

# Simulation Methods for Uncertain Decision-Theoretic Planning

Douglas Aberdeen and Olivier Buffet

National ICT Australia

Australian National University

Canberra, Australia

{douglas.aberdeen,olivier.buffet}@nicta.com.au

## Abstract

Experience based reinforcement learning (RL) systems are known to be useful for dealing with domains that are *a priori* unknown. We believe that experience based methods may also be useful when the model is uncertain (or even completely known). In this case experience is gained by *simulating* the uncertain model. This paper explores a simple way to allow experience based RL systems to cope with uncertainty in a model. The particular form of RL we consider is a policy-gradient method. The particular domains we attempt to optimise in are from temporal decision-theoretic planning. Our previous experience with military planning problems indicates that a human specified model of the planning problem is often inaccurate, especially when humans specify probabilities, thus planners that take into account this uncertainty are very useful. Despite our focus on policy-gradient RL for planning, our simple (but approximate) solution for dealing with uncertainty in the model can be applied to any simulation based RL method, such as Q-learning or SARSA. Our attempt to solve decision-theoretic planning problems with a policy-gradient approach is novel in itself, making up another contribution of this paper.

## 1 Introduction

If the true model of a Markov decision problem (MDP) is hidden we must use algorithms that train agents by *interacting* with the MDP. This is done by experiencing trajectories through the state space and forming either an explicit model (transition matrix) or an implicit model (value function or policy) of the system. These Monte-Carloesque methods can be beneficial even when the true model is completely known, especially if the model is too complex to work with directly. E.g., the state space might be too large to enumerate, or it might be continuous. In this case the model is only used in simulating the system, generating state space trajectories that the agent uses to optimise its behaviour. Another argument for simulation based optimisation is the ease of creating a simulator compared to a set of stochastic transition matrices.

This is especially true if aspects of the system are unknown or approximated.

This paper explores the idea of using an *uncertain* model to simulate trajectories, allowing an agent to directly optimise its policy in a way that minimises the impact, or risk, associated with the uncertainty in the model. Moreover, this can be achieved in highly complex domains.

The problems we consider come from temporal decision-theoretic planning, where methods that enumerate any part of the state space fail to scale to interesting problems. The model is provided in the form of a set of tasks the planner can choose from. Each task has a pre-defined duration and has probabilistic outcomes that set multiple state variables to true or false. The goal of the planner is to select actions, and schedule them concurrently, to achieve the desired *goal state* values of the state variables. Resources constrain which tasks can be run in combination, and resources are consumed as tasks end. This is a very general expression of the planning problem and only a few probabilistic planners are emerging that can operate in this setting. They can be used to optimise plans in a wide variety of situations, such as Mars rover planning [Mausam and Weld, 2005], military operations planning [Aberdeen *et al.*, 2004], or building site planning. Probabilities might arise from modelling variable battery strength, an opponent's actions, or weather. The outcome probabilities are often estimated from finite data, or guessed by human experts. Thus, the probabilities are subject to some uncertainty.

The contribution of this paper is two fold. Firstly, we describe the factored policy gradient (FPG) Planner: a novel approach to temporal decision-theoretic planning that allows very large domains to be *approximately* optimised. We achieve this by: (1) factoring the policy into simple independent policies for starting each task; (2) using a local optimisation method instead of trying to find a globally optimal solution; (3) using algorithms with memory use that scales linearly with the number of tasks, state variables, and resources, not with the state space.

The second contribution is to demonstrate how uncertainty over the probabilities described in a model can be incorporated into a simulation based optimisation. We assume that probabilities of task outcomes lie in intervals between  $[0, 1]$ . The width of the interval can be computed based on the quality of the data used, or based on how confident the human guess was. The goal is to find a policy that minimises the

risk, or variance, associated with enacting the policy over the range of models implied by the uncertainty. I.e., the policy that still performs relatively well even in the worst case scenario. The key idea is simply to simulate state space trajectories using the most *pessimistic* model. The most pessimistic model might generally be as difficult to compute as the policy, however, we show empirically that a local approximation to the pessimistic model might be sufficient. We can also compute policies based on the most *optimistic* model, to examine the differences in policy or determine how much our uncertainty could be effecting agent performance.

We start by describing background work in temporal probabilistic planning and interval methods for Markov decision problems (MDPs). Section 3 describes MDPs for planning. Section 4 describes the factored policy agents and the estimator we use to compute the gradient of the objective function. Section 5 describes preliminary experiments.

## 2 Background

Previous probabilistic temporal planners include CPTP [Mausam and Weld, 2005], Prottle [Little, 2004], and a military operations planner [Aberdeen *et al.*, 2004]. All these algorithms use some form of dynamic programming (either RTDP [Barto *et al.*, 1995] or AO\*) to associate values with each state/action pair. However, this requires that values be stored for each encountered state. Even though these algorithms do not enumerate the entire state space their ability to scale is limited by memory size. Even problems with a few tasks and state variables can produce millions of relevant states. Another probabilistic temporal planner is Tempastic [Younes and Simmons, 2004], which uses the generate, test, and debug planning paradigm. This method may suffer in domains that are highly non-deterministic.

Our FPG-Planner performs gradient ascent in the space of parameters of the factored policies (or policy agents). The policy agents can be any differentiable function approximator. We show that maximising a simple reward function naturally minimises plan durations and maximises the probability of reaching the goal. Gradients are estimated by simulating trajectories through the planning state space and calculating small contributions to the gradient at each step [Baxter *et al.*, 2001]. The FPG-Planner will be described in this paper, but is covered in more detail in Aberdeen [2005].

The use of intervals to describe uncertainty in MDPs was investigated by Givan *et al.* [2000], Hosaka *et al.* [2001], and Strehl and Littman [2004]. Our approach is most closely related to the approach of Buffet and Aberdeen [2005], who use uncertainty intervals to make RTDP robust. RTDP uses simulation to determine which state values should be updated, thus is similar in its use of simulation to help cope with large state spaces. The intervals are used to compute the most pessimistic transition probabilities given the *current* value estimates. Our work deliberately avoids storing values, thus cannot use them to approximate the worst model. Instead, we assume that the simulator can often select the probability in each interval on a state by state basis that will result in a pessimistic model being simulated. Specifically, in our planning domains we assume that each task has two outcomes: suc-

cess, which is helpful, and failure, which is harmful.

Actions in temporal planning consist of launching multiple tasks concurrently. The number of candidate actions available in a given state is the power set of the tasks that are eligible to start under the current state variables. That is, with  $N$  eligible tasks there are  $2^N$  possible actions. With only 10 eligible tasks we have 1,024 actions to choose from! Current planners try to explore this action space systematically, pruning actions that lead to low rewards (or equivalently, high costs).

A key contribution of the FPG-Planner is to deal with the explosion of the action space by replacing the single agent choosing from the power-set of tasks with a single simple agent for each task. The policy learnt by each agent is whether to start its associated task given its observation, independent of the decisions made by the other agents. This idea alone does not simplify the problem. Indeed, if the agents all receive perfect state information they could presumably predict the decision of the other agents and still act optimally. The significant reduction in complexity arises from two additional factors: (1) the agents are only provided enough information to make an approximate decision, not an optimal decision; (2) each agent is optimised locally.

## 3 POMDP Formulation of Planning

Our intention is to deliberately simplify the agents by restricting their access to state information. This requires us to explicitly consider partial observability. We now describe the partially observable MDP framework (POMDP), define the state space, our objective function, and the process for simulating the state space.

**Definition 1** *A finite partially observable Markov decision process consists of: a finite set of states  $s \in \mathcal{S}$ ; a finite set of actions  $\mathbf{a} \in \mathcal{A}$ ; probabilities  $\Pr[s'|s, \mathbf{a}] : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$  of making state transition  $s \rightarrow s'$  under action  $\mathbf{a}$ ; a reward for each state  $r(s) : \mathcal{S} \rightarrow \mathbb{R}$ ; a finite set of observation vectors  $\mathbf{o} \in \mathcal{O}$  seen by the agent in place of the complete state description; and probabilities  $\Pr[\mathbf{o}|s] : \mathbb{R}^{|\mathcal{O}|} \times \mathcal{S} \rightarrow [0, 1]$  of observing vector  $\mathbf{o}$  of dimension  $|\mathcal{O}|$ , given current state  $s$ .*

In addition, our specification of intervals around each task outcome probability induces intervals around each state transition probability  $\Pr[s'|s, \mathbf{a}]$ . As will be demonstrated later, our use of simulation on the level of planning tasks, instead of states, means we never need to explicitly compute  $\Pr[s'|s, \mathbf{a}]$  probabilities or their intervals.

*Goal states* are states where all the goal variables are satisfied. *Failure states* are states from which it is impossible to reach a goal state (usually because time or resources have run out). These two classes of state are combined to form the set of *reset* states that produce an immediate reset to the initial state  $s_0$ . A single trajectory through the state space consists of many individual trials that automatically reset to  $s_0$  each time a goal state or failure state is reached.

Policies are possibly stochastic [Singh *et al.*, 1994], mapping observation vectors to a probability over each action. Let  $N$  be the number of basic tasks available to the planner. In our setting an action  $\mathbf{a}$  is a binary vector of length  $N$ . An entry of 1 at index  $n$  means ‘Yes’ begin task  $n$ , and a 0 entry means ‘No’ do not start task  $n$ . The probability of actions is

written  $\Pr[\mathbf{a}|\mathbf{o}, \theta]$ , where conditioning on  $\theta$  reflects the fact that the policy is dictated by a set of  $p$  real valued parameters  $\theta \in \mathbb{R}^p$ . We show later how real valued parameters can control probability distributions over actions given observations, thus determining a policy. This paper assumes that all stochastic policies (i.e., all values for  $\theta$ ) reach reset states in finite time when executed from  $s_0$ . This is enforced by limiting the maximum duration of a plan. Because all policies reach a reset state, and by continuously resetting to the initial state, we ensure the underlying MDP is *ergodic*,<sup>1</sup> which is necessary for producing gradient estimates.

The aim of policy gradient algorithms is to find the set of parameters  $\theta$  that induce a policy to move from the initial state  $s_0$  to a reset state while maximising the long-term average reward. The long-term average reward is the average of all instantaneous rewards received over an infinite sample trajectory of the POMDP<sup>2</sup>

$$\eta(\theta) = \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=0}^{T-1} r(s_t).$$

In the context of planning, the instantaneous reward provides the agent with a measure of progress toward the goal. A simple reward scheme is to set  $r(s) = 1$  for all states  $s$  that represent the goal state, and 0 for all other states. To maximise  $\eta(\theta)$ , goal states must be reached as frequently as possible. This has the desired property of simultaneously minimising plan duration, as well as maximising the probability of reaching the goal (failure states achieve no reward). It is tempting to provide a negative reward for failure end states, but in this case an agent could partially maximise its reward by avoiding progress altogether, never achieving end states, and therefore never achieving negative (or positive) rewards.

We propose a reward scheme that provides a large reward (1000 in this paper) for reaching the goal as described, plus a reward at each step that heuristically awards progress toward the goal. This additional *shaping* reward provides a reward of 1 for every goal condition achieved, and -1 for every goal condition that becomes unset.

### 3.1 Planning State Space

For probabilistic temporal planning our state description contains: the state's absolute time, a queue of impending events, the status of each task, the truth value of each state variable, and the available resources. In a particular state, only a subset of the eligible tasks will satisfy all preconditions for execution. This subset is called the *eligible* task list. When a decision to start a fixed duration task is made, an end-task event is added to the time ordered event queue. The event queue holds a list of events that the planner is committed to, although the outcome of those events may be uncertain.

The generation of successor states is shown in Alg. 1. The algorithm begins by starting the tasks given by each bit in the action, implementing any immediate effects. An end-task

<sup>1</sup>Except for the aperiodic condition for ergodicity that is not important for this paper.

<sup>2</sup>Because the underlying MDP is ergodic,  $\eta(\theta)$  is independent of the starting state.

---

#### Algorithm 1 findSuccessor(State $s$ , Action $\mathbf{a}$ )

---

```

1: for each  $a_n = \text{'Yes'}$  in  $\mathbf{a}$  do
2:    $s.beginTask(n)$ 
3:    $s.addEvent(n, s.time+taskDuration(n))$ 
4: end for
5: repeat
6:   if  $s.time > \text{maximum makespan}$  then
7:      $s.failureLeaf=true$ 
8:     return
9:   end if
10:  if  $s.operationGoalsMet()$  then
11:     $s.goalLeaf=true$ 
12:    return
13:  end if
14:  if  $s.noEvents() \& \neg s.anyEligibleTasks()$  then
15:     $s.failureLeaf=true$ 
16:    return
17:  end if
18:   $e = s.nextEvent()$ 
19:   $s.time = e.time$ 
20:  selectModel( $e.PrSuccessLower, e.PrSuccessUpper$ )
21:  sample  $success Pr = p, failure Pr = 1 - p$ 
22:   $s.implementEffects(outcome)$ 
23: until  $s.anyEligibleTasks()$ 

```

---



---

#### Algorithm 2 selectModel(lowerBound, upperBound)

---

```

1: if pessimistic then
2:   return ( $e.lowerBound$ )
3: else
4:   if optimistic then
5:     return ( $e.upperBound$ )
6:   else
7:     return( $lowerBound+upperBound/2$ )
8:   end if
9: end if

```

---

event is added at an appropriate time in the queue. The state update then processes events until there is at least one task that is eligible to begin. Lines 6–17 check for reset states before the next event for the current state  $s$  is processed.

Events have probabilistic outcomes. Uncertain models provide intervals of probabilities for outcomes. The intervals are defined as part of the problem specification. Before sampling we must choose a point in this interval to base the sample on. We assume that maximising the probability of failure also minimises the long-term average reward for the current policy. Thus, to train the agent to operate well under the pessimistic model we always choose the lower bound on the probability of success as the true probability of the event (Alg. 2), and sample the outcome accordingly. Similarly, if we wish the agent to perform well if the optimistic model turns out to be correct, we select the upper bound on the probability of success. If there are more than two outcomes we could put intervals on the probability mass associated with each outcome. We then distribute the probability mass as constrained by the intervals. The worst outcomes gets the maximum probability mass, the next worst outcome gets

the maximum allowed remaining mass, and so on.

This scheme is not guaranteed to select probabilities that correspond to the worst overall model. A pessimistic choice at the current state could lead to future states with very little uncertainty in the model, whereas an optimistic choice could lead to future states with massive uncertainty and hence larger scope for poor models. We assume there is a way to approximately measure which outcomes will lead to high or low rewards. Section 6 outlines how we might learn the worst (or best) overall model in the same setting.

Line 21 of Alg. 1 samples one possible outcome from the distributions permitted by the intervals in the problem definition. Alg. 2 is the only point in the algorithm where intervals are considered. The remainder of the algorithm description is independent of our use of uncertain models.

Future states are only generated at points when tasks can be started. If an event outcome is processed and no tasks are enabled, the search recurses to the next event in the queue.

## 4 Policy Gradient Ascent

In this section we describe policy-gradient algorithms for reinforcement learning and how we use this approach for the FPG-Planner. We assume the presence of policy agents, parameterised with independent sets of parameters for each agent  $\theta = \{\theta_1, \dots, \theta_N\}$ . There are  $p$  parameters in total. We seek to adjust the parameters of the policy to maximise the long-term average reward  $\eta(\theta)$ .

Baxter *et al.* [2001] describe the GPOMDP algorithm that estimates the gradient  $\nabla\eta(\theta)$  of the long-term average reward with respect to the current set of policy parameters. Once an estimate  $\hat{\nabla}\eta(\theta)$  is computed, we maximise the long-term average reward with a gradient ascent step:  $\theta \leftarrow \theta + \alpha \hat{\nabla}\eta(\theta)$ , where  $\alpha$  is a small step size. Maximising  $\eta(\theta)$  produces better policies, both in terms of duration and probability of failure. Repeating the process of estimating the gradient, followed by a gradient ascent step, optimises the policy until a maxima in the long-term average reward is found.

### 4.1 Estimating Gradients

The GPOMDP gradient estimate algorithm works by sampling a single long trajectory through the state space. The state transitions are generated with Alg. 1 after each task agent has chosen whether to start or not. All agents receive the same reward for the new state and update their gradient estimates independently.

The parameterised policy maps observations to probability distributions over action vectors. The action vector at each step is  $\mathbf{a}_t$ , a combination of independent ‘Yes’ or ‘No’ actions made by the agents. Each agent is parameterised by an independent set of parameters that make up  $\theta \in \mathbb{R}^p$ :  $\theta_1, \theta_2, \dots, \theta_N$ . If  $a_{tn}$  represents the binary decision made by agent  $n$  at time  $t$  about whether to start its corresponding task then the stochastic policy factors into

$$\begin{aligned} \Pr[\mathbf{a}_t | \mathbf{o}_t, \theta] &= \Pr[a_{t1}, \dots, a_{tN} | \mathbf{o}_t, \theta_1, \dots, \theta_N] \\ &= \Pr[a_{t1} | \mathbf{o}_t, \theta_1] \times \dots \times \Pr[a_{tN} | \mathbf{o}_t, \theta_N]. \end{aligned}$$

It is not necessary for all agents to receive the same observation, and it may be advantageous to show different agents

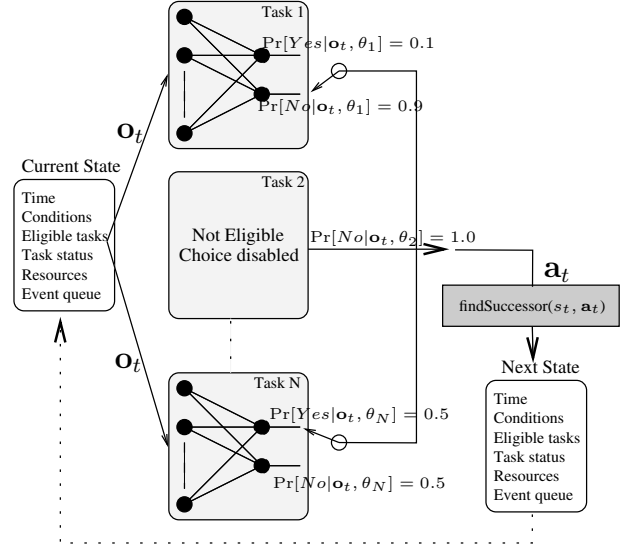


Figure 1: Task-policies receive an observation of the current state and individually choose whether to start or not. The combined decision is fed into the state simulator that probabilistically generates the next state. In this example, the joint action probability is  $\Pr[No, No, \dots, Yes | \mathbf{o}_t, \theta] = 0.45$ .

different parts of the state, leading to a decentralised planning algorithm. All our results assume that the observation vector  $\mathbf{o}$  is a binary description of the eligible tasks (15 bits) and the condition truth values (10 bits) plus a constant 1 bit to provide bias to the agents’ linear networks.

The main requirement for each policy-agent is that  $\Pr[a_{tn} | \mathbf{o}_t, \theta_n]$  be differentiable with respect to the parameters for each choice task start  $a_{tn} = \text{‘Yes’}$  or ‘No’. We choose to represent each agent with a two output linear network mapped into probabilities using a soft-max function:

$$\begin{aligned} \Pr[a_{tn} = Yes | \mathbf{o}_t, \theta_n] &= \frac{\exp(\mathbf{o}_t^\top \theta_{n, Yes})}{\exp(\mathbf{o}_t^\top \theta_{n, Yes}) + \exp(\mathbf{o}_t^\top \theta_{n, No})} \\ \Pr[a_{tn} = No | \mathbf{o}_t, \theta_n] &= \frac{\exp(\mathbf{o}_t^\top \theta_{n, No})}{\exp(\mathbf{o}_t^\top \theta_{n, Yes}) + \exp(\mathbf{o}_t^\top \theta_{n, No})}. \end{aligned}$$

This can be thought of as a two output linear network where the outputs are subsequently normalised to produce a well behaved probability distribution. If the dimension of the observation vector is  $|\mathbf{o}|$  then each  $\theta_n$  can be thought of as an  $|\mathbf{o}| \times 2$  matrix where the columns represent the network weights for the ‘Yes’ decision output and the ‘No’ decision output respectively. This expression is a form of logistic regression. The log derivatives, necessary for Alg. 3, are given in Baxter *et al.* [2001]. Initially the parameters are set to small random values: a near uniform random policy. This encourages exploration of the action space. Each gradient step typically moves the parameters closer to a deterministic policy.

Figure 1 shows the selection of actions graphically. Alg. 3 describes the algorithm for computing  $\hat{\nabla}\eta(\theta)$ . The gradient estimate provably converges to a biased estimate of  $\nabla\eta(\theta)$  as  $T \rightarrow \infty$ . The quantity  $\beta \in [0, 1)$  controls the degree of bias in the estimate. As  $\beta$  approaches 1, the bias of the

estimates drop to 0. However if  $\beta = 1$ , estimates exhibit infinite variance as  $T \rightarrow \infty$ . Thus the parameter  $\beta$  achieves a bias/variance tradeoff in the stochastic gradient estimates.

---

**Algorithm 3** Factored Planning Gradient Estimator
 

---

```

1: Set  $s_0$  to initial state,  $t = 0$ ,  $\mathbf{e}_t = [0]$ 
2: while  $t < T$  do
3:    $\mathbf{e}_t = \beta \mathbf{e}_{t-1}$ 
4:   Generate observation  $\mathbf{o}_t$  of  $s_t$ 
5:   for Each eligible task  $n$  do
6:     Compute  $\Pr[Yes|\mathbf{o}_t, \theta_n]$  and  $\Pr[No|\mathbf{o}_t, \theta_n]$ 
7:     Sample  $a_{tn} = \text{Yes}$  or  $a_{tn} = \text{No}$ 
8:     Compute  $\mathbf{e}_t = \mathbf{e}_t + \nabla \log \Pr[a_{tn}|\mathbf{o}_t, \theta_n]$ 
9:   end for
10:  Try action  $\mathbf{a}_t = \{a_{t1}, a_{t2}, \dots, a_{tN}\}$ 
11:  while mutex or resource prohibits  $\mathbf{a}_t$  do
12:    randomly turn off one task start in  $\mathbf{a}_t$ 
13:  end while
14:   $s_{t+1} = \text{findSuccessor}(s_t, \mathbf{a}_t)$ 
15:   $\hat{\nabla}_t \eta(\theta) = \hat{\nabla}_{t-1} \eta(\theta) - \frac{1}{t+1} (r(s_{t+1}) - \hat{\nabla}_{t-1} \eta(\theta))$ 
16:   $t \leftarrow t + 1$ 
17: end while
18: Return  $\hat{\nabla}_T \eta(\theta)$ 

```

---

Line 8 computes the log gradient of the sampled action probability and adds the gradient for the  $n$ 'th agent's parameters into an *eligibility trace*. The gradient for parameters not relating to agent  $n$  is 0. We do not compute  $\Pr[a_{tn}|\mathbf{o}_t, \theta_n]$  or gradients for tasks with unsatisfied preconditions. If all eligible agents decide *not* to start their tasks, we issue a null-action. If the state event queue is not empty, we process the next event, otherwise time is incremented by 1 to ensure all possible policies will eventually reach a reset state.

## 5 Experiments

This section provides some preliminary experiments that validate the ideas in this paper. We compare the present algorithm with that of our earlier RTDP based planner for military operations [Aberdeen *et al.*, 2004]. Both the current tools and our previous tool use the same code to generate states, representing exactly the same domains, providing a fair comparison.

The problem<sup>3</sup> consists of 15 tasks designed to represent the high level process of building a sky-scraper. These tasks achieve a set of 10 state variables needed for operation success. Four of the effects can be established independently by two different tasks, however, resource constraints only allow one of the tasks to be chosen. Furthermore, tasks are not repeatable, even if they fail. The probability of failure of tasks ranges between 0 and 20% with an interval on either side of 20% (unless such an interval would result in a probability of failure of less than 0%).

We have constructed this example to demonstrate the effectiveness of planning with intervals. Thus, for each effect that has two tasks that can achieve it we have selected one task

<sup>3</sup>The problem definition can be found on <http://rsise.anu.edu.au/~daa>, written using an XML version of the PDDL language).

to have a high probability of success, but also a high uncertainty. The second task has a lower probability of success, but an interval of 0 (perfect knowledge of the model). The second (lower) probability of success is chosen to be higher than the *lower bound* on the first tasks success probability. The robust plan should (and does) choose tasks with the lower probability of success, but zero interval. Table 1 shows that the results of using the FPG-Planner with different modes of optimisation: 1- No optimisation at all, the plan is to start each eligible task with a probability of 50%; 2- Optimisation based on a simulation of a pessimistic model; 3- Optimisation based on the original human model (mean model); 4- Optimisation based on a simulation of an optimistic model. Evaluations are repeated three times. The evaluations assume that the true model is: 1- the pessimistic model, 2- the original human specified model (mean model), 3- the optimistic model.

The parameters of the GPOMDP algorithm are:  $T = 50,000$  gradient estimation steps and  $\beta = 0.9$ . Optimisation time was limited to 5 minutes wall clock time on a single user 3GHz Pentium IV with 1GB ram. All optimisations ran to the complete 5 minutes. After 5 minutes optimisation was terminated and the current policy evaluated. The FPG-Planner has the 'any time' property that returns better policies the longer optimisation is allowed to run, thus it is possible we might have gotten improved results with more patience. Results quote the average duration ( $\pm$  the variance), and the percentage of plans that terminate in a failure state. Plans that fail often have short average durations because they fail early in the execution of the plan due to resource limitations or a lack of alternative courses of action. Because optimisation has a stochastic component the results presented are the average over 100 training runs and 10,000 evaluation runs of the plan.

Unsurprisingly we see that plans formed under pessimistic training perform much better than other training modes when the true (evaluation) model turns out to follow the pessimistic model. Less obviously, the plan formed under a pessimistic model has a more uniform failure probability and durations over the possible true models. This is highly desirable because it means we have less variance in the outcome of plans despite operating over a wide range of models. We emphasise that this result is largely dependent on the particular domain and is a result of a true assumption that experiencing a less pessimistic true model can only benefit the policy. We have seen similar effects on other domains using the RTDP planner [Buffet and Aberdeen, 2005].

The RTDP results are quite similar to the FPG-Planner results. The FPG-Planner is performing somewhat better than RTDP if the true model turns out to be pessimistic, but training assumed a mean or optimistic model. RTDP gets the best overall result when the true model is optimistic and training assumed a mean or optimistic model. In this case RTDP is finding a global maxima in long-term average reward, but FPG gets stuck in local maxima.

For problems of this size RTDP can enumerate the state space in memory, giving it a significant advantage because it can compute the optimal global policy. Thus, we do not expect to be able to generally perform better RTDP on this

Table 1: Average failure prob and duration of the optimised Building plan. The columns are different training conditions. The rows are different evaluation conditions. Optimisation is performed with the FPG-Planner

True model	No train		Pess. train		Mean train		Opt train	
	Fail%	Dur.	Fail%	Dur.	Fail%	Dur.	Fail%	Dur.
Pessimistic	0.657	4.80±2.22	0.549	4.42±1.78	0.688	3.35±2.89	0.694	3.42±2.37
Mean	0.403	5.97±1.69	0.420	5.16±1.34	0.378	4.98±1.64	0.381	5.03±1.67
Optimistic	0.325	6.30±1.36	0.386	5.35±1.12	0.278	5.46±1.03	0.277	5.51±1.05

Table 2: Same results as Table 1, but this time optimised with an RTDP based planner.

True model	No train		Pess. train		Mean train		Opt train	
	Fail%	Dur.	Fail%	Dur.	Fail%	Dur.	Fail%	Dur.
Pessimistic	0.657	4.80±2.22	0.541	4.62±3.12	0.729	5.32±1.46	0.724	5.40±1.76
Mean	0.403	5.97±1.69	0.428	5.09±1.80	0.272	6.49±1.12	0.272	6.54±1.19
Optimistic	0.325	6.30±1.36	0.386	5.18±1.65	0.099	6.90±0.451	0.100	6.90±0.452

problem. The fact that we outperform RTDP at all is due to the fact that we use the labelled variant of RTDP [Bonet and Geffner, 2003], with a non-zero labelling threshold that results in some degree of approximation in the policy. However, as Aberdeen [2005] demonstrates, when problems are too large to fit into main memory, the FPG-Planner can perform significantly better than RTDP based planners.

## 6 Discussion

The main requirement for learning with policy-gradient POMDP methods is that a trajectory of state observations is available. Even if we have *no model* of the planning problem we can still use FPG-Planning provided we can interact with the real-world to generate trajectories.

The greatest drawback of our work is the assumption that the poorest (or best) global model can be approximated by always trying to simulate the extremes of the intervals in each state. We can avoid this assumption by simultaneously learning the worst model at the same time as learning the best policy. This can be achieved with a second agent assigned to each planning task. The second agent learns, again using a gradient method, the most pessimistic point in the interval that should be used to simulate trajectories. We plan to try this approach soon, borrowing on the work of Bowling [2005].

To summarise, we have demonstrated an algorithm with great potential to produce policies that are robust to a degree of ‘guesswork’ in constructing the model. It is critical that real-world planning tools are tolerant of errors in the description of the model. Human beings are bad at estimating probabilities, and it is rare that we have sufficient data to perfectly estimate all parameters of a system. Further work will attempt to justify our claim that the simulation approach to dealing with uncertainty has merit in very large domains.

## Acknowledgements

National ICT Australia is funded by the Australian Government’s Backing Australia’s Ability program and the Centre of Excellence program. This project was also funded by the Australian Defence Science and Technology Organisation.

## References

- [Aberdeen *et al.*, 2004] D. Aberdeen, S. Thiébaux, and L. Zhang. Decision-theoretic military operations planning. In *Proc. ICAPS’04*, 2004.
- [Aberdeen, 2005] D. Aberdeen. Probabilistic temporal planning by factored policy gradient. Technical report, NICTA, 2005.
- [Barto *et al.*, 1995] A.G. Barto, S. Bradtke, and S. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72, 1995.
- [Baxter *et al.*, 2001] J. Baxter, P. Bartlett, and L. Weaver. Experiments with infinite-horizon, policy-gradient estimation. *JAIR*, 15:351–381, 2001.
- [Bonet and Geffner, 2003] Blai Bonet and Hector Geffner. Labeled RTDP: Improving the convergence of real-time dynamic programming. In *Proceedings of ICAPS-03*, 2003.
- [Bowling, 2005] Michael Bowling. Convergence and no-regret in multiagent learning. In *Proc. of NIPS’04*, volume 17, 2005.
- [Buffet and Aberdeen, 2005] O. Buffet and D. Aberdeen. Planning with robust (l)rtdp. In *Proc. of IJCAI’05*, 2005.
- [Givan *et al.*, 2000] R. Givan, S. Leach, and T. Dean. Bounded parameter markov decision processes. *Artificial Intelligence*, 122(1-2):71–109, 2000.
- [Hosaka *et al.*, 2001] M. Hosaka, M. Horiguchi, and M. Kurano. Controlled markov set-chains under average criteria. *Applied Mathematics and Computation*, 120(1-3):195–209, 2001.
- [Little, 2004] I. Little. Probabilistic temporal planning. Honours thesis, Australian National University, 2004.
- [Mausam and Weld, 2005] Mausam and Daniel S. Weld. Concurrent probabilistic temporal planning. In *Proc. ICAPS’05*, 2005.
- [Singh *et al.*, 1994] S. Singh, T. Jaakkola, and M. Jordan. Learning without state-estimation in partially observable Markovian decision processes. In *Proceedings of ICML 1994*, number 11, 1994.
- [Strehl and Littman, 2004] A. Strehl and M. Littman. An empirical evaluation of interval estimation for markov decision processes. In *Proc. of ICTAI’04*, 2004.
- [Younes and Simmons, 2004] Hakan L. S. Younes and Reid G. Simmons. Policy generation for continuous-time stochastic domains with concurrency. In *Proc. of ICAPS’04*, volume 14, 2004.