

# Gradient-Based Relational Reinforcement Learning of Temporally Extended Policies

Charles Gretton

NICTA, 300 Adelaide St, Brisbane QLD 4000, Australia  
charles.gretton@nicta.com.au

## Abstract

We consider the problem of computing general policies for decision-theoretic planning problems with temporally extended rewards. We describe a gradient-based approach to relational reinforcement learning (RRL) of policies for that setting. In particular, the learner optimises its behaviour by acting in a set of problems drawn from a target domain. Our approach is similar to *inductive policy selection* because the policies learnt are given in terms of relational control-rules. These rules are generated either (1) by reasoning from a first-order specification of the domain, or (2) more or less arbitrarily according to a taxonomic concept language. To this end the paper contributes a domain definition language for problems with temporally extended rewards, and a taxonomic concept language in which concepts and relations can be temporal. We evaluate our approach in versions of the miconic, logistics and blocks-world planning benchmarks and find that it is able to learn good policies. Our experiments show there is a significant advantage in making temporal concepts available in RRL for planning, whether rewards are temporally extended or not.

Decision-theoretic planning systems are often called upon to solve numerous related problems from a particular domain. This was the case at the recent International Probabilistic Planning Competitions (Younes *et al.* 2005). More importantly, it is also usually the case in practice. In the domain of energy-distribution networks (Thiébaux & Cordie 2001), we usually plan for a number of distinct systems made from common classes of objects (circuit breakers, switches, remote controllers, etc.). Because domains in planning often exhibit a strong relational structure, they are formalised using first-order languages supporting the declaration of objects and relations between these as well as the use of quantification over objects.

The cost of isolated planning in individual problems is substantial. State-of-the-art solution algorithms target either state-based (tabular) or factored propositional problem representations, thus they succumb to Bellman's curse of dimensionality – i.e. The complexity of computing the optimal policy for a problem instance can be exponential in the dimension of the problem (Littman, Goldsmith, & Mundhenk 1998). A research direction which has garnered significant

attention recently is that of *generalisation in planning*. The idea is that the cost of planning with propositional representations can be mitigated by technologies that plan for a domain rather than for individual problems. These approaches yield *general policies* which can be executed in any problem state from the domain at hand. In practice general policies are expressed in first-order/relational formalisms. Proposals to date suggest general policies can be achieved by either (1) reasoning from the domain description (Boutillier, Reiter, & Price 2001; K. Kersting, M. V. Otterlo, & L. D. Raedt 2004; Sanner & Boutillier 2005; Karabaev & Skvortsova 2005; Wang, Joshi, & Khardon 2007), or (2) by developing planners that can *learn from experience* (Khardon 1999; Martin & Geffner 2000; C. Guestrin *et al.* 2003; Hernandez-Gardiol & Kaelbling 2003; Kersting & Raedt 2004; Fern, Yoon, & Givan 2006). There has also been some work in combining the two (Gretton & Thiébaux 2004).

Reasoning approaches can achieve optimal general policies without recourse to individual problems. On the downside they rely on expensive theorem proving and cannot give guarantees about the quality and generality of policies they compute for domains where the value of a state is drawn from an infinite set. For example, this is the case in the blocks-world because the value of a state given any policy is the number of actions it takes for that policy to achieve the goal, which is proportional to the number of blocks. Inductive learning approaches have a significant advantage over reasoning approaches because they avoid theorem proving and do not rely on an exhaustive domain description. They rarely achieve optimality, but are able to compute good policies with very little effort. As was the case for reasoning approaches, learning techniques cannot learn a policy in terms of state values in domains where these are drawn from an infinite set (K. Kersting, M. V. Otterlo, & L. D. Raedt 2004).

Orthogonal to generalisation in planning, there have also been significant developments towards propositional planning with *temporally extended rewards*. In this case, rather than accepting the standard scenario where rewards are allocated to individual states, rewards are allocated to sequences of states called *rewarding behaviours*. Typical examples of rewarding behaviours occur where we reward the maintenance of some property, the periodic achievement of some objective, the achievement of an objective after a trigger has occurred (and not expired), or the first achievement of an

objective. These rewards are not supported in a reasonable way where problems are modelled using Markov decision processes (MDPs), the standard problem representation formalism. In particular, for an MDP we say both dynamics and reward are *Markovian*, because at any time both the effects of an action and the reward allocated are determined completely by the state the process is in. Moreover, although it may be possible in principle to manually compile temporally extended rewards into an MDP, by adding propositions that capture temporal events, the original structure is lost on an MDP solution algorithm that is not aware of the temporal interpretation of some state-characterising propositions. In order to address weaknesses in the MDP model where temporally extended rewards are involved, formalisms and solution methods have been proposed for decision processes with non-Markovian rewards (NMRDPs) (Thiébaux *et al.* 2006). For an NMRDP, the problem dynamics are Markovian, and reward is a compact temporal logic specification of temporally extended rewards. NMRDP solution methods exploit the temporal logic specification of the rewarding behaviours to efficiently translate NMRDPs into equivalent MDPs amenable to MDP solution methods. Consequently, NMRDP solution techniques still succumb to Bellman’s curse.

We develop ROPG (Relational Online Policy Gradient), an unsupervised RRL approach to computing temporally extended policies for domains with non-Markovian rewards. In a similar vein to some of the more fruitful techniques for RRL such as inductive policy selection (Yoon, Fern, & Givan 2002; Gretton & Thiébaux 2004), ROPG learns policies in terms of relational control rules. Each control-rule is a small expression in a first-order language that can be interpreted at an NMRDP state to provide an action prescription. We adapt two very different techniques from the literature for generating relational control rules, and evaluate each of these separately in our experimental results. The first technique is based on (Fern, Yoon, & Givan 2006). Relational control rules are generated more or less arbitrarily according to the grammar of a temporal taxonomic concept language. In order to avoid redundant taxonomic control rules, and also to avoid overwhelming the learner with too many rules, they are evaluated in small problems and only a small number of rules that behave well are passed to ROPG. The second technique is based on (Gretton & Thiébaux 2004). In this case we exploit first-order regression to generate control rules from a given domain description that are guaranteed to cover all concepts relevant to the optimal  $n$ -state-to-go value function for a given  $n$ . To this end, we develop a domain description language which accommodates non-Markovian rewards, and also extend the standard definition of first-order regression to this setting.

ROPG is a version of online policy gradient, and thus learns by acting in problems. ROPG is the first reported technique for direct relational reinforcement learning of general policies which does not rely on a state-based planning (or learning) mechanism. This means ROPG does not capture action decisions made by an optimal (or good) state-based planning agent which is inevitably better equipped to distinguish states according to propositional features that are

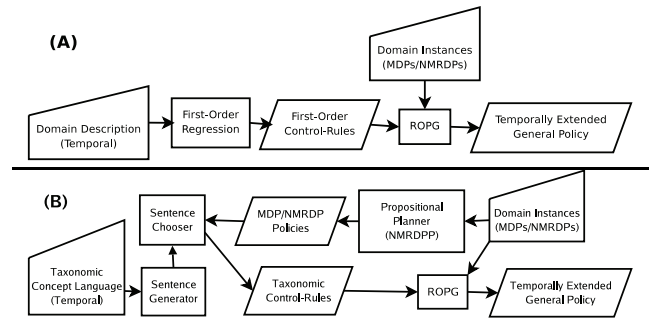


Figure 1: Data-flows for the two settings in which we evaluate ROPG. (A) Demonstrates the case where control rules are generated using first-order regression, and (B) the case where control rules are generated according to a taxonomic concept language.

not available to a relational learner. Along the same lines, ROPG also addresses some pitfalls of value-based relational reinforcement learning. Our approach directly learns a policy, thus it does not attempt to classify the infinite states from a target planning domain according to a finite set of values. Rather, it classifies those states according to an infinite set of actions prescribed by a small set of relational control rules. To summarise, ROPG learns a general policy directly by acting in domain instances. Consequently, ROPG is suited to planning in domains such as the blocks-world, where there is no known sound technique to represent a general policy in terms of a value function with finite range. Moreover, ROPG is not crippled by a reinforcement learning scheme which punishes a learner for not mimicking the actions of a “problem specialist” in the form of a state-based agent. Figure 1 shows the data-flows for using ROPG in the cases where control rules are generated by regression, and where these are generated according to a taxonomic syntax.

We evaluate our approach in Markovian, non-Markovian, stochastic and deterministic versions of the miconic, logistics and blocks-world planning benchmarks, where control rules available to the learner are generated according to our temporal taxonomic syntax, the extended taxonomic syntax, or regression. We find that ROPG can obtain a good general policy by learning in small to medium sized instances drawn from a target domain. Our experiments also show there is a significant advantage in making temporal concepts available in RRL for planning whether rewards are temporally extended or not.

The paper is organised as follows: We develop our domain description formalism for decision-theoretic planning problems with non-Markovian reward. We then develop two mechanisms for automatically computing *relational control rules*, the basis for our learner’s actions and observations in a domain. We then present our online policy gradient approach for computing a general policy for a domain given an arbitrary set of control rules. We present experimental results, and then discuss our approach further and consider future directions.

## Domains with Non-Markovian Reward

We require a *domain description language* for decision-theoretic planning domains where problem instances can have non-Markovian rewards. Our intention here is to develop a formalism that allows us to describe a domain in terms of its relational structure. To this end we use a logic and axiom schemes that let us describe (1) objects and relations between these, (2) actions, their preconditions and effects, and (3) the details of how reward is allocated to states and/or state sequences. Key to this is our *logic of event-formulae*, a first-order past-looking linear temporal logic we developed, denoted  $\mathcal{L}_{\leftarrow}^{\text{fo}}$ , with equality, first-order quantification over objects  $\{\forall, \exists\}$ , and the usual binary connectives  $\{\wedge, \vee, \neg, \rightarrow, \leftrightarrow\}$ . So that we can write about rewarding behaviours – and also so that we have a mechanism for compactly specifying the behaviour of fluents when actions are executed – sentences in  $\mathcal{L}_{\leftarrow}^{\text{fo}}$  must be able to discuss histories. To this end we consider temporal operators from PLTL the *linear temporal logic of the past*, a mechanism for expressing rewarding behaviours for NMRDP in the propositional case (Bacchus, Boutilier, & Grove 1996). In particular, this means we have  $\ominus$ , pronounced “in the last state”,  $\Box$  “always in the past”,  $\diamond$  “sometime in the past”, and  $\mathbf{S}$  “since”.<sup>1</sup>

In what follows “\*” is the Kleene star, so for example if  $\mathcal{S}$  is the set of states and  $\Gamma \in \mathcal{S}^*$ , then we have that  $\Gamma$  is a finite sequence of states. Also, where  $i$  is a natural number,  $\Gamma_i$  is the state at index  $i$  in  $\Gamma$ , and  $\Gamma(i)$  is the prefix  $\langle \Gamma_0, \dots, \Gamma_i \rangle \in \mathcal{S}^*$  of  $\Gamma$ . We develop  $\mathcal{L}_{\leftarrow}^{\text{fo}}$  for specifying rewarding behaviours and domain action physics. Consequently we need  $\mathcal{L}_{\leftarrow}^{\text{fo}}$  to specify both behaviours as sequences of states, and also trajectories of state action pairs. Henceforth, we shall refer to the latter as an *event*, denoted  $\Sigma$ . Thus, for a domain with states  $\mathcal{S}$  and actions  $\mathcal{A}$ , we can write  $\Sigma \in (\mathcal{S} \times \mathcal{A})^*$ . We develop a semantics for  $\mathcal{L}_{\leftarrow}^{\text{fo}}$  so that sentences in the language relate via a modelling relation  $\models$  to events. A formula without action symbols is a behavioural formula. These are analogues of state-formula in the situation-calculus (Reiter 2001) in the sense that state-formula are for MDPs what a behavioural formula is for NMRDPs – In the situation-calculus, state-formula discuss properties of states while behavioural formulae in  $\mathcal{L}_{\leftarrow}^{\text{fo}}$  discuss histories as sequences of states. A behavioural formula free of temporal operators is directly analogous to a state-formula in the situation-calculus.

$(\Sigma, i) \models \text{act} = A$	iff	$A$ is the action part of $\Sigma_i$
$(\Sigma, i) \models p$	iff	where $\Sigma_i = (s_i, a_i)$ , $p \in s_i$
$(\Sigma, i) \models f_1 \vee f_2$	iff	$(\Sigma, i) \models f_1$ or $(\Sigma, i) \models f_2$
$(\Sigma, i) \models \neg f$	iff	It is not the case that $(\Sigma, i) \models f$
$(\Sigma, i) \models \exists x.f$	iff	There is an object $X$ , so that $(\Sigma, i) \models f[x/X]$
$(\Sigma, i) \models \ominus f$	iff	$(\Sigma, i-1) \models f$ and $i > 0$
$(\Sigma, i) \models f_1 \mathbf{S} f_2$	iff	$\exists j \leq i$ s.t. $(\Sigma, j) \models f_2$ and $\forall j < k \leq n$ $(\Sigma, k) \models f_1$

We assume the object universe is non-empty so that for all  $\Sigma$ ,  $(\Sigma, 0) \models \exists x.\top$ . Operators  $\forall, \wedge, \rightarrow$  and  $\leftrightarrow$  are given via

<sup>1</sup>We do not consider the case of  $\mathcal{L}_{\leftarrow}^{\text{fo}}$  formulae with free variables here because they have no place in a domain description.

formula rewrites. Finally, we adopt the shorthand  $\exists : \mathbb{P}(x).f$  for  $\exists x.(\mathbb{P}(x) \wedge f)$  and  $\forall : \mathbb{P}(x).f$  for  $\forall x.(\mathbb{P}(x) \rightarrow f)$ .

We now develop axiom schemes for specifying a domain of NMRDP with examples for the logistics domain in (Boutilier, Reiter, & Price 2001):

**Action precondition axioms:** For each deterministic action  $A(\vec{y})$ , we write one axiom of the form:

$$\models \forall \vec{y}.(\text{act} = A(\vec{y}) \rightarrow \text{poss}(A(\vec{y})))$$

where  $\text{poss}(A(\vec{y}))$  is a behavioural formula characterising the preconditions of the action. For example, in logistics we have the action  $\text{Load}(b, t)$  that loads a box  $b$  onto a truck  $t$ . This is possible when  $b$  and  $t$  are in the same city, and  $t$  does not already have a box loaded on it <sup>2</sup>:

$$\models \forall : \text{Truck}(x).(\forall : \text{Box}(y).\text{act} = \text{Load}(y, x) \rightarrow \neg(\exists : \text{Box}(b).(\text{On}(b, x)))) \wedge (\exists : \text{City}(c).(\text{BIn}(y, c) \wedge \text{TIn}(x, c)))$$

**Successor-states axioms:** specify the behaviour of a fluent under the domains deterministic actions. For each fluent  $p(\vec{y})$ , there is one axiom of the form:

$$\text{For all } \Sigma \text{ and } i > 0, (\Sigma, i) \models \forall \vec{y}. \ominus \text{SSA}(p(\vec{y})) \leftrightarrow p(\vec{y})$$

where  $\text{SSA}(p(\vec{y}))$  is an event-formula characterising the truth value of  $p(\vec{y})$  in the situation resulting from performing an action in a state. In logistics we have fluent  $\text{OnT}$ , so that  $\text{OnT}(\text{News}, \text{truck})$  says that the newspaper is on the truck. A package is on a truck at some event, if it was previously on the truck and this fact is maintained by action choices excluding the unloading of the package from the truck, or if the package was legally loaded onto the truck and since then that fact has not been disturbed:

$$\text{For all } \Sigma \text{ and } i > 0, (\Sigma, i) \models \forall : \text{Box}(b).\forall : \text{Truck}(t) . (\ominus[(\text{act} = \text{Load}(b, t)) \vee (\text{OnT}(b, t) \wedge \neg(\text{act} = \text{Unload}(t)))] \leftrightarrow \text{OnT}(b, t))$$

**Unique-name axiom:** In order that we can evaluate action equality based on the action symbol and its arguments, we include unique-name axioms. For any two distinct action symbols  $A$  and  $B$  we have

$$\models \forall \vec{x} \forall \vec{y} A(\vec{x}) \neq B(\vec{y})$$

**Nature’s choice and probability axioms:** For stochastic action  $A(\vec{x})$  we specify the deterministic actions  $D_1(\vec{x}), \dots, D_k(\vec{x})$  available for nature to choose from, and the probability  $r_1, r_2, \dots, r_k$  that nature makes a particular choice. Each deterministic choice  $D_i$  uniquely identifies the stochastic action symbol  $A$  that permits nature making choice  $D_i$ .

$$\begin{aligned} &\forall \vec{x} \text{ when } (\Sigma, i) \models \text{act} = A(\vec{x}) \text{ then} \\ &\text{if } ((\Sigma, i) \models f_1(\vec{x})) [D_1(\vec{x}), r_1; \dots; D_{k^1}(\vec{x}), r_{k^1}^1]; \\ &\text{else if } ((\Sigma, i) \models f_2(\vec{x})) [D_{k^1+1}(\vec{x}), r_{k^1+1}; \dots; \\ &\quad D_{k^2}(\vec{x}), r_{k^2}]; \dots \\ &\text{else } [D_{k^{m-1}+1}(\vec{x}), r_{k^{m-1}+1}; \dots; D_{k^m}(\vec{x}), r_{k^m}]; \end{aligned}$$

For example, consider a stochastic logistics domain where it can rain. Unloading a box from a truck is non-deterministic

<sup>2</sup>Notice that  $\mathcal{L}_{\leftarrow}^{\text{fo}}$  can accommodate non-Markovian dynamics, however in this paper we shall not consider those.

so that nature decides whether the unloading is successful `UnloadS` or otherwise `UnloadF`. If it is not raining, unloading a truck is successful 90% of the time, otherwise it is only successful 30% of the time:

```

 $\forall \vec{x}$  when act = Unload( $\vec{x}$ ) then
  if(¬raining)[UnloadS( $\vec{x}$ ), 0.9; UnloadF( $\vec{x}$ ), 0.1];
  else[UnloadS( $\vec{x}$ ), 0.3; UnloadF( $\vec{x}$ ), 0.7];

```

**Reward axiom:** Rewards as they are allocated to events are conveniently expressed using the inclusive conditional form:

```

  if( $(\Sigma, i) \models f_1$ )R( $\Sigma(i)$ )+ =  $r_1$ ;
  also if( $(\Sigma, i) \models f_2$ )R( $\Sigma(i)$ )+ =  $r_2$ ;
  ...
  also if( $(\Sigma, i) \models f_n$ )R( $\Sigma(i)$ )+ =  $r_n$ ;

```

Here each  $f_j$  is a behavioural formula. Where `R` is the non-Markovian reward function so that `R( $\Sigma(i)$ )` is the reward achieved at event  $\Sigma(i)$ , the semantics for the inclusive conditional are

$$R(\Sigma(i)) = \sum_{j \text{ s.t. } (\Sigma, i) \models f_j} r_j$$

For example we could have the agent only receives reward the first time it delivers a package correctly:

```

  if( $(\Sigma, i) \models (\forall : \text{Package}(p).(\text{Delivered}(p) \wedge \ominus \boxminus \neg \text{Delivered}(p)))$ )R( $\Sigma(i)$ )+ = 100.0;

```

## Generating Control Rules for General Policies

ROPG computes a general policy over a small set of control rules which in our case are generated automatically. Policies over relational control rules already appear in the literature, for example, they can correspond to a relational version of the classic Rivest-style decision list (Kharden 1999; Martin & Geffner 2000; Fern, Yoon, & Givan 2006). A control-rule is an expression in a relational formalism which *prescribes* an action given a problem state. For example, taking an  $n$ -ary action symbol  $A$  and the  $\mathcal{L}_{\perp}^{\text{fo}}$  formula  $\phi(\vec{x})$  where  $\vec{x}$  are the only  $n$  variable symbols that appear free in  $\phi$ , we can have a control-rule “execute  $A(\vec{x})$  at  $\Sigma_i$  if  $(\Sigma, i) \models \phi(\vec{x})$ ”. For the logistics domain specifically we can have:  $A(\vec{x}) = \text{Unload}(t)$  and  $\phi(\vec{x}) = \exists : \text{City}(c). \exists : \text{Package}(p). (\text{TIn}(t, c) \wedge \text{OnT}(p, t) \wedge \text{GBIn}(p, c))$ . As a control-rule, this prescribes an action that unloads a package from a truck, if the truck is located in the city where the package is supposed to be delivered. When there is no  $\vec{x}$  so that  $(\Sigma, i) \models \phi(\vec{x})$ , we say for event  $\Sigma(i)$  the control-rule does not prescribe an action.

We develop two separate techniques for generating control rules. The first was explored for the Markovian setting in (Gretton & Thiébaux 2004). In our case it corresponds to using first-order regression, a computationally cheap mechanism for reasoning about a  $\mathcal{L}_{\perp}^{\text{fo}}$  domain definition, to compute control rules which are sufficient to build an  $n$ -state-to-go optimal policy. The second is based on (Fern, Yoon, & Givan 2006), which again only deals with the Markovian setting. We develop a method of generating taxonomic control rules according to the grammar of a temporal taxonomic language bias. In this case we give ROPG a few such control rules which seem to make good action prescriptions in small NMRDPs.

## Generating $\mathcal{L}_{\perp}^{\text{fo}}$ Control Rules

Like (Gretton & Thiébaux 2004), we use first-order regression to compute control rules from a  $\mathcal{L}_{\perp}^{\text{fo}}$  domain definition. The regression of a behavioural formula  $f$  through an action  $a$  is a behavioural formula that holds before  $a$  is executed iff  $f$  holds after the execution. Regression requires that for each fluent  $p \in f$ ,  $\text{SSA}(p)$  is given. In detail, consider a behavioural formula  $f$  and a deterministic action term  $A(\vec{y})$ . Behavioural formula  $f$  holds after we execute  $A(\vec{y})$  iff  $\text{poss}(A(\vec{y})) \wedge \text{regr}(f, A(\vec{y}))$  holds, where  $\text{regr}$  is defined in Algorithm 1.

### Algorithm 1 $\mathcal{L}_{\perp}^{\text{fo}}$ Regression

---

```

regr( $p, A(\vec{y})$ ) =  $\text{SSA}(p)$  with every occurrence of
   $\text{act} = A'(\vec{x})$  replaced with  $\text{EQUAL}(\vec{x}, \vec{y})$  if  $A'$  and  $A$ 
  are the same symbol, and otherwise replaced with  $\perp$ 
regr( $f_1 \vee f_2, A(\vec{y})$ ) =  $\text{regr}(f_1, A(\vec{y})) \vee \text{regr}(f_2, A(\vec{y}))$ 
regr( $\neg f, A(\vec{y})$ ) =  $\neg \text{regr}(f, A(\vec{y}))$ 
regr( $\exists x.f, A(\vec{y})$ ) =  $\exists x.\text{regr}(f, A(\vec{y}))$ 
regr( $\ominus f, A(\vec{y})$ ) =  $f$ 
regr( $\diamond f, A(\vec{y})$ ) =  $f \vee \ominus \diamond f$ 
regr( $\boxminus f, A(\vec{y})$ ) =  $(f \wedge \ominus \boxminus f) \vee (f \wedge \neg \ominus \top)$ 
regr( $f_1 \mathbf{S} f_2, A(\vec{y})$ ) =  $f_2 \vee (f_1 \wedge \ominus (f_1 \mathbf{S} f_2))$ 

```

---

Now, consider the set  $\{\phi_j^0\}$  consisting of the behavioural formulae in the reward specification of the domain at hand. We can compute the set of formulae  $\{\phi_j^1\}$  from  $\{\phi_j^0\}$  by regressing the  $\phi_j^0$  over all deterministic actions with existentially quantified arguments. That is, each  $\phi_j^1$  is of the form  $\exists \vec{x}.\text{poss}(D(\vec{x})) \wedge \text{regr}(\phi_j^0, D(\vec{x}))$  for some  $i$ . Any event  $\Sigma$  that is one action application from reward models  $\bigvee_j \phi_j^1$ . More usefully, a behavioural formula characterising pre-action events for each stochastic action, can be formed by considering disjunctions over formulae from  $\{\phi_j^1\}$ . Similarly we can capture longer trajectories facilitated by stochastic actions by computing  $\{\phi_j^n\}$  for  $n$  larger than 1. The set of behavioural formulae sufficient to encapsulate such trajectories are members of the set:

$$F^n \equiv \bigcup_{i=0 \dots n} \{\phi_j^i\}$$

Thus, using regression we can obtain a set of behavioural formula which are sufficient to induce a value-base classification of events that are  $n$  steps from reward. Moreover, for  $i > 0$ , elements  $\phi_j^i \in F^n$  along with information about the deterministic action symbol  $D$  from which it is generated using regression acts as the specification for a control-rule as follows: Take  $A$  to be the unique stochastic action for which  $D$  is a choice. Alone,  $\phi_j^i$  is of the form  $\exists \vec{x}.\text{poss}(D(\vec{x})) \wedge \text{regr}(\phi_j^{i-1}, D(\vec{x}))$  for some  $\phi_j^{i-1} \in F^n$ . We say that as a control-rule  $\phi_j^i$  prescribes  $A(\vec{x})$  at state  $\Sigma_i$  so that  $(\Sigma, i) \models \text{poss}(D(\vec{x})) \wedge \text{regr}(\phi_j^{i-1}, D(\vec{x}))$ . Where there are multiple  $\vec{x}$  that satisfy this condition, in practice we resolve to choose from these deterministically – i.e., If a control-rule prescribes an action at  $\Sigma(i)$ , this will always be that rule’s prescription at  $\Sigma(i)$ .

## Generating Taxonomic Control Rules

control rules can be specified in, and generated according to, a concept or taxonomic language. This was the case in (Martin & Geffner 2000) and (Fern, Yoon, & Givan 2006) where, for examples, useful general policies over taxonomic control rules were obtained for blocks-world and logistics. Here, we extend the *taxonomic syntax* from (Fern, Yoon, & Givan 2006) that we denote  $\mathcal{L}^{\text{ts}}$ , with temporal operators to obtain a language  $\mathcal{L}^{\text{ts}}$  suitable for specifying control rules for NMRDP.

Expressions in  $\mathcal{L}^{\text{ts}}$  are constructed over primitive concepts (unary predicates, e.g. `Block` in blocks-world), denoted  $C_p$ , and primitive relations (binary predicate), denoted  $R_p$ . The interpretation of a primitive concept is the corresponding predicate's extension at a given state. For example, the interpretation of the concept `Block` at a block-world state is simply the set of all the blocks in that world. Similarly the interpretation of a role at a state is the corresponding binary predicates extension at a state – i.e. A set of pairs of objects. The expressions  $\mathcal{L}^{\text{ts}}$  we will build from these primitives are given by the following grammar. Again, we use the PLTL temporal modalities  $\ominus$ ,  $\diamond$ ,  $\boxplus$  and **S** yielding the following grammar:

$$\begin{aligned} R &::= R_p | \mathbf{Id} | R^{-1} | R \cap R | R^* | \ominus R | RSR | \diamond R | \boxplus R \\ C &::= C_p | \mathbf{a\text{-}thing} | \neg C | \ominus C | CSC | \boxplus C | \diamond C | RC | C \cap C \end{aligned}$$

In order to give formal semantics for  $\mathcal{L}^{\text{ts}}$  we need a few more notations. Take  $\mathcal{O}$  as a set of domain objects, for example the set of blocks and the table in blocks-worlds, and the set of cities, trucks and packages in the logistics. We write  $o$  for an element in  $\mathcal{O}$ . Whereas  $\mathcal{L}^{\text{ts}}$  expressions are interpreted at an MDP state<sup>3</sup>,  $\mathcal{L}^{\text{ts}}$  expressions are interpreted at the  $i$ 'th state of a history  $\Gamma$  (i.e. a sequence of states). This is achieved using a function  $I : (\mathcal{S}^* \times \mathbb{N} \times \mathcal{L}^{\text{ts}}) \rightarrow (2^{\mathcal{O}} \cup 2^{\mathcal{O}^2})$ , defined below. We write  $C(o) \in s$  and  $R(o, o') \in s$ , if the unary proposition  $C(o)$  and binary proposition  $R(o, o')$  label the state  $s$  respectively. Thus, we obtain the interpretation of the primitive concept `Block` in blocks-world at the  $i$ 'th state of history  $\Gamma$  as the set  $I(\Gamma, i, \text{Block}) := \{o | o \in \mathcal{O}, \text{Block}(o) \in \Gamma_i\}$ . Finally in the semantics below,  $'^*$  on the left-hand side of an assignment  $' := '$  is for transitive closure, *otherwise* it is a Kleene star. We omit the semantics for interpreting sentences with a temporal operator applied to a role because they follow in the obvious way.

$$\begin{aligned} I(\Gamma, i, \mathbf{a\text{-}thing}) &:= \mathcal{O} \\ I(\Gamma, i, \mathbf{Id}) &:= \{(o, o) | o \in \mathcal{O}\} \\ I(\Gamma, i, C) &:= \{o | o \in \mathcal{O}, C(o) \in \Gamma_i\} \\ I(\Gamma, i, R) &:= \{(o, o') | o, o' \in \mathcal{O}, R(o, o') \in \Gamma_i\} \\ I(\Gamma, i, \neg C) &:= \{o | o \notin I(\Gamma, i, C)\} \\ I(\Gamma, i, RC) &:= \{o | \exists o' \in I(\Gamma, i, C), \\ &\quad (o', o) \in I(\Gamma, i, R)\} \\ I(\Gamma, i, C_a \cap C_b) &:= I(\Gamma, i, C_a) \cap I(\Gamma, i, C_b) \\ I(\Gamma, i, R^{-1}) &:= \{(o, o') | (o', o) \in I(\Gamma, i, R)\} \\ I(\Gamma, i, R_a \cap R_b) &:= I(\Gamma, i, R_a) \cap I(\Gamma, i, R_b) \\ I(\Gamma, i, R^*) &:= I(\Gamma, i, \mathbf{Id}) \cup \{(o, o') | \exists k \in \mathbb{N}, \\ &\quad \psi \in \mathcal{O}^*, \psi_0 = o, \psi_k = o', \\ &\quad \forall i \in \mathbb{N} \text{ s.t. } 0 \leq i < k \\ &\quad (\psi_i, \psi_{i+1}) \in I(\Gamma, i, R)\} \end{aligned}$$

$$\begin{aligned} I(\Gamma, i, \diamond C) &:= \{o | \exists j \leq i. o \in I(\Gamma, j, C)\} \\ I(\Gamma, i, \boxplus C) &:= \{o | \forall j \leq i. o \in I(\Gamma, j, C)\} \\ I(\Gamma, i, \ominus C) &:= \{o | o \in I(\Gamma, i-1, C)\} \\ I(\Gamma, i, C_a SC_b) &:= \{o | \exists j \leq i \text{ s.t. } o \in I(\Gamma, j, C_b), \forall j < k \leq i. \\ &\quad o \in I(\Gamma, k, C_a)\} \end{aligned}$$

<sup>3</sup>i.e., the *domain of discourse* is an MDP state

**Taxonomic Control Rules** The length of a sentence is the number of operators appearing in it. In order to generate taxonomic control rules we start by computing a set of sentences in  $\mathcal{L}^{\text{ts}}$  (or  $\mathcal{L}^{\text{ts}}$ ) up to a given limiting length. Computation is bottom up, starting with zero length sentences – i.e. Primitive domain concepts and roles. Sentences of length  $1, 2, \dots, i$ , are used along with operators from  $\mathcal{L}^{\text{ts}}$  to build concept and role expressions up to length  $2i + 1$ . In practice we avoid generating logically equivalent sentences via simple syntactic checks and by interpreting sentences over a collection of histories taken from small to medium sized problems. We build taxonomic control rules by arbitrarily choosing, for an  $n$ -ary action,  $n$  generated concept-expressions so that the  $i$ 'th action argument is to be taken from the interpretation of the  $i$ 'th concept-expression. When a control-rule is executed at a history, the  $i$ 'th argument is chosen arbitrarily from the interpretation of the  $i$ 'th concept-expression at that history. For example in blocks-world, we could have a control-rule that places blocks on the table if they have not been there before

$$\begin{aligned} a_1 \in I(?\Gamma, ?i, \neg \diamond \text{On}^{-1} \text{table}), \\ a_2 \in I(?\Gamma, ?i, \text{table}) \implies \text{move}(a_1, a_2) \end{aligned}$$

Executing this at the  $i$ 'th state of some  $\Gamma$ , we input  $i$  for  $?i$  and  $\Gamma$  for  $??\Gamma$  in the above expression, and thus can obtain some  $a_1$  and  $a_2$ . We say this control rule then prescribes action  $\text{Move}(a_1, a_2)$  at  $\Gamma_i$ . Unlike (Martin & Geffner 2000), we treat control rules deterministically – If they apply a particular action at  $\Gamma(i)$ , this will always be the control's action at  $\Gamma(i)$ .

In order to collect a small set of taxonomic control rules to give to the reinforcement learner, we generate a small set that seem to make useful prescriptions according to the optimal policies and value functions computed by the propositional planner NMRDPP (Thiébaux *et al.* 2006). There is insufficient space to give complete details here.

## ROPG

Here we model learning to act in a domain of NMRDPs given a set of control rules as learning to act in a partially observable Markov decision process (POMDP) with a deterministic observation model. The POMDP is a six-tuple  $\langle \mathcal{S}, \mathbb{A}, Pr, R, \mathbb{O}, v \rangle$  where  $\mathcal{S}$  is the set of domain states and  $\mathbb{A}$  is a small set of control rules.  $P_S$  is the star-state distribution, and we denote  $P_s$  the probability of starting in state  $s$ .<sup>4</sup> Where  $s, s' \in \mathcal{S}$  and  $\varrho \in \mathbb{A}$ ,  $Pr(s, \varrho, s')$  is the unknown probability of a transition from state  $s$  to  $s'$  given control-rule  $\varrho$  is executed at state  $s$ .  $R$  is a bounded real-valued reward function  $R : \mathcal{S}^* \rightarrow \mathbb{R}$ .<sup>5</sup>  $\mathbb{O}$  is the set of observations, each observation corresponding to an element in the powerset of  $\mathbb{A}$ . Intuitively, an observation is the set of applicable control rules. In a typical POMDP, for each  $s \in \mathcal{S}$ , an observation  $o \in \mathbb{O}$  is generated independently according

<sup>4</sup>For this paper we assume starting states are drawn uniformly from a small set.

<sup>5</sup>Because NMRDPs can be expanded into equivalent MDPs (Thiébaux *et al.* 2006), no difficulties arise using  $R : \mathcal{S}^* \rightarrow \mathbb{R}$  instead of  $R : \mathcal{S} \rightarrow \mathbb{R}$ .

to some probability distribution  $v(s)$ . We denote  $v_o(s)$  the probability of getting observation  $o$  in state  $s$ . In our case the observation at a state is the set of control rules in  $\mathbb{A}$  that prescribe an action, and is thus deterministically generated.

Because we cannot generally compute an optimal policy for our POMDP, we consider a technique for solving POMDPs that concentrates policy search on parameterised reactive “memoryless” policies  $\mu : (\mathbb{O} \times \theta) \rightarrow P_{\mathbb{A}}$ .  $\theta \in \mathbb{R}^{|\mathbb{A}|}$  is an  $|\mathbb{A}|$  length vector of real parameters/weights and  $P_{\mathbb{A}}$  is a distribution over the POMDP actions.

With each control-rule  $\varrho_i$  in  $\mathbb{A}$ , we associate a single weight  $\theta_i$  which is used to compute the probability that  $\varrho_i$  is executed given an observation  $o$ . The probability  $\mu_{\varrho_i}(o, \theta)$  that rule  $\varrho_i$  with weight  $\theta_i$  is chosen to prescribe an action for observation  $o$  is given by a discrete Boltzmann distribution

$$\mu_{\varrho_i}(o, \theta) = \kappa(\varrho_i, o) \frac{e^{\theta_i}}{\sum_j \kappa(\varrho_j, o) e^{\theta_j}}$$

where  $\kappa$  is 1 or 0 depending on whether  $\varrho_i$  is in  $o$  or not. Taking  $0/0 = 0$ ,  $\mu(o, \theta)$  is sometimes called the *soft-max* distribution. In our case, finding the best soft-max policy equates to finding  $\theta \in \mathbb{R}^{|\mathbb{A}|}$  that maximise the expected discounted reward over an infinite horizon of acting according to  $\mu(\theta)$  in the domain. We denote the value of a policy parameterised by  $\theta$  as  $\eta(\theta)$ . There will usually be no obvious way to compute  $\eta(\theta)$  for arbitrary starting state distribution  $P_S$ , however a good estimate  $\hat{\nabla}\eta$  of its *gradient* with respect to parameters  $\theta$ ,  $\nabla\eta$ , can be computed by executing  $\mu(\theta)$  in problems drawn according to  $P_S$ . The online REINFORCE (Williams 1992) style gradient ascent optimisation strategy of Algorithm 2,<sup>6</sup> can indeed find a local maximum of the utility function  $\eta(\theta)$  by computing a sequence  $\theta^0, \dots, \theta^n$  so that for a small step-size  $\alpha$ .

$$\theta^{i+1} = \theta^i + \alpha \hat{\nabla}\eta(\theta^i)$$

## Experimental Evaluation

In this section we evaluate ROPG in Markovian, non-Markovian, stochastic and deterministic domains, where control rules available to the learner are generated according to  $\mathcal{L}_{\pm}^{\text{ts}}$ ,  $\mathcal{L}^{\text{ts}}$ , or regression. We include  $\mathcal{L}_{\pm}^{\text{ts}}$  here even for Markovian domains because we want to see the advantage of providing the learner with temporal features. Questions we address include: (1) Can ROPG obtain a good general policy in a set of small to medium sized instances drawn from a target domain? (2) What are the costs and benefits of using control rules drawn from the different mechanisms we have discussed? (3) Can ROPG achieve good generalisation? (4) What are the benefits of including temporal features even when the domain is Markovian? To this end, we have implemented our approach in C++ and will email this on request. This includes functionality for generating control rules according to  $\mathcal{L}_{\pm}^{\text{ts}}$ ,  $\mathcal{L}^{\text{ts}}$ , and regression. It also includes the domain descriptions and problem instances from [this paper](#).

<sup>6</sup>In ROPG, setting  $\gamma < 1$  is a standard strategy for addressing the temporal credit assignment problem (Baxter & Bartlett 2001). If this is inappropriate,  $\gamma = 1$  is admissible in our setting.

---

### Algorithm 2 Relational Online Policy Gradient (ROPG)

---

- 1: Given:
    - Initial parameters  $\theta \in \mathbb{R}^n$  and policy  $\mu(\theta)$ .
    - Distribution over starting states  $P_S$
    - Reward and trace discount factors  $0 < \beta, \gamma < 1$
    - Step-size parameter  $\alpha$
  - 2: **repeat**
  - 3:   Sample an NMRDP starting state  $s_0$  from  $P_S$
  - 4:   Generate a set  $\mathcal{T}_\mu$  of observation( $o$ )-action( $\varrho$ ) trajectories that yield reward  $r$ , starting at the observation deterministically generated from  $s_0$ , and using the current policy  $\mu(\theta)$
  - 5:    $\hat{\nabla}\eta_0 \leftarrow 0$
  - 6:   **for** Each  $T \in \mathcal{T}_\mu$  (length  $|T|$ ) with reward  $r_T$  **do**
  - 7:      $z_0 \leftarrow 0$
  - 8:     **for** Each  $o_j, \varrho_j \in T$  chronologically **do**
  - 9:        $z_{j+1} \leftarrow \gamma z_j + \frac{\nabla \mu_{\varrho_j}(o_j)}{\mu_{\varrho_j}(o_j)}$
  - 10:    **end for**
  - 11:     $\hat{\nabla}\eta_{i+1} \leftarrow \hat{\nabla}\eta_i + \frac{1}{i+1} [\beta^{|T|} \cdot r_T \cdot z_{|T|} - \hat{\nabla}\eta_i]$
  - 12:    **end for**
  - 13:     $\theta \leftarrow \theta + \alpha \hat{\nabla}\eta|_{\mathcal{T}_\mu}$
  - 14: **until**  $\hat{\nabla}\eta|_{\mathcal{T}_\mu} = 0$
- 

## Domains and Problems

We experiment with 6 domains including four variants of miconic elevator scheduling systems (Koehler & Schuster 2000) distinguished by whether they are stochastic ‘*stc-mic*’ or deterministic ‘*det-mic*’, and whether they feature complex non-Markovian ‘*mic-rea*’ or simple ‘*mic-sim*’ reward schemes. The stochastic and non-Markovian elements of miconic are from (Thiébaux *et al.* 2006). We also experiment with deterministic Markovian versions of the classic benchmarks logistics from (Boutilier, Reiter, & Price 2001) and blocks-world (Slaney & Thiébaux 2001) with the usual goal achievement and maintenance reward schemes.

Our experiments examine the behaviour of ROPG in a small group of *training problems* from each domain. The problems are sufficiently small for the optimal planner NMRDPP to solve. This permits a comparison of the policy learnt by ROPG with the optimal. With this objective we sample 11 problems from each of the miconic variants. For each miconic domain, in 6 of the problems there are 2 people and a single elevator, with the number of floors ranging from 2, ..., 7. For the other 5 problems we have the number of people and floors ranging simultaneously from 3, ..., 7. For logistics we sample 13 random problems each with 2 trucks. With the number of boxes ranging from 2, ..., 4, we consider problems with 2, ..., 5 cities. We also have a problem with 5 boxes and 2 cities. We experiment with two different sets of problems from the blocks-world. The first, called blocks-world-10, is a set of 10 problems: with 2 problems with 2, 3 and so on up to 6 block worlds. Where  $n$  is the number of blocks in a world, for each  $n \in 2..6$  we include one randomly generated problem in which the goal is distinct from the initial state. For each size  $n$  we also include a problem that requires the planner to reverse an  $n$  block stack on the table. The second set, called blocks-world-15, is a set of 15

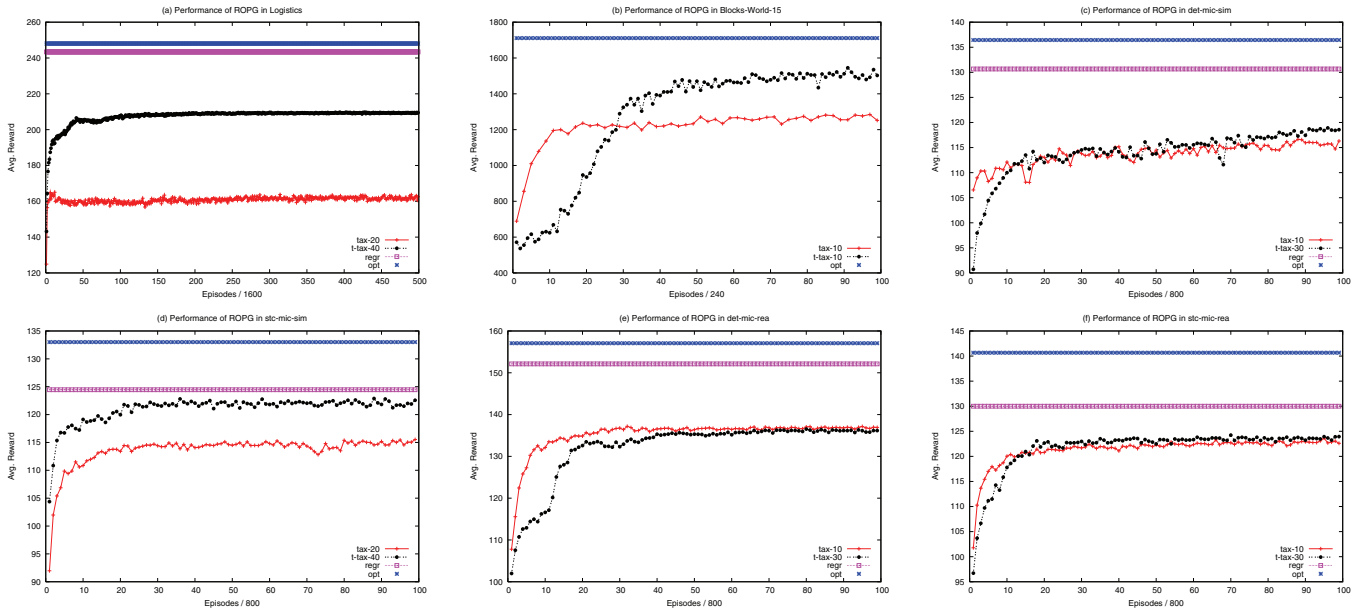


Figure 2: For  $\mathcal{L}_{\leftarrow}^{\text{t},\text{s}}$  ( $t\text{-tax-}N$ ) and  $\mathcal{L}^{\text{t},\text{s}}$  ( $\text{tax-}N$ ) control rules in  $\mathbb{A}$ , we report convergence of average discounted reward experienced as ROPG undertakes episodes in training problems.  $N$  reports the size of  $\mathbb{A}$ . (*opt*) the optimal performance according to NMRDPP, and (*regr*) the performance of the best policy obtained by ROPG with  $\mathbb{A}$  comprising control rules generated by regression. We omit *regr* in the case of blocks-world because control rules based on regression do not work in that domain [Gretton and Thiébaux, 2004].

distinct random problems. In this set we include 5 problems with 3, 4 and 5 blocks respectively.

## Results and Discussion

Our experiments were conducted on an AMD Athlon(tm) 64 Processor 3200+ machine with 2 gigabytes of RAM. Both the trace and reward discount factors were set to  $\gamma, \beta = 0.95$ . The step-size  $\alpha$  was  $10^{-3}$  for learning with control rules based on regression, and  $10^{-4}$  in all domains except in logistics, where we used  $\alpha = 10^{-5}$ . For each problem set,  $P_S$  is configured so that problems are drawn uniformly at random by the learner.

Performance results are summarised in Figure 2.<sup>7</sup> These show that if we measure policy performance in terms of expected discounted cumulative reward (*utility*), sampling uniformly in training problems, the best is achieved using control rules generated via regression, the next best is obtained with  $\mathcal{L}_{\leftarrow}^{\text{t},\text{s}}$  control rules, and then  $\mathcal{L}^{\text{t},\text{s}}$  rules. The only exception occurs for *det-mic-rea* where the  $\mathcal{L}^{\text{t},\text{s}}$  policy is slightly better than the  $\mathcal{L}_{\leftarrow}^{\text{t},\text{s}}$  policy. Convergence is not always to the optimal when control rules are generated according to regression because (1) ROPG only guarantees local convergence, and (2) more significantly we limit the number of regression steps in generating control rules, thus these rules do not always facilitate optimality. Not shown in Figure 2, is performance in terms of how much computation time it takes the policies to execute. We find policies con-

<sup>7</sup>In that figure we have not shown convergence in the case where rules are generated by regression because that case has much faster convergence than the case where rules are generated according to a concept language.

Language	Problem	Time	Utility
$\mathcal{L}_{\leftarrow}^{\text{t},\text{s}}$	<i>det-mic-sim-30</i>	1885m57	327.231
$\mathcal{L}^{\text{t},\text{s}}$	<i>det-mic-sim-30</i>	971m31	99.3418
$\mathcal{L}_{\leftarrow}^{\text{t},\text{s}}$	<i>det-mic-rea-30</i>	1136m4	305.538
$\mathcal{L}^{\text{t},\text{s}}$	<i>det-mic-rea-30</i>	798m51	340.22
$\mathcal{L}_{\leftarrow}^{\text{t},\text{s}}$	<i>stc-mic-sim-30</i>	2208m45	246.922
$\mathcal{L}^{\text{t},\text{s}}$	<i>stc-mic-sim-30</i>	758m22	116.114
$\mathcal{L}_{\leftarrow}^{\text{t},\text{s}}$	<i>stc-mic-rea-30</i>	1089m28	554.717
$\mathcal{L}^{\text{t},\text{s}}$	<i>stc-mic-rea-30</i>	896m59	399.742

Table 1: Runtime and utility experienced using general policies on large test problems. Policies are executed for 1000 episodes, where each episode lasts for 100 actions.

structed using rules generated via regression are by far the slowest to execute. For example, in training problems from *det-mic-rea*, to undertake  $10^3$  episodes of ROPG using regressed control rules takes 1026 minutes while using  $\mathcal{L}_{\leftarrow}^{\text{t},\text{s}}$  rules only takes around 110 minutes. This occurs because of the operator-length (and relative complexity) of sentences used to describe  $\mathcal{L}_{\leftarrow}^{\text{t},\text{s}}$  control rules, and not because the number of control rules is large. We also find that it is between 1.4 and 2 times slower on average to undertake episodes of learning with  $\mathcal{L}_{\leftarrow}^{\text{t},\text{s}}$  control rules than with their  $\mathcal{L}^{\text{t},\text{s}}$  counterparts. This is because, (1) the learner was given more rules in the  $\mathcal{L}_{\leftarrow}^{\text{t},\text{s}}$  case, and (2) the interpretation of temporal rules can be relatively expensive.

To determine how well policies ROPG learns in training problems generalise, we evaluated the blocks-world policies in problems with 10, 15 and 20 blocks, the miconic policies in problems with 15, 20 and 30 passengers, and the logistics policies in problems with 15 packages, trucks and cities.

The results we obtained for miconic test problems are summarised in Table 1.<sup>8</sup> Not shown in that table is the fact that non-Markovian rewards were being sought out and achieved by the learnt policies. We found that policies based on regression were very (sometimes prohibitively) expensive in computation time to execute, relative to taxonomic policies. The latter generalised well for all domains except blocks-world. The taxonomic policies computed for the blocks-world-10 problems only generalised reliably where a stack of blocks had to be reversed on the table. In a similar vein, the generalisation achieved by the taxonomic policies computed in the block-world-15 problems was unreliable. On a final positive note, the temporal blocks-world policies completed episodes more quickly than their atemporal counterparts in large problems. Not surprisingly, factors which we find can negatively effect generalisation include bias in training problems, and the overall quality of control rules available to ROPG.

### Concluding Remarks

We developed ROPG, an approach to unsupervised planning-as-learning for generalisation in non-Markovian domains. This operates given a set of control rules which we proposed be computed automatically. To this end we created a domain definition language based on  $\mathcal{L}_{\perp}^{fo}$  for axiomatising non-Markovian domains and extended first-order decision-theoretic regression to this setting for the purpose of automatically generating control rules along the lines of (Gretton & Thiébaux 2004). We also extended the taxonomic syntax from (Fern, Yoon, & Givan 2006) to accommodate temporal concepts and relations for the purpose of automatically generating taxonomic control rules suited to NMRDPs and useful in MDPs. We evaluate our approach in a number of planning benchmarks and find that it is able to learn good general policies. ROPG is attractive and unique because it can both (1) generalise from experience without recourse to state values, and (2) policy improvement occurs at a first-order, resp. propositional, level. A pressing item for future work is to investigate ways of mitigating slow convergence of ROPG. Also, future work should try to address the cost of model checking for control rules, which is a significant bottle neck of our approach.

### Acknowledgements

Thanks to Doug Aberdeen and Sylvie Thiébaux for useful discussions. We would also like to acknowledge the support of NICTA. NICTA is funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council.

### References

Bacchus, F.; Boutilier, C.; and Grove, A. 1996. Rewarding behaviors. In *AAAI-96*.

Baxter, J., and Bartlett, P. L. 2001. Infinite-horizon policy-gradient estimation. *J. Artif. Intell. Res. (JAIR)* 15:319–350.

<sup>8</sup>The runtime results, while competitive with a state-based planner that would plan from scratch for each episode, are slightly misleading because our  $\mathcal{L}_{\perp}^{fo}$  problem simulator was not implemented to be as efficient as possible.

Boutilier, C.; Reiter, R.; and Price, B. 2001. Symbolic dynamic programming for first-order MDPs. In *IJCAI-01*.

C. Guestrin; D. Koller; C. Gearhart; and N. Kanodia. 2003. Generalizing plans to new environments in relational MDPs.

Fern, A.; Yoon, S.; and Givan, R. 2006. Approximate policy iteration with a policy language bias: Solving relational Markov decision processes. *J. Artif. Intell. Res. (JAIR)* 25.

Gretton, C., and Thiébaux, S. 2004. Exploiting first-order regression in inductive policy selection. In *UAI*.

Hernandez-Gardiol, N., and Kaelbling, L. P. 2003. Envelope-based planning in relational mdps. In *NIPS-17*.

K. Kersting; M. V. Otterlo; and L. D. Raedt. 2004. Bellman goes relational. In *ICML*, 59.

Karabaev, E., and Skvortsova, O. 2005. A Heuristic Search Algorithm for Solving First-Order MDPs. In *UAI*.

Kersting, K., and Raedt, L. D. 2004. Logical markov decision programs and the convergence of logical td(lambda). In *ILP*, 180–197.

Khardon, R. 1999. Learning action strategies for planning domains. *Artificial Intelligence* 113(1-2):125–148.

Koehler, J., and Schuster, K. 2000. Elevator control as a planning problem. In *AIPS*.

Littman, M. L.; Goldsmith, J.; and Mundhenk, M. 1998. The computational complexity of probabilistic planning. *J. Artif. Intell. Res. (JAIR)* 9:1–36.

Martin, M., and Geffner, H. 2000. Learning generalized policies in planning using concept languages. In *KR*, 667–677.

Reiter, R. 2001. *Knowledge in action : logical foundations for specifying and implementing dynamical systems*. Cambridge, Mass.: MIT Press.

Sanner, S., and Boutilier, C. 2005. Approximate linear programming for first-order mdps. In *UAI*.

Slaney, J., and Thiébaux, S. 2001. Blocks world revisited. *Artificial Intelligence* 125:119–153.

Thiébaux, S., and Cordie, M. 2001. Supply restoration in power distribution systems – a benchmark for planning under uncertainty. In *EPC-01*.

Thiébaux, S.; Gretton, C.; Slaney, J.; Price, D.; and Kabanza, F. 2006. Decision-theoretic planning with non-markovian rewards. *J. Artif. Intell. Res. (JAIR)* 25:17–74.

Wang, C.; Joshi, S.; and Khardon, R. 2007. First order decision diagrams for relational MDPs. In *IJCAI-07*.

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

Yoon, S. W.; Fern, A.; and Givan, R. 2002. Inductive policy selection for first-order mdps. In *UAI*, 569–576.

Younes, H. L. S.; Littman, M.; Weissmann, D.; and Asmuth, J. 2005. The first probabilistic track of the IPC. In *J. Artif. Intell. Res. (JAIR)*, volume 24, 851–887.