
Reinforcement Learning using Optimistic Process Filtered Models

Funlade T. Sunmola
Jeremy L. Wyatt

FTS@CS.BHAM.AC.UK
JLW@CS.BHAM.AC.UK

School of Computer Science, University of Birmingham, Edgbaston, Birmingham B15 2TT, UK.

Abstract

An important problem in reinforcement learning is determining how to act while learning sometimes referred to as the exploration-exploitation dilemma or the problem of optimal learning. The problem is intractable, usually solved through approximation such as by being optimistic in the face of uncertainty. In environments with inherent determinism, arising for example from known process templates, acting conforms to certain acceptable conventions that limit exploration. We present an algorithm for the learning problem in which action selection is through optimistic models filtered to conform to conventions specified by a process language. We show results to illustrate the approach and its benefits for task transfers.

1. Introduction

A reinforcement learning (RL) agent learns by iteratively performing actions in the world and using resulting experiences to decide future actions. The experiences encapsulate among other things a reward or reinforcement on the actions taken. The environments in which agents act are usually stochastic and typically modelled as Markov Decision Processes (MDPs). Model-based RL agents build MDP models of their environments then plan and act using the models. Model-based approaches are advantageous in domains where real-world actions are expensive and computation time is relatively cheap.

Central to reinforcement learning is the challenges of exploration-exploitation trade off (Wyatt, 1997) and seeking to profit from structure inherent in the task environment (Dearden, 2000). The former is crucial for controlling action selection and is, typically for a reinforcement learning agent, an intractable problem of knowing how to act while learning so as to maximise lifetime performance. This inevitably involves balanc-

ing exploitation of current knowledge and exploration to discover new knowledge that might lead to better performance in the future. The latter contributes to alleviating the problems of scaling up to large domains by exploiting the natural organisation of the task environment.

In Markov Decision Processes (MDPs) the optimal Bayesian solution to the trade off problem is well known, but intractable (Martin, 1967; Bellman, 1961). Many approximate ways of dealing with the trade off in RL have been proposed. A new heuristic method was recently introduced (Wyatt, 2001) that brings together ideas from several recent approaches (Kaelbling, 1990; Wiering & Schmidhuber, 1998; Kearns & Singh, 1991). The heuristics provides a framework for exploration control in reinforcement learning based on optimistic model selection (OMS) from a density over possible models. The heuristics was shown to outperform existing methods when optimised. A structured version of the heuristics for factored MDPs using dynamic Bayesian networks is presented in (Sunmola & Wyatt, 2003). Essentially, through OMS, it is possible to perform a structured optimistic search on a density of possible process models while allowing a prior distribution over the space of models to be incorporated in the learning steps.

In this paper we study the leverage offered to the agents while learning to carry out a new task in an environment with inherent determinism arising, for example, from known process templates, value and policy constraints acquired typically from experiences on related tasks. We focus here on process (transition) model determinism and the constraints imposed on possible process models due to availability of conventions to which the learnt process models must conform. The templates are specified by a process (action) language to which there is an underlying context free grammar. The process templates provide added information and it would seem likely that acting based on optimistic models filtered through the templates should improve learning.

The paper is organised as follows: in section 2 we start by introducing reinforcement learning and the exploration control problem. Also contained in the section is an overview of the standard OMS algorithm. We concentrate on explicitly represented state-based MDPs. In section 3 we discuss the idea of constraining exploration and present a new OMS algorithm called *most optimistic first* that allows for process filtering. Section 4 describes empirical results on a circular flags world task and the paper is concluded in section 5 with directions for future research.

2. Reinforcement Learning

Our agent is learning to control a stochastic environment modelled as an MDP. An explicitly represented, state based, MDP is simply a 4-tuple (S, A, P, R) . In the tuple, S is a set of distinct states, A is a set of actions, $p_{ij}^a \in P$ is a transition function that captures the probability of reaching state j after executing an action a at state i such that $i, j \in S$, and $r \in R$ is a reward function mapping S into real-valued rewards. The state transition coefficients p_{ij}^a obey standard stochastic constraints hence they have the following properties $0 \leq p_{ij}^a \leq 1$ and $\sum_j p_{ij}^a = 1$. The decision problem for an agent in the MDP is to find a policy π that optimises the expected discounted total reward $V = E(\sum_{t=1}^{\infty} \gamma^{t-1} r_t)$, where r_t is the reward received t steps into the future and $\gamma \in [0, 1]$ is a discount factor.

2.1. Exploration Control Problem

In Bayesian formulation of the problem, we assume that there is a space \mathcal{P} of possible transition functions (parametric models) for the MDP and that there exists a prior probability density over this space. Given that a state $i \in S$ in the process has N possible succeeding states when an action a is taken, then the transition function from that state action pair is a multinomial distribution over the outcomes:

$$\vec{p}_i^a = \{p_{i1}^a, p_{i2}^a, \dots, p_{iN}^a\} \quad (1)$$

The possible transition functions from i, a are the possible \vec{p}_i^a .

A convenient, natural conjugate, choice of prior distribution over the \vec{p}_i^a is Dirichlet density:

$$f(\vec{p}_i^a | \vec{m}_i^a) = \frac{\Gamma(\sum_{j=1}^N m_{ij}^a)}{\prod_{j=1}^N \Gamma(m_{ij}^a)} \prod_{j=1}^N (p_{ij}^a)^{m_{ij}^a - 1} \quad (2)$$

and the density is parameterised by the $m_{ij}^a > 0$ for all j . The transition functions \vec{p}_i^a may be chosen based on

prior information available for the process or initialised to non-informative uniform values.

We observe process transitions. From our choice of likelihood function and prior distribution, it follows directly that the ensuing posterior distribution will be Dirichlet too with parameters $m_{ij}^{a''}$. For a single transition $i \xrightarrow{a,r} j$ the update rule is $m_{ij}^{a''} = m_{ij}^{a'} + 1$. The density for the multi-state case follows directly from this since the densities over the one step transition functions for all state action pairs are independent. The density $f(P|M)$ for a possible transition function $P \in \mathcal{P}$ for the MDP is therefore simply the product of the $f(\vec{p}_i^a | \vec{m}_i^a)$ over all i . The density is parameterised by the matrix $M = [m_{ij}^a]$ where $M \in \mathcal{M}$. In a Bayesian framework, we choose a prior matrix M' which specifies our prior density over the space of possible models. The additional information from a sequence of observations is captured in a count matrix F . The posterior density given these observations is therefore simply parameterised by $M'' = M' + F$.

Given the usual squared error loss function, the Bayesian estimator of expected return under the optimal policy is the expectation of the value function \tilde{V}_i in our MDP with unknown transition probabilities:

$$V_i(M) = E[\tilde{V}_i | M] = \int_{\mathcal{P}} V_i(P) f(P|M) dP \quad (3)$$

where $V_i(P)$ is the value of i given the transition function P . We know from the central result of both Bellman and Martin that when this integral is evaluated we transform our problem into one of solving an MDP with unknown transition probabilities, defined on the information space $\mathcal{M} \times \mathcal{S}$:

$$V_i(M) = \max_a \left\{ \sum_j \vec{p}_{ij}^a(M) (r_{ij}^a + \gamma V_j(T_{ij}^a(M))) \right\} \quad (4)$$

in which, for convenience, the transformation on M due to a single observed transition $i \xrightarrow{a,r} j$ is denoted $(T_{ij}^a(M))$, $\vec{p}_{ij}^a(M)$ is the marginal expectation of the Dirichlet, and r_{ij}^a is the reward associated with the transition $i \xrightarrow{a,r} j$. This shows how the Bayesian estimate of value elegantly incorporates the value of future information. The optimal solution to our exploration-exploitation trade off problem is thus to act greedily with respect to the Bayes Q-values. The solution to the problem is however intractable because it involves dynamic programming over a tree of information states. Approximate solutions may be found by either simply using the certainty equivalent (CE) estimate constructed by replacing $T_{ij}^a(M)$ with M , approximate the integral by random sampling, or select models optimistically in the face of uncertainty.

2.2. Optimistic Model Selection

The goal of model selection is to identify the one model, from a set of competing models, that best captures the regularities underlying the cognitive process of interest. The OMS method (Wyatt, 2001) integrates ideas from a popular family of approximate approaches to the exploration-exploitation trade off which typically use some instantiation of the heuristic ‘be optimistic in the face of uncertainty’ (Kaelbling, 1990; Wiering & Schmidhuber, 1998; Meuleau & Bourguine, 1999). OMS integrates the instantiation idea with the Bayesian view of exploration by selecting an optimistic model P_{opt} from \mathcal{P} using probability intervals calculated based on $f(P|M)$. Wyatt suggests two ways of selecting P_{opt} , which were termed simple and full OMS. In simple OMS we are optimistic only about hypothesised transition to an imaginary terminal state. In full OMS we can be optimistic about transitions to other states too.

2.3. Full OMS

Following an initialisation step (step 1), the main loop (steps 2-5) of an OMS-oriented solution approach to the exploration-exploitation trade off problem is organised as follows:

Step 1: initialise process values and model parameters.

Step 2: select action based on the current value estimate, interact with environment and observe transition.

Step 3: update parameter matrix in standard way, based on the observed transition.

Step 4: select an optimistic model P_{opt} .

Step 5: do value backup using P_{opt} .

In the full OMS, P_{opt} is selected as follows. 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. We construct an optimistic transition function by allocating the maximum probability mass to states early in the ordering while keeping all probabilities within their lower and upper bounds. For example, consider a process shown in Figure 1 at state $i = 1$ under action a and with four observed successor states $j = [2, 4, 3, 1]$. The successor states are ordered according to their value function estimates in descending order and we select a full optimistic model $\vec{p}_{opt,1}^a = [0.38, 0.32, 0.20, 0.10]$.

Once $\vec{p}_{opt,i}^a$ is selected, the state action pair i, a is then backed up using the optimistic one step transition function that results. The relevant Bellman equation is:

$$\xi_i^a(M) = \sum_j p_{opt,ij}^a(M)(r_{ij}^a + \gamma \max_{a'} \{\xi_j^{a'}(M)\}) \quad (5)$$

where $p_{opt,ij}^a(M)$ are the transition probabilities according to P_{opt} . The agent selects the action with the highest optimistic value ξ_i^a .

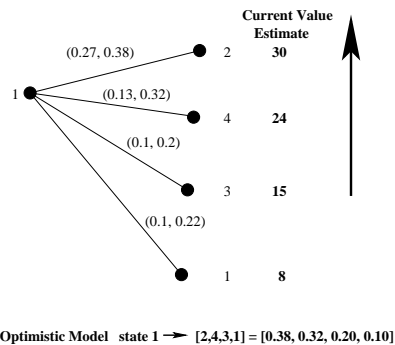


Figure 1. An example process at state $i = 1$ under action a with four successor states. Bounds on probabilities for each transition are shown as (lowerbound, upperbound).

3. Constraining Exploration

We consider feasible regions $\mathcal{P}' \in \mathcal{P}$ defined as the set of points in the model space for which the MDP satisfies all specifications on its behavior. By selecting models that belong only to the feasible region we constrain exploration to relevant parts in the space of possible process models. The integral of equation 3 is constrained to the following.

$$V_i(M) = E[\tilde{V}_i|M] = \int_{\mathcal{P}'} V_i(P) f(P|M) dP \quad (6)$$

$\mathcal{P}' \in \mathcal{P}$ represents the feasible region where the model parameters are such that the MDP and, consequently, the learning agent satisfy behavioral requirements.

In this work, we do not attempt to evaluate the above integral. Instead, the feasible region is approximated using a process template and we attempt to select optimistic models that comply with the approximated feasible region. The optimal policy is the point P at which the expected return $V_i(M)$ is maximised given the feasible region defined by the process template. In the sections that follow, we introduce a process description language that specifies the template and describe an algorithm for selecting conforming optimistic models.

3.1. Process Description Language

A process description language is a specialised language (or grammar) that helps express an MDP precisely and compactly in a system of words that can be understood by an agent and its learning algorithm. Central to the process description of an MDP is an action language component. The action description language (Gelfond & Lifschitz, 1993) which was introduced in 1992 became very popular particularly for model checking (Bultan, 2000) and has been designed to come up with a specification language which describes the effect of actions in a simple, elegant and natural way. In general, action languages are formal models specifically designed to specify actions and their effects.

Formal languages (or automata) constitute a cornerstone of computer science (Salomaa, 1973). A formal language is normally defined by an alphabet and formation rules. The alphabet of a formal language is a set of symbols on which the language is built. Some of the symbols in an alphabet may have a special meaning. The formation rules specify which strings of symbols should be considered as well-formed, often called words, expressions, formulas, or terms. The formation rules are usually recursive, they postulate that such and such expressions belong to the language in question or establish how to build well-formed expressions from other expressions belonging to the language. Formation rules are sufficient for defining simple languages. More syntactically complex languages can be defined by means of finite automata, grammars, regular expressions or certain operations.

In a grammar system we have an initial symbol and a set of rewriting rules, which state how a word is derived from another. A grammar defining formal language \mathcal{L} is a quadruple (L_N, L_T, L_R, L_S) , where L_N is a finite set of nonterminals, L_T is a finite set of terminal symbols, L_R is a finite set of productions, and L_S is an element of L_N . The set L_T of terminal symbols is L 's alphabet. Nonterminals are symbols representing language constructs. The sets L_N and L_T should not intersect. L_S is called the start symbol. Productions are rules of the form: $prod_a \rightarrow prod_b$, where both $prod_a$ and $prod_b$ are strings of terminals and nonterminals, $prod_a$ contains at least one nonterminal.

3.2. Optimistic Process Filtered Models

Filtering is the problem of estimating the state of a system as a set of observations becomes available online. A process filter is predicated on the ideas of the general filtering problem. Given a process template based upon a process description language \mathcal{L} , a process

Table 1. Main loop of the OMS algorithm with Process Filtering

Optimistic Process Filtered Model Selection

begin

Initialise t , prior parameter matrix m , process description language \mathcal{L} , and exploratory value function ξ

repeat

observe current state x .

choose an action a based on value function ξ_x , breaking ties randomly.

do a and observe transition $x \xrightarrow{a,r} y$.

update parameter matrix m .

until your ARTDP algorithm stops

choose state i

for each action b

find \vec{p}_{opt}^b using most-optimistic-first

update ξ_i^b using equation 5

update $t \rightarrow t + 1$.

end

filter estimates the importance of a process model P_i for an MDP, attaching weight w_i to the model. A weight is conceptually the probability of the process model given the language.

$$w_i = Pr(P_i|\mathcal{L}) \quad (7)$$

The model with maximum weight is selected as the most conforming process model. Feasible process models are approximated as those whose weights are greater than a specified threshold β i.e. $Pr(P_i|\mathcal{L}) > \beta$, with β typically small ≈ 0 .

We add a filter component to the standard OMS algorithm. The filter component discourages process models which have a zero conformance probability. It takes as input an optimistic model, passes the model through the process language and return state transitions in the optimistic model that are not in conformance with the specified language. The main loop of the OMS algorithm incorporating process filtering in an asynchronous real time dynamic programming (ARTDP) framework is shown in table 1. Any other asynchronous methods may be used with the OMS algorithms in a convenient way.

Search of the feasible model space is conducted by using a full OMS procedure to first obtain the most optimistic model for the MDP at the current time step. The most optimistic model is then passed through the process filter component. If all transitions of the optimistic model conform to the language the most optimistic model is selected and the state action pair is

Table 2. A description of process shown in Figure 1.

Language fragment for a sample process

** *greatestsuccstate* is the state that has the largest transition probability in the set of successors to the current state under action (*act*)

** $\text{Prob}(i \xrightarrow{\text{act}} j)$ is the transition probability for state *i* to state *j* under action *act*

var := {*greatestsuccstate*}

operators := {mult,>}

state_1transitions(act):

greatestsuccstate := 2;

$\text{Prob}(1 \xrightarrow{\text{act}} 1) > \text{mult}(0.5, \text{Prob}(\text{greatestsuccstate}))$;

then backed up using the selected optimistic model. Otherwise, if there are non-conforming transitions, we reapply the full OMS algorithm on the non-conforming transitions.

To illustrate, assume the process language of table 2 applies to the example of Figure 1. The language specifies transition from state 1 to state 2 (i.e. $1 \xrightarrow{\text{act}} 2$) as having the largest probability in the set of successors of state 1. Using the current optimistic probability estimates for transitions from state 1, the transition probability of $1 \xrightarrow{\text{act}} 1$ is expected to be greater than $0.5 * 0.38 = 0.16$ according to the process language. The lower bound for the transition $1 \xrightarrow{\text{act}} 1$ is reset to 0.16, defining a feasible region for the model. The current optimistic process model is updated to $\vec{p}_{opt,1}^{act} = [0.38, 0.32, 0.14, 0.16]$ as shown in Figure 2.

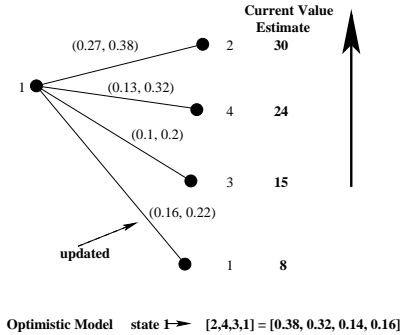


Figure 2. An extension of example shown in Figure 1, allowing for the language fragment described in table 2

4. Empirical Results

To test the optimistic methods presented in this paper, we used the optimistic model selection algorithms with and without process filtering in learning how to behave in a reinforcement driven environment. We employed a flags world domain similar in principle to the one described in (Dearden, 2000) where an agent attempts to collect flags and get them to a goal.

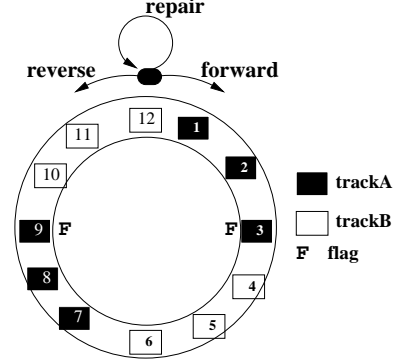


Figure 3. The reinforcement learning task in a three action circular flags world composed of twelve cells and two binary flags positioned at cells 3 and 9. Cell 12 is the start and goal cell.

We devised a circular flags world shown in Figure 3 that contains twelve cells and two binary flags. The agent navigates the flags world using an automatic guided vehicle (agv) that may occasionally breakdown. At any given time, flags in the world may be either set or unset and the agv may have a status of either faulty or not faulty. In all, the flags world have $12 \times 2^2 \times 2 = 96$ possible states. In the flags world instance of Figure 3, the start and goal positions are located in cell 12 while the flags (marked 'F' in the Figure) are positioned in cells 3 and 9. The cells in the flags world are divided into sections which in our experiments we labeled as *track A* and *track B*. The agent receives reward on reaching the goal cell based on the number of flags collected.

Initially, the agent starts at the start cell with an agv that is not faulty and all flags unset. The agent navigates the world with three possible actions *forward*, *reverse*, and *repair* moving the agent forward to the next cell, backwards to the previous cell and carrying out repairs when the agv is faulty, respectively. The *i*'th flag is set only if the agv is not faulty and when the agent executes an action at the cell containing the flag. Once set, the flag may only be unset at the goal cell. All the flags are unset once the agent executes an action at the goal cell. Transitions from one cell to an-

Table 3. A description of the circular flags world instance used in the experiments

Circular flags world instance with 96 states and 2 flags.

** `greatestsuccstate` is the state that has the largest transition probability in the set of successors to the current state under action (`act`)
 ** `Prob(i,act,j)` is the transition probability for state *i* to state *j* under action *act*

`action := { forward | reverse | repair }`
`operators := {move, not, override, tol, add, mult, toggle, trackfactor, ingoalcell, inflagcell, flagisset, faulty}`
`var := {greatestsuccstate,curstate,altsucstate}`

doingForwardReverse(action,curstate):
`greatestsuccstate := move(action, curstate);`
`altsucstate := move(toggle(action, 'action'), curstate);`
`if (inflagcell, not(flagisset), not(faulty))`
`greatestsuccstate:=override(greatestsuccstate,flagisset);`
`altsucstate := override (altsucstate, flagisset);`
`if (ingoalcell)`
`greatestsuccstate:=override(greatestsuccstate,`
`not(flagisset));`
`if (faulty)`
`Prob(curstate,action,altsucstate) := tol(0.14, 0.02);`
`if (action == 'forward')`
`Prob(curstate,action,curstate) := tol(0.2, 0.01);`
`else Prob(curstate,action,curstate) := tol(0.3, 0.01);`
`Prob(curstate,action,greatestsuccstate):=`
`add(mult(trackfactor(curstate,action),Prob(curstate,`
`action, curstate)), Prob(curstate,action,altsucstate));`

doingRepair(action,curstate):
`if (faulty)`
`if (ingoalcell) greatestsuccstate := toggle (over-`
`ride(curstate,not(flagisset)), 'fault');`
`else greatestsuccstate := toggle(curstate,'fault');`
`Prob(curstate,action,greatestsuccstate):=tol(0.945,0.013);`
`Prob(curstate,action,curstate) := tol(0.045, 0.013);`

other is stochastic for all three actions. Inherent in the flags world are structural dependencies. For example, due to the flags, the probability of transitions from one cell to another does not depend on flags setting.

We carried out our experiments on a specific instance of the circular flags world of Figure 3. The task to learn is specified with a reward of 10 for setting a single flag and 100 for setting both flags. We choose a non-informative prior matrix with zero transitions for all entries except for transitions leading to an imaginary state which we set to 1. The template provides information about the process particularly at states in which the agv is faulty.

In addition, available to the agent, is a template

(shown in table 3) that specifies a fragment of the process. The process fragment described in table 3 consists of two action modules. The `doForwardReverse` module applies to the forward and reverse actions, and the `doingRepair` module applies to the repair action. In each of the modules, given the current state (`curstate`) and the current action, the agent can determine the state (`greatestsuccstate`) which has the largest transition probability amongst the successor states to the current state. In deriving the `greatestsuccstate`, operators `move` and `override` are used. `Move` is a function that takes as input an action and the current state then returns the next state in the flags world when the action is performed successfully, ignoring requirements for flag settings. `Override` takes as input a state and a flag setting then returns a corresponding state in which the flag variable has the same setting as the flag setting parameter passed to it.

Also specified in the action modules are guides for the transition probability from the current state to the `greatestsuccstate` under a given action. The specification uses function `tol` which takes two floats (fl_1, fl_2) and returns a tolerance range $fl_1 \pm fl_2$. Also used is the function `trackfactor` that returns a number for a given action and state. `Trackfactor` is set as follows. For forward action, `trackfactor` returns 3.0 if the current state is in track A and 2.4 for track B. For reverse action, `trackfactor` returns 1.33 for both tracks A and B.

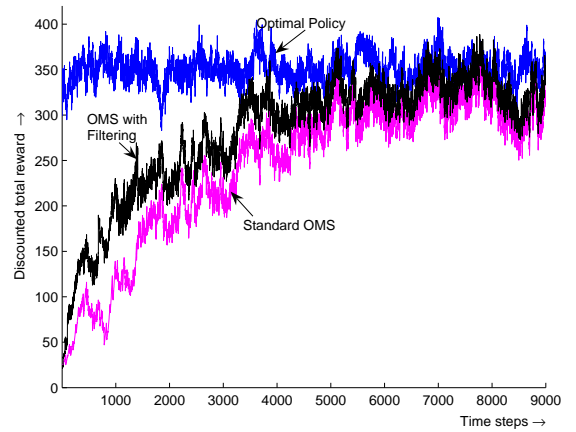


Figure 4. Plot of discounted total rewards over time for the OMS algorithms in comparison with Optimal Policy on a circular flags world instance averaged over 10 trials.

Performance of the learning agent can be measured in several ways. To account for exploration and exploitation tradeoff we measured the discounted total

reward-to-go at each point at each time step. More precisely, suppose the agent receives the following rewards r_1, r_2, \dots, r_t in a run of time length t . The reward-to-go at time t' is defined to be $\sum_{t' \geq t} r'_t \gamma^{t'-t}$. This estimate is reliable only for points that are far enough from the end of the run. In Figure 4, we plot the discounted total reward-to-go for each of the OMS algorithms in comparison to the optimal policy, as a function of time averaged over 10 runs. As expected, the optimum policy gave the largest discounted total reward, averaging 350 from start. The most optimistic first algorithm based on optimistic model selection with filtering resulted in an improved performance when compared to standard OMS without filtering. The slight gain is attributed to the additional information made available to the most optimistic first algorithm. Whilst these are preliminary results, it thus offer prospects for accomplishing task transfers as the template provides a constraining influence on exploration early in learning for related tasks covered by the template.

5. Conclusions and Future Work

We have extended the standard optimistic model selection algorithm to incorporate a process filter. The process filter discourages the selection of models that do not conform to a specified process description language. We studied a version of the OMS algorithms called most optimistic first. Preliminary results indicate that by filtering the process models the most optimistic first algorithm is able to constrain exploration with prospects of improving learning performance. Areas of further work include establishing the efficiency of the most optimistic first algorithm, improving its performance, and studying other filtering methods. In addition, we are studying additional leverage obtained using factored representations. Finally, other important areas of future work are the sensitivity of learning to variations in process templates and the linkage between the process templates and the prior information matrix.

References

- Bellman, R. E. (1961). *Adaptive control processes: A guided tour*. Princeton University Press.
- Bultan, T. (2000). Action language: A specification language for model checking reactive systems. *Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000)* (pp. 335–344).
- Dearden, R. (2000). *Learning and planning in structured world*. Department of Computer Science, University of British Columbia, Canada: Ph.D.Thesis.
- Gelfond, M., & Lifschitz, V. (1993). Representing action and change by logic programs. *Journal of Logic Programming*, 17, 301–321.
- Kaelbling, L. P. (1990). *Learning in embedded systems*. Dept. of Computer Science, Stanford: Ph.D.Thesis.
- Kearns, M., & Singh, S. (1991). Near-optimal reinforcement learning in polynomial time. *Proceedings of the Fifteenth International Conference on Machine Learning*, 12, 993–1001.
- Martin, J. (1967). *Bayesian decision problems and Markov chains*. New York: Wiley.
- Meuleau, N., & Bourguin, P. (1999). Exploration of multi-state environments: Local measures and back-propagation of uncertainty. *Machine Learning*, 35, 117–154.
- Salomaa, A. (1973). *Formal languages*. Academic Press.
- Sunmola, F., & Wyatt, J. (2003). Optimistic model selection in structure based reinforcement learning. *Proceedings of the Sixth European Workshop on Reinforcement Learning* (pp. 31–32).
- Wiering, W., & Schmidhuber, J. (1998). Efficient model-based exploration. *From Animals to Animats 5: Proceedings of the Fifth International Conference on Simulation of Adaptive Behaviour* (pp. 223–228).
- Wyatt, J. L. (1997). *Exploration and inference in learning from reinforcement*. University of Edinburgh, Dept. of Artificial Intelligence, Edinburgh University: Ph.D.Thesis.
- Wyatt, J. L. (2001). Exploration control in reinforcement learning using optimistic model selection. *International Conference on Machine Learning (ICML 2001)*, 593–600.