A Two-Teams Approach for Robust Probabilistic Temporal Planning

Olivier Buffet and **Douglas Aberdeen**

National ICT Australia &
The Australian National University
firstname.lastname@nicta.com.au

Abstract. Large real-world Probabilistic Temporal Planning (PTP) is a very challenging research field. A common approach is to model such problems as Markov Decision Problems (MDP) and use dynamic programming techniques. Yet, two major difficulties arise: 1- dynamic programming does not scale with the number of tasks, and 2- the probabilistic model may be uncertain, leading to the choice of unsafe policies. We build here on the *Factored Policy Gradient* (FPG) algorithm and on robust decision-making to address both difficulties through an algorithm that trains two competing teams of learning agents. As the learning is simultaneous, each agent is facing a non-stationary environment. The goal is for them to find a common Nash equilibrium.

1 Introduction

Probabilistic Temporal Planning [1] can be efficiently modeled using Markov Decision Problems (MDPs) [2]. Dynamic programming is a common approach to find plans (policies) optimising the expected long-term utility of an MDP.

In particular, PTP leads to a type of MDP taking the form of stochastic shortest-path problems for which algorithms such as *Real-Time Dynamic Programming* (RTDP) [3] are particularly appropriate. Unfortunately, the "relevant" part of the state space (from the optimal policies point of view) will grow exponentially with the number of tasks to handle, which makes it difficult to scale to large real world problems. A possible solution is to switch to a factored algorithm where each individual task has to learn when it should be triggered, regardless of the policy for the remaining tasks [4; 5]. Here we consider this method as a multi-agent reinforcement learning algorithm.

Another difficulty with real-world PTP is that the model may be inaccurate, resulting in overly-optimistic plans in risk-adverse domains. This uncertainty usually lies in the transition probabilities, and is due to learning models statistically or by asking. A simple way of representing such uncertain probabilities is through intervals in which the true model values lie with a given confidence.

Taking this uncertainty into account while planning usually means looking for a robust plan, i.e., searching for the best plan while an opponent is looking for the worst possible model. Indeed, this game-theoretic point of view has

been successfully applied by modifying dynamic programming algorithms such as *Value Iteration* [6] or *RTDP* [7].

As can be observed, making a planning algorithm scalable or robust leads to different "multi-agent" solutions. We present an algorithm, Two-Teams-FPG (TT-FPG), that attempts to tackle both problems by merging these two solutions in a single framework where two teams of learning agents are competing. Because agents learn simultaneously, each of them is facing a non-stationary environment. A difficult problem is to ensure convergence to a Nash equilibrium.

We first introduce Markov Decision Problems and Probabilistic Temporal Planning. Section 3 describes the two domains we are building on: the *Factored Policy Gradient* algorithm and robustness. Our two-teams algorithm and its experimental validation follow in Sections 4 and 5 before a conclusion.

2 Background

2.1 Markov Decision Problems

A Markov Decision Problem [2] is defined here as a tuple $\langle S, A, T, r\rangle$. It describes a control problem where S is the finite set of **states** of the system considered and A is the finite set of possible **actions** a. Actions control transitions from one state s to another state s' according to the system's probabilistic dynamics, described by the **transition function** T defined as $T(s,a,s') = Pr(s_{t+1} = s' | s_t = s, a_t = a)$. The aim is to optimise a performance measure based on the **reward function** $r : S \times A \times S \to \mathbb{R}$.[1]

An algorithm optimising an MDP has to find a **policy** that maps states to probability distributions over actions $\pi : S \to \Pi(A)$ which optimises the chosen performance measure, here the **value** V defined as the average *reward* gained during a transition. Other performance measures include the discounted sum of rewards on an infinite horizon ($V_{dis} = \sum_{t=0}^{\infty} \gamma^t r_t$, where $\gamma \in [0,1)$) and the cost ($c(s,a,s') = -r(s,a,s')$) to the goal J when our aim is to reach a goal state with minimal cost.

Dynamic Programming — Well-known stochastic dynamic programming algorithms such as *value iteration* (VI) make it possible to find a deterministic policy maximising V_{dis} (or minimising J). *Value iteration* works by computing the value function $V^*_{dis}(s)$ that gives the expected reward of the optimal policies. It is the unique solution of the fixed point equation [8][2]

$$V(s) = \max_{a \in A} \sum_{s' \in S} T(s,a,s') \left[r(s,a,s') + \gamma V(s') \right]. \tag{1}$$

Updating V_{dis} with this formula leads to the optimal value function. For convenience, we also introduce the Q-value: $Q(s,a) = \sum_{s' \in S} T(s,a,s')[r(s,a,s') + W(s')]$, where W can be V, γV_{dis} or J depending on the performance measure.

[1] As the model is not sufficiently known, we do not make the usual assumption $r(s,a) = \mathbb{E}_{s'}[r(s,a,s')]$. The expectation depends on the (unknown) true model.

[2] An equivalent formulation exists for minimising the long-term cost J.

Policy-Search — Apart from dynamic programming, there exists various direct policy-search algorithms. They often amount to the problem of optimising a function (the performance measure) in a space of parameters on which the policy depends. William's *REINFORCE* [9] is the first example of tuning the parameters of a connexionist controller based on immediate reinforcement signals. These algorithms do not necessarily rely on the knowledge of the exact state of the system: they can find a (locally) optimal policy using partial observations.

In our work, we have mainly used the on-line policy-gradient algorithm *OL-POMDP* [10; 11]. Here, the policy is defined as a function $\mu_a(\theta, o)$ giving the probability of choosing action a under observation o, where θ is the policy's vector of parameters. *OL-POMDP* works by approximating the gradient after each simulation step and following it. At time t, if action a_t is performed while observing o_t and receiving reward r_t, the update is achieved by computing:

$$z_{t+1} = \beta z_t + \frac{\nabla \mu_{a_t}(\theta_t, o_t)}{\mu_{a_t}(\theta_t, o_t)} \tag{2}$$

$$\theta_{t+1} = \theta_t + \alpha_t r_t z_{t+1} \tag{3}$$

where β and α_t are learning rates.

2.2 Probabilistic Temporal Planning

Our planning domain is described by: a set of condition variables $\mathcal{C} = \{C_1, \ldots, C_{|\mathcal{C}|}\}$, a set of resources $\mathcal{R} = \{R_1, \ldots, R_{|\mathcal{R}|}\}$ and a set of tasks $\mathcal{T} = \{T_1, \ldots, T_{|\mathcal{T}|}\}$.

A task t can be triggered if certain conditions hold (pre-conditions) and if certain resources are available. After the task's duration, one of several possible outcomes $out_T(1), out_T(2) \ldots$ arise, depending on a probability distribution $Pr(out_T(k))$, and setting some conditions or consuming some resources. Tasks can run simultaneously and can be repeated if necessary.

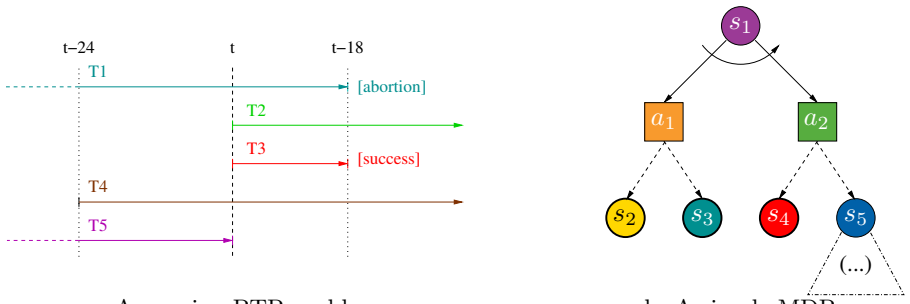
Our objective is to reach a goal state specified as a conjunction of conditions. Indeed, among previous probabilistic temporal planners — as CPTP [12], Prottle [13], Tempastic [14] and a military operations planner [15] — most are based on the MDP framework and dynamic programming algorithms.

PTP with MDPs — A simple way of turning such a probabilistic temporal planning problem into an MDP is, as done in [16; 15], to define:

- States as instants when some tasks end, resulting in a new situation where a decision can be made. A state is then defined by: current conditions, current resources and currently running tasks.
- Actions as the decision of triggering a set of eligible tasks. Triggering no task is valid as it amounts to waiting for the next decision point.
- Transition probabilities depend on each task's probability distribution over outcomes.
- Rewards are defined as: 0 by default and +1 when the goal is reached.

One could also give a negative reward for resource consumption or when a plan fails, what we detect by setting a time limit (on goal reachability, see [17]). In our implementation, +1000 is given when the goal is reached, and a progress estimator is used to help the planner move towards the goal (see [5]).

To give a graphical view of this transcription from PTP to MDP, Figure 1 presents side-by-side a representation of evolving tasks as from a probabilistic temporal planning problem, and a simple MDP.



a- A running PTP problem b- A simple MDP

Fig. 1. In this work, PTP problems (a) are turned into MDPs (b).

3 Two Multi-Agent Approaches

We now come back to the two problems we want to address — scalability and robustness of the planning algorithm — and discuss the solutions that are the basis of our contribution.

3.1 Factored Policy Gradient

Most probabilistic temporal planners are based on MDPs and dynamic programming. This leads to state- and action-spaces that grow exponentially with the number of tasks. Indeed, if n tasks are eligible in state s, there are 2^n possible actions. The number of possible next states from s grows similarly. This is a major problem even if algorithms do not necessarily develop the complete state-space. Ensuring optimality requires keeping in memory more than the relevant states (states which can be visited by some optimal policy).

Reducing Spatial Complexity — To tackle this problem, the first idea behind the *Factored Policy-Gradient planner* (FPG) [5] is to avoid enumerating states by using a direct policy-search.[3] In fact, reducing the algorithm's spatial complexity

[3] Another approach is to compute an approximation of the Q-values.

requires reducing the number of degrees of freedom through the choice of the policy's parameterisation.

FPG's parameterisation is mainly influenced by the distributed perspective of PTP: it is quite natural to see this kind of problem as the problem of having multiple "task-agents" coordinating their decisions to trigger their respective tasks. An interesting property of this factorization is that the number of possible deterministic policies remains the same: at each decision point, if n tasks are eligible, there are 2^n local deterministic strategies. This is good news because the optimal policy will still be obtainable given a sufficiently rich parameterisation for each agent.

If we are prepared to sacrifice the existence of an optimal deterministic policy, we may introduce partial observability and reduce the number of parameters (degrees-of-freedom). We do this to simplify the agents, hence speeding up learning for very large domains.

Multi-Agent Policy Search — A lot of work has been done on multi-agent decision making, often based on dynamic programming. Yet, convergence guarantees of dynamic programming often require a stationary environment, which is not satisfied in multi-agent settings as the environment comprises agents with changing behaviors. But a factored policy-gradient setting can still be seen as a single learning agent.

Examples of successful applications of direct policy search to cooperative multi-agent settings are [18], [19] and [20]. All three are based on policy-gradient algorithms. It is important to observe that they can differ in how reward is distributed among agents: in [19], the algorithm attempts to guess if an agent should get more reward than another.

A last remark is that a multi-agent policy search makes it easier to reuse knowledge when the planning domain changes. Assuming a partially observable setting in which an agent does not know about currently running tasks, new tasks can be easily added while the policies linked to already existing tasks may need only moderate changes to remain efficient. This is a strong argument as real-world planning domains often evolve with time. Also, task-dependent policies may give a more understandable solution than a monolithic policy.

3.2 Robustness

Earlier Work — The second aspect of real-world domains we want to take into account is the uncertainty of the model. This uncertainty is sometimes represented as a set of possible models, each assigned a model probability [21]. The simplest example is a set of possible models that are assumed equally probable [6; 22]. But, rather than construct a possibly infinite set of models we represent model uncertainty by allowing each probability to lie in an interval [23; 24]. That is, we know that $T(s, a, s') \in [Pr^{\min}(s'|s, a), Pr^{\max}(s'|s, a)]$.

Uncertain probabilities have been investigated in resource allocation problems [21] — investigating efficient exploration [25] and state aggregation [23]

— and policy robustness [6; 24; 22]. We focus on the later, considering a zero-sum two-player game where the opponent chooses from the possible models to counter the planner's actions.

In previous work [7], we have focussed on making the *Real-Time Dynamic Programming* algorithm (RTDP) [3] robust. As other robust dynamic programming algorithms, the two-player game is made easier thanks to the assumption that next-state distributions $T(s, a, \cdot)$ are independent from one state-action pair (s, a) to another. This makes it possible to find worst models locally, i.e., for each state-action pair independently. It is all the more beneficial that the resulting *sequential* game only requires looking for pure strategies (deterministic policies) for both players.

The Case of Probabilistic Temporal Planning — In the case of probabilistic temporal planning, the model uncertainty is specified at the level of independent tasks. For our planning problems, a convenient representation is that of interval of probabilities in which the true probability of a given outcome is known to lie. Based on this uncertain per task model, probability intervals can be computed for the corresponding MDP state transitions.

Unfortunately, the independence assumption used in robust dynamic programming does not hold anymore with PTP. Indeed, as tasks can be triggered from different states, two transition probabilities $T(s_1, a_1, s'_1)$ and $T(s_2, a_2, s'_2)$ may depend on a common task outcome probability $Pr(out_{T_l}(k))$. In a framework where the "bad" task outcomes may be different depending on the decision point (current state), this independence assumption may be unrealistic and make the opponent's task too easy. As described in next section, our direct policy search attempts to remove this assumption.

4 The Two-Team Approach

4.1 Principle

In robust PTP, the opponent is controlling outcome probabilities, not transition probabilities. For each task, it has to tune as many probabilities as there are different possible outcomes. The constraints are that they must sum to one:

$$\sum_k Pr(out_{T_l}(k)) = 1$$

and each of them has to remain in its dedicated interval:

$$Pr(out_{T_l}(k)) \in [Pr^{\min}(out_{T_l}(k)), Pr^{\max}(out_{T_l}(k))]$$

Fig. 2 illustrates these constraints with three possible outcomes. An interesting characteristic of the resulting set of possible distributions is its convexity, which will be helpful when searching for optimal solutions.

We would like to avoid the independence assumption between the MDPs state-action pairs. Another choice of assumption should hold in most cases: that

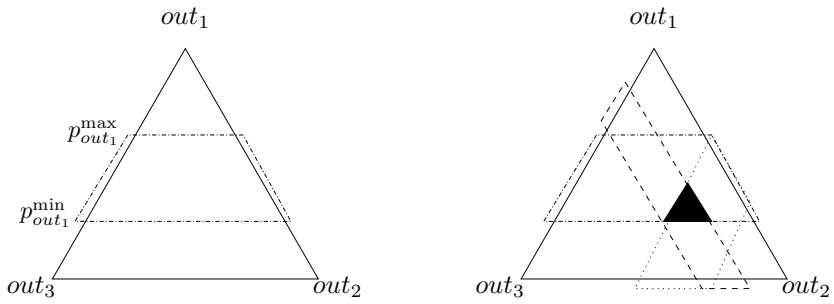


Fig. 2. A triangle is a probability simplex representing all possible probability distributions with three different outcomes ($Pr(out_i) = 1$ at the out_i vertex). The left triangle is the trapezium showing the interval constraint for out_1. The right triangle shows possible models at the intersection of the three interval constraints.

probability distributions over outcomes are independent from one task to another. It may be wrong with two quite similar tasks, but such tasks will often be associated with similar worst case models. Using this new assumption, it is natural to factor the opponent's control problem by assigning one opponent-agent to each task, i.e., applying an FPG-like factorization on the side of the opponent.

Immediately, the view of the robust probabilistic temporal planning framework as two opposing teams follows, with one player by task in each team.

4.2 The Game Being Played

Here we clarify the rules of the "game". A starting point in any game is deciding whether it is simultaneous or sequential. As robust planning is meant to deal with a worst case scenario, it is logical to consider a sequential game where the planner plays first, before the opponent observes this choice and then responds.

This does not mean that the game considered requires making a decision at each decision point, which would lead to the unrealistic situation of task models depending on the current state of the system. In fact, this game consists in:

- choosing a — possibly stochastic — policy for the planner, and
- choosing a global model (considering all tasks) for the opponent;

then the value of a game is the average reward gained by the planner under these two choices, and the opposite for the opponent.

The point of view adopted here is that of a game where player A's strategy may be known to player B, but is a mixed strategy. As the actual move will known only after player B's own choice, there is little difference in this setting between a sequential and a simultaneous game. In fact, it can simply be considered as a **repeated simultaneous game** as it makes it possible for B to learn A's strategy. Conversely, the planner can learn B's strategy and adapt to it.

4.3 Algorithm

The game-theoretic framework just described would ideally require learning at a quite high level, one player's action being the choice of a policy, the other player's action being the choice of a complete model, and their respective rewards being based on the average payoff. In practice, it seems preferable to make use of new experiences as soon as possible, i.e., to reinforce actions on-line while executing both player's policies.

Coming back to the two-teams view of our approach, we have used classical direct policy search algorithm for each agent:

- For each member of the planning team, the on-line policy-gradient presented in Sec. 2.1 is employed, using a simple parameterisation based on a perceptron. Observations are a subset of the state's description (conditions, resources, running tasks). Actions consist of triggering a task or not.
- For each member of the model-manipulating team, we applied a modified GIGA algorithm [26] (with eligibility traces). These agents are blind, to ensure that final task models do not depend on the system's state. Actions consist in setting outcome probabilities within legal values (as in Fig. 2).

The original FPG planner is using a different policy-gradient algorithm alternating between evaluating the gradient and performing a line-search. Having two such distinct and long phases does not seem favorable to our simultaneous game setting. The on-line algorithm we use (OLPOMDP) is probably more appropriate in this non-stationary framework. This is because the line search tends to drive to the extremes of policies, and it will reach the extreme for one player before the other, preventing the extreme player from recovering quickly.

Concerning the opponents, it is easier to apply a GIGA-like algorithm, especially to implement the constraints. The eligibility traces are used to reward decisions made in the past, remembering that these actions are performed when a task's outcome has to be decided, i.e., often when the task ends.

5 Experiments

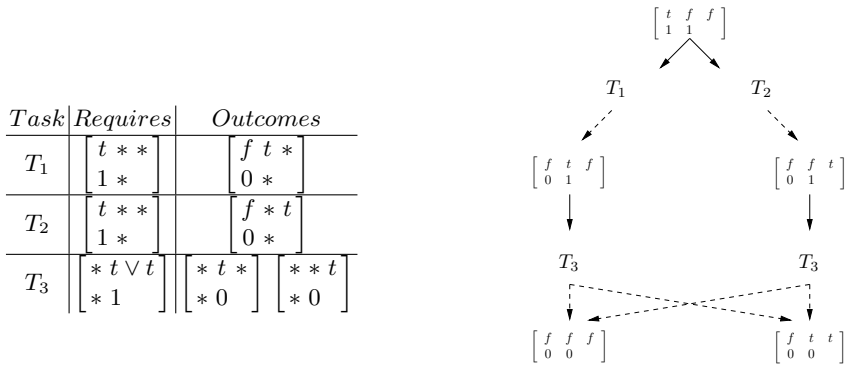
5.1 Toy Domain

This first domain is here to illustrate why the usual independence assumption of robust planning can be misleading. It consists of: 3 condition variables C_1, C_2, C_3, 2 resources R_1, R_2, and 3 tasks T_1, T_2, T_3. No two tasks can be run simultaneously, so that decision points (states) correspond to the end of a task running alone. Thus, the initial state s_0 can be described by the values of condition variables, and resources: $C_1 = true,\ C_2 = false,\ C_3 = false$ and $R_1 = 1,\ R_2 = 1$, what will be noted:

$$s_0 = \begin{bmatrix} t & f & f \\ 1 & 1 & \end{bmatrix}$$

Using a $*$ to indicate conditions or resources which are not requirements to start a task T or which are not changed by this task, the three tasks can be

defined as on Fig. 3-a. Then, Fig. 3-b shows possible evolutions of the system, starting from s_0. The goal is to get $C_2 = true$ and $C_3 = true$. Only T_3's outcome is probabilistic and uncertain, all probability distributions over its two outcomes being allowed.



a) Description of tasks b) Possible evolutions of the system

Fig. 3. Toy problem showing why the assumption that the probability distribution $T(s, a, \cdot)$ is independent from one state-action pair to another can be dangerous in PTP.

In this setting, the opponent's optimal policy is to give equal probabilities to each of T_3's outcomes. Otherwise the planner may benefit from the opponent's preference toward one outcome.

5.2 Building Domain

These preliminary experiments are based on the same domain used to validate the FPG-planner in [4], where it is compared to an RTDP based planner for military operations [15]. All three tools are based on the same code to simulate the domain.

The problem 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 failure probability below 0%).

This example is designed to demonstrate the effectiveness of robust planning. Thus, for each effect that has two tasks that can achieve it we have selected one task to have a higher probability of success than the other, but also a higher uncertainty. Tab. 1 shows the results of using our FPG-Planner (based on OL-POMDP) 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 results are very close to results in [4], whereas OL-POMDP was here running at most 2 minutes while GPOMDP was initially limited to 5 minutes. One must pay attention to the fact that the algorithm is optimising how often a goal state is reached, i.e., a compromise between plan failure probability and duration.

Table 1. Average failure probability and duration of the optimised Building domain. Columns are different training conditions. Rows are different evaluation conditions. Optimisation is performed with the FPG-planner, using OLPOMDP.

True model	No train		Pess. train		Mean train		Opt train	
	Fail%	Dur.	Fail%	Dur.	Fail%	Dur.	Fail%	Dur.
Pessimistic	0.690	4.22±2.33	0.568	4.39±1.87	0.629	4.27±1.88	0.623	4.30±1.82
Mean	0.381	5.91±1.62	0.426	5.23±1.41	0.353	5.45±1.44	0.358	5.38±1.41
Optimistic	0.279	6.39±1.00	0.383	5.45±1.16	0.279	5.74±1.11	0.278	5.70±1.11

Tab. 2 shows results with TT-FPG, considering that the opponent could also be a friend (looking for the best model). The trained (or not-trained) planner is still evaluated against the same three models, but also against the second team. As can be observed, the second team properly behaves like an opponent or a friend. In this simple example, it usually finds the pessimistic and optimistic models. The slightly degraded results of the planner in TT-FPG should be solved with more learning time.

Table 2. Same results as Table 1 but with the optimisation performed with the TT-FPG planner.

True model	No Train vs Opp		No Train vs Friend		Train vs Opp		Train vs Friend	
	Fail%	Dur.	Fail%	Dur.	Fail%	Dur.	Fail%	Dur.
vs Player	0.700	4.18±2.32	0.285	6.38±1.01	0.618	4.29±1.84	0.273	5.71±1.09
Pessimistic	0.690	4.22±2.33	0.690	4.22±2.33	0.612	4.33±1.83	0.628	4.26±1.84
Mean	0.381	5.91±1.62	0.381	5.91±1.62	0.363	5.35±1.42	0.357	5.39±1.40
Optimistic	0.279	6.39±1.00	0.279	6.39±1.00	0.282	5.67±1.10	0.278	5.69±1.09

6 Conclusion

A first remark is that, even if these two teams are opponents, they have the common objective of finding an equilibrium and should not try to exploit possible

defaults in the other team's learning algorithm as described in [27]. This aspect has not been particularly taken into account in our choice of the individual learning algorithms. A main direction to look at is how to ensure convergence and no-regret through the use of the "Win or Learn Fast" principle from [28] and the ideas behind GIGA [26] for example (see also [29]).

An important point to keep in mind regarding TT-FPG is that, even if the planning-team has complete observability, it is not facing a completely observable MDP and its optimal policy is probably not deterministic. The team mates would then benefit from the ability to synchronize their actions, but at the cost of getting back to the complexity of the original non-factored approach.

To summarize, TT-FPG proves to be a good candidate for robust probabilistic temporal planning. It is designed to be more scalable than dynamic programming approaches, and does not depends on an independence assumption of most robust approaches, assumption which does not hold in PTP. Future work should mainly focus on ensuring convergence to a Nash equilibrium, as robustness puts learning agents in a non-stationary environment.

Acknowledgments — This work was supported by National ICT Australia (NICTA) and the Australian Defence Science and Technology Organisation (DSTO) in the framework of the joint Dynamic Planning Optimisation and Learning Project (DPOLP). NICTA is funded through the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council.

References

1. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory and Practice. Morgan Kauffmann Publishers (2004)
2. Bertsekas, D., Tsitsiklis, J.: Neurodynamic Programming. Athena Scientific (1996)
3. Barto, A., Bradtke, S., Singh, S.: Learning to act using real-time dynamic programming. Artificial Intelligence **72** (1995)
4. Aberdeen, D., Buffet, O.: Simulation methods for uncertain decision-theoretic planning. In: Proc. of the IJCAI 2005 Workshop on Planning and Learning in A Priori Unknown or Dynamic Domains. (2005)
5. Aberdeen, D.: Policy-gradient methods for planning. In: Advances in Neural Information Processing Systems 19 (NIPS'05). (2005)
6. Bagnell, J., Ng, A.Y., Schneider, J.: Solving uncertain Markov decision problems. Technical Report CMU-RI-TR-01-25, Robotics Institute, Carnegie Mellon U. (2001)
7. Buffet, O., Aberdeen, D.: Robust planning with (l)rtdp. In: Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI'05). (2005)
8. Bellman, R.: Dynamic Programming. Princeton U. Press, Princeton, New-Jersey (1957)
9. Williams, R.: Simple statistical gradient-following algorithms for connectionnist reinforcement learning. Machine Learning **8** (1992) 229–256
10. Baxter, J., Bartlett, P.: Infinite-horizon policy-gradient estimation. Journal of Artificial Intelligence Research **15** (2001) 319–350

11. Baxter, J., Bartlett, P., Weaver, L.: Experiments with infinite-horizon, policy-gradient estimation. Journal of Artificial Intelligence Research **15** (2001) 351–381
12. Mausam, Weld, D.: Concurrent probabilistic temporal planning. In: Proc. of the 15th Int. Conf. on Planning and Scheduling (ICAPS'05). (2005)
13. Little, I., Aberdeen, D., Thiébaux, S.: Prottle: A probabilistic temporal planner. In: Proc. of the 20th American Nat. Conf. on Artificial Intelligence (AAAI'05). (2005)
14. Younes, H., Simmons, R.: Policy generation for continuous-time stochastic domains with concurrency. In: Proc. of the 14th Int. Conf. on Automated Planning and Scheduling (ICAPS'04). (2004)
15. Aberdeen, D., Thiébaux, S., Zhang, L.: Decision-theoretic military operations planning. In: Proc. of the 14th Int. Conf. on Automated Planning and Scheduling (ICAPS'04). (2004)
16. Hoffmann, J., Nebel, B.: The FF planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research **14** (2001) 253–302
17. Buffet, O.: Fast reachability analysis for uncertain ssps. In: Proc. of the IJCAI 2005 Workshop on Planning and Learning in A Priori Unknown or Dynamic Domains. (2005)
18. Peshkin, L., Kim, K., Meuleau, N., Kaelbling, L.: Learning to cooperate via policy search. In: Proc. of the 16th Conf. on Uncertainty in Artificial Intelligence (UAI'00). (2000)
19. Dutech, A., Buffet, O., Charpillet, F.: Multi-agent systems by incremental gradient reinforcement learning. In: Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI'01). (2001)
20. Tao, N., Baxter, J., Weaver, L.: A multi-agent, policy-gradient approach to network routing. In: Proc. of the 18th Int. Conf. on Machine Learning (ICML'01). (2001)
21. Munos, R.: Efficient resources allocation for Markov decision processes. In: Advances in Neural Information Processing Systems 13 (NIPS'01). (2001)
22. Nilim, A., Ghaoui, L.E.: Robustness in Markov decision problems with uncertain transition matrices. In: Advances in Neural Information Processing Systems 16 (NIPS'03). (2004)
23. Givan, R., Leach, S., Dean, T.: Bounded parameter Markov decision processes. Artificial Intelligence **122** (2000) 71–109
24. Hosaka, M., Horiguchi, M., Kurano, M.: Controlled Markov set-chains under average criteria. Applied Mathematics and Computation **120** (2001) 195–209
25. Strehl, A.L., Littman, M.L.: An empirical evaluation of interval estimation for Markov decision processes. In: Proc. of the 16th Int. Conf. on Tools with Artificial Intelligence (ICTAI'04). (2004)
26. Zinkevich, M.: Online convex programming and generalized infinitesimal gradient ascent. In: Proc. of the 20th Int. Conf. on Machine Learning (ICML'03). (2003)
27. Chang, Y.H., Kaelbling, L.: Playing is believing: the role of beliefs in multi-agent learning. In: Advances in Neural Information Processing Systems 14 (NIPS'01). (2001)
28. Bowling, M., Veloso, M.: Multiagent learning using a variable learning rate. Artificial Intelligence (2002) 215–250
29. Bowling, M.: Convergence and no-regret in multiagent learning. In: Advances in Neural Information Processing Systems 17 (NIPS'04). (2004) 209–216