Cognitieve Kunstmatige Intelligentie

Cognitieve Kunstmatige Intelligentie

Cognitieve Kunstmatige Intelligentie

Cognitieve Kunstmatige Intelligentie

Cognitieve Kunstmatige Intelligentie

Cognitieve Kunstmatige Intelligentie

Proceedings of the Fifth European Workshop on Reinforcement Learning

Edited by
Marco A. Wiering

Intelligent Systems Group
Institute of Information and Computing Sciences
Utrecht University, The Netherlands

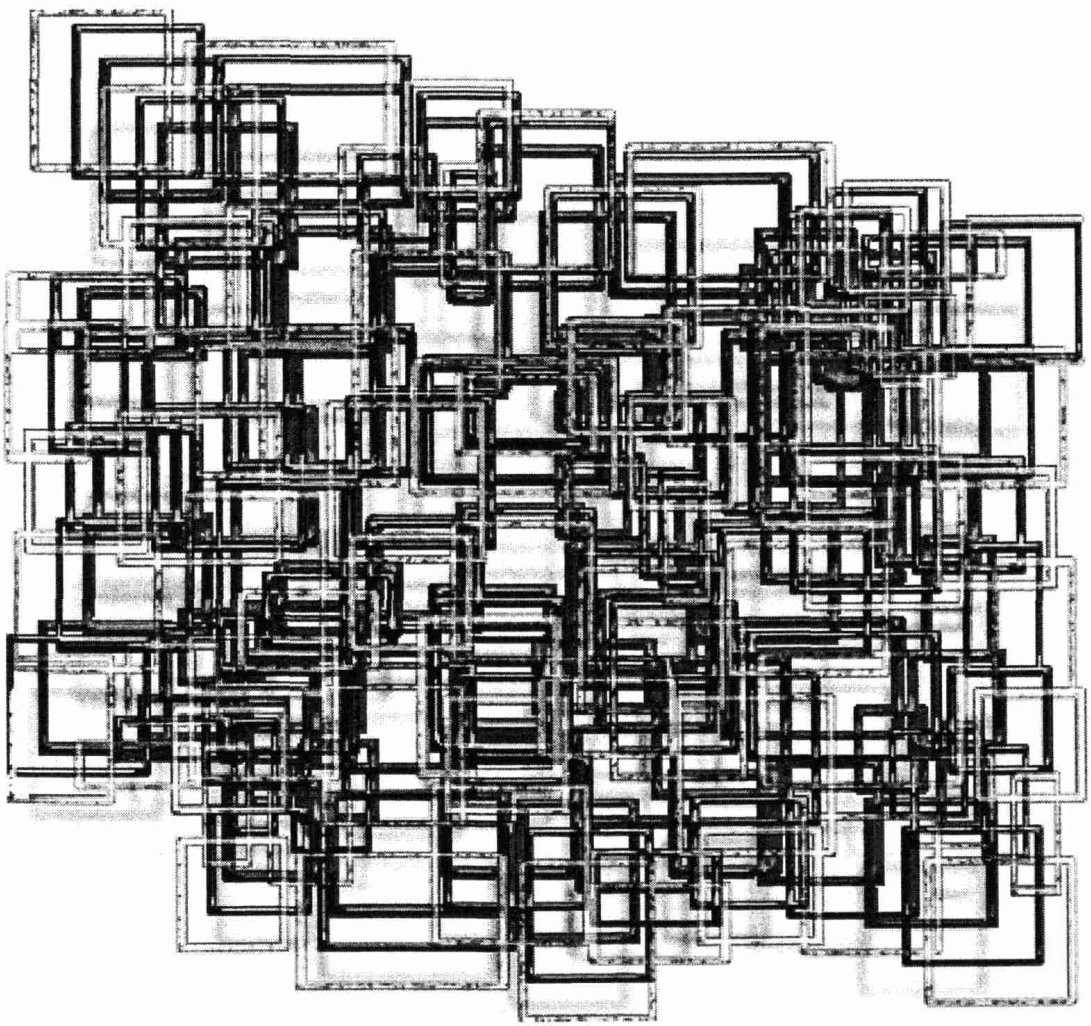
Preprint nr. 027 5-6 october 2001

Onderwijsinstituut CKI

ISBN 90-393-2874-9
ISSN 1389-5184
Prof.dr. A. Visser, Editor

Onderwijsinstituut CKI
Utrecht University
Heidelberglaan 8
3584 CS Utrecht
The Netherlands

PROCEEDINGS OF THE FIFTH EUROPEAN WORKSHOP ON REINFORCEMENT LEARNING



October 5-6, 2001 Utrecht University

Proceedings of the Fifth European Workshop on Reinforcement Learning

Edited by

Marco A. Wiering

Intelligent Systems Group
Institute of Information and Computing Sciences
Utrecht University

5-6 October, 2001
Utrecht University, The Netherlands

Contents

Preface

The Fifth European Workshop on Reinforcement Learning (EWRL-5) has gathered a wide variety of researchers interested in many different topics in reinforcement learning (RL). First of all, several papers describe RL algorithms for solving POMDPs. In this category, there is a paper by Hartley and Wyatt describing using Hidden Markov Models (HMMs), where state splitting is used to learn the right model. A paper by Dutech and Scherrer describes using contextual information for handling POMDPs, and a paper by Bakker describes using LSTM recurrent networks for solving POMDPs.

A different session is on using function approximators in RL. Arleo and Gerstner use a Hippocampal Spatial Model (similar to using Radial Basis functions) for Robotic Reinforcement Learning. Herrera and Santos study the clustering aliasing problem (CAP) which is caused by function approximators mapping different states with different optimal actions to (nearly) the same internal representation. Their approach grows Cell Structures to continue partitioning the state space to diminish the negative effects of the CAP. Shapiro studies modelling the expected time for receiving reward, which is very useful for modelling animal learning experiments.

The third session is on multi-agent RL. Gutjahr and Eder study dyadic interaction learning in which two software agents offer a single product to a population of buyers and learn to set their prices to a number of possible levels. Verbeeck et al. study learning agents in multi-agent systems where the overall performance is as good as that of the poorest performing agent. Finally, Preux et al. incrementally learn an artifact to perceive its environment, move in it, and handle and use objects. For this they use a single agent for manipulating each individual "organ" of the agent.

A fourth session is on exploration issues in RL. Wyatt and Summola study selecting an optimistic model (or using Monte Carlo sampling in the space of possible models) for exploration and describe several extensions of this method such as combining it with options (multi-time models), and Dynamic Bayesian Networks. Reynolds studies the Curse of Optimism which arises when models are initialized to large Q-values. His insight is that it is very difficult to overcome the initial value biases if these biases are optimistic.

Another session concentrates on reinforcement learning methods which search in the policy space in an unconventional manner. Strens describes a method using state-parameter evaluation functions (SPEFs) which map environmental states and policy parameters to expected return acquired by evaluating state-parameter settings. Then, optimizing the policy is done by searching for the best performing SPEF. Kwee et al. study gradient-based reinforcement planning (GREP), where an agent plans ahead and improves its policy before it actually acts in the environment. Matt and Regensburger study policy improvement for several environments at the same time. They show how an optimal stochastic policy for several environments can be computed. Then there is also some special work in RL which is not based on using value functions or conventional RL methods at all, but which describes using theoretical computational science algorithms for RL. Hutter describes an agent which can optimally learn to make sequential decisions in unknown environments by using Solomonoff's universal semi-measure for computing prior probability distributions over a set of possible models.

There is also some work on hierarchical RL. Buffet and Dutech look for scalable agents which can reuse prior knowledge and can learn to choose or combine particular behaviors. Schoknecht and Riedmiller study using multi-step actions and show that this method provides a useful way for speeding up RL in particular domains. Driessens and Blockeel describe a hierarchical RL method for learning the computer game of Digger. For this they use relational reinforcement learning (RRL), where each subproblem is first learned separately, after which they are combined by RRL.

Several novel RL algorithms for backing up state values are described by Garcia and Reynolds. Garcia describes ATD and AQ-learning, two methods for average reward RL for discounted reward problems. Reynolds describes a novel way for backing up Q-values which extends Backwards replay and truncated λ-return estimates, and shows that this method can learn faster than $Q(\lambda)$ methods.

Santos et al. describe a method for automatically tuning vector parameterized reward functions

for robotic RL applications. Finally there are several papers describing applications of RL. Hunger and Riedmiller describe using RL for a scheduling problem, and empirically evaluate the usefulness of this approach. Hing and van Harten describe an application for accepting orders in a business control framework in which agents should sometimes delay accepting orders to be able to accept more convenient orders in the future. Finally, Bennane et al. describe using RL for adapting learning situations in an intelligent tutoring system.

Organising Committee:

Marco Dorigo
Jürgen Schmidhuber
Marco Wiering

Program Committee:

Marco Dorigo
Yassine Faihe
Nicolas Meuleau
Remi Munos
Stuart Reynolds
Martin Riedmiller
Jürgen Schmidhuber
Malcolm Strens
Marco Wiering
Jeremy Wyatt

Sponsors:

The workshop was partially sponsored by SIKS (School of Information and Knowledge Systems), the Netherlands (www.siks.nl).

Other Contributors:

Department of Philosophy
Institute of Information and Computing Sciences
Utrecht University

Acknowledgments

Special thanks go to Leslie Kaelbling for giving the invited talk. We also want to thank Monique Dixon for helping us with the local organization. We are grateful to Maarten Janssen for his help with producing the proceedings. Finally, we want to thank the students Tijn van der Zant, Walter de Back, Lars Zwanepol, and Arne Koopman for helping with different aspects in the organization of EWRL-5.

Hippocampal Spatial Model for State Space Representation in Robotic Reinforcement Learning

Angelo Arleo ANGELO.ARLEO@EPFL.CH

Wulfram Gerstner WULFRAM.GERSTNER@EPFL.CH

Lab. of Computational Neuroscience, Swiss Federal Inst. of Technology Lausanne, EPFL, CH-1015, Switzerland

Abstract

We study reinforcement learning for cognitive navigation. The state space representation is constructed by unsupervised learning during exploration. As a result of learning, a stable representation of the continuous two-dimensional manifold in the high-dimensional input space is found. The representation consists of a population of localized overlapping place fields. This state-space coding is a biologically inspired model of place fields in the Hippocampus of the rat. Place fields are learned by extracting spatio-temporal properties of the environment from visual sensory inputs. Visual ambiguities are eliminated by taking into account self-motion signals via path integration. This solves the hidden-state problem and provides a robust representation suitable for applying Q-learning in continuous space. Reward-based function approximation takes place to drive action units one synapse downstream from place cells. The teaching error models the dopaminergic reward-expectation signal from neurons in the brainstem. Several action modules share the same space representation and guide the robot to multiple targets. The experimental validation of the system is done by implementing it on a mobile Khepera robot.

1. Introduction

Problems with high-dimensional continuous state space are a critical issue in reinforcement learning (Santamaría et al., 1998). In these cases, optimizing the value function (i.e., deriving the function predicting the optimal long-term reward for a given state when selecting a specific action and following an optimal policy thenceforth) implies to learn the optimal mapping over infinitely many state-action pairs. Since exploring the whole state-action space is unfeasible, a key issue in these problems is the ability of the agent to estimate the expected value function for never experienced state-action pairs. Combining reinforcement learning with function approximation methods provides such a *generalization* property (Santamaría et al., 1998; Sutton & Barto, 1998; Lin, 1992; Rummery & Niranjan, 1994).

We study reward-based action learning to provide a robot with cognitive navigation capabilities. We ask two questions: *(i)* How can an agent learn a suitable state-space representation based on its high-dimensional and continuous sensory inputs? *(ii)* How can goal-oriented action selection be done based on the learned state-space coding?

We adopt a neuro-mimetic approach and we study spatial learning capabilities of rodents (Arleo & Gerstner, 2000). In particular, we model *(i)* the functional role of the hippocampal formation in mediating space coding in rats (O'Keefe & Nadel, 1978), *(ii)* the interaction between the hippocampus and the ventral striatum (which includes the nucleus accumbens) for mapping spatial information into actions (e.g., goal-oriented movements). Since neurons in the ventral part of the striatum are primarily activated in relation to the expectation of rewards (Schultz et al., 1992), we model such interaction through reinforcement learning.

2. State-space Representation

Neurophysiological findings suggest the spatial self-localization of rodents is supported by place-sensitive and direction-sensitive cells. *Place cells* in the rat Hippocampus provide a spatial representation in allocentric coordinates (O'Keefe & Nadel, 1978). A place cell exhibits a high firing rate only when the animal is in a specific region of the environment, which defines the *place field* of the cell. *Head-direction cells* encode the animal's allocentric heading in the azimuthal plane (Taube et al., 1990). A directional cell fires maximally only when the animal's heading is equal to the cell's *preferred direction*, regardless of the orientation of the head relative to the body, of the rat's location, or of the animal's behavior.

We present a computational model which is consistent with several neurophysiological data concerning place and head-direction cells (Arleo & Gerstner, 2000; Arleo & Gerstner, 2001). Place-coding and directional sense are provided by two coupled neural systems that interact with each other to form a unitary spatial learning system. We stress the importance of integrating allothetic (e.g., visual stimuli) and idiothetic (e.g., self-motion cues) information to establish a stable space code.

High-dimensional visual inputs (images have a resolution of 422×316 pixels) are processed by means of an unsupervised learning technique that allows the robot to detect

a convenient low-dimensional view manifold representing the visual input space. However, when relying on visual data only, the state-space representation encoded by place cells does not fulfill the Markov hypothesis (McCallum, 1996). Indeed, distinct areas of the environment may provide identical visual cues and lead to singularities in the view manifold (*sensory input aliasing*). We employ idiothetic signals (i.e., path integration) along with visual inputs in order to remove such singularities and solve the hidden-state problem. On the one hand, unreliable visual data are compensated for by means of path integration. On the other hand, reliable visual information can calibrate the path integrator system and maintain the dead-reckoning error bounded over time. Correlational learning is applied to combine visual cues and path integration over time.

After learning, the space representation consists of a population of localized overlapping place fields (Fig. 1) covering the two-dimensional workspace densely. Our place fields provide a "natural" set of basis functions in the sense that we do not have to choose parameters like width and location of the basis functions. Rather, the basis functions are created automatically by unsupervised learning. In order to interpret the information represented by the ensemble pattern of activity of our place cells, we employ *population vector coding* (Wilson & McNaughton, 1993).

3. Action Learning: Goal-oriented Navigation

Our spatial learning system provides an *incrementally learned coarse coding representation* suitable for applying reinforcement learning for continuous state spaces. Learning an action-value function over a continuous location space endows the system with spatial generalization capabilities. Since our state-space code solves the problem of ambiguous input or partially hidden states, the current state is fully known to the system and reinforcement learning can be applied in a straightforward manner.

We take a population $\mathrm{A} = \{a \,|\, 1 \leq a \leq m\}$ of *locomotor action neurons* one synapse downstream from our place cells p. Each cell a provides an allocentric directional motor command (e.g., go north). Then, the navigation problem is: How can we establish a mapping function $\mathcal{M} : \mathcal{P} \rightarrow \mathcal{A}$ from the place cell activity space $\mathcal{P}$ to the action space $\mathcal{A}$ to achieve goal-directed behavior? We apply Q-learning to approximate this function from the continuous space of physical locations to the activity space of action cells based on the agent's experience. The robot interacts with the environment, and reward-related stimuli elicit the synaptic changes of the $p \rightarrow a$ projections w_p^a to adapt the action-selection policy to the task. After training, the hippocampally-dependent activity of cells $a \in A$ provides a map to support goal-oriented navigation and obstacle avoidance.

Let r_p be the activation of a place cell p. Each state $\vec{s}$ is encoded by the ensemble activity vector $\vec{r}(\vec{s}) = (r_1(\vec{s}), \ldots, r_n(\vec{s}))$, where n is the number of place cells. The state-action value function $Q_w(\vec{s}, a)$ is of the form

$$Q_w(\vec{s}, a) = (\vec{w}^a)^T \vec{r}(\vec{s}) = \sum_{p=1}^{n} w_p^a \, r_p(\vec{s}) \qquad (1)$$

where $\vec{s}, a$ is the state-action pair, and $\vec{w}^a = (w_1^a, \ldots, w_n^a)$ is the adjustable parameter vector. The learning task consists of updating the vector $\vec{w}^a$ to approximate the optimal function $Q_w^*(\vec{s}, a)$. The state-value prediction error is

$$\delta_t = R_{t+1} + \gamma \max_a Q_t(\vec{s}_{t+1}, a) - Q_t(\vec{s}_t, a_t) \qquad (2)$$

where R_{t+1} is the immediate reward, and $0 \leq \gamma \leq 1$ is a constant discounting factor. At each time step the weight vector $\vec{w}^a$ changes according to

$$\vec{w}_{t+1}^a = \vec{w}_t^a + \alpha \delta_t \vec{e}_t \qquad (3)$$

where $0 \leq \alpha \leq 1$ is a constant learning rate parameter, and $\vec{e}_t$ is the eligibility trace vector. During learning, the exploitation-exploration trade-off is determined by an ϵ-greedy policy, with $0 \leq \epsilon \leq 1$. As a consequence, at each step t the agent might either behave greedily (*exploitation*) with probability $1 - \epsilon$, by selecting the best action a_t^* with respect to the Q-value functions, $a_t^* = \mathrm{argmax}_a Q_t(\vec{s}_t, a)$, or resort to uniform random action selection (*exploration*) with probability equal to ϵ. The update of the eligibility trace depends on whether the robot selects an exploratory or an exploiting action. Specifically, the vector $\vec{e}_t$ changes according to (we start with $\vec{e}_0 = \vec{0}$)

$$\vec{e}_t = \vec{r}(\vec{s}_t) + \begin{cases} \gamma\lambda\vec{e}_{t-1} & \text{if } \textit{exploiting} \\ 0 & \text{if } \textit{exploring} \end{cases} \qquad (4)$$

where $0 \leq \lambda \leq 1$ is a trace-decay parameter. Learning consists of a sequence of paths starting at random positions and determined by the ϵ-greedy policy. When the robot reaches the target, a new training path begins at a new random location. After learning, population vector decoding is applied to map $\mathcal{A}$ into a continuous action space $\mathcal{A}'$ by averaging the ensemble action cell activity. Fig. 2 shows an example of navigation map learned after only 5 training trials. The vector field representation of Fig. 2 has been obtained by rastering uniformly over the environment. Many of sampled locations were not visited by the robot during training which confirms the generalization capabilities of the method: the robot was able to associate appropriate goal-oriented actions to never experienced spatial states.

We assume that the anatomical counterpart of our action cells a are neurons in the nucleus accumbens (NA). NA neurons receive dopaminergic afferents related to the occurrence of unconditioned or conditioned reward-related

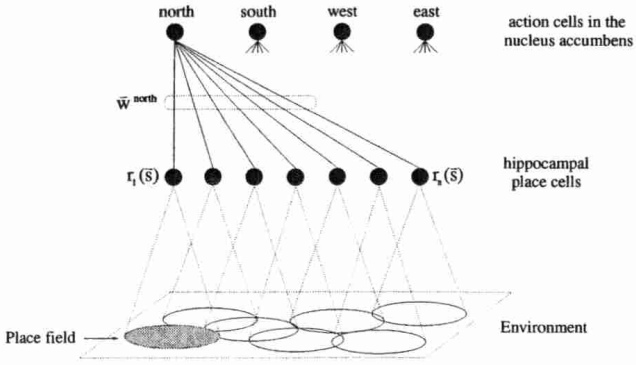


Figure 1. The overlapping place fields work as basis functions. A discrete set of actions $A = \{north, south, west, east\}$ is considered, which results in four Q-functions $Q(\vec{s}, north)$, $Q(\vec{s}, south)$, $Q(\vec{s}, west)$, $Q(\vec{s}, east)$ to be approximated.

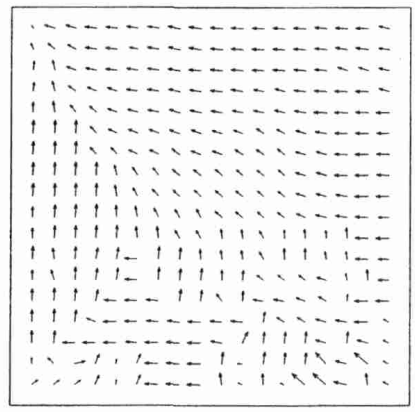


Figure 2. Vector field representation of a navigational map learned after 5 trials. The target area (about 2.5 times the area occupied by the robot) is the upper-left corner of the arena.

events (Schultz et al., 1992). The presence of dopamine-dependent plasticity in NA suggests that dopamine responses might be involved in plasticity changes yielding reward-based learning. Indeed, since the activity of dopamine neurons is a function of the unpredictability of a reward, dopaminergic activity might work as a prediction error signal (i.e., the difference $R(t) - R'(t)$ between the actual reward $R(t)$ at time t and the predicted reward $R'(t)$) suitable for learning.

4. Conclusions

Reinforcement learning takes long training time when applied directly on high-dimensional input spaces (Sutton & Barto, 1998). We have shown that by means of an appropriate state space representation, based on localized overlapping place fields, the robot can learn goal-oriented behavior after only 5 training trials. This is similar to the escape platform learning time of rats in Morris water-maze (Morris et al., 1982).

References

Arleo, A., & Gerstner, W. (2000). Spatial cognition and neuro-mimetic navigation: A model of hippocampal place cell activity. *Biological Cybernetics, Special Issue on Navigation in Biological and Artificial Systems*, *83*, 287–299.

Arleo, A., & Gerstner, W. (2001). Spatial orientation in navigating agents: Modeling head-direction cells. *Neurocomputing*, *38-40(1-4)*, 1059–1065.

Lin, L. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, *8*, 293–321.

McCallum, R. (1996). Hidden state and reinforcement learning with instance-based state identification. *IEEE Systems, Man, and Cybernetics*, *26(3)*, 464–473.

Morris, R., Garrud, P., Rawlins, J., & O'Keefe, J. (1982). Place navigation impaired in rats with hippocampal lesions. *Nature*, *297*, 681–683.

O'Keefe, J., & Nadel, L. (1978). *The Hippocampus as a cognitive map*. Oxford: Clarendon Press.

Rummery, G., & Niranjan, M. (1994). *On-line Q-learning using connectionist systems* (Technical Report CUED/F-INFENG/TR 166). Engineering Department, Cambridge University.

Santamaría, J., Sutton, R., & Ram, A. (1998). Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive Behavior*, *6(2)*, 163–218.

Schultz, W., Apicella, P., Scarnati, E., & Ljungberg, T. (1992). Neuronal activity in monkey ventral striatum related to the expectation of reward. *Journal of Neuroscience*, *12*, 4595–4610.

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

Taube, J., Muller, R., & Ranck, J. (1990). Head direction cells recorded from the postsubiculum in freely moving rats. I. Description and quantitative analysis. *Journal of Neuroscience*, *10*, 420–435.

Wilson, M., & McNaughton, B. (1993). Dynamics of the hippocampal ensemble code for space. *Science*, *261*, 1055–1058.

Reinforcement learning in non-Markovian domains using LSTM recurrent neural networks

Bram Bakker BBAKKER@FSW.LEIDENUNIV.NL
Department of Psychology, Leiden University, P.O. Box 9555, 2300 RB, Leiden, The Netherlands

Abstract

Hidden state in non-Markovian reinforcement learning problems (POMDPs) may be resolved by maintaining a memory of past events, e.g. using recurrent neural networks. This work employs the recently proposed Long Short Term Memory recurrent neural network for that purpose, using it to approximate the value function of model-free Advantage(λ) learning. The results show that this combination works well in tasks with complex and long-term dependencies between past events and the optimal policy.

1. Introduction

There are a number of techniques for solving non-Markovian (partially observable, hidden state) reinforcement learning problems. Many of them rely on remembering (part of) the history of past observations and actions, e.g. using fixed or variable size history windows (delay lines), memory bits that the controller can switch on and off, finite state automata, or recurrent neural networks. The general idea is that the Markovian environmental state may be inferred from the combination of the current, possibly ambiguous observation and the representation of the past.

The learning task becomes increasingly difficult for most if not all of these techniques when a relevant piece of information lies further back in the past. For example, it might be that the decision whether to go left or right at a T-junction in a maze should depend on an observation from many timesteps ago. It also becomes difficult when the relationship between past observations/actions and the best action at the current timestep is complex. For example, it might be that the decision whether to go left or right at the T-junction should depend not on a single observation in the past, but on a specific combination of past observations.

2. LSTM

Similar problems appear in supervised learning of timeseries data. This work explores the use of one recently developed technique that was successful in overcoming these problems in that context: Long Short Term Memory (LSTM) recurrent neural networks (Hochreiter & Schmidhuber, 1997). An LSTM network can learn complex, long-term dependencies between relevant events in timeseries data, because it enforces non-decaying error flow propagated back in time in a number of specialized units, called memory cells. Nevertheless, LSTM's learning algorithm is local in space and time (which fits in nicely with online reinforcement learning), and its update complexity is only $O(1)$.

In this work, an LSTM network is used to directly approximate the value function of model-free reinforcement learning, in the same spirit as other model-free recurrent neural network approaches (Lin & Mitchell, 1993). LSTM could also be used as the basis of a model-based system, where it could learn a predictive model of the environment, with predictions depending on information from long ago. One possible advantage of a model-free approach over a model-based approach is that the system may learn to only disambiguate states insofar as that is useful for obtaining higher returns, rather than waste time and resources in trying to predict features of the environment that are irrelevant with respect to returns. The outputs of the LSTM network represent the values of different actions. The state of the environment is approximated by the current observation, which is the input to the network, together with the representation of the past, encoded by the recurrent activations in the network (see Bakker, 2001 for further details).

3. Advantage(λ) learning

The particular reinforcement learning algorithm used here is (direct) Advantage learning (Harmon & Baird, 1996), which was originally designed as a variation of

and improvement on Q-learning for continuous-time problems. It turns out to also perform much better than Q-learning in the discrete-time problems investigated in this work. Similar to continuous-time problems, discrete-time problems with long paths to rewards result in small differences between the values of adjacent states, and Advantage learning deals with that more effectively.

Eligibility traces have often been shown to improve learning considerably, especially in non-Markovian tasks. For this reason, they were added to Advantage learning, yielding Advantage(λ) learning. Furthermore, a directed exploration scheme was used, which effectively increases exploration when estimated Advantage values for the current observation or in the nearby future have a lot of uncertainty.

4. Test problems

The ability to learn long-term dependencies was investigated in a simple delayed reward T-maze navigation task, where the best action at the T-junction depended on a single observation from long ago. The system was able to learn this relationship reliably when the minimal number of intervening state-action pairs was as high as 50, even in the presence of severe noise in the intervening observations. Beyond 50 intervening steps, performance gradually becomes worse.

A similar T-maze task was used to test the ability to deal with more complex temporal regularities. The sequence of observations in the corridor leading up to the T-junction was either "grammatical" or "ungrammatical", where grammatical sequences corresponded to a simple non-regular language, $a^n b^n$. In the limit, such a language cannot be represented by a finite state automaton (FSA), but requires a higher class of automaton (with a stack or counter). Grammatical sequences indicated that the agent should turn right at the T-junction, ungrammatical sequences indicated that the agent should turn left. The agents reliably learned this task. Moreover, they could generalize to sequences (corridors) of much greater length than the ones they experienced during learning—which is something an induced FSA would not automatically do. Inspection of the internal state of the networks shows that they induce counters. This appears to be the first time that reinforcement learning systems induce automata of a higher computational class than FSAs, and it shows that a reinforcement learning system based on LSTM can learn complex relationships between sequences of past observations and the optimal policy in non-Markovian domains.

A third learning task was a difficult variation of pole balancing. It had two sources of hidden state. First, the (continuous) cart velocity and pole velocity were not part of the observation (as in Lin & Mitchell, 1993). Second, the controller had to learn to operate in two different modes, in which the two actions corresponded to opposite push directions. The (discrete) information which mode the controller was operating in was only part of the observation for the first second of simulated time, and had to be remembered indefinitely after that. In all cases, the controllers learned to balance the pole for hundreds or thousands of timesteps, and in some cases, the controller even learned to balance the pole indefinitely in both modes of operation. This shows that the system can learn to approximate the continuous velocities, and at the same time learn to remember the discrete mode information for long periods of time or even indefinitely.

5. Conclusion

In all experiments, when the LSTM network was replaced by a Simple Recurrent Network trained using backpropagation through time (Lin & Mitchell, 1993), learning performance was significantly worse, in most cases not reaching a satisfactory policy. Furthermore, Advantage learning worked much better than Q-learning. The system with eligibility traces ($\lambda > 0$) significantly outperformed the system without eligibility traces ($\lambda = 0$). Finally, using directed exploration rather than undirected exploration was crucial for the two T-maze tasks. In conclusion, the LSTM recurrent neural network, combined with model-free Advantage(λ) learning and directed exploration, seems to be a promising technique for dealing with non-Markovian reinforcement learning problems with complex and long-term dependencies between past events and the optimal policy.

References

Bakker, B. (2001). *Reinforcement learning with LSTM in non-Markovian tasks with long-term dependencies* (Technical report). Dept. of Psychology, Leiden University.

Harmon, M. E., & Baird, L. C. (1996). *Multi-player residual Advantage learning with general function approximation* (Technical report). Wright-Patterson Air Force Base.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation, 9 (8)*, 1735–1780.

Lin, L.-J., & Mitchell, T. (1993). Reinforcement learning with hidden states. In *Proc. of the 2nd int. conf. on simulation of adaptive behavior*. MIT Press.

Applying Reinforcement Learning To ITS For Adapting Learning Situations

BENNANE Abdellah ; MICHIELS Isabel ; MANDERICK Bernard & D'HONDT Theo

ABENNANE@VUB.AC.BE ; ISABEL.MICHIELS@VUB.AC.BE ; BERNARD@ARTI.VUB.AC.BE & TJDHONDT@VUB.AC.BE

Vrijie Universiteit Brussel

2, pleinlaan, B–1050 Brussels

Belgium

Abstract

During the development of intelligent tutors, several techniques were applied resulting from the artificial intelligence and machine learning. The goal is to develop a system, which imitates human behavior in these various cases and in fact in (1) the reasoning process and in (2) the level of interactivity with its environment.

An intelligent tutor is composed of the following entities: (1) the domain expert (2) the tutor (pedagogical module) (3) the student model and (4) an interface of communication between expert – tutor on the one hand and student the other.

The expert interacts with the learner according to the situation selected by the tutor. Following the action of the learner, the expert reacts to evaluate the behavior of the learner knowing that the action of this last is carried out successfully or failure. In both cases, the expert presents an adequate feedback to the learner. The tutor observes the interaction between learner and the expert and tries to direct the communication according to the objective to be reached and the abilities of the learner.

In this paper, we expose the entities subjects of interaction in an intelligent tutor system and we discuss how reinforcement learning can be applied to the tutoring system in order to individualize and to adapt to the generation of learning situations.

Key words: learning Situation, Tutoring, Reinforcement learning, ITS, Agent, Environment.

1. Introduction

Using communication and information technology for teaching and training is an idea, which dates at least from the beginning of the last century. The majority of researchers (BRUI, 97; DEP, 87) consider that the starting point was with the machine built by Pressey in 1926. The objective of Pressey was to build a device which makes it possible to raise a question, and after the answer of the pupil, to communicate the correctness of the answer immediately to him. It considers that the true work of the teacher is to concentrate on the activities stimulating thought, to release it from the repetitive spots and to delegate the training exercises to a machine.

At the beginning of the Sixties, the computer–assisted instruction (CAI) comes to finally seem a true marriage of programming instruction and data processing (ALBE, 92).

The two aspects interesting of the CAI were (1) the individual character of the program of teaching: the learner, with the programmed machine, has the feeling to be with a system charged to provide him a particular course, adapted to his needs and his level; and (2) the interactive aspect which, if it is agreeably exploited, avoids with the learner sinking in passivity (LESC, 85).

The applications of CAI, with or without audio–visual dimension, were limited to the organization of information presented at the screen according to a preset sequence often under the program testing (DEP, 99). In the Seventies, a limited number of researchers defined new and an ambitious objective for the CAI. They adopted the human tutor like that of the educational model and applied the artificial intelligence techniques.

The objective of the Intelligent Tutoring systems (ITS) is to engage the students in continuous activities of reasoning and to interact with the student based on a deep comprehension of its behaviors (ANDE, 97).

The possibility of storing and executing the teaching domain knowledge constitutes the keystone of intelligent tutorials. The representation of this knowledge makes it possible not only to treat the answers, but to penetrate in the process even of problems solving (DILL, 93). One must be conscious owing to the fact that research on the intelligent tutors systems is far from the ideal goal which is to create a completely autonomous system in its teaching reasoning (DEP, 99).

In this paper, we expose the entities subjects of interaction in an intelligent tutor and we discuss how reinforcement learning can be applied to the tutoring system in order to individualize and to adapt to the generation of learning situations.

2. Components of the interaction unit

Recall that, the pedagogical module is a set of rules and protocols that manages resources and interactive communication and following the learner's needs.

The pedagogical module has to fulfill the following tasks:

· Evaluate the learner actions and determine the values of the transition parameters, the rewards, and the learner path;

· Select the learning situations from the database and follow the orders of the evaluation unit and present it to the user;

· Look for and send the rewards to learner following the orders of the evaluation unit.

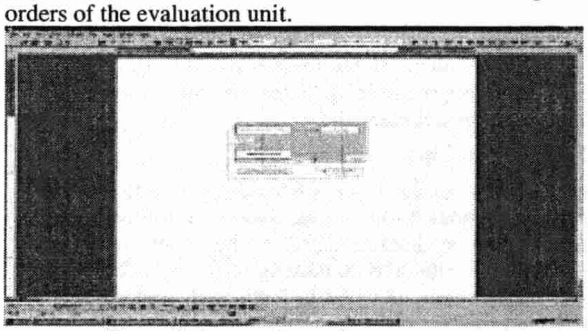


Figure 1: interaction unit components

2.1 Evaluation unit

The evaluation unit is the core of the pedagogical module. It is connected to the selection and presentation unit of the training situation, to the reward unit and to the student model.

It is designed according the principles of the inferences and the transitions in a training sequence.

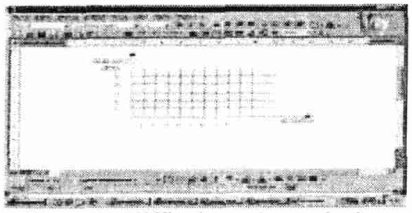


Figure 2: Difficulty and complexity

We have four principles: success, failure, remediation and high–level principles (BENN, 2000).

Success principle:

For every situation executed with success, the transition to the following situation, it doesn't to make vertically, but does generally from low toward the high and on the right, but also horizontally and follow the case of the following situation variable.

Failure principle:

For every situation executed with failure, the transition to the following situation, makes following the cases presented below:

1. One decrements the degree of situation difficulty and one maintain the degree of complexity.
2. If the degree of situation difficulty in progress is inferior to three (3), and if the situation is executed with success, then one increments the degree of difficulty and one maintain the degree of complexity.

We consider that, all situations having a degree of difficulty lower than the average (3) are conceived in order to be learned, to surmount the difficulties and to reach the optimal level.

High–level principle:

If the difficulty degree of the current situation reaches the ceiling and if it is executed with success, then the transition to the following situation makes of the following manner: one increases the meter of the measurement situations (complexity degree) and one maintains the maximal rank (partial rank) of the situation difficulty degree.

The remediation principle:

If the difficulty degree in progress situation reaches the lowest level and if the activity of the learner is executed with failure, then the transition to the following situation makes of the following manner: one decreases the meter of the measurement situations (transition to the column precursor) in order to see again the previous situation for to remedied the deficiencies.

2.2 Transition Unit

The transition unit loads on one hand, the selection of the training situations from the database and following the orders of the evaluation unit, on the other hand, the presentation of situation containing to the user. This unit has two objectives, the one is functional, and concerns search and the recuperation of the information and send it to the presentation function, the other is ergonomic and concerns the choice of the forms associated to the situation in progress and following situation parameters.

2.3 Reward Unit

The object of the reward unit is choosing and sending the adequate feedback to the learner and following his action. The sent message to the learner could be (1) an encouragement following the action executed with success, or (2) some indications in order to complete the instructions to undertake, or (3) a message containing the correct answer. The messages generate since the orders received from the evaluation unit.

3. Tutoring as a reinforcement learning problem

Reinforcement learning is an interesting technique for solving learning problems. It requires a signal of reinforcement that is the only feedback of the environment toward the agent. The agent receives continuously some sensory information of the environment called states. It chooses and selects the actions that act on the environment and after every action it receives the environment rewards (SUTT, 98). The goal of learning is to obtain the optimal policy that maximizes the rewards and the agent's performance.

3.1. Mechanism of training as reinforcement learning

We begin by defining the mechanism of training like a reinforcement–learning problem and by ending we will generate a target functions algorithm. In this order, what is the graph of a pedagogical sequence?

A training sequence is defined like a directed graph :

- Every situation represents a node (or state) of graph;
- The transition relations between nodes represent the actions and their associate rewards (links).
- Every sequence admits an initial and final situation, respectively the start node (start state) and goal node (goal state).

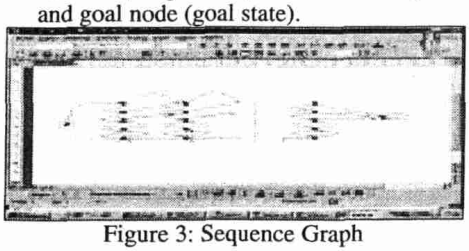


Figure 3: Sequence Graph

3.2. Transition function

One could represent this graph by a matrix line–column, then determine the representation of states, actions, rewards.

States representation:

S is a matrix of 5 lines and N columns representing the situations of a training sequence.

Actions representation:

Normally the actions achieved by the student are:

- Choose an answer (Closed question: MCQ[1], T–F[2], MAQ[3]);
- Type an answer (Open question);
- Unroll a demonstrative situation.

The agent could execute some other actions, like asking for help or giving hints, make some counts using a utilitarian and come back to the situation, etc.

But our interest here is the answer to the following question: Are the action is executed by success or failure? Knowing that { state + action} gives like result the following state, and our goal is to determine the outcome of every pair state–action. We can summarize by giving the actions set: $\{ A^{success}, A^{Failure} \}$

Rewards representation:

[1] Multiple Choices Question.
[2] True – False.
[3] Multiple Answers Question.

Recall that we have five situations s1, s2, s3, s4, s5. Each one of them, is executed with some probabilities: α, β, λ, μ, ρ.
The reward $R^{a}_{ss'}$ from the current state (s) to next state (s') by executing the action (a) are: $R^{success}$, $R^{failure}$.

3.3. Integration of the success and failure functions

In a concrete situation of learning, the student chooses an answer (closed question) or types an answer (open question) ended by validation. The feedback of the answer validation gives like outcome the rightness of the answer: success or failure. Below, we address the algorithm for these two functions.
What data do we need, if we want to make the connection with a reinforcement–learning problem?
We think that we need four things, a set of states, a set of actions, a reward function and policy. Our interest is to develop the policy algorithm, denoted as T.
Then, what is the algorithm of this mapping?
First, we signal that the set of actions will be {0, 1} such that the element 0 represent the failure action and the element 1 represent the success action.

```
T(s, i, j, a)                    // 1 ≤ i ≤ 5; 1 ≤ j ≤ N; a ∈ {0, 1}.
{
case a = 0                       //failure
{
load s(i,j) // current state;
if (i >1)
then s'← s(i −1,j);
if (i = 1)
then
{
        (1) load previous situation s(.,j−1); // Previous column.
                (2) s'← s(partial_min, j−1); // 1 ≤ partial_min ≤ 3.
}
case a = 1                       //success
{
load s(i,j); // current state;
if (i >= AVG) //AVG: Average..
then
{
load next situation s(.,j+1); // next column.
 partial_max ← max (s(.,j+1)); // 3 ≤ partial_max ≤ 5.
if (i < partial_max)
then s'← s(i+1,j+1)
else s'← s(partial_max, j+1);
        }
        if (i < AVG)
        then s'← s(i+1, j);
}
}
```

4. Conclusion

In this paper we concentrated on the modelisation of a tutoring systems and the interaction between its components. We demonstrate that the use of modern reinforcement learning techniques can give interesting results for enhancing tutoring systems. We believe that this combination was possible because we reformulated the question like a problem of the reinforcement learning by determining the whole of the states and the corresponding actions, the rewards and the policy and the algorithm correspondents. We believe that the contest and the multiplication of the efforts in concrete experiments can conclude the introduction and the use of the new techniques resulting from the IA in the design process of the intelligent tutors.

5. References

(ALBE, 92): ALBERTINI J. M. , La pédagogie n'est plus ce qu'elle sera. Le Seuil, Presses du CNRS, 1982.
(ANDE, 97): ANDERSON J. R., CORBETT A. T. & KOEDINGER K. R.: Intelligent tutoring systems. Handbook of human–computer interaction, Second completely revised Edition, ELSEVIER, 1997.
(BENN, 2000): BENNANE A., Processus de conception des tuteurs intelligents et l'apprentissage renforcé ; VUB Bruxelles 1999.
(BRUI, 97): BRUILLARD E., Les machines à enseigner. Editions Hermes, Paris, 1997.
(DEP, 87): DEPOVER C. , L'ordinateur média d'enseignement, DeBoeck, Bruxelles, 1987.
(DEP, 99): DEPOVER C. , GIARDINA M. & MARTON P. , Les environnements d'apprentissage multimédia ; L'Harmattan, Paris, 1999.
(DILL, 93): DILLENBOURG P., Evolution épistémologique en EIAO. Ingénierie Educative. Sciences et Techniques Educatives. 1 (1), 39–52, 1993.
(LESC, 85): LESCORT A. , Intelligence Artificielle et systèmes experts ; CEDIC NATHAN, Paris, 1985.
(SUTT, 98): SUTTON, R. & BARTO A. G.; Reinforcement Learning, A Introduction; A Bradford Book, 1998.

Looking for Scalable Agents

Olivier Buffet BUFFET@LORIA.FR

Alain Dutech DUTECH@LORIA.FR

LORIA/INRIA, BP 239, 54506 Vandœuvre-lès-Nancy, FRANCE

Introduction

Reinforcement Learning intends to ease and possibly to perform automatically the design of systems such as software or robot agents. An important aspect is the ability of learning agents to adapt to their environment and to the task they have to accomplish. This kind of learning is unfortunately restrained by problems like combinatorial explosion of the state space that limits the number of sensors or objects an agent can reasonably deal with, especially in the case of Multi-Agent Systems.

Considering Markov Decision Processes, different solutions exist to overcome the difficulties related to large state spaces: hierarchical structures (Parr, 1998) or factored representations (Kearns & Koller, 1999) of the agent's behavior for example. Nevertheless, these tools require manual preparations before going through the learning step, and result in an agent designed for a specific task.

The work presented here intends to define agents able to be efficient in several complex situations, reusing prior knowledge (see also (Dixon et al., 2000)). The design is based on the use of many basic behaviors which the agent will have to manage, each behavior corresponding to a different motivation. For now, Reinforcement Learning is mainly employed for the "recording" of these basic behaviors. Nevertheless there is place for improvements of our framework through other uses of Reinforcement Learning.

Basic behaviors

For each motivation, a basic behavior b is learned as a stochastic policy mapping perceptions to actions (Dutech et al., 2001). And for a given behavior, perceptions only consider subsets of the set of all objects and are called **configurations**. Useful data concerning each behavior are stored in two tables:

- $\theta_b(c, a)$: gives the probability to choose action a while seeing configuration c, and
- $Q_b(c, a)$: is the expected discounted reward when choosing action a for the configuration c.

For a given behavior, both tables are learned through Reinforcement Learning. More precisely, a gradient descent is used to search for the stochastic policy π_{θ_b}, and afterwards the Q_b-table can be learned while the agent behaves according to the stochastic policy.

An agent confronted to a single configuration c of a behavior b will simply use $\theta_b(c, .)$ to make the choice of its next action. If this agent has to deal with more configurations, from possibly different behaviors, the θ-tables are not sufficient to weight the relative importance of the different decisions that can be taken. The Q-tables will thus give a measure of current motivations.

Scene decomposition

When choosing among its available actions, an agent has to analyse the present situation. This is done by searching in the scene which configurations are linked to behaviors, and are therefore of interest. If $\mathcal{P}$ is the perception, the useful configurations are in

$$\mathcal{C} = \{c \subset \mathcal{P} \mid \exists b\,;\; Q_b(c, .) \textit{ and } \theta_b(c, .) \textit{ are well defined}\}.^1$$

A configuration may be related to different behaviors, leading to the use of

$$\mathcal{B}(c) = \{b \mid Q_b(c, .) \textit{ and } \theta_b(c, .) \textit{ are well defined}\}.^1$$

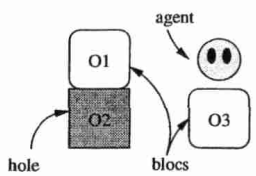


Figure 1. A not so simple scene
In the scene shown on this figure, the agent perceives objects O_1, O_2 and O_3. With two behaviors: *a*voiding holes (b_a) and *p*ushing blocs in holes (b_p), the agent has to manage three different $(behavior, configuration)$ pairs: $(b_a, \{O_2\})$, $(b_p, \{O_1, O_2\})$ and $(b_p, \{O_3, O_2\})$.

[1] By "well defined", we mean that the Q_b and θ_b tables are defined for configuration c. For a given behavior b, only a restricted subset of all possible configurations are of interest, and are therefore defined.

Basic behaviors combination

Once the scene is decomposed, the agent has to use its knowledge to decide its next action. This is done by calculating a weighted distribution $\Theta(a) = \frac{1}{K}\sum_{c\in C}\sum_{b\in B(c)} w(b,c,a).\theta_b(c,a)$ for each action a, resulting in a probability distribution over actions. In this formula, the $w(b,c,a)$ are positive functions that can be defined in various ways, and K is a normalizing factor.

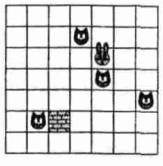
The basis of this computation is a heuristic lacking of theoretical support. Nevertheless, this part of the method is subject to discussion, as tools like fuzzy logic (Zadeh, 1973) or learned parameters could be efficiently used. For now our experiments only present different w functions that helped us analyze the problem.

Some experiments

One task we tried to deal with thanks to this method was for four predators to catch a prey while avoiding a wall. Two basic behaviors: *hunt* and *avoid* were already known (learned θ- and Q-tables) and had only to be adequately combined to obtain Θ. Five methods have been tried:

1. *hunt* behavior alone: $\Theta(a) = \theta_{hunt}(c_h, a)$,
2. *avoid* behavior alone: $\Theta(a) = \theta_{avoid}(c_a, a)$,
3. both behaviors with equal weights: $\Theta(a) = \frac{1}{2}[\theta_{hunt}(c_h, a) + \theta_{avoid}(c_a, a)]$,
4. both behaviors weighted by $w(b,c) = \max_a\{abs(Q_b(c,a))\}$, and
5. *hunt* behavior with certain moves forbidden if $Q_{avoid}(c,a) < 0$.

Some results are shown on table 1 for this particular application.



method	*#captures*	*#hits*
hunt	802	349
avoid	1	0
3	91	65
4	790	220
5	834	0

Table 1. Results in a 7 × 7 grid-world
(Number of prey-captures and wall-hits for one predator in 10000 time steps.)

Results and Perspectives

The method was tried on two different problems: 1- predators catching a prey while avoiding a wall, and 2- agents having to merge pairs of blocs (with different possible fusions at the same time). These examples involve various forms of motivations (reaching a goal or avoiding an injury) and various relations between them (different levels of priority, or equal priority motivations). Whereas it illustrated the complexity of a behavior composed of many basic behaviors, the efficiency obtained through this method is encouraging.

The next step will be to learn parameters so as to better tune the relative importance of the different behaviors. But we also would like to dynamically learn hierarchies of basic behaviors, and automatize the creation of a behaviors' library.

References

Dixon, K., Malak, R., & Khosla, P. (2000). *Incorporating prior knowledge and previously learned information into reinforcement learning agents* (Technical Report). Carnegie Mellon University, Institute for Complex Engineered Systems.

Dutech, A., Buffet, O., & Charpillet, F. (2001). Multi-agent systems by incremental gradient reinforcement learning. *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI-01.*

Kearns, M., & Koller, D. (1999). Efficient reinforcement learning in factored mdps. *Proceedings of the 16th International Joint Conference on Artificial Intelligence, IJCAI'99, Stockholm.*

Parr, R. (1998). *Hierarchical control and learning for markov decision processes.* Doctoral dissertation, Computer Science, University of California at Berkeley.

Zadeh, L. A. (1973). Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Transactions on Systems, Man and Cybernetics, SMC-3*, 28–44.

Learning Digger using Hierarchical Reinforcement Learning for Concurrent Goals.

Kurt Driessens KURT.DRIESSENS@CS.KULEUVEN.AC.BE
Hendrik Blockeel HENDRIK.BLOCKEEL@CS.KULEUVEN.AC.BE
Department of Computerscience, K.U.Leuven, Celestijnenlaan 200A, B-3001 Leuven, Belgium

Hierarchical Reinforcement Learning is often used for reinforcement learning problems that are too large to be handled directly. Most techniques (Kaelbling et al., 1996; Tadepalli & Dietterich, 1997) consist of learning to handle subgoals on a lower level and having a higher level module, often a planner, choose between different subgoals. This requires the subgoals to have a sequential nature, i.e., one can be achieved after another has been completed (not necessarily in a strict order). This imposes a restriction on the types of problems where these techniques can be applied. W-learning (Humphrys, 1995) on the other hand does deal with multiple parallel goals by generating separate Q-learners for each goal and have them learn a Weight-factor for each state which determines the importance of following a certain agents advice in that state.

In this work we study the Digger game, an example problem where several subgoals have to be dealt with concurrently. We design a new hierarchical reinforcement learning method that makes use of Relational Reinforcement Learning (RRL) and show some preliminary results of this technique.

1. The Digger Game

Digger[1] is a computer game created in 1983, by Windmill Software. It is one of the few old computer-games which still hold a fair amount of popularity. In this game, the player controls a digging machine or "Digger" in an environment that contains emeralds, bags of gold, two kinds of monsters (nobbins and hobbins) and tunnels. The object of the game is to collect as many emeralds and gold as possible while avoiding or shooting monsters. In our tests we removed the hobbins and the bags of gold from the game. Although hobbins are more dangerous than nobbins for human players because they can dig their own tunnels and reach Digger faster as well as increase the mobility of the nobbins, they are less interesting for learning purposes, because they reduce the implicit penalty for digging new tunnels (and thereby increasing the mobility of the monsters) when trying to reach certain rewards. The bags of gold we removed to reduce the complexity of the game, but we do plan to reintegrate them in the future.

[1] http://www.digger.org

2. Relational Reinforcement Learning

We make use of the RRL-TG algorithm as described in (Driessens et al., 2001). Relational Reinforcement Learning combines relational learning with reinforcement learning. It uses first order representations for the states, actions and the Q-function, allowing it to generalise over different states, actions and goals and exploit the structural aspects of the environment.

Ordinary Q-learning techniques would have a hard time dealing with the large state-space of the Digger Game. While the game-field is naturally divided into 10*15 squares, the presence of tunnels, an unknown number of emeralds and monsters causes a standard Q-table to become very large. Moreover, the connectivity of tunnels and the relative position of the monsters and emeralds is more important for playing the game than their absolute positions. Also, tunnels aren't restricted to clearing a square of the game field all at once, but can stop half way into a square. Therefore it becomes very hard to describe all possible tunnel configurations in a standard attribute value style.

RRL on the other hand can exploit these structural aspects of the game. Supplying the learning algorithm with the distance to monsters or emeralds boils down to adding some background knowledge to the system. The same goes for whether a monster is in the line of fire or finding the direction of the nearest emerald. Another advantage of using RRL is that we can learn on all the levels of the Digger game at once.

3. Hierarchical Reinforcement Learning

While we can use RRL in its original form to learn a strategy on the entire Digger game at once, this is a hard learning task and requires a large amount of

exploration and experience to converge. However, one can see that the Digger game is very naturally subdivided into several parts. One goal of Digger is to collect emeralds, while another goal is to avoid or kill monsters. The difficulty in the Digger game is that both subgoals have to be handled in parallel, not sequentially as is usually required by hierarchical reinforcement learning techniques.

In the solution we propose, a first step consists of letting RRL learn on the subproblems separately. In our version of the Digger game these are: collecting emeralds and dealing with monsters. We let RRL generate a Q-tree for each of those subgoals separately.

One way of combining the separate Q-trees would be to use the sum of the predicted values as the Q-function for the whole problem. This already gives OK results but little can be said about its optimality as this relies too much on the exact rewards related to the two subproblems.

To obtain better results, but still make use of the already generated Q-trees we devised a new hierarchical reinforcement learning technique. After learning Q-trees on the sub-problems, we start a new learning episode on the full scale problem and add the already learned Q-trees to RRL as background knowledge. We then allow RRL to compare the two values, to negate them, to add them together with or without the use of a factor and then compare them to a series of constant values for building a new tree to represent the Q-function of the complete problem. One advantage of this approach is that it can detect actions that are non-optimal for either of the subproblems (subgoals) but become optimal when all subgoals are handled simultaneously. The W-learning technique (Humphrys, 1995) differs from our technique at this point since it will always take an action that is optimal to one of the agents.

In the case of the Digger game, we also presented the regular features of the Digger game to RRL to be used as test for the Q-tree but — as expected — RRL preferred tests related to the Q-function predictions over other tests it was allowed to use. Figure 1 shows some (preliminary) results. One can see that convergence to a higher yielding strategy is faster with the hierarchical learning strategy. However, it is not yet clear to the authors whether this learning technique will result in more optimal policies.

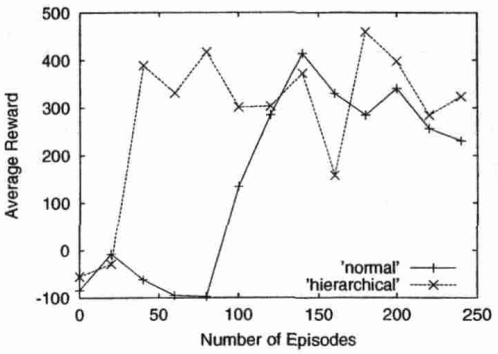


Figure 1. Average rewards from normal and hierarchical learning.

4. Conclusion

For future work we would like to investigate the applicability constraints of our hierarchical reinforcement learning technique for different kinds of concurrent subgoals and further compare its convergence speed with regular Q-learning strategies. Also, we are looking into other representation possibilities for the Q-function in RRL.

References

Driessens, K., Ramon, J., & Blockeel, H. (2001). Speeding up relational reinforcement learning through the use of an incremental first order decision tree learner. *Proceedings of the 13th European Conference on Machine Learning* (pp. 97–108). Springer-Verlag.

Humphrys, M. (1995). *W-learning: Competition among selfish q-learners* (Technical Report 362). University of Cambridge, Computer Laboratory.

Kaelbling, L., Littman, M., & Moore, A. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research, 4*, 237–285.

Tadepalli, P., & Dietterich, T. (1997). Hierarchical explanation-based reinforcement learning. *Proceedings of the 14th International Conference on Machine Learning.*

Learning to use Contextual Information for Solving Partially Observable Markov Decision Problems

Alain Dutech DUTECH@LORIA.FR

Bruno Scherrer SCHERRER@LORIA.FR

LORIA/INRIA, BP 239, 54506 Vandoeuvre les Nancy, FRANCE

Introduction

While a reinforcement learning agent can rely on memoryless policies to solve a completely observable Markov problem, it is no longer the case with partially observable problems. An alternative is to use the history of previous observations. In theory, an infinite memory of the past is required, but to prevent combinatorial explosions, a finite memory, either with finite or infinite horizon (remember a finite number of events which can be arbitrary old) is usually used.

The work presented here belongs to the last category as the agents must choose an action given the present observation *and* contextual information. This contextual-memory architecture should be able to solve many usual partially observable Markov problems by allowing the agent to distinguish between ambiguous observations. But the learning agent is still confronted to a highly non-Markovian task. Figure 1 illustrates this on a simple example.

To that effect, several learning algorithms have been used (Lanzi, 2000), (Lusena et al., 1999), (Peshkin et al., 1999) but Monte-Carlo-like methods seem the best compromise as the specific features of non-Markovian problems are not very well understood.

Following this track of thought, we propose an algorithm to help the agent with the crucial task of choosing the right context. To that effect we introduce the notion of *pertinence* for an observation. Ideally, the pertinence should be a mesure of the rightfulness of choosing the current observation as a context for future decisions. In practice, as described below, the pertinence of an observation is based on the information brought by this observation on the current state of the process.

Our algorithm is made of two stages. First, a model of the process is learned off-line using a Bayesian approach and the pertinence of each observation is computed. Then, an extended-version of Q-Learning is used to learn a policy on-line. The following sections give more details on various parts of the algorithm.

Contextual Information and POMDP

Let $(\mathcal{S}, \mathcal{A}, T, r, \mathcal{O}, \Phi)$ be a Partially Observable Markov Decision Problem (POMDP). $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ a finite set of actions, $T(s, a, s')$ the probability of transition from state s to s' having chosen action a, r a reward function of s, $\mathcal{O}$ a finite set of observations and $\Phi(s, o)$ the probability of observing o in state s. Instead of looking for a policy of the history of observations, we introduce a set of contexts $\mathcal{C}$ (here $\mathcal{C} = \mathcal{O}$) and look for policies of the form: $\mathcal{O} \times \mathcal{C} \longrightarrow \mathcal{A}^+$ where $\mathcal{A}^+$ contains actions which alter the context. The idea here is to use the context as a mean to choose between ambiguous situations, like observation B in Figure 1. Of course, the agent must learn to change its context so as to make a pertinent use of it.

Pertinent Context

Ideally, a context is *pertinent* if it brings valuable information for choosing the current action, given the current observation. As such, it is very hard to quantify this notion, especially since the context must be memorized *before* it will actually be used.

Therefore, to help the agent with the task of learning to use the right context, we measure the information brought by a particular observation o, noted $I(o)$. This will be our *pertinence*. The reason behind this choice is to bias the context towards the most informative observations of the process.

The pertinence is thus set to the mutual information between two consecutive state distributions of the process S_1 and S_0 when S_0 is a uniform distribution and o is observed:

$$I(o) = I(S_0; S_1|o) = K + K' \sum_{s,s'} [\Phi(s', o) \times \{\sum_a T(s, a, s')\} log \frac{\sum_a T(s,a,s')}{\sum_{s'',a} T(s'',a,s')}]$$

Algorithm

If we assume that the pertinence of each observation is known, we use a two-step algorithm. The goal of the first step is to explore and learn to use the context. Among the various ways to privilege the memorization of pertinent observations, we have been mainly working with the two methods described below. One

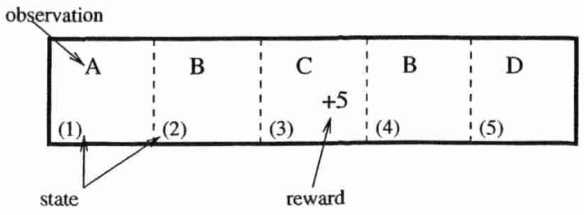


Figure 1. **Simple non-Markovian problem.** When relying only on the observation, the learning agent cannot chose an optimal action when seeing observation B. But, if it learns to memorize the fact that it was either in A or D as a contextual knowledge, the agent can decide to go either left or right when in B. Still, the stochastic process $(Context, Observation)_t$ is non-Markovian while the agent is learning to use and modify its context.

method (step 2.a of the algorithm) is based on memorizing the observation with the highest pertinence, but, to prevent ending with the memorization of only one observation, the pertinence of the context is iteratively decreased. Another method (step 2.b of the algorithm) focusses on the difference in pertinence between two consecutive observations, promoting the memorization of observation which increases the "local" pertinence.

1. From o_t and c_t, choose randomly an a_t. Obtain new o_{t+1} and r_{t+1}.
2.a If $I(o_{t+1}) > I(c_t)$ then $c_{t+1} \leftarrow o_{t+1}$, else $c_{t+1} \leftarrow c_t$ and $I(c_{t+1}) = \beta I(c_t)$.
OR
2.b If $I(o_{t+1}) > I(o_t)$ then $c_{t+1} \leftarrow o_{t+1}$, else $c_{t+1} \leftarrow c_t$ with no decrease in I.
3. Update $Q(o_t, c_t, a_t) \leftarrow (1-\alpha)Q(o_t, c_t, a_t) + \alpha[r_{t+1} + \gamma max_{a'} Q(o_{t+1}, c_{t+1}, a')]$
Then, the Q function can be used to chose an optimal action.

A Model for the Pertinence

A model of the process is required to compute the pertinence of each observation. In the framework of reinforcement learning, this model has to be learned. But, the importance of the hidden states for computing the pertinence of the observations makes this learning quite hard : one has to learn the structure *and* the parameters of the model. We have chosen an incremental model learning algorithm derived from the work of (Stolcke & Omohundro, 1994) based on the maximization of the *a posteriori* probability of the model. The choice of a prior probability for the structure of the model is essential to merge new trajectories of observations to the current model without over-generalization.

Results and Perspectives

We have validated the use of pertinent contextual observations to solve POMDP on academic examples of a path-finding agent in a virtual grid world when the model is known. The two algorithms (version 2.a and 2.b) behave correctly. We are now investigating on the incremental learning of a model and preliminary results show the importance of the prior used in learning the model. Our algorithms, along with others not described here, need to be tested more thoroughly, especially to test the impact of the different parameters.

Among possible perspectives, we want to focus on model learning and on the incremental aspect of our algorithm. In particular, we would like to investigate on using a neural network derived from the neural gas algorithm (Fritzke, 1995) to learn the topological, or structural, aspect of the model.

References

Fritzke, B. (1995). A growing neural gas network learns topologies. *Advances in Neural Information Systems 7*. The MIT Press.

Lanzi, P. (2000). Adaptive agents with reinforcement learning and internal memory. *Proceedings of the Sixth International Conference on the Simulation of Adaptive Behavior (SAB2000)*. Paris (France).

Lusena, C., Li, T., Sittinger, S., Wells, C., & Goldsmith, J. (1999). My brain is full: When more memory helps. *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*.

Peshkin, L., Meuleau, N., Kim, K., & Kaelbling, L. (1999). Learning policies with external memory. *Machine Learning: Proceedings of the Sixteenth International Conference*. Morgan Kaufmann, San Francisco, CA.

Stolcke, A., & Omohundro, S. (1994). *Best-first model merging for hidden markov model induction* (Technical Report TR-94-003). ICSI, Berkeley, CA.

ATD and AQ-learning: reward baseline reinforcement learning algorithms for discounted reward problems

Frédérick Garcia FGARCIA@TOULOUSE.INRA.FR
INRA, Unité de Biométrie et Intelligence Artificielle, BP 27, Auzeville, 31326 Castanet Tolosan Cedex, France

1. Introduction

TD(λ), Q-learning or Q(λ) are well known temporal difference learning algorithms that have been designed for efficiently learning (optimal) expected sum of discounted rewards for Markov decision processes. These algorithms have been deeply studied, and for TD(λ) and Q(λ), efficient implementations based on eligibility traces have been proposed for look-up table representations.

Average reward RL algorithms constitute another family of algorithms designed for an average reward criterion. The main difference between average reward algorithms and discounted reward algorithms consists in the presence of a reward baseline in the temporal difference term that drives the learning process. This reward baseline is generally updated in order to approximate the average reward of the (limit) policy. An important literature has recently emerged on such algorithms, and the general belief, suggested by several experiences - but still not clearly theoretically founded, is that there seems to be computational advantages in using average reward reinforcement learning methods for discounted reward problems, at least for values of the discount factor γ close to 1.

We have recently developed two new reinforcement learning algorithms, called ATD and AQ-learning, that are designed for the discounted reward criterion for any value of $\gamma < 1$, but that includes a baseline reward in their update rules. The theoretical justifications of these algorithms have been respectively described in (Garcia & Serre, 2000) and (Garcia & Serre, 2001). We have shown that ATD corresponds to an efficient implementation of an asymptotic approximation of TD(λ) with $\lambda = 1$ (Monte-Carlo), and that AQ-learning is an efficient implementation of Watkins' Q(λ) with $\lambda = \frac{1}{\tau}$, where τ is the probability of following a greedy policy at the current step. Simulations made on random MDPs have established that these algorithms could outperform classical TD(λ) or Q(λ) methods.

We now intend in this presentation to go further in the understanding and the evaluation of these two reward baseline reinforcement learning algorithms.

2. ATD and AQ-learning

Let us consider a Markov Decision Process (S, A, P, R), and a policy π from S to A. The ATD algorithm learns the γ-discounted value function V^π of π from the observation of a trajectory $\{(s_n, s_{n+1}, r_n)\}$, where s_n is the current state at time n, s_{n+1} the resulting state obtained when applying the action $\pi(s_n)$ on s_n and r_n the transition reward. ATD maintains a relative value function $W_n(s)$ and a scalar ρ_n, with the following update rules:

$$d_n = r_n - \rho_n + \gamma W_n(s_{n+1}) - W_n(s_n),$$

$$\begin{aligned} W_{n+1}(s_n) &\leftarrow W_n(s_n) + \frac{1}{N(s_n)} d_n, \\ \rho_{n+1} &\leftarrow \rho_n + \frac{\gamma}{n} d_n, \end{aligned}$$

where $N(s_n)$ is the number of time state s_n has been visited at time n. In practice, (W_n, ρ_n) converges to (W^π, ρ^π) such that $V^\pi = W^\pi + \frac{\gamma}{1-\gamma}\rho^\pi$ is the discounted value function of the policy π. This convergence can be theoretically established when the learning rate $\frac{1}{N(s_n)}$ is replaced by its asymptotic value $\frac{1}{n}\frac{1}{\mu(s_n)}$ where $\mu(s)$ is the steady-state probability of state s (Garcia & Serre, 2000). In that case, ρ_n converges to the average gain $\rho^\pi = \lim_{n\to\infty} E[\frac{1}{n}\sum_{p=0}^{n-1} r_p]$.

Similarly, AQ-learning learns the optimal value function of the MDP from the observation of a trajectory $\{(s_n, a_n, s_{n+1}, r_n)\}$, where a_n is the random action applied in state s_n. Like ATD, AQ-learning maintains a relative value function $W_n(s, a)$ and a

scalar ρ_n. At time n, the current optimal action $a_n^* = \text{argmax}_a W_n(s_n, a)$ is chosen with probability τ. W_n and ρ_n are then updated as follows:

$$d_n = r_n - \rho_n + \gamma \max_a W_n(s_{n+1}, a) - W_n(s_n, a_n),$$

$$\begin{aligned} W_{n+1}(s_n, a_n) &\leftarrow W_n(s_n, a_n) + \frac{1}{N(s_n, a_n)} d_n, \\ \rho_{n+1} &\leftarrow \rho_n + \frac{\gamma}{\tau} \frac{1}{n} d_n \quad \text{if } a_n = a_n^*. \end{aligned}$$

AQ-learning converges experimentally toward (w^*, ρ^*) such that $Q^* = W^* + \frac{\gamma}{1-\gamma}\rho^*$ is the optimal Q-value function of the problem.

3. Minimal variance algorithms

We now show that ATD and AQ-learning implement a simple approximation of optimal gain matrix variants of TD(0) and Q-learning that minimize the asymptotic variance of the estimate. TD(0) and Q-learning algorithms can be described as simple stochastic algorithms

$$J_{n+1} = J_n + \alpha f(J_n, X_n)$$

where α is the learning rate, J_n is the parameter vector and X_n the input random vector that brings some information on J_n at time n. For TD(0), $J_n = V_n$ is the current estimate of V^π and $X_n = (s_n, s_{n+1}, r_n)$. For Q-learning, $J_n = Q_n$ is the current estimate of Q^* and $X_n = (s_n, a_n, s_{n+1}, r_n)$.

The convergence of this learning process can be easily analysed with the ordinary differential equation method (ODE). Under general assumptions, J_n converges with probability 1 to the unique solution J^* of some equation

$$E[f(J, X)] = 0 \tag{1}$$

where $E[f()]$ is the expectation of the vector $f()$. For TD(0), $J^* = V^\pi$ and (1) is a linear equation in J. For Q-learning, $J^* = Q^*$ and (1) is the Bellman equation.

Another potential application of the ODE method concerns the optimization of the rate of convergence of these stochastic algorithms. One can show that for learning rates α asymptotically proportional to $\frac{1}{n}$, and assuming some general stability conditions (Benveniste et al., 1990), we have

$$\sqrt{n}(J_n - J^*) \underset{n\to\infty}{\longrightarrow} \mathcal{N}(0, M)$$

where M is a symmetric covariance matrix. This relation implies that for large n,

$$E[\| J_n - J^* \|^2] \approx \frac{1}{n} \text{trace}(M).$$

In order to accelerate the convergence of J_n toward J^*, trace(M) should be minimized. This can be done by considering the new algorithm

$$J_{n+1} \leftarrow J_n + \frac{1}{n}\Gamma^* f(J_n, X_n)$$

where Γ^* is the gain matrix defined by

$$\Gamma^* = -(\nabla_J \lim_{n\to\infty} E[f(J, X_n)])^{-1}(J^*).$$

In practice, this algorithm exhibits a very fast convergence toward J^*.

This optimal gain matrix Γ^* can be calculated for TD(0) or Q-learning (see (Garcia & Serre, 2000; Garcia & Serre, 2001)). For TD(0), Γ^* is a function of the transition probability matrix P_π, of the stationary distribution of this chain, and of γ. For Q-learning, Γ^* is a function of the transition probability matrices P, of the optimal policy π^*, of the stationary distribution of the Markov chain defined by π^* and by τ, and of γ. Of course, these optimal gain matrices cannot be a priori calculated since the transition probability matrices are not known. Nevertheless, the gain matrix Γ^* can be simply approximated by replacing exact transition probabilities by steady-state probabilities that can be simply estimated on line. In that case we show that the resulting near optimal gain matrix variants of TD(0) and Q-learning exactly correspond respectively to ATD and AQ-learning. This result appears to be a first formal justification for the use of a reward baseline in value function reinforcement learning in the context of the discounted criterion.

The experimental study we conducted on random MDPs or classical benchmarks confirm the very fast convergence of ATD and AQ-learning, compared to different TD(λ) or Q(λ) methods.

References

Benveniste, A., Metivier, M., & Priouret, P. (1990). *Adaptive Algorithms and Stochastic Approximation.* Berlin,New York: Springer-Verlag.

Garcia, F., & Serre, F. (2000). Efficient asymptotic approximation in temporal difference learning. *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI 2000).* Berlin.

Garcia, F., & Serre, F. (2001). From Q(λ) to Average Q-learning: Efficient implementation of an asymptotic approximation. *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001).* Seattle, USA.

A Markov Model for Dyadic Interaction Learning

Walter J. Gutjahr walter.gutjahr@univie.ac.at

Dept. of Statistics and Decision Support Systems, University of Vienna, Universitätsstraße 5/9, A–1010 Wien, Austria

Anselm Eder anselm.eder@univie.ac.at

Dept. of Sociology, University of Vienna, Universitätsstraße 7, A–1010 Wien, Austria

1. Introductory Example

Let us consider an electronic market where two software agents offer a single product to a population of buyers (cf. Sairamesh & Kephart, 2000). In an alternating sequence, the two agents set their prices to one of three possible levels 1, 2 or 3. The payoff for each agent (we label them by 0 and 1) depends on the last two decisions, i.e., its own previous decision and the subsequent decision of its opponent, as given by the matrices below:

Therein, entry $a^{(k)}(i,j)$ of the matrix is the payoff for agent k, given agent k has first chosen price level i, and agent $1-k$ has then chosen price level j. For example, if the sequence of price decisions is

agent 0: 3, agent 1: 1, agent 0: 2, agent 1: 3, agent 0: 3
then the resulting payoff values are:

agent 0: 0, agent 1: 2, agent 0: 4, agent 1: 3, ...

In order to *learn* an optimal pricing behavior, each agent $k = 0,1,\ldots$ keeps a [3 x 3]–matrix where the entry in line i and column j indicates the probability of reacting on previous decision i by agent $1-k$ with decision j. As in the S–R–C scheme of psychological learning theory, it is assumed that a stimulus–reaction pair is *reinforced* to a degree proportional to the reward obtained as a consequence of the reaction. (Here, the stimulus is the other agent's action.) E.g., the payoff of 2 units for agent 1 in the example above increases the probability of agent 1 of reacting on decision "3" of agent 0 with decision "1" by 2·const, and the subsequent payoff of 4 units for agent 0 increases the probability of agent 0 of reacting on decision "1" of agent 1 with decision "2" by 4·const. After each probability increase, the corresponding probability vector (the line in the probability matrix) is re–normalized to a sum equal to unity. (In terms of Nowak, Sigmund & El–Sedy, 1995, cf. also Hofbauer & Sigmund, 1998, each agent applies here a "memory–one learning rule".) As it can be seen, this results in a process of co–learning, where each agent forms the "environment" for the other agent.

In the literature on repeated 2–person games, similar dynamic processes have been studied (often in the context of so–called *win–stay, lose–shift rules*) by several authors, e.g., Staddon (1983), Roth & Erev (1995), Wedekind & Milinsky (1996), Posch (1997), Posch, Pichler & Sigmund (1999), or Greenwald, Friedman & Shenker (2001). The model studied here differs from the mainstream of these approaches by the fact that we assume strictly *alternating moves* of the two players, and by admitting payoff matrices of *arbitrary sizes*, which do not need to have a specific structure (as, e.g., that of the prisoner's dilemma) any longer and need not to be equal. In this way, our model is able to capture very general situations of dyadic social interactions among humans, animals or artificial agents. Sociological aspects of our model are discussed in Eder, Gutjahr & Neuwirth (2001), where also a detailed computational example for a specific pair of payoff matrices is presented.

2. Limiting Behavior

In its general form, the problem studied here can be considered as a two–agent reinforcement learning (RL) problem with specifically structured policies, reward functions and value functions (see, e.g., Sutton & Barto, 1998): Let $A^{(k)} = (a^{(k)}(i,j))$ $(k = 0,1;\ 1 \le i \le N_k,\ 1 \le j \le N_{1-k})$ be the payoff matrices for agents $k = 0,1$, and let $X_n^{(k)} = (x_n^{(k)}(i,j))$ $(n = 0,1,\ldots;\ k = 0,1;\ 1 \le i \le N_k,\ 1 \le j \le N_{1-k})$ be their probability matrices after their nth move $(n = 0,1,\ldots)$. The start matrices $X_0^{(k)}$ are arbitrary, provided that the lines are probability vectors. The matrices $X_n^{(k)}$ represent the current policies of the two agents; the matrices $A^{(k)}$ determine the reward functions, and the value functions can be defined, e.g., as long–run discounted total rewards. The sequence of actions (decisions) of agent k is denoted by $(i_0^{(k)}, i_1^{(k)}, \ldots)$. We are interested in the limiting behavior of the process outlined in Section 1 as $n \to \infty$. First of all, it can be shown that the two stochastic processes with the quadruples

$$(i_{n-1}^{(1)}, X_n^{(0)}, i_n^{(0)}, X_n^{(1)}) \quad (n = 1,2,\ldots) \text{ resp.}$$

$$(i_{n-1}^{(0)}, X_{n-1}^{(1)}, i_{n-1}^{(1)}, X_n^{(0)}) \quad (n = 1,2,\ldots)$$

as states are *Markov processes* in discrete time. We distinguish two types of stationary distributions of these Markov processes: Type (a) is characterized by the property that $X_\cdot^{(0)}$ and $X_\cdot^{(1)}$ have point mass distributions, i.e., $x_n^{(k)}(i,j)$= const for fixed k, i and j in the steady state, whereas type (b) is characterized by the property that either $X_\cdot^{(0)}$ or $X_\cdot^{(1)}$ has a distribution that is not a point mass. In our computer experiments,

we only observed the type (a) case. For this case, we can show the following result. On the condition that $a^{(k)}(i,j) > 0$ for all k, i and j, a stationary distribution of type (a) has the property that there are index sets $I^{(0)} \subseteq \{1,\ldots,N_0\}$ and $I^{(1)} \subseteq \{1,\ldots,N_1\}$ and bijective functions $③^{(1)}$: $I^{(0)} \to I^{(1)}$ and $③^{(0)}$: $I^{(1)} \to I^{(0)}$, such that $③^{(1)}(i)$ is the fixed response (chosen with probability one) of agent 1 on decision $i \in I^{(0)}$ of agent 0, and $③^{(0)}(j)$ is the fixed response (chosen with probability one) of agent 0 on decision $j \in I^{(1)}$ of agent 1.

Informally speaking, this result predicts a *limit cycle* as the final interaction sequence, at least in the case where the stationary distribution is of type (a). After some time, the two agents start to reproduce the same cycle of alternating actions again and again. (The result is in accordance with the typical occurrence of cyclical price wars, as they have been observed in Sairamesh & Kephart, 2000, in a situation generalising our introductory example.) While approaching the steady state, the process shows a successive reduction of (Shannon) *entropy*, which allows interesting conclusions on interactions in social systems.

Let us mention that the RL algorithm considered here is a simple stochastic trial–and–error mechanism and does not apply more elaborated RL techniques such as Temporal Difference Learning or Function Approximation. It might be desirable to extend our results to agents implementing these more refined techniques.

3. Relations with ACO

In heuristic optimization, the paradigm of ACO (*Ant Colony Optimization*) plays a role of growing importance. It may be of interest that it is possible to express our interaction model in close relation to the ACO framework. The ACO metaheuristic, as formulated by Dorigo & Di Caro (1999), is a search process realized by randoms walks of "ants" in a directed graph, where *learning* takes place by changes of the so–called *pheromone values* assigned to the arcs of the graph, which govern the transition probabilities. We can put our model into this context as follows: Only a single ant is considered. The nodes of the graph are the triples (k, i, j) ($k = 0,1$; $1 \le i \le N_k$, $1 \le j \le N_{1-k}$), and the arcs of the graph are of the form $\langle (k, i, j), (1-k, j, m) \rangle$ with $1 \le m \le N_k$. Pheromone values correspond to the values of the matrices $X_n^{(k)}$, where each entry $x_n^{(k)}(i,j)$ is assigned to all arcs with endnode (k, i, j). Reinforcement of a stimulus–reaction pair trans–lates into pheromone increase on the corresponding arcs in a straightforward way. The re–normalization to probability vectors with sum equal to unity corresponds to (a variant of) the so–called pheromone evaporation in ACO. The only necessary adaption is that instead of *online step–by–step pheromone update*, as described in Dorigo & Di Caro (1999), we require an update mechanism that could be called *online delayed step–by–step pheromone update*: Each time an arc $\langle v,w \rangle$ is traversed, pheromone update is done for all arcs of the form $\langle u,v \rangle$. It can be shown that the resulting ACO–type process is equivalent to the dynamic process described in the previous sections. We conjecture that also in standard ACO implementations, *self*–en–forcement (as a consequence of non–terminating walks) leads asymptotically to limit cycles.

References

Dorigo, M., Di Caro, G., "The Ant Colony Optimi–zation metaheuristic", in: *New Ideas in Optimization*, D. Corne, M. Dorigo, F. Glover (eds.), pp. 12–32, McGraw–Hill , 1999.

Eder, A., Gutjahr, W.J., Neuwirth, E., "Modeling social interactions by learning Markovian matrices", *Proc. 8th Int. Conf. on Facet Theory*, Prague, July 15–18, pp. 95–106, 2001.

Greenwald, A., Friedman, E.J., Shenker, S., "Learning in network contexts: experimental results from simu–lations", *Games and Economic Behavior* 35, pp. 80–123, 2001.

Hofbauer, J, Sigmund, K., *Evolutionary Games and Population Dynamics*, Cambridge UP, 1998.

Nowak, M.A., Sigmund, K., El–Sedy, E., "Automata, repeated games and noise", *J. Math. Biol.* 33, pp. 703–722, 1995.

Posch, M., "Cycling in a stochastic learning algorithm for normal form games", *J. Evolutionary Econ.* 7, pp. 193–207, 1997.

Posch, M., Pichler, A., Sigmund, K., "The efficiency of adapting aspiration levels", *Proceedings of the Royal Society,* Series B, 266, pp. 1427–1436, 1999.

Roth, A.E., Erev, I., "Learning in extensive–form games: experimental data and simple dynamic models in the intermediate term", *Games and Economic Behavior* 8, pp. 164–212. 1995.

Sairamesh, J., Kephart, J.O., "Price dynamics and quality in information markets", *Decision Support Systems* 28, 35–47, 2000.

Staddon, J.E.R., *Adaptive Behavior and Learning*, New–York: Cambridge UP, 1983.

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

Wedekind, C., Milinski, M., "Human cooperation in the simultaneous and the alternating Prisoner's Dilemma: Pavlov versus generous tit–for–tat", *Proc Nat. Acad. Sci. USA*, 93, pp. 2686–2689, 1996.

Issues in Dynamic Hidden Markov Modelling

Adrian Hartley and Jeremy Wyatt
School of Computer Science
University of Birmingham
Birmingham, UK, B15 2TT
{arh,jlw}@cs.bham.ac.uk

Abstract

The starting point of this paper is to note that all traditional Hidden Markov Model (HMM) optimisation algorithms [1] work on the assumption of a static number of states within the model. Traditional approaches optimise a given HMM with respect to a sequence resulting in the most likely model given the sequence. However they have no way of changing the resolution of the model; adding more states to better a models description of the sequence. This paper discusses how a previous dynamic HMM algorithm can be improved for more reliable results in reinforcement learning tasks with hidden state. This allows the agent to start with a minimal model and expand it when it is necessary; rather than starting with a massively over general model.

Introduction

Hidden Markov Models extend the Markov framework to allow us to model n^{th} *order Markov processes*. We note that stochastic process modelling is still very popular within reinforcement learning (RL) as a way of representing the dynamics of an agent-world interaction. The HMM framework is appealing as it gives extra descriptive power for representing those problems with hidden state.

If a process contains hidden state then the problem of learning a model is harder than the completely observable case. Techniques based on the Baum-Welch algorithm can be applied to sequences of observations drawn either from discrete or continuous spaces. Basic HMM techniques require that the number of states in the model be known in advance. In addition they are only guaranteed to converge to a local maximum likelihood model, though techniques that employ a set of randomly generated initial models can give improved results. However, methods such as Lonnie Chrisman's perceptual distinctions algorithm (Chrisman 1992) and Andrew McCallum's Utile Distinctions (UDM) algorithm allow the refinement of the number of model states.

From a reinforcement learning perspective McCallum's (McCallum 1996) approach is appealing since it recognises that the required model need only provide accurate predictions of the utility of state-action pairs. We refer to such techniques as task driven dynamic HMM algorithms. However, such algorithms have weaknesses. They require long sequences of data and may not detect n^{th} order hidden state unless there is an improvement in predictive accuracy for each intermediate order over the previous one. Directly searching the space of possible models with hidden state up to order n for a good starting model is clearly computationally intractable.

The power of this Dynamic Hidden Markov Modelling approach comes from allowing an agent to start with minimal assumptions (e.g. a minimal HMM[2]). States are only added when they will benefit the agent in solving a given task. The end result should be compact models with a high utility.

Dynamic Hidden Markov Models

A Partially Observable Markov Decision Process, (POMDP[3]) is a HMM with action. A POMDP can be described by the tuple $\langle S, A, T, R, Z, O \rangle$. $S = \{s_1, ..., s_N\}$ is the finite set of states of the world. $A = \{a_1 ..., a_C\}$ is the finite set of actions actions available to the agent. T is the transition function, $Pr : S \times A \rightarrow \prod(S)$, giving for each world state and action a probability distribution over the world states. R is the reward function, $R : S \times A \rightarrow \mathbb{R}$, giving the expected immediate reward gained by the agent for taking an action in a state. $Z = \{z_1, ..., z_M\}$ is the finite set of possible observations the agent can experience of its world. O is the observation function $O : S \times A \rightarrow \prod(\Omega)$ giving, for each action a and resulting state s' a probability distribution over possible observations. N is the number of states.[4] C is the length of the sequence of actions. M is the length of the sequence of observations.

S_t denotes the random variable denoting the state at time t. Z_t denotes the random variable denoting the observation at time t. σ denotes the sequence of actual observations $(o_1, ..., o_T)$. POMDP's use the same distributional parameters as MDPs: $T = \{T_{ij}\}$ such that $T_{ij} = Pr(S_{t+1} = s_j | S_t = s_i)$, $O = \{O_i\}$ such that $O_i(k) = Pr(Z_t = p_k | S_t = s_i)$. But POMDP'ss have the additional distributional parameter: $\pi = \{O_i\}$ such that $\pi_i = Pr(S_0 = s_i)$, this is the initial state distribution. Also let $\lambda = \{T, O, \pi\}$

[1] For example the Baum Welch algorithm or the Segmental K-means Algorithm.

[2] A minimal HMM is a HMM with just a single state for each possible observation and is essentially the Markov estimate on the process.

[3] See (Littmann 1994) and (Littamn 1996) for more details and examples of their use in reinforcement learning.

[4] The number of states is known or estimated initially.

which describes a given POMDP with fixed $|S|$ and $|Z|$.

In hidden state problems an agent will have to extract the true structure and dynamics of its world from the observations it makes. However, perceptual aliasing means that single observations cannot be relied on to determine outcomes. The agent will have to infer complicated underlying dynamics which are not evident from just the sequence of observations.

In the Dynamic HMM as proposed by (McCallum 1996) the agent starts with a minimal HMM. Data is gathered on estimated reward values for states *(s')* given transitions from all proceeding states *(s)* under any action and that a specific action *(a)* is taken. The data is used to form 95% confidence intervals for each tuple (s, s', a), with a large enough sample. Then we search the set of sets of confidence intervals $\{\{(s, s', a)|s \in S\}|s' \in S, a \in A\}$, looking for disjoint or overlapping confidence intervals. Disjoint confidence intervals indicate the presence of hidden state and so the state is split. This is illustrated in Figure 1.[5] The model can then be re-optimised using Baum-Welch and this repeated as an iterative process.

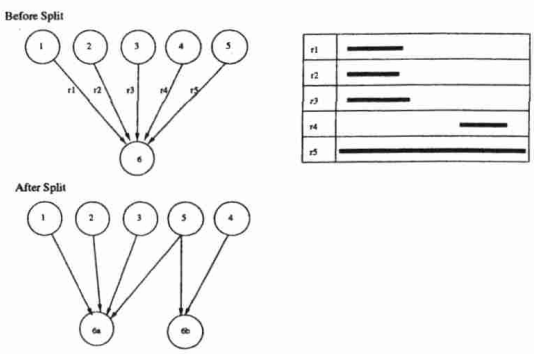


Figure 1: Illustration of state splitting

Problems of State Splitting in Learning a POMDP

The problems associated with state splitting can be split into the following classes:

- The selection of the statistical test for deciding on a split.
- Deciding on the best state to split
- How to rebuild the model incorporating the new states.

McCallum uses a standard confidence interval which incorporates the assumption that data is normally distributed. Unfortunately for states that are hidden, estimated return values are often drawn from a multinomial distribution rather than a normal distribution. This means that a single data set (s, s', a) can actually contain disjoint intervals if the data set is modelled as a multinomial.

After statistical analysis there may be more than one possible state to split. For all the sets $\{(s, s', a)|s \in S\}$ more than one may conform to conditions for splitting. Do we split all possible states or just one; splitting all will increase the model size many times where as splitting just one may reduce the variance enough that, on successive iterations, no more splits are necessary. We have currently made the assumption that the greatest magnitude between the disjoint intervals indicates the best state to split.

Once a state is chosen for splitting, how do we rebuild the model to incorporate the split state. McCallum's diagram (Figure 1) suggests that based on the confidence intervals incoming transitions are partitioned across the split states; disjoint intervals go to different states and overlapping ones go to both. But McCallum also states that a fully connected HMM is maintained.

There is definitely a space of possible algorithms for rebuilding the HMM after a state is split. At one end we use the minimal possible number of transitions. New states inherit out-going transitions from the original state that was split and incoming transitions are based just on the transition ratios and the confidence intervals. In this case we no longer have a fully connected HMM as there will only be certain paths into and out of split states. At the other end we use the maximal number of transitions. Using a uniform probability of transition from all states into and out of the new states. This maintains a fully connected HMM and relies on optimisation to resolve the structure of the model.

We propose a method in between these two methods. Statistics are based on interquartile ranges generated for each of the possible underlying distributions of the multinomial data set $\{(s, s', a)|s \in S\}$ where data sets are formed through clustering. The model is then rebuilt using the minimal possible number of transitions to express the intervals, as illustrated in Figure 1.

Conclusion

Although McCallum goes on to pursue tree based representational methods we argue that due to the power of graphically based models it is worth pursuing these methods. Preliminary results suggest that we can stably achieve good compact models while employing a different statistical approach than McCallum's.

References

Chrisman, L. 1992. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 183–188.

Littamn, M. L. 1996. *Algorithms for Sequential Decision Making*. Ph.D. Dissertation, Brown University.

Littmann, M. 1994. The witness algorithm: Solving partially observable markov decision processes. Technical report CS-94-40, Brown University, Department of Computer Science.

McCallum, A. K. 1996. *Reinforcement Learning with Selective Perception and Hidden State*. Ph.D. Dissertation, University of Rochester.

[5] Figure 1 is taken from (McCallum 1996) p52.

Order Acceptance with Reinforcement Learning

Marisela Mainegra Hing, Aart van Harten, M.MAINEGRAHING,A.VANHARTEN@SMS.UTWENTE.NL
Faculty of Technology and Management, University of Twente, The Netherlands

1. Introduction

Order Acceptance (OA) is one of the main functions in a business control framework. It is typically a decision making problem at the interface of customer relations management and production management. Basically, OA involves for each order a 0/1 (i.e., reject / accept) decision (Bertrand, Wortmann & Wijngaard, 1990). Traditionally this problem is solved as follows: always accept an order if sufficient capacity is available. However, always accepting an order when there is available capacity could unable the system to accept more convenient orders in the future. In Operations Research literature the idea of opportunity losses is recognized but not really worked out (Wouters, 1997). Another important aspect is the availability of information to the decision maker. Generally in the literature information regarding negotiation with the customer such as an estimate of the work content of an order, a norm for the necessary processing time, the price and the due date are assumed to be known or estimated, and a model of the production process is also considered to be known beforehand as in Raaymakers (1999). But most of the time it is difficult to have such information. How to find a good trade-off between long term opportunity costs and immediate yield in case of order acceptance under different degrees of uncertainty due to lack of information is a central problem in this study.

There are some aspects that make Reinforcement Learning (RL) appealing to our problem. The idea of learning without the necessity of complete model information and the possibility that the agent learns even from delayed rewards (when the effects of an action can be known only in the future) allows us to consider different degrees of uncertainty and to take into account the opportunity cost problem in a natural way. RL is quite a new field and successful applications are not always fully understood at a theoretical level. It means that convergence properties of the corresponding algorithms and procedures for tuning the parameters in the algorithms have to be explored. Hence, a lot of work still has to be done in order to understand how RL can best be applied to our field and to get insight into why some domains are considerably more tractable for RL than others.

2. Modeling OA

2.1 Prototype model

A first prototype problem is used to gain insight into RL. Production capacity is modeled as a single server that can only process one job at a time. Orders arrive in a single arrival process from a set of N order types. Order type j is characterized by a small set of attributes: processing time (t_j), profit for acceptance (r_j) and arrival chance (p_j). Arrivals are checked every fixed period of time, and a decision should be taken immediately at that moment if the server is idle: accept or reject the arrived order. If the server is busy, the arrived order is lost. This prototype problem can easily be modeled as a discrete time Semi Markov Decision Problem (SMDP). State s identifies the arriving order and there are two possible actions, reject or accept ($a = 0,\ a = 1$). The state transition is described in the table below:

actual state	s	
action (a)	0	1
immediate reward $rew(s,a)$	0	r_s
time until next decision $d(s,a)$	1	t_s
next state	s'	
transition probability $p(s,a,s')$	$p_{s'}$	

The performance of the system is measured as the expected value of the total discounted reward. The Bellman equation for the value function V^π of policy π is given by $V^\pi(s) = rew(s,\pi(s)) + \gamma^{d(s,\pi(s))} \sum_{s'} p_{s'} V^\pi(s')$ with γ the discount factor. The optimal policy in this case has a very special form: an order of type s should be accepted if and only if $r_s \geq (\gamma - \gamma^{t_s})\beta$, where $\beta = \sum_s p_s V^*(s)$ is defined by the problem instance.

2.2 Extended model

An extended model is formulated that considers multi-server capacity per production stage (C_{max}) in a finite planning horizon (H), with a stochastic perturbation (p) in the capacity profile, and batch order arrivals from a set of N order types. Order type i is characterized by arrival rate (λ_i), due-date(t_i), requested capacity (w_i), and profit for acceptance (r_i).

The state at each decision moment is defined by $s = (k, c)$, where $k = (k_1, ..., k_N)$ is the order list and k_i represents the amount of orders of type i requesting service, $c = (c_1, ..., c_H)$ is the capacity profile and c_j is the total capacity already allocated at stage j. To reduce the size of the action space to a polynomial size in the number of order types we impose that decisions are created sequentially by selecting only one order from the order list. Allocation is done through an $alloc(c, a)$ procedure. Action a$\in$ $[1..N]$ is possible if order type a is present in the order list and capacity is available (i.e., S(s,a)=$\{a \in [1..N] \mid k_a \neq 0, alloc(c, a)$ is possible$\}$. The rejection action ($a = 0$) is always possible, it rejects the complete order list, so new arrivals (A) and a perturbation (p) in the capacity profile would be considered. The perturbation p introduces a penalization ($pen(c, p)$) in the reward structure and participates in a reallocation procedure ($ralloc(c, p)$) that changes the capacity profile. The state transition is described in the table below:

s	(k, c)	
a	$a \in S(s, a)$	0
rew(s,a)	r_a	pen(c,p)
d(s,a)	0	1
s'	(k-e_a, alloc(c,a))	(A, ralloc(c,p))
p(s,a,s')	1	Pr{A}Pr{ralloc(c,p)}

It is clear that in this case the state space can be tremendously large. As a consequence an optimal policy can be a very complex rule. A simple structure as found in case of the prototype problem can no longer be expected. This model is still under study.

3. Experimental Results

Figure 1 shows the total average reward of 6 agents for a case of the prototype model. We compare 4 learning agents (2 direct Q-learning (using neural networks (NN) and using backup table (BT)), a Dyna-BT, and a Naive model-based as in Sutton and Barto (1998)) with 2 agents following fixed policies (heuristic rule *always accept*, and an optimal one). Learning methods outperformed the rough heuristic "always accept" and they all show convergence to the optimal solution. Although model-based methods converge faster to the optimal solution than direct methods, direct methods still converge and they can be a better option for more complex problems in which the use of an estimated model is not that straightforward as in this prototype model.

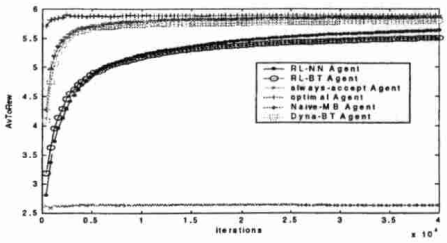


Figure 1. Agents comparison in model 1 for N=10

Figure 2 shows a comparison for the extended model. The learning agent outperforms the greedy heuristic "choose best reward per capacity request from the present order list" and the elitist which only accepts orders of type 1 and 3. The learning agent converges to the elitist which only choose orders type 1.

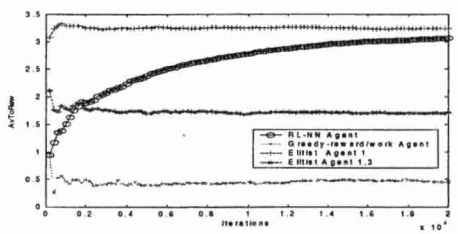


Figure 2. Agents comparison in model 2 for N=3, Cmax=2, H=3.

References

Bertrand, J.W.M., Wortmann, J.C. and Wijngaard, J. (1990). Production Control, A structural and design oriented approach, Elsevier, Amsterdam.

Wouters, M.J.F. (1997). Relevant cost information for order acceptance decisions. Production Planning and control, vol. 8, No. 1, 2-9

Sutton, R.S., and Barto, A.G. (1998). Reinforcement Learning: An introduction. MIT Press, Cambridge, Massachusetts.

Raaymakers, W. (1999). Order acceptance and capacity loading in batch process industries. Pd.D Thesis. Technische Universiteit Eindhoven.

Scheduling with Adaptive Agents – an Empirical Evaluation

Werner Hunger HUNGER@IRA.UKA.DE
Martin Riedmiller RIEDML@IRA.UKA.DE
Institute of Logic, Complexity and Deductive Systems
University of Karlsruhe
D-76128 Karlsruhe

Introduction

Learning in multi-agent systems is a field where established methods for the solution of common applications are scarce for now. To gather further insight in reinforcement learning in complex domains, we did empirical research using a setup which comprehends the following aspects of learning systems: model-free learning, compression of a large state space, local versus global problem view, cooperation of multiple agents and exploration versus exploitation of already learned knowledge. This means working in a domain that is being dictated by practical application and that is hard to cope with analytically.

We are simulating a job shop production scheduling problem out of the field of Operations Research which is NP-hard to solve in the considered instance and that can only be approximately solved in real world applications. Instead of using a global approach for the search for an optimal solution, we are breaking the problem down to a sequence of local decisions which compose the global solution, trying to yield a nearly optimal result.

The approach of local decision making is broadly used in production scheduling. It can be seen as an instantiation of the heuristic rule of thumb: "Find a good solution in your sphere of influence and look ahead, this will hopefully optimize the composed overall solution". Classically, the necessary local decisions of which job to process next on a given resource are made with the help of heuristic rules. But the design of such rules can be exceedingly difficult for real world applications, because there is not only a model for the process to be found, but also the influence of a local decision strategy on the global quality of the solution to be determined.

To get around this, we are coupling an agent to every resource that is subject to planning. The agent has the task of learning to dispatch the jobs that are waiting in the resource's queue without using an explicit process model. To accomplish this, we are using the method of Q-learning and define the agent's state to be composed of parameters describing general characteristics of the waiting jobs (tightness and distribution of the job's due dates) and of parameters describing the current situation of the agent (estimated makespan and average slack of the waiting jobs). These features leave the agent with a highly compressed view of it's current state. An action vector is built of parameters describing characteristics of the selected job with respect to the waiting jobs (due date index and relative slack), so that the selection of similar jobs corresponds to similar action vectors. Because of the continuous state and action space we employ a multi-layer perceptron for function approximation.

We expect this system to be able to exploit inherent regularities of the jobs to be scheduled and to adapt itself dynamically to a changing environment.

Empirical evaluation

The empirical study is showing some interesting results. The learned decision making is in some cases a real improvement over sophisticated heuristic rules. We examined many planning scenarios, differing in the number of adaptive and non-adaptive agents, the applied learning methods and the learning parameters. We used up to eight agents in total and rated the overall performance as well as the ability to generalize. As long as a scenario employs only a few adaptive agents, the following assumptions can be made:

- The system finds good policies that, in most cases, perform better than the considered non-adaptive ones.
- The learning algorithm shows good generalisation abilities.
- The learning algorithm converges quickly and shows a rather constant behaviour.

On an increasing number of adaptive agents in a scenario, the learning process becomes harder because of the following:

- Every agent acts in a non-stationary environment, which breaks a precondition for the reinforcement

learning algorithm. This can be circumvented by a round-robin learning mode.

- The reinforcement signal for the learning process does not reflect the quality of one agent's contribution, but the overall result instead. This can be circumvented only in part by a round-robin learning mode. The general problem of a cumulated reinforcement signal for agents that depend on each other's policies remains untouched, but seems to have great influence.

- The effect of an agent's policy change gets weaker on a raising number of resources, so the reinforcement learning algorithm has to cope with more noise.

The more adaptive agents in the system, the tighter the learning parameter domain for which the system is stable and well-performing. But even with eight agents in total there are many scenarios that are showing good generalisation, good convergence and a better performance than their corresponding non-adaptive benchmark versions.

Conclusion

In its present form, the depicted approach of an adaptive scheduling system shows problematic behaviour for larger problem instances. Additional techniques have to be found for the stabilization of the learning process, be it a better rotation policy (adaptive, maybe), a pre-training along some heuristic priority rule or any method of stabilizing the learning environment that is perceptible to an adaptive agent. It seems to be of great importance to improve the generation of the reinforcement signal, perhaps by partitioning the set of adaptive agents. For smaller problem instances, the approach works promisingly good.

References

Barto, A. G., Sutton, R. S., & Watkins, C. J. C. H. (1989). Learning and sequential decision making. *COINS Technical Report 89 – 95.*

Brauer, W., & Weiß, G. (1998). Multi-machine scheduling – a multi-agent learning approach. *Proceedings of the 3rd International Conference on Multi-Agent Systems* (pp. 42–48).

Dietterich, T., & Zhang, W. (1995). A reinforcement learning approach to job-shop scheduling. *Proceedings of the IJCAI.*

Hunger, W. (2000). Scheduling mit lernenden Agenten – eine empirische Untersuchung. Studienarbeit am Institut für Logik, Komplexität und Deduktionssysteme der Universität Karlsruhe.

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

Klotz, U. (1999). Verteiltes Reinforcement Learning zur Lösung von Scheduling Problemen. Diplomarbeit am Institut für Logik, Komplexität und Deduktionssysteme der Universität Karlsruhe.

Mahadevan, S., & Theocharous, G. (1998). Optimizing production manufacturing using reinforcement learning. *Proceedings of the Eleventh International FLAIRS Conference* (pp. 372–377). AAAI Press.

Pinedo, M. (1995). *Scheduling: theory, algorithms, and systems.* Prentice Hall international series in industrial and systems engineering.

Riedmiller, M. (1997). *Selbständig lernende neuronale Steuerungen.* No. 626 in Fortschr.-Ber. VDI Reihe 8. VDI Verlag Düsseldorf.

Riedmiller, S., & Riedmiller, M. (1999). A neural reinforcement learning approach to learn local dispatching policies in production scheduling. *Proceedings of the Int. Joint Conference on Artificial Intelligence.*

Schneider, J., Boyan, J., & Moore, A. (1998). Value function based prduction scheduling. *International Conference on Machine Learning.*

Universal Sequential Decisions in Unknown Environment

Marcus Hutter MARCUS@IDSIA.CH
Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA), Galleria 2, CH-6928 Manno-Lugano, Switzerland

Introduction: Every inductive inference problem can be brought into the following form: Given a string $x_1x_2...x_{t-1} \equiv x_{1:t-1} \equiv x_{<t}$, take a guess at its continuation x_t. We will assume that the strings which have to be continued are drawn from a probability distribution μ. The maximal prior information a prediction algorithm can possess is the exact knowledge of μ, but often the true distribution is unknown. Instead, prediction is based on a guess ρ of μ. We expect that a predictor based on ρ performs well, if ρ is close to μ or converges to μ.

Universal probability distribution: Let $\mathcal{M} := \{\mu_1, \mu_2, ...\}$ be a finite or countable set of candidate probability distributions on strings. We define a weighted average on $\mathcal{M}$,

$$\xi(x_{1:n}) := \sum_{\mu_i \in \mathcal{M}} w_{\mu_i} \cdot \mu_i(x_{1:n}), \qquad \sum_{\mu_i \in \mathcal{M}} w_{\mu_i} = 1, \quad w_{\mu_i} > 0.$$

We call ξ universal relative to $\mathcal{M}$, as it multiplicatively dominates all distributions in $\mathcal{M}$, i.e. $\xi(x_{1:n}) \geq w_{\mu_i} \cdot \mu_i(x_{1:n})$ for all $\mu_i \in \mathcal{M}$. In the following, we assume that $\mathcal{M}$ is known and contains the true distribution from which $x_1x_2...$ is sampled, i.e. $\mu \in \mathcal{M}$. The condition $\mu \in \mathcal{M}$ is not a serious constraint if we include *all* computable probability distributions in $\mathcal{M}$ with high weights assigned to simple μ_i. Solomonoff's universal semi-measure is obtained if we include all enumerable semi-measures in $\mathcal{M}$ with weights $w_{\mu_i} \sim 2^{-K(\mu_i)}$, where $K(\mu_i)$ is the length of the shortest program for μ_i (Hutter, 2001a). One can show that the conditional ξ and μ probabilities rapidly converge to each other:

$$\xi(x_t|x_{<t}) \to \mu(x_t|x_{<t}) \quad \text{with } \mu \text{ probability 1.} \quad (1)$$

Since the conditional probabilities are the basis of the decision algorithms considered in this work, we expect a good prediction performance if we use ξ as a guess of μ.

Bayesian decisions: Let $\ell_{x_t y_t} \in [0,1]$ be the received loss when predicting $y_t \in \mathcal{Y}$, but $x_t \in \mathcal{X}$ turns out to be the true t^{th} symbol of the sequence. Let $L_{n\Lambda_\rho}$ be the total expected loss for the first n symbols of the Bayes predictor Λ_ρ which minimizes the ρ expected loss. For instance for $\mathcal{X} = \mathcal{Y} = \{0,1\}$, Λ_ρ is a threshold strategy with $y_t^{\Lambda_\rho} = 0/1$ for $\rho(1|x_{<t}) \gtrless \gamma$, where $\gamma := \frac{\ell_{01} - \ell_{00}}{\ell_{01} - \ell_{00} + \ell_{10} - \ell_{11}}$. Let Λ be *any* prediction scheme (deterministic or probabilistic) with no constraint at all, taking *any* action $y_t^\Lambda \in \mathcal{Y}$ with total expected loss $L_{n\Lambda}$. If μ is known, Λ_μ is obviously the best prediction scheme in the sense of achieving minimal expected loss $L_{n\Lambda_\mu} \leq L_{n\Lambda}$ for any Λ. For the predictor Λ_ξ based on the universal distribution ξ, on can show $L_{n\Lambda_\xi}/L_{n\Lambda_\mu} = 1 + O(\sqrt{K(\mu)/L_{n\Lambda_\mu}})$, i.e. Λ_ξ has optimal asymptotics for $L_{n\Lambda_\mu} \to \infty$ with rapid convergence of the quotient to 1. If $L_{\infty\Lambda_\mu}$ is finite, then also $L_{\infty\Lambda_\xi}$ (Hutter, 2001a).

More active systems: Prediction means guessing the future, but not influencing it. One step in the direction to more active systems was to allow the Λ system to act and to receive a loss $\ell_{x_t y_t}$ depending on the action y_t and the outcome x_t. The probability μ is still independent of the action, and the loss function ℓ^t has to be known in advance. This ensures that the greedy Λ_μ strategy is still optimal. The loss function can also be generalized to depend on the history $x_{<t}$ and on t.

Agents in known probabilistic environments: The full model of an acting agent influencing the environment has been developed in (Hutter, 2001b). The probability of the next symbol (input, perception) x_t depends in this case not only on the past sequence $x_{<t}$ but also on the past actions (outputs) $y_{1:t}$, i.e. $\mu = \mu(x_t|x_{<t}y_{1:t})$. We call probability distributions of this form *chronological*. The total μ expected loss is $\sum_{x_{1:n}} (\ell^1 + ... + \ell^n)\mu(x_{1:n}|y_{1:n})$, where we assumed a total number of n interaction cycles. Action $y_t(x_{<t}y_{<t})$ and loss function $\ell^t(x_{1:t}y_{1:t})$ may depend on the complete history, which allows planning and delayed loss assignment.

Sequential decision theory: The goal is to perform the actions which minimize the total μ expected loss:

$$y_t := \arg\min_{y_t} \sum_{x_t} ... \min_{y_n} \sum_{x_n} (\ell^1 + ... + \ell^n)\mu(x_{1:n}|y_{1:n}), \quad (2)$$

$$L_{n\Lambda_\mu} = \min_{y_1} \sum_{x_1} ... \min_{y_n} \sum_{x_n} (\ell^1 + ... + \ell^n)\mu(x_{1:n}|y_{1:n}). \quad (3)$$

The minimization over y_t is in chronological order to correctly incorporate the dependency of x_t and y_t on the history. Note that y_t only depends on the known history $x_{<t}y_{<t}$, whereas minima and expectations are taken over the unknown $x_{t:n}y_{t:n}$ variables. The policy (2) (called AIμ model) is optimal in the sense that no other policy leads to lower μ-expected loss.

Bellman equations: In the case that ℓ^t is independent of $y_{<t}$ and μ is independent of $y_{1:n}$, policy (2) reduces to the greedy Bayes Λ_μ strategy. For (completely observable) Markov Decision Processes $\mu = \mu(x_t|x_{t-1}y_t)$ (2) and (3) can be written as recursive Bellman equations of sequential decision theory with state space $\mathcal{X}$, action space $\mathcal{Y}$, state transition matrix $\mu(x_t|x_{t-1}y_t)$, rewards $-\ell^t$, *etc.* The general (non-MDP) case may also be (artificially) reduced to Bellman equations by identifying complete histories $x_{<t}y_{<t}$ with states and $\mu(x_t|x_{<t}y_{1:t})$ with the state transition matrix. Due to the use of complete histories as state space, the AIμ model neither assumes stationarity, nor the Markov property, nor complete accessibility of the environment. But since every state occurs at most once in the lifetime of the system the explicit formulation (2) is more useful than a pseudo-recursive Bellman equation form. There is no principle problem in determining y_k as long as μ is known and computable and $\mathcal{X}$, $\mathcal{Y}$ and n are finite.

Reinforcement learning for unknown environment: Things dramatically change if μ is unknown. Reinforcement learning algorithms are commonly used in this case to learn the unknown μ (or directly a value function). They succeed if the state space is either small or has effectively been made small by generalization or function approximation techniques. In almost all approaches, the solutions are either *ad hoc*, or work in restricted domains only, or have serious problems with state space exploration versus exploitation, or have non-optimal learning rate. Below we propose the AIξ model as a universal and optimal solution to these problems.

Unknown loss function: Furthermore, the loss function $\ell^t(x_{1:t}y_{1:t})$ may also be unknown, but there is an easy "solution" to this problem. The specification of the loss function can be absorbed in the probability distribution μ by increasing the input space $\mathcal{X}$. Let $x_t \equiv x'_t l_t$, where x'_t is the regular input, l_t is interpreted as the loss, $\ell^t(x_{1:t}y_{1:t})$ is replaced by l_t in (2) and (3), and μ is only non-zero if l_t is consistent with the loss, i.e. $l_t = \ell^t(x_{1:t}y_{1:t})$. In this way all possible unknowns are absorbed in μ.

The universal AIξ model: Encouraged by the good performance of the universal sequence predictor Λ_ξ, we propose a new model, where the probability distribution μ is learned indirectly by replacing it with a universal prior ξ. We define $\xi(x_{1:n}|y_{1:n}) := \sum_{\mu_i \in \mathcal{M}} w_{\mu_i} \cdot \mu_i(x_{1:n}|y_{1:n})$ as a weighted sum over chronological probability distributions in $\mathcal{M}$. Convergence $\xi(x_n|x_{<n}y_{1:n}) \to \mu(x_n|x_{<n}y_{1:n})$ can be proven analogously to (1). Replacing μ by ξ in (2) the *AIξ system* outputs

$$y_t := \arg\min_{y_t} \sum_{x_t} ... \min_{y_n} \sum_{x_n} (l_t + ... + l_n)\xi(x_{1:n}|y_{1:n}) \quad (4)$$

in cycle t given the history $x_{<t}y_{<t}$, where $x_t \equiv x'_t l_t$. The largest class $\mathcal{M}$ which is necessary from a computational point of view is the set of all enumerable chronological semi-measures with weights $w_{\mu_i} \sim 2^{-K(\mu_i)}$, where $K(\mu_i)$ is the Kolmogorov complexity of μ_i. Apart from the dependence on the horizon n and unimportant details, the AIξ system is uniquely defined by (4) without adjustable parameters. It does not depend on any assumption about the environment apart from being generated by some computable (but unknown!) probability distribution in $\mathcal{M}$.

Universally optimal AI systems: We want to call an AI model *universal*, if it is μ-independent (unbiased, model-free) and is able to solve any solvable problem and learn any learnable task. Further, we call a universal model, *universally optimal*, if there is no program which can solve or learn significantly faster (in terms of interaction cycles). As the AIξ model is parameterless, ξ rapidly converges to μ in the sense of (1), the AIμ model is itself optimal, and we expect no other model to converge faster to AIμ (in some sense) by analogy to the sequence prediction case, we risk the conjecture that AIξ is such a universally optimal system. Further support is given in (Hutter, 2001b) by a detailed analysis of the behaviour of AIξ for various problem classes, including prediction, optimization, games, and supervised learning. We discuss in which sense AIξ overcomes some fundamental problems in reinforcement learning, like generalization, optimal learning rates, exploration versus exploitation, *etc.* Computational issues are also addressed.

References

Hutter, M. (2001a). General loss bounds for universal sequence prediction. *Proceedings of the 18th International Conference on Machine Learning (ICML-2001)*, 210–217. ftp.idsia.ch/pub/techrep/IDSIA-03-01.ps.gz.

Hutter, M. (2001b). Towards a universal theory of artificial intelligence based on algorithmic probability and sequential decisions. *Proceedings of the 12th Eurpean Conference on Machine Learning (ECML-2001)*, 226–238. ftp.idsia.ch/pub/techrep/IDSIA-14-00.ps.gz.

Gradient-based Reinforcement Planning in Policy-Search Methods

Ivo Kwee IVO@IDSIA.CH
Marcus Hutter MARCUS@IDSIA.CH
Juergen Schmidhuber JUERGEN@IDSIA.CH
Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA), Galleria 2, CH-6928 Manno, Switzerland

1. Introduction

It has been shown that *planning* can dramatically improve convergence in reinforcement learning (RL) (Schmidhuber, 1990; Sutton & Barto, 1998). However, most RL methods that explicitly use planning that have been proposed are value (or Q-value) based methods, such as *Dyna-Q* or *prioritized sweeping*.

Recently, much attention is directed to so-called *policy-gradient* methods that improve their policy directly by calculating the derivative of the future expected reward with respect to the policy parameters. Gradient based methods are believed to be more advantageous than value-function based methods in huge state spaces and in POMDP settings. Probably the first gradient based RL formulation is class of REINFORCE algorithms of Williams (Williams, 1992). Other more recent methods are, e.g., (Baird, 1998; Baxter & Bartlett, 1999; Sutton et al., 2000). Our approach of deriving the gradient has the flavor of (Ng et al., 1999) who derive the gradient using future state probabilities.

Our novel contribution in this paper is to combine gradient-based learning with explicit planning. We introduce "gradient-based reinforcement planning" (GREP) that improves a policy *before* actually interacting with the environment. We derive the formulas for the exact policy gradient and confirm our ideas in numerical experiments. GREP learns the action probabilities of a probabilistic policy for discrete problem. While we will illustrate GREP with a small MDP maze, it may be used for the hidden state in POMDPs.

2. Derivation of the policy gradient

PROJECTION MATRIX

Let us denote the discrete space of *states* by $\mathcal{S} = \{1,...,N\}$. Our belief on $\mathcal{S}$ is decribed by a probability vector $\mathbf{s}$ of which each element s_i represents the probability of being in state i. We also define a set of actions $\mathcal{A} = 1,...,K$. The stochastic policy $\mathcal{P}$ is represented by a matrix $\mathbf{P} : N \times K$ with elements $P_{ki} = p(a_k|s_i)$, i.e. the conditional probability of action k in state i.[1] Furthermore, let environment $\mathcal{E}$ be defined by transition matrices $\mathbf{T}_k$ $(k = 1,...,K)$ with elements $T_{kji} = p(s_j|s_i, a_k)$, i.e. the transition probability to s_j in state s_i given action k.

Now we define the *projection matrix* $\mathbf{F}$ with elements

$$F_{ji} = \sum_k T_{kji} P_{ki}. \tag{1}$$

Important is that matrix $\mathbf{F}$ is *not* modelling the transition probabilities of the environment, but models the induced transition probability using policy $\mathcal{P}$ in environment $\mathcal{E}$. The induced transition probability F_{ji} is a weighted sum over actions k of the transition probabilities T_{kij} with the policy parameters P_{ki} as the weights.

EXPECTED STATE OCCUPANCY

Using the projection matrix $\mathbf{F}$, states $\mathbf{s}_t$ and $\mathbf{s}_{t+1}$ are related as $\mathbf{s}_{t+1} = \mathbf{F}\mathbf{s}_t$ and therefore $\mathbf{s}_t = \mathbf{F}^t\mathbf{s}_0$, where $\mathbf{s}_0$ is the state probability distribution at $t = 0$. We can now define the *expected state occupancy* as

$$\mathbf{z} = E[\mathbf{s}|\mathbf{s}_0] = \sum_{t=0}^{\infty}(\gamma\mathbf{F})^t\mathbf{s}_0 = (\mathbf{I} - \gamma\mathbf{F})^{-1}\mathbf{s}_0 \tag{2}$$

where γ is a discount factor in order to keep the sum finite. In the last step, we have recognized the sum as the Neumann representation of the inverse. Notice that $\mathbf{z}$ is a solution of the linear equation

$$(\mathbf{I} - \gamma\mathbf{F})\mathbf{z} = \mathbf{s}_0 \tag{3}$$

[1] For computational reasons, one often reparameterizes the policy using a Boltzmann distribution. Here in this paper the probability $p(a_k|s_i)$ is just given by P_{ki} and we do not use reparameterization in order to keep the analysis clear.

which is just the familiar Bellman equation for the expected occupancy probability $\mathbf{z}$.

EXPECTED REWARD FUNCTION

In *reinforcement learning* (RL) the objective is to maximize future reward. We define a reward vector $\mathbf{r}$ in the same domain as $\mathbf{s}$. Using the expected occupancy $\mathbf{z}$ the future expected reward H is simply

$$H = \langle \mathbf{r}, \mathbf{z} \rangle \tag{4}$$

where $\langle \cdot, \cdot \rangle$ is the scalar vector product. Because $\mathbf{z}$ is a solution of Eq. 3 it is dependent on $\mathbf{F}$ which in turn depends on policy $\mathbf{P}$. Given $\mathbf{r}$ and $\mathbf{s}_0$, our task is to find the optimal $\mathbf{P}^*$ such that H is maximized, i.e.

$$\mathbf{P}^* = \arg\max_{\mathbf{P}} H. \tag{5}$$

CALCULATION OF THE POLICY GRADIENT

A variation $\delta\mathbf{z}$ in the expected occupancy can be related to first order to a perturbation $\delta\mathbf{P}$ in the (stochastic) policy. To obtain the partial derivatives $\partial\mathbf{z}/\partial P_{ik}$, we differentiate Eq. 3 with respect to P_{ik} and obtain:

$$-\gamma \frac{\partial \mathbf{F}}{\partial P_{ik}} \mathbf{z} + (\mathbf{I} - \gamma \mathbf{F}) \frac{\partial \mathbf{z}}{\partial P_{ik}} = 0. \tag{6}$$

The right hand side of the equation is zero because we assume that $\mathbf{s}_0$ is independent of P_{ik}.

From Eq. 4 and Eq. 6, together with the chain rule, we obtain the gradient of the RL error with respect to the policy parameters P_{ik}:

$$\frac{\partial H}{\partial P_{ik}} = \frac{\partial H}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial P_{ik}} = \gamma \left\langle \mathbf{K}^* \mathbf{r}, \frac{\partial \mathbf{F}}{\partial P_{ik}} \mathbf{z} \right\rangle \tag{7}$$

where A^* means the adjoint operator of A defined by $\langle u, Aw \rangle = \langle A^* u, w \rangle$. Let us define: $\mathbf{q} = \mathbf{K}^* \mathbf{r}$. While $\mathbf{K}$ maps the initial state $\mathbf{s}_0$ to the future expected state occupancy $\mathbf{z}$, its adjoint, $\mathbf{K}^*$, maps the reward vector $\mathbf{r}$ back to *expected reward* $\mathbf{q}$. The value of q_i represents the (pointwise) expected reward in state s_i for policy $\mathbf{P}$.[2]

Finally, differentiating Eq. 1 gives us $\partial\mathbf{F}/\partial P_{ik}$. Inserting this into Eq. 7 yields:

$$G_{ik} = \frac{\partial H}{\partial P_{ik}} \propto z_i \sum_j T_{kji} q_j. \tag{8}$$

[2]Indeed, this is a different way to define the traditional *value-function*. Note that generally, neither $\mathbf{r}$ nor $\mathbf{q}$ are probabilities because their 1-norm is generally not 1.

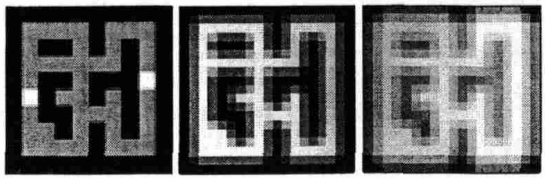


Figure 1. Left: 10 × 10 toy maze with start at left and goal at right side. Center: plot of expected occupancy $\mathbf{z}$. Right: plot of expected reward $\mathbf{q}$. White corresponds to higher probability. [Blurring is due to visualisation only].

In words, the gradient of H with respect to policy parameter P_{ik} (i.e. the probability of taking action a_k in state s_i) is proportional to the expected occupancy z_i times the weighted sum of expected reward q_j over next states ($j = 1, ..., N$) weighted by the transition probabilities T_{kji}.

3. Computation of the optimal policy

If the environment transition probabilities T_{kji} are known, the agent may improve its policy using GREP. Our GREP planning algorithm consist of two steps:

1. *Plan ahead:* Compute the policy gradient $\mathbf{G}$ in Eq. 7 and improve current policy

$$\mathbf{P} \leftarrow \mathbf{P} + \alpha \mathbf{G} \tag{9}$$

   where α is a suitable step size parameter; for efficiency we can also perform a linesearch on α.

2. *Evaluate policy:* Repeat above until policy is optimal.

Matrix $\mathbf{P}$ describes a probabilistic policy. We define the *maximum probable policy* (MPP) to be the deterministic policy by taking the maximum probable action at each state. It is not obvious that the MPP policy will converge to the global optimal solution but we expect MPP at least to be near-optimal.

4. Numerical experiments

We performed some numerical experiments using GREP. Our test problem was a planning task in a 10 × 10 toy maze (see Fig. 1) where the probabilistic policy $\mathbf{P}$ represents the probability of taking a certain action at a certain maze position. The same figure also shows typical solutions for the quantities $\mathbf{z}$ and $\mathbf{q}$, i.e. the expected occupancy and expected reward respectively (for certain $\mathbf{P}$).

After each GREP iteration, i.e. after each gradient calculation and $\mathbf{P}$ update, we checked the obtained policy

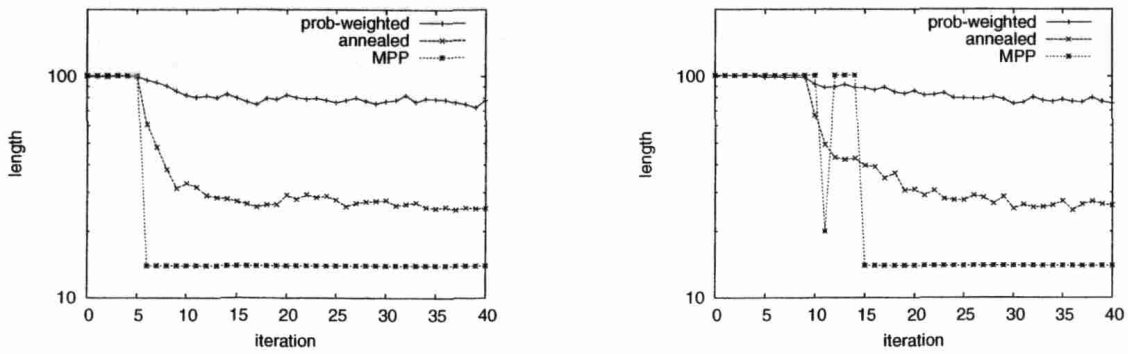


Figure 2. Plot of simulated path length versus GREP iteration of a small toy MDP maze for the probability weighted (PW) policy, annealed PW policy and MPP policy. The shortest path to goal is 14. Left: starting from initial uniform policy. Right: starting from initial random policy.

by running 20 simulations using the current value of $\mathbf{P}$. The probability weighted (PW) policy selects action k at state i proportional to P_{ik}, while the annealed PW policy uses an annealing factor of $T = 4$; we also simulated the MPP solution. Figure 2 shows the average simulated path length versus GREP iteration of the PW, the annealed PW policy and the derived MPP policy. In the left plot the initial policy $\mathbf{P}$ was taken uniform. The right plot in the same figure shows the simulated path lengths from a random policy; also here the MPP finds the optimal solution but slightly later.

We clearly see from the figure that in both cases the probability-weighted (PW) policy is improving during the GREP iterations. However, the convergence is very slow which shows the severe non-linearity of the problem. The annealed PW does perform better than PW. Finally, we see that MPP finds the optimal solution quickly within a few iterations. Using Dijkstra's method, we confirmed that the found MPP policy was in agreement with the global shortest path solution.

5. Conclusions

We have introduced a learning method called "gradient-based reinforcement planning" (GREP). GREP needs a model of the environment and plans ahead to improve its policy *before* it actually acts in the environment. We have derived formulas for the exact policy gradient.

Numerical experiments suggest that the probabilistic policy indeed converges to an optimal policy—but quite slowly. We found that (at least in our toy example) the optimal solution can be found much faster by annealing or simply by taking the most probable action at each state.

Further work will be to incorporate GREP in online RL learning tasks where the environment parameters, i.e. transition probabilities T_{kji}, are unknown and have to be learned. While an analytical solution for $\mathbf{q}$ and $\mathbf{z}$ are only viable for small problem sizes, for larger problems we probably need to investigate Monte Carlo or DP methods.[3]

References

Baird, L. C. (1998). Gradient descent for general reinforcement learning. *Advances in Neural Information Processing Systems.* MIT Press.

Baxter, J., & Bartlett, P. (1999). *Direct gradient-based reinforcement learning: I. gradient estimation algorithms* (Technical Report). Research School of Information Sciences and Engineering, Australian National University.

Ng, A., Parr, R., & Koller, D. (1999). Policy search via density estimation. *Advances in Neural Information Processing Systems.* MIT Press.

Schmidhuber, J. (1990). An on-line algorithm for dynamic reinforcement learning and planning in reactive environments. *Proc. IEEE/INNS International Joint Conference on Neural Networks, San Diego* (pp. 253–258).

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning. An introduction.* MIT Press, Cambridge.

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

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

[3]This work was supported by SNF grants 21-55409.98 and 2000-61847.00

Policy Improvement for several Environments

Andreas Matt ANDREAS.MATT@UIBK.AC.AT
Georg Regensburger GEORG.REGENSBURGER@UIBK.AC.AT
Institute of Mathematics[1], University of Innsbruck, Austria

Abstract

In this paper we state a generalized form of the policy improvement algorithm for reinforcement learning. This new algorithm can be used to find stochastic policies that optimize single-agent behavior for several environments and reinforcement functions simultaneously. We first introduce a geometric interpretation of policy improvement, define a framework to apply one policy to several environments, and propose the notion of balanced policies. Finally we explain the algorithm and present examples.

1. Idea

Until now reinforcement learning has been applied to learn behavior within one environment. Several methods to find optimal policies for one environment are known (Kaelbling et al., 1996; Sutton & Barto, 1998).

In our research we focus on a general point of view of behavior that appears independently from a single environment. As an example imagine that a robot should learn to avoid obstacles, a behavior suitable for more than one environment. Obviously a policy for several environments cannot - in general - be optimal for each one of them. Improving a policy for one environment may result in a worse performance in an other. Nevertheless it is often possible to improve a policy for several environments. Compared to multiagent reinforcement learning as in Bowling and Veloso (2000), where several agents act in one environment, we have one agent acting in several environments.

2. Equivalent and Improving Policies

We fix a finite Markov Decision Process $(S, \mathbf{A}, \mathbf{P}, \mathbf{R})$, use the standard definitions of value function V and Q-value and write $\pi(a \mid s)$ for the probability that action a is chosen in state s. We say that two policies π and $\tilde{\pi}$ are equivalent if their value functions coincide, i.e. $V^{\pi} = V^{\tilde{\pi}}$.

Theorem 1 *Two policies $\tilde{\pi}$ and π are equivalent if and only if*

$$\sum\nolimits_{a \in A} Q^{\pi}(a, s)\tilde{\pi}(a \mid s) = V^{\pi}(s) \text{ for all } s \in S.$$

This gives us a description of the equivalence class of a policy. We interpret $\pi(- \mid s)$ as a point on a standard simplex and the equivalence class as the intersection of the hyperplane H defined by $Q^{\pi}(- \mid s)$ and $V^{\pi}(s)$ with the simplex. See Figure 1 *left* for an example with three actions a_1, a_2 and a_3. The following theorem is

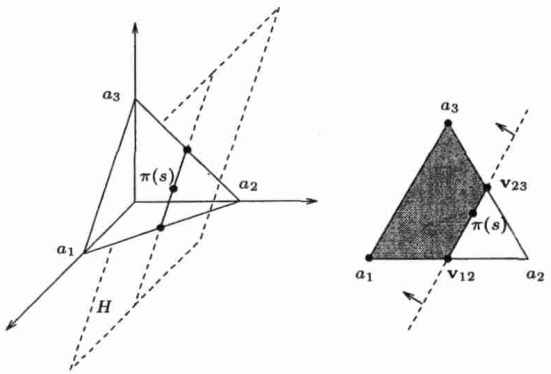


Figure 1. left: A policy in state s and its equivalence class *right:* Improving policies in state s

a general version of the policy improvement theorem.

Theorem 2 *Let π and $\tilde{\pi}$ be policies such that*

$$\sum\nolimits_{a \in A} Q^{\pi}(a, s)\tilde{\pi}(a \mid s) \geq V^{\pi}(s) \text{ for all } s \in S.$$

Then $V^{\tilde{\pi}} \geq V^{\pi}$. If additionally there exists an $s \in S$ such that $\sum Q^{\pi}(a, s)\tilde{\pi}(a \mid s) > V^{\pi}(s)$ then $V^{\tilde{\pi}} > V^{\pi}$.

We define the *set of improving policies* for π in s by

$$C^{\pi}_{\geq}(s) = \left\{ \tilde{\pi}(- \mid s) : \sum Q^{\pi}(a, s)\tilde{\pi}(a \mid s) \geq V^{\pi}(s) \right\},$$

[1]We wish to thank Prof. Ulrich Oberst for his motivation, comments and support. This research was partially supported by "Forschungsstipendien an österreichische Graduierte" and Project Y-123 INF.

and in analogy the *set of strictly improving policies* $C^\pi_>(s)$ and the *set of equivalent policies* $C^\pi_=(s)$ for π in s. We define the *set of strictly improving actions* of π in s by $A^\pi_>(s) = \{a : Q^\pi(a,s) > V^\pi(s)\}$.

The set of improving policies $C^\pi_\geq(s)$ is a polytope given by the intersection of a half-space and a standard simplex. Its vertices are $\text{vert}\left(C^\pi_\geq(s)\right) = \text{vert}(C^\pi_=(s)) \cup A^\pi_>(s)$. See Figure 1 *right*, where $A^\pi_>(s) = \{a_1, a_3\}$, $\text{vert}(C^\pi_=(s)) = \{\mathbf{v}_{12}, \mathbf{v}_{23}\}$ and $C^\pi_\geq(s)$ is the shaded area, the side marked by the small arrows.

3. Policies for several Environments

Consider a robot and its sensors to perceive the world. All possible sensor values together represent all possible states for the robot. In each of these states the robot can perform some actions. We call all possible states and actions the *state action space (SAS)* $\mathbb{E} = (S, \mathbf{A})$. Now we put the robot in a physical environment, where we can observe all possible states for this environment, a subset $S_E \subset S$ of all possible states in general, and the transition probabilities $\mathbf{P}_E$. We call $E = (S_E, \mathbf{A}, \mathbf{P}_E)$ a *realization* of an SAS.

Let $\mathbf{E} = (E_i, \mathbf{R}_i)_{i=1\ldots n}$ be a finite family of realizations of an SAS with rewards $\mathbf{R}_i$. Since the actions are given by the SAS it is clear what is meant by a policy π for $\mathbf{E}$. For each $(E_i, \mathbf{R}_i)$ we can calculate the value function, which we denote by V_i^π. We define the *set of improving policies* of π in $s \in S$ by

$$C^\pi_\geq(s) = \bigcap\nolimits_{i\in[n],\ s\in S_i} C^\pi_{i,\geq}(s)$$

and the *set of strictly improving policies* of π in s by

$$C^\pi_>(s) = \left\{ \begin{array}{c} \tilde\pi(- \mid s) \in C^\pi_\geq(s) \text{ such that} \\ \exists\, i \in [n] \text{ with } \tilde\pi(- \mid s) \in C^\pi_{i,>}(s) \end{array} \right\},$$

where $[n] = \{1, \ldots, n\}$. The set of improving policies of π in s is the intersection of a finite number of half-spaces through a point with a standard simplex.

Theorem 3 *Let $\tilde\pi$ be a policy for $\mathbf{E}$ such that*

$$\tilde\pi(- \mid s) \in C^\pi_\geq(s) \text{ for all } s \in S.$$

Then $V_i^{\tilde\pi} \geq V_i^\pi$ for all $i \in [n]$. If additionally there exist an $s \in S$ with $\tilde\pi(- \mid s) \in C^\pi_>(s)$ then there exists an $i \in [n]$ such that $V_i^{\tilde\pi} > V_i^\pi$.

In order to describe $C^\pi_\geq(s)$ and find an $\tilde\pi(- \mid s) \in C^\pi_>(s)$ we consider its vertices. We call the vertices of $C^\pi_\geq(s)$ *improving vertices* and define the *strictly improving vertices* by $\text{vert}(C^\pi_>(s)) = \text{vert}(C^\pi_\geq(s)) \cap C^\pi_>(s)$. There exist several algorithm to find all vertices of a polytope (Fukuda, 2000). Linear Programming methods can be used to decide whether there exist strictly improving vertices and to find one (Schrijver, 1986). Observe that for a single environment the strictly improving vertices are just the set of strictly improving actions.

Let $s \in S$. We define π_s as the set of all policies that are arbitrary in s and equal π otherwise. We call a policy *balanced* if and only if for all $s \in S$ and all $\tilde\pi \in \pi_s$ either $V_i^{\tilde\pi} = V_i^\pi$ for all $i \in [n]$ or there exists $i \in [n]$ such that $V_i^{\tilde\pi} < V_i^\pi$. This means that if one changes a balanced policy in one state s it is the same for all environments or it gets worse in at least one. Compare to the notion of an equilibrium point in game theory (Nash, 1951). Note that for one environment the notions of optimal and balanced policies coincide.

Theorem 4 *A policy π is balanced if and only if there are no strictly improving policies, i.e. $C^\pi_>(s) = \emptyset$ for all $s \in S$.*

4. General Policy Improvement

We state a generalized form of the policy improvement algorithm for a family of realizations of an SAS which we call *general policy improvement* (algorithm 1). The idea is to improve the policy by choosing in each state a strictly improving vertex. If there are no strictly improving vertices the policy is balanced and the algorithm terminates.

Input: a policy π and a family of realizations $(E_i, \mathbf{R}_i)$
Output: a balanced policy $\tilde\pi : V_i^{\tilde\pi} \geq V_i^\pi$ for all $i \in [n]$

```
π̃ ⟵ π
repeat
    calculate V_i^π̃ and Q_i^π̃ for all i ∈ [n]
    for all s ∈ S do
        if vert(C>^π̃(s)) ≠ ∅ then
            choose π'(− | s) ∈ vert(C>^π̃(s))
            π̃(− | s) ⟵ π'(− | s)
until vert(C>^π̃(s)) = ∅ for all s ∈ S
```

Algorithm 1: General Policy Improvement

In each step of the algorithm we try to choose a strictly improving vertex. Different choices may result in different balanced policies and influence the number of improvement steps before termination. The algorithm includes policy improvement for one environment as a special case.

A geometric interpretation of one step of the general policy improvement algorithm for three states and two realizations can be seen in Figure 2. In state s_1 there are no strictly improving vertices. In state s_2 there are three strictly improving vertices, one of them is the action a_3. In state s_3 there are only two, both of

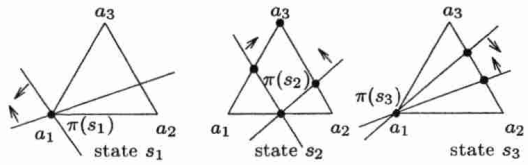


Figure 2. Improving policies for two realizations

them a stochastic combination of a_2 and a_3.

5. Examples

All experiments are made with a 10x10 gridworld simulator to learn an obstacle avoidance behavior. The robot has 4 sensors, forward, left, right, and back, with a range of 5 blocks each. There are 3 actions in each state: move forward, left and right. The robot gets rewarded if it moves away from obstacles, it gets punished if it moves towards obstacles.

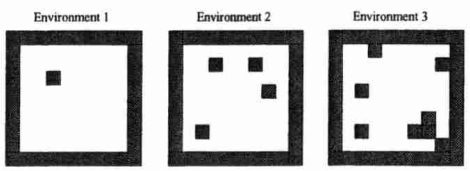


Figure 3. Three different environments

We choose three environments (see Figure 3) with the same reinforcement function and run the algorithm. In all experiments we start with the random policy. We calculate in each step all strictly improving vertices and choose one randomly. In order to evaluate and compare policies we consider the average utilities of all states, and normalize it, with 1 being an optimal and 0 the random policy in this environment. Four sample experiments show performances in each environment of the different balanced policies learned.

Experiment:	1	2	3	4
Environment 1	0.994	0.771	0.993	0.826
Environment 2	0.862	0.876	0.788	0.825
Environment 3	0.872	0.905	0.975	0.878

Figure 4 shows the progress of the algorithm for each environment in experiment 2. In all experiments the algorithm terminates after 6 to 8 iteration steps.

6. Discussion

The general policy improvement algorithm can be used to improve a policy for several realizations of a state

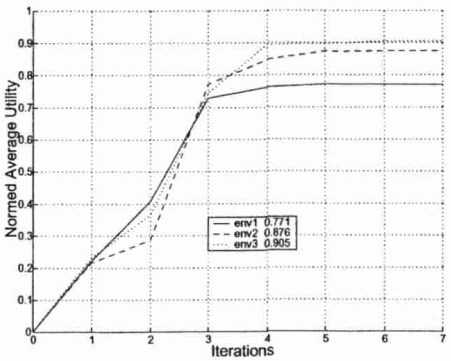


Figure 4. The progress of general policy improvement for each environment.

action space simultaneously. This means that it can be used to learn a policy for several environments and several reinforcement functions together. A useful application is to add a new environment or behavior to an already optimal policy, without changing its performance. We have already implemented Value Iteration for several realizations which leads to an extension of Q-learning. Our future research focuses on the implementation of on-line algorithms, methods to find the strictly improving vertices and to decide which of them are best regarding to learning speed. For more detailed information please consult the extended version of this paper on `http://mathematik.uibk.ac.at/~rl`.

References

Bowling, M., & Veloso, M. (2000). *An analysis of stochastic game theory for multiagent reinforcement learning* (Technical Report CMU-CS-00-165).

Fukuda, K. (2000). Frequently asked questions in polyhedral computation. `http://www.ifor.math.ethz.ch/~fukuda/polyfaq/polyfaq.html`.

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

Nash, J. (1951). Non-cooperative games. *Ann. of Math. (2), 54*, 286–295.

Schrijver, A. (1986). *Theory of linear and integer programming.* Chichester: John Wiley & Sons Ltd.

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

The Clustering Aliasing Problem in Reinforcement Learning for Robots

Rosana Matuk Herrera[1] RMATUK@DC.UBA.AR
[1] Departamento de Computación; Facultad de Ciencias Exactas y Naturales; Universidad de Buenos Aires; Argentina.
Juan Miguel Santos[1,2] JMSANTOS@DC.UBA.AR
[2] Center for Engineering Science; Advanced Research (CESAR–CSMD); Oak Ridge National Laboratory; P.O. Box 2008; Oak Ridge, TN 37831–635; USA.

1. Introduction

For the last ten years, Reinforcement Learning (RL) has been extensively used for the behavior synthesis in robot systems. One of the problems which has called the attention of several researchers has been to deal with the representation of huge state or state–action spaces in robot learning. Originally, techniques such as Q–Learning (Watkins, 1989) were proposed for finite state–action spaces and typically the Q–function could be represented directly in memory using tables. However, real robots can have a completely huge set of possible states and actions which makes it absolutely necessary to consider strategies to explore such space and memorize it in a compact way. The use of Artificial Neural Networks (ANN) to memorize and generalize huge state–action spaces has had a good impact on this issue. Basically, ANN work as function approximators (Lin, 1993) of the Q–function or as clustering techniques to partition the domain of such function. This strategy allows the robot to avoid the exhaustive exploration of its environment by using the generalization capability of the neural approaches. The use of ANN to cluster the Q–function domain has been implemented using different models such as CMAC (Albus, 1975), RBF (Kretchmar & Anderson, 1996) and Kohonen Maps (Sehad, 1996). On the other hand, Moore & Atkenson (1995) proposed a method to get a partitioning of the state space using a grid with a non homogeneous resolution. In this approach the task is specified by a goal region instead of a reward function and this could be a restrictive fact, depending on the task that the agent has to learn.

In spite of the clustering method used, to get clusters of state–action pairs, leads to another problem called *cluster aliasing* (CA). The CA problem arises when the reinforcement function (RF) provides different reinforcement for configurations belonging to the same cluster. For example, let us consider an RL scheme where the RF delivers immediate reinforcements. Now, let us suppose two state–action pairs p_1 and p_2 which are generalized by the same cluster and the RF delivers a reward for the pair p_1 and a penalty for the pair p_2. Obviously, the computation of the return term, in fact the Q–value, associated to the cluster will be distorted by this fact, except that there was a way to divide this cluster into two. Therefore, we need a way to measure this sort of situations and a method to continue partitioning the space in order to avoid a decrease of the learning performance. This is the subject of the work presented here.

2. Cluster Aliasing measure

We have proposed a CA measure (in immediate reinforcement schemes). Let c be a cluster, which has mapped, at least, a state–action pair with a reward associated, then CA measures how many state–action pairs associated with null or negative reinforcement have been also mapped by c. In order to weight the influence of null reinforcements or penalties, two factors are added. Formally,

$$CA(c) = \frac{p_- \# ref_- + p_0 \# ref_0}{\# ref_+ + \# ref_- + \# ref_0},$$

where $\#ref_-$, $\#ref_+$, $\#ref_0$ are the number of penalties, rewards and null reinforcements respectively, for the pairs mapped by the cluster c; and p_- and p_+ are parameter factors.

3. Algorithm CA–gcs

We have proposed a method which we called CA–gcs, to obtain a clustering of the state–action space which minimizes the clustering aliasing measure. Our proposal is based on the Growing Cell Structures (GCS) introduced some years ago by Fritzke (1994). In fact, there is an antecedent of our work by Großmann and Poli (2000) in which they propose a pre–learning phase where a robot collects experiences in a random way and then, these experiences are used to find a suitable segmentation of the input state space using GCS. However, these authors do not specify when the algorithm finds such a *suitable* segmentation.

Our algorithm has four stages:

1) *State Space Clustering Construction.* This stage implements the vector quantization variant of GCS (Fritzke, 1996) to get a partition of the state space. At the end of this stage the following constraint must be satisfied for any input ξ,

$$\forall c \in C, \quad \xi \in M(c) \Rightarrow |c_i - \xi_i| \le k, \quad i{:}1...\#\text{robot sensors}$$

where ξ_i is the i–component of the input ξ, c_i is the i–component of the cluster c, $M(c)$ is the set of inputs mapped by c, k is a parameter with $k \in \Re$, $k>0$, and C is the set of all clusters.

2) *Individual cluster training*. In this stage, the action a_c associated to each cluster c is tuned so that the reinforcement function delivers the maximum number of rewards for the pairs (s, a_c) where s is each one of the states mapped by cluster c.
3) *Partition guided by the clustering aliasing measure*. In this stage, all clusters with a CA measure greater than a parameter value will be partitioned. The aim is to minimize the number of states that are mapped by clusters whose actions associated do not match (or are not close to) a sub–optimal one.
4) *Assignation of Q–values*. Based on the aliasing measure, we assign a Q–value to each individual cluster.

These stages are preceded by a phase where the robot – using a random walk in its environment – collects experiences. Each one is stored as a tuple with three elements: the state and the conducted action at time t, and the state at time $t+1$. After that, the method uses this memory based on a Lazy Q–Learning strategy (Aha, 1997; Touzet, 1999).

4. Experiments

We have carried out a series of experiments using miniature Khepera robots to validate our method. System state sensed by the robots included a noise component (approximately ±10% of the measurement). We have considered two behaviors: *moving toward the maximal light source* and *obstacle avoidance*. In both cases, we have obtained GCS networks with decreasing clustering aliasing measures (controlling the parameter values) and with increasing performance of the synthesized behavior (respectively). Additionally, we have used the Kohonen networks to get a clustering of the state–action space and have measured their clustering aliasing. In this way, we could compare the performance of the synthesized behaviors with GCS and Kohonen (with equivalent number of neurons) concluding that systematically, the clustering representations obtained using CA–gcs allow the robot to learn the desired behaviors better. In Figure 1, we show the performance results for the obstacle avoidance behavior. In Figure 2, we show the performance results for the light searching behavior. Numbers beside each reference point in the figures represent the number of clusters (neurons) of the networks.

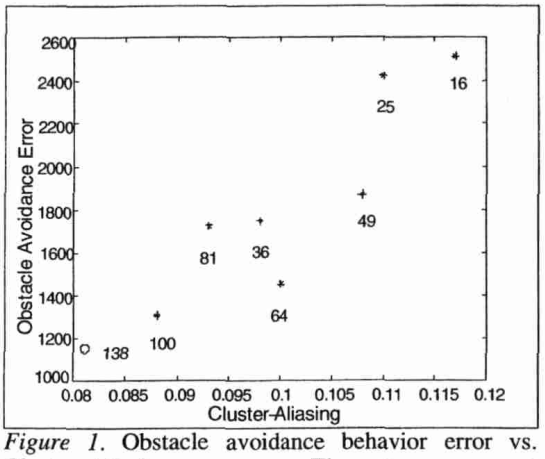


Figure 1. Obstacle avoidance behavior error vs. Cluster–Aliasing measure. The stars represent Kohonen networks and the circle the CA–gcs network. Values are averages obtained from series of 5 experiments.

5. Conclusions

In this work, we have introduced the CA–gcs method to deal with the cluster–aliasing problem in RL schemes which use clustering as a strategy to represent huge state–action spaces. We have given a measure of CA which is minimized by the proposed method. We obtained behaviors with a better performance in RL schemes which had a lower CA measure in their clustering of the state–action space.

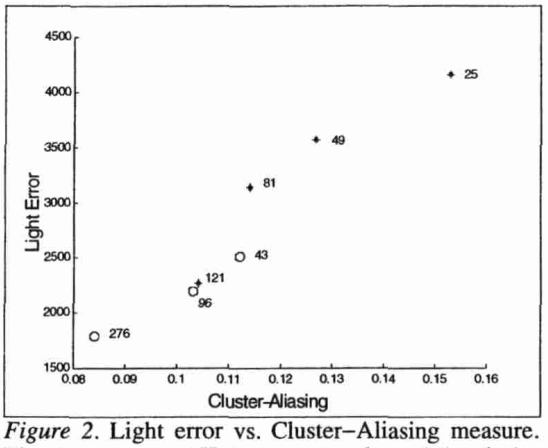


Figure 2. Light error vs. Cluster–Aliasing measure. The stars represent Kohonen networks and the circles the CA–gcs networks. Values are averages obtained from series of 5 experiments.

References

Aha, D. (ed.) (1997). *Lazy Learning*. Kluwer.

Albus, J. (1975). A new approach to manipulator control: The cerebellar model articulation controller (CMAC). *Trans. ASME, J.Dynamic Sys., Meas., Contr., (97).*

Fritzke, B. (1994). Growing Cell Structures – A self–organized network for unsupervised and supervised learning. *Neural Networks, 7* (9):1441–1460.

Fritzke, B. (1996). Unsupervised ontogenic networks. In E. Fiesler, & R. Beale, *Handbook of Neural Computation*. Institute of Physics Publishing and Oxford University Press.

Großmann, A., & Poli, R. (2000). Learning a navigation task in changing environments by multi–task reinforcement learning. *Lecture Notes in Artificial Intelligence*, (1812):23–43.

Kretchmar, R.M., & Anderson, C.W (1996). Comparison of CMACs and Radial Basis Functions for local Function Approximators in Reinforcement Learning. In *Proc. of the Inter.l Conference on Neural Networks'97*.

Lin, Long–Ji (1993). *Reinforcement Learning for Robots Using Neural Networks*. PhD Thesis, Carnegie–Mellon University.

Moore, A. W., & Atkeson, C. G. (1995). The Parti–game Algorithm for Variable Resolution Reinforcement Learning in Multidimensional State–spaces. *Machine Learning, 21*.

Sehad, S. (1996). *Contribution à l'étude et au développement de modèles connexionnistes à apprentissage par renforcement: Applications à l'acquisition de comportements adaptatifs*. PhD thesis, Université de Montpellier II.

Touzet, C. (1999). Programming Robots with Associative Memories. *IJCNN'99*, Washington DC, USA.

Watkins, C.J.C.H (1989). *Learning from Delayed Rewards*. PhD thesis, King´s College, Cambridge, UK.

A non supervised multi-reinforcement agents architecture to model the development of behavior of living organisms

Philippe Preux PREUX@LIL.UNIV-LITTORAL.FR
Christophe Cassagnabère CASSAGNA@LIL.UNIV-LITTORAL.FR
Samuel Delepoulle DELEPOULLE@LIL.UNIV-LITTORAL.FR
Laboratoire d'Informatique du Littoral, Université du Littoral Côte d'Opale, UPRES-JE 2335, B.P. 719, 62228 Calais Cedex, France

Jean-Claude Darcheville DARCHEVILLE@UNIV-LILLE3.FR
Unité de Recherche sur l'Évolution des Comportements et des Apprentissages, Université de Lille 3, UPRES-EA 1059, B.P. 149, 59653 Villeneuve d'Ascq Cedex, France

Our project is based on computer science and psychology. Its aim is to assess to which extent reinforcement learning (RL) can account for animal dynamics of behavior, that is, how behaviors are acquired, learnt, and evolve during lifespan. To this end, we are incrementally building an artificial creature that we call MAABAC which behavior is entirely controlled by RL algorithm. As such, MAABAC is subject to its enviroment to which it continuously adapt itself. We want MAABAC to be able to learn to reach and grasp objects, to walk, to run, ... in its "physical" environment. We have interest in the ability to reach an object as it is one of the first complex motor behavior that is exhibited by human babies. In the same time, we want that some features of learning are shared by animals and MAABAC (Multi-Agent Animat for Behavioral Arm Control). We put a strong emphasis on the development of new behaviors, that is, behaviors have to be acquired, or learnt, rather than merely being added as new components to MAABAC. This idea is shared with the BabyBot project (Metta et al., 2000). At first sight, it might seem that we follow a project which is close to the COG project at MIT (Brooks et al., 1998). However, as argued below, we want to restrict ourselves to the use of RL algorithms because that is of interest for the study of animal behavior, while COG has a more ingineer, pragmatic approach to build a robot. In the sequel, we present motivations and methodological choices, the basic architecture of MAABAC, the first stages of development of MAABAC, the first experiments and current conclusions. Along the text, we outline some points directly related to the research being performed in the RL community.

Reinforcement algorithms (Sutton & Barto, 1998) provide an interesting implementation of a fundamental law of animal behavior, namely the law of effect (Thorndike, 1911). Then, assuming that the behavior of living organisms is fundamentally based on the law of effect, we can try to build artefacts based on TD that learn their behavior through interacting with their environment.

Basically, the architecture of MAABAC is a non supervised multi-agent system (MAS). There are two reasons for this. The first reason regards modeling: it is natural that each "organ" maps one agent. The second is that of reducing the size of search space; assuming that each organ can be in one of any N states, a system made of k agents would yield a search space of size N^k, whereas a MAS approach leads to k problems of size N, thus $k \times N$, which is far smaller than N^k; interactions between the agents guarantee that constraints are respected; in summary, a MAS approach simplifies the search of space by each individual agent. Each agent is controlled by a TD algorithm.

We have designed an arm, originally introduced by S. Delepoulle in his PhD thesis (Delepoulle, 2000). This arm is made of two joint segments (bones); one extremity is the shoulder, the other is the fist (no hand for the moment), while the joint is the elbow. The position of each segment is controlled by two muscles, one flexor and one tensor. Each muscle can either contract, or relax itself a little bit more, or remains in its current state of contraction: these are the 3 possible actions of the agent, and the number of states of contraction is $N = 50$. The position of each segment of the arm is simply given by the difference of contraction of the two muscles controlling it. Each muscle is

an agent which behavior is controlled by a Q-learner so that to follow the law of effect. When the arm is activated, each state of contraction leads to a certain energetic cost, the more contracted, the more costly; the energetic cost is handled as a negative reward in each Q-learner; when the arm puts its fist in a certain target zone of the space, it gets a positive reward, while otherwise, it only receives a negative reward (energetic cost). Among the various tasks that could have been used, we consider the one aiming at putting the fist into the target zone. As far as each agent has only access to its current state of contraction, the whole task is non markovian. The whole architecture is thus a hybrid of non supervised and reinforcement learning.

This overall presentation being made, let us present and discuss the simulations that have been performed. When activated in an initially random position, the arm first shows erratic behaviors until its fist reaches the target zone for the first time. Then, the arm receives a positive reward for the first time and at that time, it "learns" that it can receive a reward. More precisely, the 4 muscles receive the same positive reward, either they have contributed to receive it, or not. After resetting its position and after a certain amount of reachings of the target zone, the movement of the arm becomes smooth and straightforward towards the zone. We call a "movement" the whole sequence of behaviors from the initial position to the target zone. Then, we have shown that the arm exhibits various abilities and perform multi-task reinforcement: learning to reach different positions of the target zone, learning to reach them from different initial positions, acquiring movements with an optimal trajectory (with regards to their energetic cost, considering this precise representation of space), extinction, generalization, shaping, tracking a moving target, escaping an uncomfortable zone, and avoiding such a zone. Clearly, these behaviors and movements are the outcome of the use of TD algorithms. The coordination and self-organization of muscles come from the fact that the reward is shared by the muscles while, in the same time, the negative reward due to energetic cost leads to the optimization of movements, thus their straightforwardness and their smoothness. The precision of the movements is bounded by the number of states of contraction of the muscles: the more states, the more precise the movement. To obtain finer movements while avoiding too long learning time, we have added a mechanism of state splitting (Moore & Atkeson, 1995).

Then, we have added a second arm to the artefact, itself driven by 4 muscles (exactly the same architecture as the first arm). We have done the same experiments as with the first arm. To reduce the time to learn correct movements for the two arms, we have also used the following idea. Once one arm had learnt several movements, we have used its Q-values to initialize the second arm. Then, the second arm is readily able to perform the same movements as the first, whereas both may still adapt freely their own behaviors to their own contingencies.

The third step has been to add a body on which the two arms are attached by their shoulders. The body can turn on itself under the action of a muscle, again controlled by a Q-learner. The whole artefact is then controlled by 9 Q-learners, without any central supervisor. It has shown its ability to learn optimal movements. We have noted that the learning time for this third step artefact scales very favorably with regards to the simple arm: actually, a movement is acquired faster by this artefact than by the version made of only two arms (without a body).

The next steps will be to add legs to the artefact as well as senses to let it perceive its environment. We are also considering using TD(λ) instead of Q-learning, foreseeing that it will speed-up learning, while remaining realistic with regards to the law of effect. We are also paying some attention to training to speed-up the initial exploratory phase, and study the interaction between supervised, reinforcement, and non supervised learnings.

References

Brooks, R., (Ferrell), C. B., Irie, R., Kemp, C., Marjanovic, M., Scassellati, B., & Williamson, M. (1998). Alternate essences of intelligence. *AAAI.*

Delepoulle, S. (2000). *Coopération entre agents adaptatifs ; étude de la sélection des comportements sociaux, expérimentations et simulations.* Doctoral dissertation, Université de Lille 3, URECA, Villeneuve d'Ascq. Thèse de doctorat de Psychologie.

Metta, G., Manzotti, R., Panerai, F., & Sandini, G. (2000). Development: is it the right way towards humanoid robotics? *IAS-6.* Venice, Italy.

Moore, A., & Atkeson, C. (1995). The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning, 21.*

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

Thorndike, E. (1911). *Animal intelligence: Experimental studies.* Mac Millan.

The Curse of Optimism

Stuart I. Reynolds SIR@CS.BHAM.AC.UK
School of Computer Science, The University of Birmingham, Birmingham, B15 2TT, UK

1. The max Operator and Optimistic Initial Biases

This paper introduces the following simple insight:

> RL algorithms that update their value estimates based upon a return estimate involving $\max_a Q(s,a)$ find it more difficult to overcome their initial value biases if these biases are optimistic.

To see that this is so, consider the example in Figure 1. Assume that all transitions yield a reward of 0. A learning algorithm is applied that adjusts $Q(s_1,a_1)$ towards $E[\max_a Q(s_2,a)]$. If all Q-values are initialised optimistically, to 10 for example, then the Q-values of *all* actions in s_2 must be readjusted (i.e. lowered towards zero) before $Q(s_1,a_1)$ may be lowered. However, if the Q-values are initialised pessimistically by the same amount (to -10), then $\max_a Q(s_2,a)$ is raised when the value of a *single* action in s_2 is raised. In turn, $Q(s_1,a_1)$ may then also be raised.

In general, it is clear that it is easier for RL algorithms employing $\max_a Q(s,a)$ in their return estimates to raise their Q-value predictions than to lower them. In effect, the max operator causes a resistance to change in value updates that can inhibit learning.

It is also clear that the effect of this is further compounded if *i*) the Q-values in s_2 are themselves based upon the over-optimistic values of their successors, or, *ii*) states have many actions available, and so many Q-values to adjust before $\max_a Q(s,a)$ may change.

Examples of methods that use $\max_a Q(s,a)$ in their return estimates and are affected by this phenomenon are: value-iteration (Sutton & Barto, 1998), Q-learning (Watkins, 1989), R-learning (Schwartz, 1993), Watkins' $Q(\lambda)$ (Watkins, 1989; Sutton & Barto, 1998), Peng and Williams' $Q(\lambda)$ (Peng & Williams, 1996) and their many derivatives.

Experimental results with value-iteration and Q-learning have shown that, when the effects of optimism on the agent's exploration strategy are accounted for, convergence towards the optimal Q-function generally proceeds more quickly if non-optimistic initial Q-values are provided. In some cases speedups of a factor 20 have been found with value-iteration by initialising the value function pessimistically than with the same amount of optimistic bias (Reynolds, 2001).

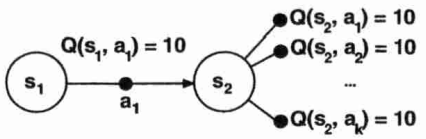


Figure 1. A simple process in which optimistic initial Q-values slows learning.

2. Discussion

It is not suggested that general improvements to RL algorithms can be achieved simply by making the agent's initial Q-function less optimistic. The reason for this is that in practical RL settings, agents often have the difficult task of managing a tradeoff between exploration (to gain more information about the rewards it can collect and the structure of the environment) and exploitation (taking actions that appear to lead to the highest return based upon the prior experience). Most effective exploration strategies are defined in terms of the current Q-function, so in general the initial choice of Q-function plays an important role determining how this tradeoff is managed. Moreover, optimism is often deliberately introduced into value estimates as a way of encouraging exploration, for example: (Meuleau & Bourgine, 1999; Wyatt, 1996; Wiering, 1999; Strens, 2000; Wyatt, 2001)).

We note that better methods may follow that explicitly model optimistic biases and Q-functions separately (Meuleau & Bourgine, 1999; Strens, 2000). If the Q-function is initialised non-optimistically, we can benefit by finding accurate Q-values more quickly. In turn, we might feed this improved Q-function into the exploration method. Improvements in the exploration strategy could arise since better value predictions should allow the agent to better predict the relative value of ex-

ploiting instead of exploring. To an extent these benefits may only be afforded to model-free techniques. At any time, model-learning methods may use their current model to generate value predictions without optimistic biases.

Distinguishing value-prediction from optimism for exploration generally seems like a good idea as we can now deal with these two conceptually different quantities separately (and it adds little to the overall computational complexities of the algorithms). It allows us to make explicit separations between exploration and exploitation – at any time we can decide to stop exploring completely and decide to exploit given the current Q-function. For example, this is useful in gambling or financial trading problems where we may wish to learn the relative return available for making bets or trading shares by initially exploring the problem with a small amount of capital. Later, if we decided to play the game for real and bet the farm for the expected return indicated by the learned Q-values, we might be extremely disappointed to find that this was in fact a gross over-estimate because of the optimistic biases introduced for exploration.

There are also other applications for which accurate Q-values are needed, but in which exploration is still required. An example is deciding whether or not (or where) to refine the agent's internal Q-function representation. This can be done based upon the differences of Q-values in adjacent parts of the space (Reynolds, 2000; Munos & Moore, 1999; Chapman & Kaelbling, 1991). In different RL frameworks, the agent may be learning to improve several independent policies that maximise several separate reward functions (Humphrys, 1996). Deciding which policy to follow at any time is done based upon the Q-values of the actions in each policy.

Initial experiments with algorithms that separate value estimates from optimism are encouraging. In particular, it is easy to construct algorithms that maintain identical exploration strategies to existing methods but typically reduce the distance to the optimal Q-function far more quickly. The experimental results mentioned in this paper can be found in (Reynolds, 2001).

References

Chapman, D., & Kaelbling, L. P. (1991). Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence* (pp. 726–731). Morgan Kaufmann, San Mateo, CA.

Humphrys, M. (1996). Action selection methods using reinforcement learning. *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior* (pp. 135–144). MIT Press/Bradford Books, MA., USA.

Meuleau, N., & Bourgine, P. (1999). Exploration of multi-state environments: Local measures and back-propagation of uncertainty. *Machine Learning, 35*, 117–154.

Munos, R., & Moore, A. (1999). Variable resolution discretization in optimal control. *Machine Learning.* (to appear).

Peng, J., & Williams, R. J. (1996). Technical note: Incremental Q-learning. *Machine Learning, 22*, 283–290.

Reynolds, S. I. (2000). Decision boundary partitioning: Variable resolution model-free reinforcement learning. *Proceedings of the Seventeenth International Conference on Machine Learning* (pp. 783–790). San Fransisco: Morgan Kaufmann.

Reynolds, S. I. (2001). Optimistic initial Q-values and the max operator. *Proceedings of the UK Workshop on Computational Intelligence, Edinburgh, UK.* (to appear).

Schwartz, A. (1993). A reinforcement learning algorithm for maximising undiscounted rewards. *Proceeding of the Tenth International Conference on Machine Learning* (pp. 298–305). Morgan Kaufmann, San Mateo, CA.

Strens, M. (2000). A bayesian framework for reinforcement learning. *Proceedings of the 17th International Conference on Machine Learning* (pp. 943–950). San Fransisco: Morgan Kaufmann.

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

Watkins, C. (1989). *Learning from delayed rewards.* Doctoral dissertation, King's College, Cambridge, UK.

Wiering, M. (1999). *Explorations in efficient reinforcement learning.* Doctoral dissertation, Universiteit van Amsterdam.

Wyatt, J. (1996). *Exploration and inference in learning from reinforcement.* Doctoral dissertation, Department of Artificial Intelligence, University of Edinburgh, UK.

Wyatt, J. (2001). Exploration control in reinforcement learning using optimistic model selection. *Proceedings of the Eighteenth International Conference on Machine Learning (ICML-2001)*, 593–600.

Experience Stack Reinforcement Learning: An Online Forward λ-Return Method

Stuart I. Reynolds SIR@CS.BHAM.AC.UK
School of Computer Science, The University of Birmingham, Birmingham, B15 2TT, UK

Reinforcement learning (RL) algorithms that employ eligibility traces, such as $TD(\lambda)$, $Q(\lambda)$ and $SARSA(\lambda)$, provide a means to efficiently propagate observed return information to many previously visited states. Thus they usually provide learning speedups compared to their single-step counterparts, $TD(0)$, Q-learning and $SARSA(0)$ (Sutton, 1984; Sutton & Barto, 1998; Watkins, 1989; Peng & Williams, 1996; Rummery, 1995). Recently Fast $Q(\lambda)$ and its derivatives have provided a means of precisely implementing eligibility trace algorithms at a hugely reduced computational cost (Wiering, 1999). However, eligibility traces themselves are an inconvenience introduced only to allow the algorithms to learn online. If offline learning is allowed or the learner's environment is acyclic then far simpler and naturally efficient techniques that directly employ the λ-return estimate can be used (Cichosz, 1997; Lin, 1993).

A brilliantly simple example is backwards replay (Lin, 1993; Sutton & Singh, 1994). This method records experience and replays it offline. Experience is replayed backwards, such that a newly updated value of a state can be used in determining a new estimate for its predecessor. Even if single-step updates are used, a single piece of new reward information can be quickly propagated to many predecessor states. However, backwards replay is not a method which is generally regarded as suitable for learning online (Cichosz, 1997). In many problems performing learning updates online is extremely important in order to aid exploration. Methods which explore the environment uninformed of recently collected rewards can suffer huge losses in performance as a result.

Experience Stack RL is a novel *online* learning technique that combines elements of backwards replay (Lin, 1993; Cichosz, 1997), truncated λ-return estimates (Sutton, 1984; Watkins, 1989; Sutton & Barto, 1998) and operates without the need for eligibility traces. It is a generic method in the sense that analogues can be created of most existing eligibility trace methods. The method records and replays experience as backwards replay but, to allow it to learn online, value updates are made in a lazy fashion (see Figure 1). Experience is replayed only when states are revisited which appear in the experience history. Only at this point can an exploration strategy benefit from updates to the value-function or Q-function. All of the states up to the one revisited are updated in the reverse order to which they were observed. The updates are made using a λ-return estimate and the experiences required to make these updates are removed from memory. Updates to states prior to the one revisited are delayed further with the expectation that the experience yet to come will provide a better value-function or Q-function with which to make these updates.

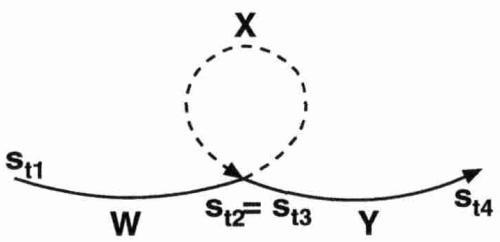


Figure 1. A sequence of experiences where s_{t2} is revisited at $t3$. Sequence X will be updated and removed from memory upon entering s_{t3} to obtain a new value estimate that is immediately useful for exploration. Updates to sequence W are delayed pending the possibly useful experiences in sequence Y. Sequence Y will be updated before sequence W.

The combination of λ-return estimates and backwards replay provides the method with two separate and complimentary mechanisms for propagating return information. With low values of λ the method employs mainly backwards replay. With high values of λ mainly the λ-return mechanism is used. As a result, experiments have shown the performance of the new method to be relatively insensitive to values of λ in most set-

tings compared to related eligibility trace methods.

With low values of λ, the method can be expected to provide significant improvements upon eligibility trace methods in most cases. This follows from noting that, given the same experience, the algorithm makes updates based upon nearly identical return estimates but gains an additional speedup since the return will typically be based on a more up-to-date value or Q-function.

With high values of λ in cyclic online learning tasks, performance may often be worse due to features of the method that allow it to function online. In this case, the backwards replay mechanism will provide little benefit since the stored (up-to-date) state-values or Q-values weigh little in the return estimate. However, in acyclic environments or where offline learning is allowed, the method is expected to improve upon, or do no worse than, (accumulating) trace methods for *all* values of λ in most settings. This follows from the equivalence of forwards and backwards methods in offline and acyclic tasks (Sutton & Barto, 1998) – again, the method essentially learns from the same return estimate but with more up-to-date value estimates.

In addition to this, the method also remains extremely cheap in computation terms (on average it has a cost of $O(|A|)$ per step where $|A|$ is the mean number of actions available at each step). This is as efficient as Fast $Q(\lambda)$ yet the method remains far simpler in its use of the return estimate. It directly implements the forward view of truncated λ-returns (Sutton & Barto, 1998) whereas Fast $Q(\lambda)$ uses this return only indirectly and is conceptually far less tractable as a result. It is expected that convergence proofs of the new online algorithm will be easier to obtain because of this.

An issue which is also becoming increasingly important is the need for efficient off-policy methods (Precup et al., 2000). These are *exploration insensitive* techniques which can learn about the reward obtainable under one policy while experience is generated by another. However, most eligibility trace methods are exploration sensitive and cannot be guaranteed to learn useful value or Q-functions while following arbitrary policies. For control optimisation tasks, an exception is Watkins' $Q(\lambda)$ (Watkins, 1989; Sutton & Barto, 1998). Yet this method is hampered by the need to reset its eligibility trace every time an off-policy (i.e. non-greedy) action is taken. In cases where non-greedy actions are frequently taken, Watkins' $Q(\lambda)$ provides little benefit over the original single-step Q-learning algorithm from which it was derived. The most significant results for the experience stack method have been obtained for an off-policy analogue of Watkins' $Q(\lambda)$. In this case, the backwards replay mechanism counteracts the problems caused by frequently truncating the return estimate. These results and further details of the algorithm are available in (Reynolds, 2001).

References

Cichosz, P. (1997). *Reinforcement learning by truncating temporal differences.* Doctoral dissertation, Warsaw University of Technology.

Lin, L.-J. (1993). *Reinforcement learning for robots using neural networks.* Doctoral dissertation, Carnegie Mellon University, Pittsburgh.

Peng, J., & Williams, R. J. (1996). Technical note: Incremental Q-learning. *Machine Learning, 22,* 283–290.

Precup, D., Sutton, R. S., & Singh, S. (2000). Eligibilty trace methods for off-policy evaluation. *Proceedings of the 17th International Conference of Machine Learning.* Morgan Kaufmann.

Reynolds, S. I. (2001). *Experience stack reinforcement learning* (Technical Report). School of Computer Science, The University of Birmingham, UK. (In preparation).

Rummery, G. A. (1995). *Problem solving with reinforcement learning.* Doctoral dissertation, Department of Engineering, University of Cambridge.

Sutton, R. S. (1984). *Temporal credit assignment in reinforcement learning.* Doctoral dissertation, University of Massachusetts.

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

Sutton, R. S., & Singh, S. P. (1994). On step-size and bias in temporal difference learning. *Proceedings of the Eighth Yale Workshop on Adaptive and Learning Systems* (pp. 91–96).

Watkins, C. (1989). *Learning from delayed rewards.* Doctoral dissertation, King's College, Cambridge, UK.

Wiering, M. (1999). *Explorations in efficient reinforcement learning.* Doctoral dissertation, Universiteit van Amsterdam.

Tuning Vector Parametrized Reinforcement Functions

Juan M. Santos[1,4] JMSANTOS@DC.UBA.AR
Andreas Matt[2] ANDREAS.MATT@UIBK.AC.AT
Claude F. Touzet[3,4] TOUZETC@NEWSUP.UNIV-MRS.FR
[1]Depto.de Computación, FCEN; University of Buenos Aires; Cdad.Universitaria, Pab.I; Buenos Aires, Argentina
[2]Institute of Mathematics, University of Innsbruk, Viktor-Franz-Hess Haus, Technikerstr.25/7, A-6020 Innsbruck, Austria
[3]Laboratoire de Neurobiologie Humaine – UMR 6265 ; University of Provence ; Av. Escadrille Normandie-Niemen ; F – 13397 Marseille Cedex 20, France
[4]CESAR-CSMD; Oak Ridge National Laboratory; P.O. Box 2008, Oak Ridge; TN 37831-635, USA

1. Introduction.

There is an issue that has recently called the attention of some researchers: the reinforcement function (RF) definition. Since the RF has the responsibility of delivering the reinforcements during learning, its definition becomes a critical and delicate task for the Reinforcement Learning (RL) scheme designer. The difficulty in the RF design is that – given an informal expression of a desired behavior – the designer would like to find an easy computable formal function that makes the agent learn the desired behavior as fast as possible. The by-hand-process in order to express the semantic formulation of the desired behavior as a formal mathematical representation of the RF is an intuitive process, which often results in long time experimenting by the RL Designer. In previous works (Santos & Touzet, 1998; Santos 1999) introduced a general expression for RFs (General Constraint Representation) based on parametrized constraints associated separately with positive and negative reinforcements, and additionally a method – the Update Parameters Algorithm (UPA) – for tuning a two parameter RF in such a way that it will be optimal during the exploration phase (Santos & Touzet, 1999).

This paper presents an approach which is a natural extension and generalization of that previous one. On one hand, we will introduce a method that can tune RFs with an arbitrary number of parameters. We have called it the Vectorized Update Parameter Algorithm (VUPA).

On the other hand, we will introduce some definitions and connections of different RF representations. The connections are stated using two lemmas. Their more important consequence is that VUPA can be applied to tune the RF of any Reinforcement Learning scheme in order to facilitate the learning process.

2. Reinforcement Function representations and their equivalencies

Definition (Classic Reinforcement Function): The $RF_{classic} : (s,a) \rightarrow R$ is a function between a situation-action-pair and a real number that represents the associated reinforcement. s is usually a vector of all sensor values perceived by the agent and a a vector of action values.

Definition (Extended Reinforcement Function): The $RF_{extended} : (s_1,a_1,s_2,a_2,...,s_n,a_n) \rightarrow R$ is a function between a $2n$-tuple and a real number that represents the associated reinforcement. n is a natural number lower or equal to the total number of performed actions.

Since the situation vector does not match usually with the state vector (partially observed environments), the dimension of s can be extended using estimators of the unknown state variables (Watkins, 1989).

Definition (Restricted Reinforcement Function): The $RF_{restricted} : (s_1,a_1,s_2,a_2,...,s_n,a_n) \rightarrow \{-1,0,1\}$ is a function between a $2n$-tuple and an element of the set $\{-1,0,1\}$ that represents the associated reinforcement.

Definition (General Constraint Representation): The $RF_{constraint} : (s_1,a_1,s_2,a_2,...,s_n,a_n) \rightarrow \{-1,0,1\}$ has the same domain and image as the Restricted Reinforcement Function. The formal expression is

$$RF_{constraint}(s_1,a_1,s_2,a_2,...,s_n,a_n) = \begin{cases} prior(c_+,c_-) & if \quad (c_+ \quad or \quad c_-) \quad is \quad true \\ 0 & otherwise \end{cases}$$

where c_+ and c_- are constraint lists combined with logic operators: $c_{+/-} = [not]e_1 \quad and/or \quad ... \quad [not]e_{cn}$. The expressions $e_1,...,e_{cn}$ are of the form $e_i := (g_k \quad op \quad \theta_k)$, cn is the cardinal of constraints, $g_k : (s_1,a_1,s_2,a_2,...,s_n,a_n) \rightarrow V$ is an arbitrary function, $op \in \{<,>,\leq,\geq,=\}$, V usually the real space $\Re$, and θ_k a parameter with bounded values in $\mathrm{Im}(g_k) = V$. $prior : B \times B \rightarrow \{-1,+1\}$, B is the set of Boolean values (true or false). $prior$ is a predicate which evaluates the priorities between c_+ and c_-. In general, we could define *prior* according to:

$$prior(c_+,c_-) = \begin{cases} +1 & if \quad c_+ \quad and \quad \neg c_- \\ -1 & if \quad \neg c_+ \quad and \quad c_- \\ pv & if \quad c_+ \quad and \quad c_- \end{cases},$$

where pv is +1 or −1 according to the RF designer criterion. Usually negative reinforcements are preferred (a "good and bad" action is a "bad" one).

Lemma 1: For every behavior, there exists at least one restricted RF so the agent learns the same behavior as with a given classic RF.

Lemma 2: There is at least one General Constraint RF for a $RF_{restricted}$ (and due to Lemma 1 also for a $RF_{classic}$).

Proofs are omitted here due to size restriction. The first one relates the classic RF representation with the Restricted RF representation. That is, any RF which is an instance of the Classic RF representation can be expressed as an instance of the Restricted RF representation. The second lemma relates the Restricted RF representation with our proposed General Constraint Representation. That is, any RF which is an instance of the Restricted RF representation can be expressed as an instance of the General Constraint one. Therefore, there exists an equivalence between RFs expressed in Classical way with the RFs expressed by means of our proposed representation.

3. Vectorized Update Parameter Algorithm.

We consider two performance estimators: one for the proportion of rewards received by the agent and another for the proportion of penalties received by the agent, both, over time. The basic idea behind VUPA is the monotonic relation between these estimators and the parameter values of the RF. Thus, VUPA can adjust incrementally the RF parameters of the system with the object of leading the performance estimators to some desired values (one, close to 1, for rewards and close to 0, for penalties). Since our main application is behavior synthesis in robotics, such estimators must be observed and we have to obtain a measure of them experimentally during learning. These values depend on the received sensor values and therefore the convergence of the method will be influenced by stochastic components. Following this fact we consider the use of the main concepts of stochastic methods for function approximation, in particular, the Robbins–Monro Procedure which uses a monte–carlo method for non–linear function minimization (Fukunaga, 1990; Kushner & Clark, 1978). We have taken advantage of this fact to prove the convergence of VUPA when the time (iterations) tends to infinity.

Let θ be the RF parameter vector (i.e. θ^+ for rewards and θ^- for penalties), and $f(\theta)$ the monotonic function between the performance estimator and the parameter vector θ, which is possible to sample for arbitrary values of θ with a bounded random error and whose statistical expectation is the null value.

Definition: The Vector Update Parameter Algorithm (VUPA) is defined by the stochastic function approximation Robbins–Monro Procedure in vector form for the function $(f-p)$ and the single components of θ.

That is, $^{n+1}\theta_i^+ = {}^n\theta_i^+ - \lambda_n\left(f\left({}^n\theta^+\right) - p^+\right)$, with $i = 1...l$ is the dimension of , and where $^n\theta^+ = \left({}^n\theta_1^+, {}^n\theta_2^+, ..., {}^n\theta_l^+\right)$ is the parameter approximation corresponding to the step n of the Robbins–Monro Procedure and

$$\#r_{\Delta t}^+ = \sum_{k=t}^{t-\Delta t} \delta\left(1, RF\left|{}_{n\theta^+, {}^n\theta^-}\left(s^t, a^t\right)\right.\right)$$

is the number of rewards obtained during the last Δt steps of the pure random exploration assuming that the RF is parameterized at $\left[{}^n\theta^+, {}^n\theta^-\right]$. λ_n is a coefficient that decay over time so that the convergence can be guaranteed. The $\delta()$ is the Kronecker function defined as $\delta(a,b) = \begin{cases} 1 & if\ a = b \\ 0 & else \end{cases}$.

We note $RF\left|{}_{n\theta^+, {}^n\theta^-}\left(s^t, a^t\right)\right.$ to express that the values of $^n\theta^+$ and $^n\theta^-$ will remain unchanged during the computation of the RF for all value of t.

Applying VUPA, $^n\theta$ converge w.p.1 to the value $^*\theta$ (root of $(f-p)$) for $n \to \infty$ under the following conditions: $\sum_{n=0}^{\infty}\lambda_n = \infty$, $\sum_{n=0}^{\infty}\lambda_n{}^2 < \infty$ and $Lim_{n\to\infty}\lambda_n = 0$.

4. Experiments

We have conducted two series of experiments with the miniature robot Khepera using Q–learning to synthesize the *obstacle avoidance* and *wall following* behaviors. In both cases, we have checked the practical consequences of VUPA: the tuning of RF parameters improves the Q–Learning convergence toward the desired behaviors.

References

Fukunaga, K. (1990). Statistical Pattern Recognition. (2nd ed.). Academic Press.

Kushner, H.J. & Clark, D.S. (1978). Stochastic Approximation Methods for Constrained and Unconstrained Systems, Springer–Verlag New York Heidelberg Berlin.

Santos, J.M. & Touzet, C. (1998). Automatic Tuning of the Reinforcement function. Proceedings of the Fourth International Conference on Neural Networks and their Applications, (NEURAP'98), 103–110, Marseille.

Santos, J.M. (1999). Contribution to the study and the design of reinforcement functions. Ph.D. thesis. Universidad de Buenos Aires and Universite d'Aix Marseille–III.

Santos, J.M. & Touzet, C. (1999). Exploration Tuned Reinforcement Function. Neurocomputing, 28, 93–105.

Watkins, C.J.C.H (1989). *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK.

Using Multi-step Actions for Faster Reinforcement Learning

Ralf Schoknecht SCHOKN@IRA.UKA.DE
Martin Riedmiller RIEDML@IRA.UKA.DE
Institute of Logic, Complexity and Deduction Systems, University of Karlsruhe, 76128 Karlsruhe, Germany

Introduction Standard Reinforcement Learning algorithms, like trajectory based *Q-learning*, scale very badly with increasing problem size. Among others, one intuitive reason for this is that according to the problem size the number of decisions from the start state to the goal state increases. In order to keep the number of decisions to the goal tractable hierarchical approaches based on temporal abstraction have been proposed. The main contributions are the *Option* approach, the *Hierarchy of Abstract Machines* (HAM) (Parr, 1998) and the *MAXQ* approach (Dietterich, 2000). They are all based on the notion that the whole task is decomposed into subtasks each of which corresponds to a subgoal. The existing hierarchical RL approaches are able to solve problems of the following two types:

1. **Abstract actions given**: Abstract actions for achieving subgoals are given in terms of actions that are lower in the hierarchy.
2. **Subgoals given**: The concrete realization of abstract actions in terms of subordinate actions is not known but a decomposition of the whole task in subtasks is given.

The minimal requirement for the application of existing hierarchical RL algorithms is that a decomposition of the whole problem into subproblems is known. Thus, problems from technical process control, e.g. cart-pole balancer, mountain car or acrobot, as well as general navigation tasks often cannot profit from those approaches because in such 'unstructured' domains subgoals are not known in advance or do not exist at all. In case no subgoals are given, efficient RL algorithms are not known to date.

Even if no subgoals are available many control problems show the following characteristic. When controlling the cart-pole balancer, for example, on the one hand it is necessary to use a temporal resolution that is fine enough to provide the needed reactivity when a switch of action is required. On the other hand between those action switches the same action will be applied for several consecutive time steps. Hence, the temporal resolution could be coarser which would result in less decisions to the goal. This dilemma cannot be resolved by existing RL approaches. In this paper we propose a new hierarchical approach to RL that is suited for such problems where no decomposition in subproblems is known in advance.

Multi-step Actions in Reinforcement Learning In the following we consider a discrete time RL problem with a primitive time step Δt. Primitive actions last exactly one such primitive time step. The set of primitive actions is denoted as $\mathcal{A}^{(1)}$. We define the set of all *multi-step actions* (MSAs) of *degree* n as $\mathcal{A}^{(n)} = \{a^n | a \in \mathcal{A}^{(1)}\}$ where a^n denotes the MSA that arises if action a is executed in n consecutive time steps. The next decision is only made after the whole MSA has been executed. Thus, the MSA has a time-dependent termination condition after n primitive time steps. In the general option framework defined in (Sutton et al., 1999) MSAs can therefore be modelled as special semi-Markov options. In order to retain the possibility of learning optimal policies different time scales are combined in *one* action set. We denote such *heterogeneous* action sets as $\mathcal{A}^{(n_1,\ldots,n_k)} = \mathcal{A}^{(n_1)} \cup \ldots \cup \mathcal{A}^{(n_k)}$.

MSA-Q-Learning In the following we demonstrate how the concept of MSAs can be integrated in learning algorithms like Q-learning. The agent maintains Q-functions for all time scales it is acting on. When executing action a^j of degree j in a state s the agent goes to state s' and updates the corresponding Q-value according to $Q^{k+1}(s,a^j) = (1-\alpha)Q^k(s,a^j) + \alpha[r(s,a^j) + \gamma^j \max_{a' \in \mathcal{A}} Q^k(s',a')]$ where j denotes the number of time steps elapsed between s and s'. The reward accumulated on the primitive time scale is denoted by $r(s,a^j) = \sum_{\tau=0}^{j-1} \gamma^\tau r(s_\tau, a), s_0 = s$. In order to use the experience from the transitions more efficiently we apply *intra-MSA* methods (Schoknecht, 2001). This enables to extract updates for MSAs on lower level time scales from a high level transition.

Results We illustrate the behaviour of MSA-Q-learning using the mountain car benchmark. The state space is bounded by position $|p| \leq 1.0$ and velocity $|v| \leq 3.0$. If the car exceeds these bounds the trajectory is aborted and a terminal reinforcement of -15.0 is received. The immediate reward is 5.0 within the goal region, $0.5 \leq p \leq 0.7$, and -0.1 outside.

In this benchmark domain there are obviously no known subgoals that the agent should try to reach on its way to the goal region. Approaches that are based on a task decomposition are therefore not applicable here. On the other hand an optimal policy will only involve about two action switches until the goal region is reached. This is a typical situation where MSAs are suited. The number of decisions to the goal region can be reduced by selecting larger time scale actions but the necessary reactivity is retained because lower time scale actions are also available. In the following we investigate how different heterogenous action sets influence the speed of MSA-Q-learning in the mountain car domain. As the state space is continuous, function approximation has to be used for the Q-function. We choose a grid based approach where the state space is divided into simplices by the Kuhn triangulation. In a trajectory based learning approach the origin of a transition $(s, a^j) \rightarrow s'$ does not necessarily lie on a grid point. The Kaczmarz update rule (Pareigis, *1998*) is suitable to adapt the Q-function in this situation. Initially all Q-values are set to zero. The discount factor γ is set to 0.9. Experimentally, $\alpha = 0.3$ was determined to be a good value for the learning rate in the mountain car domain[1]. In order to provide enough exploration actions are selected according to an ϵ-greedy method with $\epsilon = 0.1$. The following experiment is carried out. All trajectories start on the bottom of the valley in state $(-0.5, 0)$. This is a 'difficult' starting state because the thrust is not sufficient to reach the goal directly by driving up the hill. Instead, the car has to gain energy by moving up the opposing hill. After each step MSA-Q-learning is performed to update the Q-function. If no constraint violation occurs the trajectories are aborted after 80 steps. A test trajectory is carried out every 10 training trajectories. The performance of the learned policy is measured as the accumulated discounted reward per test trajectory plotted against the accumulated number of primitive training time steps.

Table 1 shows quantitative results for the learning speed with different action sets. Using an optimal policy the agent reaches the goal region after 29 steps and does not leave it any more. Thus, the optimal reward that can be achieved in 80 steps is about 1.392. A policy that needs one step more to reach the goal region has a reward of about 1.152. In order to determine the speed of learning we measure after how many training time steps a policy of this quality is permanently achieved on an average learning curve. It can be seen that if the degree of the actions in a homogeneous action set becomes too large ($\mathcal{A}^{(4)}$) optimal policies cannot be learned any more. On the other hand, heterogeneous action sets with MSAs lead to a considerable speed-up of learning. With $\mathcal{A}^{(1,8)}$, for example, the number of training time steps to reach the 1.152 level is reduced by about 84% compared to $\mathcal{A}^{(1)}$.

[1] Although the mountain car is a deterministic domain $\alpha < 1$ is required because the effects of transitions from different states in the same cell have to be averaged.

action set	'good' policy after step	learning speed relative to $\mathcal{A}^{(1)}$	optimal policy
$\mathcal{A}^{(1)}$	212759	100%	yes
$\mathcal{A}^{(2)}$	99193.3	46.6%	yes
$\mathcal{A}^{(4)}$	-	-	no
$\mathcal{A}^{(1,2)}$	78877.5	37.1%	yes
$\mathcal{A}^{(1,4)}$	51821.8	24.4%	yes
$\mathcal{A}^{(1,8)}$	33734.5	15.9%	yes

Table 1. Learning with different action sets. The second column shows after how many primitive training time steps a 'good' policy of quality 1.152 is permanently achieved on an average learning curve obtained from 100 independent learning runs. In the third column the learning speed relative to $\mathcal{A}^{(1)}$ is depicted. The fourth column indicates if an optimal policy is possible with the chosen action set.

Conclusions We extended Q-learning by integrating the concept of multi-step actions (MSAs) together with the intra-MSA method. The new MSA-Q-learning algorithm efficiently uses training experience from different time scales. In the mountain car benchmark domain learning could be considerably accelerated by using MSA-Q-learning together with action sets that combine primitive actions and MSAs. The success of the MSAs is due an implicit reduction of problem size, which enables the agent to reach the goal with less decisions.

References

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research, 13*, 227–303.

Pareigis, S. (1998). Adaptive choice of grid and time in reinforcement learning. *Advances in Neural Information Processing Systems.* The MIT Press.

Parr, R. E. (1998). *Hierarchical control and learning for markov decision processes.* Doctoral dissertation, University of California, Berkeley, CA.

Schoknecht, R. (2001). Hierarchical reinforcement learning with multi-step actions. *Workshop on Hierarchy and Memory in Reinforcement Learning, Eighteenth International Conference on Machine Learning.*

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

Reinforcement Learning and the Perception of Time Intervals

Jonathan Shapiro JLS@CS.MAN.AC.UK
Department of Computer Science, Manchester University, M13 9PL U.K.

Abstract

One component of the delayed-reward reinforcement learning problem is learning the expected time to the reward. Animal experiments on delayed-reward timing tasks show that many animals can learn this, but they do it in a characteristic way. They exhibit the scalar property, which states that variability in their responses increases in proportion to the length of the interval being measured. The data also shows a more striking property — results from different time intervals collapse into a single curve when the data is scaled relative to the time interval. A simple model of a network accumulator consisting of noisy, linear, spiking neurons can produce the scalar property. When coupled to TD(λ) reinforcement learning algorithm, the model simulates experimental results from animal experiments.

1. Motivation

An aspect of delayed-reward reinforcement learning which has a long history of study in animal-learning experiments, but has been overlooked by learning theorists, is the learning of the expected time to the reward. In a number of animal experiments there are benefits to the animal to learn the time to the reward. In one example, an animal needs to wait a given time interval after a stimulus before performing an action in order to receive a reward. In another example, an animal receives rewards for a particular action at a given rate. After a random interval, the rewards stop and the animal has to change actions. Here the animal needs to learn the inter-reward interval in order to change actions with minimum delay.

There are two reasons why it might be desirable for a learner to learn the expected time to receive a reward. First, it allows it to perform the action for an appropriate length of time. Second, it allows it to compare the rate of rewards between two different actions. An example studied experimentally by numerous researchers is foraging (for a review, see (Kacelnik & Bateson, 1996)). An animal which had no sense of the typical time to receive food could remain too long at a depleted foraging site and starve; conversely, it could fail to find food by leaving too frequently. In addition, the expected time to rewards for two foraging patches allows them to be compared. The ability to learn reward times is important in more general applications for similar reasons. It allows the system to recognise when the current strategy has failed and a new strategy should be found; it allows comparison between two strategies in a form which is easily amenable to decision-making.

Of course, in order to perform this task, the animal needs an internal clock or mechanism of perceiving time intervals, as well as a learning system which can tackle this as well as more familiar aspects of the delayed-reward reinforcement learning problem. The experiments show that a wide range of animals can learn the expected time to rewards, but they do so in a characteristic way. The data, when suitable scaled, collapses onto universal curves. This is called the scalar property of interval time. This property is found in a wide range of experiments and species, so it may reveal something fundamental about animal clocks, or some desirable property of solutions to this learning problem.

2. The Scalar Property of Interval Timing

In the animal experiments, what is typically found is the following. First, animals can do these tasks. They learn to respond and tend to give highest response is at a time proportional to the waiting time interval, usually with proportionality factor very close to one. However, there is variability in the response times; the animal does not respond at exactly the correct time even when the reinforcement schedules are deterministic. In addition, the variability increases in proportion to the measured interval. I.e. the ratio of the standard deviation in the response time to the mean response

time is constant independent of the length of the time interval. This is one form of the *scalar property* (the weak form).

A more striking form of the scalar property (the strong form) states that when the average response rate is multiplied by the time interval and plotted against the time divided by the time interval, the data from different time intervals collapses into one curve. This is also called super-imposition. This states that not only is the ratio of the standard deviation to the mean response rates invariant, but *all* scale-independent shape measures, including the skewness, the kurtosis, and higher order ones, are invariant. In other words, the response curves have an invariant shape, when scaled in terms of relative time. For a review of interval timing phenomena, see (Gibbon & Church, 1990).

The strong scalar property can be stated mathematically as follows. Let T be the actual time elapsing, and $\tilde{T}$ be the subjective time perceived by the animal. The experiments show that $\tilde{T}$ varies for given T. The scalar property, in particular super-imposition, implies that the conditional probability of $\tilde{T}$ given T depends upon T in a special way,

$$P(\tilde{T}|T) \approx \frac{1}{T} P_{\text{inv}} \left(\frac{\tilde{T}}{T} \right).$$

Here P_{inv} is the function which describes the shape of the scaled curves. One question which remains unanswered is: what is the origin of this property? It is found in many species, including rats, pigeons, turtles; humans will show similar results if the time intervals are short or if they are prevented from counting through distracting tasks. It is found in many types experiments. It is also hard to understand, since any model build around an accumulation of independent errors, such as a clock with a variable pulse rate will not have the scalar property. It has been very difficult to find a mechanism which gives rise to this.

We can show that a simple model of a neural clock can give rise to the scalar property (Shapiro et al., 2001). Time is measured by an accumulator consisting of noisy, linear, spiking neurons. The total activity of the neural network measures the length of time passed. When the rate of growth of activity is set to be proportional to the time passed, the network activity as a function of time obeys the strong scalar property.

We also show, analytically and via simulations, that when the output of the network is used as a stimulus in a TD(λ) learning system such as those used to model Pavlovian eye-blink experiments and other examples of classical conditioning (Moore et al., 1990; Sutton & Barto, 1990), the scalar property is preserved, and most prominent features of the animal experiments is reproduced.

References

Gibbon, J., & Church, R. M. (1990). Representation of time. *Cognition, 37*, 23–54.

Kacelnik, A., & Bateson, M. (1996). Risky theories - the effects of variance on foraging decisions. *American Zoologist.*

Moore, J. W., Berthier, N. D., & Blazis, D. E. J. (1990). Classical eye-blink conditioning: Brain systems and implementation of a computational model. In M. Gabriel and J. Moore (Eds.), *Learning and computational neuroscience: Foundations of adaptive networks*, A Bradford Book, 359–387. The MIT Press.

Shapiro, J., Wearden, J., & Barrone, R. (2001). A simple model exhibiting scalar timing. *Connectionist Models of Learning Development and Evolution.* Springer-Verlag.

Sutton, R. S., & Barto, A. G. (1990). Time-derivative models of pavlovian reinforcement. In M. Gabriel and J. Moore (Eds.), *Learning and computational neuroscience: Foundations of adaptive networks*, A Bradford Book, 497–537. The MIT Press.

Policy Search using a State–Policy Evaluation Function

Malcolm J A Strens MJSTRENS@QINETIQ.COM

CRMV, Room 1032, X107 Building, QinetiQ, Cody Technology Park, Farnborough, Hants., GU14 0LX, U.K.

1. State–Policy Evaluation Function

Policy search for reinforcement learning in partially observable environments requires optimisation of a stochastic function (the return from a trial) in a high–dimensional space (the space of policy parameter vectors). Therefore it is important to find an efficient method which minimises learning time and exploits the inherent structure in the learning problem. Using paired comparisons of policies can accelerate the search in episodic problems: each policy is evaluated using the same set of start states to reduce variance in the outcome (Strens & Moore, 2001). Furthermore, if learning with a simulation, the same random number sequence can also be used for each policy (Ng & Jordan, 2000) to eliminate variance in returns, at the cost of some bias.

The paired comparison methods are taking advantage of the fact that performance of a policy in any given trial is a function of both policy parameters *and* (start) state. To generalise this idea I propose using a function approximator to estimate a state–policy evaluation function (SPEF), in order to aid policy search. The SPEF models the expected return for combinations of (hidden) states and policy parameters. This function captures all relevant information about the relationship between the state, the policy parameters, and the expected return. (The method is most applicable to problems with compact, multi–dimensional, continuous–valued, state and policy spaces.) The SPEF can be defined over the space of allowable start states or over the whole state space (assumed here). In an episodic task, the SPEF can model expected return per episode or average return per time step. In continuing tasks expected discounted return is used and the SPEF must be defined over the whole state space.

The SPEF satisfies the Markov property if it is defined over the full state space of the underlying system. (This property implies that that value of a policy in a particular state does not depend on how that state was reached.) Defining the SPEF over the full state space is possible if (i) the system is fully observable; or (ii) learning is with a simulation for which the hidden state is made visible to the learner. If learning is through real–world experience rather than simulation, the function approximator can only use the partially observable state. In this case, the best policy for the system will not necessarily be found by using the SPEF method. (In episodic tasks, the start state may be fully observable even if subsequent states are not; hence the SPEF can be defined over start–states only.)

2. Predicting a policy's performance

Suppose that a state–policy evaluation function can be estimated during learning. By integrating this function with respect to some state occupancy distribution (e.g. the on–line distribution), we obtain a policy evaluation function. (If modelling expected return per episode, empirical state occupancy must be weighted down by trial duration to obtain an unbiased estimate.) At any time during learning, minimising this policy evaluation function with respect to the policy parameters yields an estimate of the best (exploitative) policy, and can help to choose the next experiment in exploration of the policy space. Therefore the estimated policy evaluation function becomes a substitute for the true (noisy) function in the search procedure. The main advantage of this is that it is unnecessary to perform more than one trial for each candidate policy because a smooth interpolating surface is available. This use of a substitute function has many characteristics in common with the STAGE algorithm for optimisation (Boyan & Moore, 2000).

Additionally, if we are able to represent uncertainty in the parameters of the policy evaluation function, then samples of size 1 (hypotheses), rather than maximum likelihood estimates, can be used in the above process. By integrating each hypothesised SPEF and minimising the resulting policy evaluation function, exploratory rather than exploitative experiments are obtained automatically. Strens (2000) showed that this 'exploring by hypothesis' is an effective strategy.

When integrating the state–policy evaluation function to obtain a policy evaluation function, the choice of state occupancy distribution is important. In some problems, it will be sufficient to use the empirical on–line state occupancy obtained during learning; a decay method may be used to give recent experience more weighting. An alternative is to use the empirical distribution obtained only during trials with similar policies to the one of interest (the "query"). This yields a localised policy evaluation function, and provides a means to guarantee convergence.

4. Approximating the SPEF during learning

A benefit in learning performance can be obtained only if a function approximator with useful generalisation performance across policy parameters *and* state can be found to represent the SPEF. Therefore the function

approximator should be chosen to exploit local smoothness and global patterns in the SPEF. In particular, we might expect one or more policy parameters to be irrelevant to the expected discounted return in large regions of the state–space; generalisation can take place over those policy parameters within that region.

For example, suppose during learning in a continuing task that a single region of state space is visited several times, and the policies being followed in each case differ only in (locally) irrelevant policy parameters. Then the noisy discounted returns from each of these occurrences can be averaged by the function approximator to obtain a lower variance prediction.

It is important to use an appropriate representation for the SPEF, and we are only starting to explore the possibilities. To guarantee convergence, local representations are preferred (e.g. locally weighted regression). This ensures that the complexity of the estimated SPEF can increase during learning, as new information is gathered, and that representation will be more accurate in the most interesting (highest return) regions of the policy space. Local representations also have some drawbacks: (i) all trajectories must be stored in memory; (ii) integrating the SPEF over a state distribution has a high computational cost because it involves many queries; (iii) consistent sampling is difficult; (iv) global generalisation capability may be lost.

Using global representations (e.g. polynomials over the joint state–policy space) would solve these drawbacks but the guarantee of convergence is lost, and the control of model complexity is difficult. The choice of representation must also take into account the curse of dimensionality: the SPEF is defined over the joint space of states and policy parameters, and so may be high dimensional.

5. Initial experiments

The method has been applied to a simple test problem (the mountain car) using global linear (or quadratic) function approximation to represent the SPEF. The differential evolution method (Storn & Price, 1995) was used to search the policy space. The population (of policy parameter vectors) could be retained from one trial to the next because the policy evaluation function changes slowly. When successful, good policies were found very rapidly. However, in some configurations only a locally optimal policy was found, providing evidence that local representations are required; these could be introduced progressively during learning.

6. Conclusion & future work

I have described an approach to policy search for reinforcement learning that makes use of much of the structure in the learning problem, including the Markov property, by estimating a state–policy evaluation function during learning. The method has the potential to greatly reduce the number of trials required to find a good policy in difficult control tasks, and is likely to be particularly effective when learning with a simulation. The main outstanding issues before the method can be applied to difficult tasks are:

(i) Choice of appropriate function approximator for the SPEF.

(ii) Type of state occupancy distribution used in estimating the value of a policy.

(iii) Choice of method for minimising the policy evaluation function.

Acknowledgements

This work was part–funded by the UK Ministry of Defence Corporate Research Programme (Technology Group 3).

References

Boyan, J. A., & Moore A. W. (2000). Learning evaluation functions to improve optimization by local search. *Journal of Machine Learning Research 1:77–112.*.

Ng A., & Jordan M. (2000). PEGASUS: A policy search method for large MDPs and POMDPs. *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence.*

Storn, R., & Price, K. (1995). *Differential evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces* (Technical Report TR–95–012). International Computer Science Institute, Berkeley, California.

Strens, M. J. A. (2000). A Bayesian framework for reinforcement learning. *Proceedings of the Seventeenth International Conference on Machine Learning.* San Francisco: Morgan Kaufmann.

Strens, M. J. A., & Moore, A. W. (2001). Direct policy search using paired statistical tests. *Proceedings of the Eighteenth International Conference on Machine Learning.* San Francisco: Morgan Kaufmann.

Learning Fair Periodical Policies

Katja Verbeeck KAVERBEE@VUB.AC.BE
Ann Nowé ASNOWE@INFO.VUB.AC.BE
Johan Parent JOHAN@INFO.VUB.AC.BE
Computational Modeling Lab (COMO), Vrije Universiteit Brussel, Belgium, http://como.vub.ac.be

Abstract

Periodical policies were recently introduced as a solution for the coordination problem in games which assume competition between the players, but where the overall performance can only be as good as the performance of the poorest player. Instead of converging to just one Nash equilibria, which may favor just one of the players, a periodical policy switches between periods in which all interesting Nash equilibria are played. As a result the players are able to equalize their pay-offs and thus a fair solution is build. Moreover social players can learn this policy with a minimum of communication.

1. Periodical Policies

This work studies the fairness of strategies learned by agents in dynamical multi-agent systems, in which agents experience conflicting objectives, but where the overall performance can only be as good as that from the poorest performing agent. The kind of applications we have in mind is for example autonomous task allocation or job scheduling. Clearly, agents have to collaborate to distribute the available resources among them. However since communication has its price in distributed systems, our goal is to limit the communication required, yet allowing coordination so that pay-offs are equalized and the global goal is achieved.

For learning in a multi-agent system, two extreme approaches can be recognized. On the one hand, the presence of other agents can be completely ignored. On the other hand, the presence of other agents can be modeled explicitly. The first approach may lead to oscillating policies, while in the second approach each agent learns in a joint action space which may become very large.

An added difficulty is that it is not always clear what agents should learn. When the agents' interactions are modeled in a game, this game may posses different Nash equilibria, which may be fair to some, but uninteresting for other players. So how should agents behave in these situations? Playing pure Nash equilibria is often unfair to at least one of the players, while playing a mixed strategy doesn't give any guarantee for coordination and usually results in a sub-optimal payoff for all agents.

We therefore introduce the notion of periodical policies, see (Nowé et al., 2001). In a periodical policy players alternate between playing different Nash equilibria if these equilibria are optimal for some players but suboptimal for others. In a fair periodical policy each player gets the chance of playing his best Nash equilibrium and these equilibria should be alternated such that the average payoff of all players is equalized, even when they don't give the same payoff to the players.

2. Homo Egualis Reinforcement Learners

We show how a periodical policy can be learned by social reinforcement learners, which are inspired on the Homo egualis society from sociology. In many decision-making and strategy-settings people do not behave like the self-interested "rational" actor depicted in neoclassical economics and classical game theory (Gintis, 2000). In a Homo egualis system, agents do not only care about their own payoff, but also about how it compares to the payoff of others. A Homo egualis agent may be willing to reduce his own payoff to increase the degree of equality in the group. On the other hand he is also displeased when receiving a lower payoff than the other group members. As a result altruists appear in ultimatum and public good games, (Gintis, 2000).

We used this idea of inequality aversion to develop an algorithm, which is able to let a group of distributed agents learn a fair periodical policy, with a minimum

of communication. In our algorithm agents are selfish utility optimizing reinforcement learners who will converge to some Nash equilibrium of the game. This is assured by results from Learning Automata theory[1] (Narendra & Thathachar, 1989). After convergence to one of the Nash equilibria, a communication phase takes place only to compare performances, see Figure 1. Based on Homo equalis arguments agents reduce their action space to go find an alternative Nash equilibria in another subspace and which will favor another agent. The algorithm proposed is able to learn the length of the period in which an agent must play his best Nash equilibrium in order to equalize the average payoff. We use Q-learning (Sutton & Barto, 1998) to let the agents converge to one of the equilibrium points, but after communication agents use the communication results instead of their Q-values to decide what do to, reduce the action set, re-install it or do nothing. Interesting to note is that while agents are playing the periodical policy, the mixed strategy policy is learned off-line.

```
## COMMUNICATION PHASE
if (time_to_communicate) {
communicate_to_all (cumulative_payoff , last_payoff );
payoff_all := receive_from_all (cumulative_payoff );
last_period_payoff_all :=
  receive_from_all(last_period_payoff );
if (not_equal_payoffs (payoff_all ) ) {
  if (and (has_best_payoff (payoff_all ))
       (has_best_payoff (last_period_payoff_all)))
    actionSet := actionSet - best_action;
  else
    if ( not (has_best_payoff (payoff_all )))
      actionSet := original_actionSet;

   INITIALIZATION ;
  }
}
```

Figure 1. Pseudo code of the communication part of the algorithm executed by each Homo egualis Q-learner.

3. Results

The algorithm has been applied on multi-agent games with conflicting equilibrium points. Synchronous action taking with immediate feedback, as well as asynchronous action taking with delayed feedback are tested. For the latter job scheduling games of the common pool resource type as in Figure 2 were simulated. The agents where able to learn a periodical policy such that their overall payoff was equalized optimally, see Figure 3.

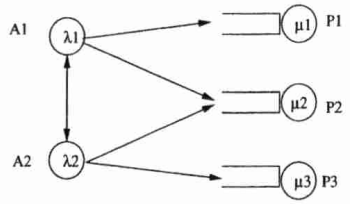


Figure 2. A simple job scheduling game. Two agents can use either their private processor or a common processor. The agents generate jobs according to an exponential law with a mean value of 1.5 time units. Both the private processors handle the jobs with a time consumption chosen from an exponential distribution with mean value of 1 time units, while the common processor works with a mean of 0.5 time units.

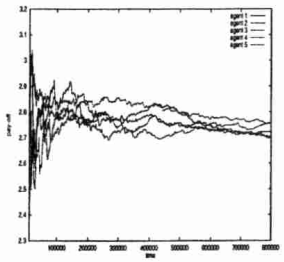


Figure 3. Average job turnaround time for the job scheduling game with 5 players. Each player can choose between a private or common processor. Load and processor consumption time are the same as in Figure 2

References

Gintis, H. (2000). *Game theory evolving: A problem-centered introduction to modeling strategic behavior.* Princeton, New Jersey: Princeton University Press.

Narendra, K., & Thathachar, M. (1989). *Learning automata: An introduction.* Prentice-Hall International, Inc.

Nowé, A., Parent, J., & Verbeeck, K. (2001). Social agents playing a periodical policy. *Proceedings of the 12th European Conference on Machine Learning.* Freiburg, Germany: Springer-Verlag LNAI2168.

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

[1] Learning Automata theory assures that players in an N-person nonzero-sum game who use a reward-inaction update scheme with an arbitrarily small step size will always converge to one of the equilibrium points. Which one depends on the initial conditions.

Advances in Exploration control in Reinforcement Learning

Jeremy L. Wyatt JLW@CS.BHAM.AC.UK
Funlade Sunmola FTS@CS.BHAM.AC.UK
School of Computer Science, University of Birmingham, Birmingham, B15 2TT, United Kingdom

Abstract

In this paper we present three ways of solving the problem of exploration control for reinforcement learning (RL). In optimistic model selection (OMS) from a density over possible models. We give results for the basic algorithm, and describe extensions to the case of optimistic multi-time models; and to dynamic bayes networks. The second method is based on directly taking Monte Carlo samples from the space of possible models. The third method is based on taking Monte Carlo samples by running the MDP and taking trajectories through the information space. We will sketch all algorithms, give an indication of the trade-offs, and present new results on the extensions to OMS and the latter two algorithms. We will relate all these methods to the Bayesian formulation of the exploration-exploitation trade-off.

1. Introduction

The problem of how to act while learning is a class of optimal control problems with a long history (Bellman, 1961). In RL it has taken the form of problems of (i) how to act so as to maximise performance during the learning agent's lifetime; and (ii) how to act to identify as good a policy as possible within the learning period. In Markov decision processes (MDPs) the optimal Bayesian solution to problem (i) is well known, but intractable (Martin, 1967; Bellman, 1961). Many approximations have been proposed. The domain is a finite state MDP with an unknown transition function and a known reward function. All the methods considered here are model-based.

2. Background

The Bayesian approach is based on there being a space $\mathcal{P}$ of possible transition functions (or models) P for the MDP, and a well-defined prior probability density over that space. The probability density over the space of possible finite state MDPs for a known state space S is a density $f(P|M)$. This is simply a product of Dirichlet densities, one for each state action pair in the MDP. In a Bayesian framework we choose a prior matrix M', which specifies our prior density over the space of possible models. The value function in an MDP with unknown transition probabilities is thus itself a random variable, $\tilde{V}_i$. Given the usual squared error loss function the Bayesian estimator of expected return under the optimal policy is the expectation of $\tilde{V}_i$:

$$V_i(M) = \mathrm{E}[\tilde{V}_i|M] = \int_{\mathcal{P}} V_i(P) f(P|M) dP \qquad (1)$$

where $V_i(P)$ is the value of i given the transition function P. The central result of both Bellman and Martin was that when this integral is evaluated we transform our problem into one of solving an MDP with known transition probabilities, defined on the information space $\mathcal{M} \times \mathcal{S}$. The optimal solution to the well-known exploration-exploitation trade-off (problem (i) above) is thus to act greedily with respect to the Bayes Q-values. Because the solution involves dynamic programming over a tree of information states the problem is intractable.

3. Optimistic Model Selection

We integrate the idea of Model Based Interval Estimation (Wiering & Schmidhuber, 1998) with the Bayesian view of exploration by selecting an optimistic model P_{opt} from $\mathcal{P}$ using probability intervals calculated based on $f(P|M)$. Since the Dirichlet is the natural conjugate density for sampling from a multinomial distribution, this allows us to correctly incorporate prior knowledge about the transition function, and to explore using OMS from the outset. One problem is how to choose the prior matrix M in the case where we have very little knowledge about P. We create a single additional terminal state k to represent possible unobserved transitions. By making this state

highly rewarding we can also induce a distal exploration value function that will drive the learner toward novel state action pairs.

How exactly should we select an optimistic model? In the full OMS algorithm the model can be optimistic about the transition probabilities for any of the successors of i, a. To achieve this we employ the idea of bounded parameter MDPs (Givan et al., 1997). Rather than perform interval value iteration, we compute only the optimistic value function. Given state action pair i, a we order its successors by the current estimate of the value function, in descending order. We then calculate the lower and upper bounds of the $(1 - \alpha)$ probability interval for each transition. An optimistic transition function is then constructed by sending as much probability mass as possible to the states early in the ordering, while keeping all probabilities within their lower and upper bounds. The state action pair i, a is then backed up using the optimistic one-step transition function that results.

We compared simple and full OMS with Wiering's MBIE algorithm and Meuleau's variance based and worst case IEDP+ algorithms on four tasks based in two environments. We measured performance according to three criteria. Results showed OMS significantly outperformed the other techniques on most criteria in most cases.

4. Extending OMS

There are two necessary and important extensions of OMS. In the first we describe how in a hierarchy of multi-time models we need to construct optimistic multi-time models at all levels in order to control exploration between options appropriately. We suggest a simple approach is to calculate an optimistic model at the level of the lowest MDP, and that all optimistic multi-time models above this are obtained by treating it as the correct model. This should work because the only independent parameters in the model are all at the level of the lowest level MDP.

In the second extension we sketch a version of the algorithm for use with Dearden and Boutillier's structured prioritised sweeping using Dynamic Bayes Networks to represent the one-step model compactly. We hope to present an algorithm in which the reward function can be of an arbitrary form. It is likely it will be problematic to cleanly extend OMS to the DBN representation, and we explain why we currently believe this.

5. Monte Carlo methods

Two very simple approaches to approximating the exploration value function are based on Monte Carlo sampling. In the first, we follow Dearden and Strens (Dearden et al., 1999; Strens, 2000) by sampling directly from the density over models. Here we describe how to extend this to the case of using a Dynamic Bayes Network to encode the one step transition model. Since the DBN allows us to compactly represent the model and the value function sampling from this small space of models is more feasible for reasonable problems than doing it directly on the space of unfactored models.

The second approach is to run the MDP defined over the space of information states and then to infer the Bayes value function given the sample. This allows us to use a variety of techniques to infer the value function, including temporal difference learning. The hypothesis is that we may obtain some computational leverage over direct Monte Carlo sampling, because we are only paying attention to the parts of the model space that affect our current exploratory choices.

References

Bellman, R. E. (1961). *Adaptive control processes: A guided tour.* Princeton University Press.

Dearden, R., Friedman, N., & Andre, D. (1999). Model-based bayesian exploration. *Proceedings of the Fifteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-99)* (pp. 150–159). San Francisco, CA: Morgan Kaufmann Publishers.

Givan, R., Leach, S., & Dean, T. (1997). Bounded parameter markov decision processes. *Recent Advances in AI Planning: 4th European Conference on Planning.* Springer Verlag.

Martin, J. (1967). *Bayesian decision problems and markov chains.* New York: Wiley.

Strens, M. (2000). A bayesian framework for reinforcement learning. *Proceedings of the Seventeenth International Conference on Machine Learning* (pp. 943–950). Morgan Kaufmann.

Wiering, M., & Schmidhuber, J. (1998). Efficient model-based exploration. *From Animals to Animats 5: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior.*