The Factored Policy Gradient planner (IPC-06 Version)

Olivier Buffet and **Douglas Aberdeen**
National ICT Australia & The Australian National University
Canberra, Australia
firstname.lastname@nicta.com.au

Abstract

We present the Factored Policy Gradient (FPG) planner: a probabilistic temporal planner designed to scale to large planning domains by applying two significant approximations. Firstly, we use a "direct" policy search in the sense that we attempt to directly optimise a parameterised plan using gradient ascent. Secondly, the policy is factored into a per action mapping from a partial observation to the probabilility of executing, reflecting how desirable each action is. These two approximations — plus memory use that is independent of the number of states — allow us to scale to significantly larger planning domains than were previously feasible. Unlike other probabilistic temporal planners, FPG can attempt to optimise *both* makespan and the probability of reaching the goal. The version of FPG used in the IPC-06 competition optimises the makespan only, and turns off concurrent planning.

Introduction

To date, only a few planning tools have attempted to handle general probabilistic temporal planning domains. These tools have only been able to produce good or optimal policies for relatively small or easy problems. We designed the Factored Policy Gradient (FPG) planner with the goal of creating tools that produce good policies in real-world domains rather than perfect policies in toy domains. We achieve this by: 1) using gradient ascent for direct policy search; 2) factoring the policy into simple approximate policies for starting each action; 3) presenting each policy with critical observations instead of the entire state (implicitly aggregating similar states); and 4) using Monte-Carlo style algorithms with memory requirements that are independent of the state space size.

The AI planning community is familiar with the value-estimation class of reinforcement learning (RL) algorithms, such as RTDP (Barto, Bradtke, & Singh 1995), and arguably AO*. These algorithms represent probabilistic planning problems as a state space and estimate the long-term value, utility, or cost of choosing each action from each state (Mausam & Weld 2005; Little, Aberdeen, & Thiébaux 2005). The fundamental disadvantage of such algorithms is the need to estimate the values of a huge number of state-action pairs. Even algorithms that prune most states still fail to scale due to the exponential increase of relevant states as the domains grow.

On the other hand, the FPG planner borrows from Policy-Gradient reinforcement learning. This class of algorithms does not estimate planning state-action values. Instead, policy-gradient RL algorithms estimate the gradient of the unique long-term average reward of the process. In the context of stochastic shortest path problems, such as the IPC-06 domains, we can view this as estimating the gradient of long-term value of only the initial state. Gradients are computed with respect to the parameters governing the choice of actions at each decision point. These parameters summarise the policy, or plan, of the system. Stepping the parameters in the direction given by the gradient increases the expected return, or value from the initial state. Specifically, we will use the OLPOMDP policy-gradient RL algorithm described by Baxter, Bartlett, & Weaver (2001).

The policy takes the form of a parameterised function that accepts a description of the planning state as input, and returns a probability distribution over legal actions. In our temporal planning setting, an *action* is defined as a single *grounded* durative-action (in the PDDL 2.1 sense).

We factor the parameterised policy by using a function approximator for each action. Only when an action's preconditions are satisfied do we evaluate the desirability (as a probability of executing at this decision point) of the action. By doing this, the number of policy parameters grows linearly with the number of actions and predicates. Our parameterised action policy is a simple multi-layer perceptron that takes the truth value of the predicates at the current planning state, and outputs a probability distribution over whether to start the action. We require that the truth value of the predicates be a good (but not necessarily complete) indicator of the total state of the plan so far.

Background

Input Language

FPG's input language is the complete language handled by `mdpsim`, i.e., PDDL with some minor extensions (Younes & Littman 2004; Younes *et al.* 2005). Indeed, FPG is using mdpsim's data structures and functions to implement the planning domain simulator.

POMDP Formulation of Planning

We deliberately use factored policies that only consider partial state information. Policy gradient methods still converge under partial observability, but their value-based counterparts may not (Singh, Jaakkola, & Jordan 1994).

A finite partially observable Markov decision process consists of: a finite set of states $s \in \mathcal{S}$; a finite set of actions $a \in \mathcal{A}$; probabilities $\mathbb{P}[s'|s,a]$ of making state transition $s \to s'$ under action a; a reward for each state $r(s) : \mathcal{S} \to \mathbb{R}$; and a finite set of observation vectors $\mathbf{o} \in \mathcal{O}$ seen by action policies in lieu of complete state descriptions. FPG draws observations deterministically given the state, but more generally observations may be stochastic. *Goal states* occur when the predicates match a goal state specification. From *failure states* it is impossible to reach a goal state, usually because time or resources have run out. These two classes of state combine to form the set of *terminal* states that produce an immediate reset to the initial state s_0. A single trajectory through the state space, used to estimate gradients, consists of concatenated simulated plan executions that reset to s_0 when a terminal state is reached.[1]

Policies are stochastic, mapping observation vectors $\mathbf{o}$ to a probability over actions. For FPG, an action a is an integer in $[1, N]$, where N is the number of available grounded actions. The probability of action a is $\mathbb{P}[a|\mathbf{o},\boldsymbol{\theta}]$, where conditioning on $\boldsymbol{\theta}$ reflects the fact that the policy is controlled by a set of real valued parameters $\boldsymbol{\theta} \in \mathbb{R}^p$. The maximum makespan of a plan is limited, ensuring that all stochastic policies reach reset states in finite time when executed from s_0.

The GPOMDP algorithm maximises the long-term average reward

$$R(\boldsymbol{\theta}) := \lim_{T\to\infty} \frac{1}{T}\, \mathbb{E}_{\boldsymbol{\theta}} \left[\sum_{t=1}^{T} r(s_t) \right], \qquad (1)$$

where the expectation $\mathbb{E}_{\boldsymbol{\theta}}$ is over the distribution of state trajectories $\{s_0, s_1, \dots\}$ induced by $\boldsymbol{P}(\boldsymbol{\theta})$. In the context of planning, the instantaneous reward provides the action policies with a measure of progress toward the goal. Our simple reward scheme is to set $r(s) = 1000$ for all states s that represent the goal state, and 0 for all other states. To maximise $R(\boldsymbol{\theta})$, *goal* states must be reached as frequently as possible. This has the desired property of simultaneously minimising plan duration and maximising the probability of reaching the goal (failure states achieve no reward).

Planning State Space

As already mentionned, this implementation of FPG is using `mdpsim`'s data structures and functions (Younes *et al.* 2005). Thus, a state includes all true predicates as well as the value of each function. The observation vector used by FPG needs to have one entry for each predicate that could be true at some point. Thus, a first step (once the problem has been loaded) is to generate these predicates, which is done simultaneously when `mdpsim` grounds actions.

To estimate gradients we need a plan execution simulator to generate a trajectory through the planning state space. Here again, FPG's simple solution is to use `mdpsim`'s `next()` function, which samples a next state s' given current state s and chosen action a.

Policy-Gradient Reinforcement Learning

Each action a determines a stochastic matrix $\boldsymbol{P}(a) = [\mathbb{P}[s'|s,a]]$ of transition probabilities from state s to state s' given action a. The gradient estimator discussed in this paper does not assume explicit knowledge of $\boldsymbol{P}(a)$ or of the observation process.

All policies are stochastic, with a probability of choosing action a given state s, and parameters $\boldsymbol{\theta} \in \mathbb{R}^n$ of $\mathbb{P}[a|\mathbf{o},\boldsymbol{\theta}]$. During the course of optimisation the policy becomes more deterministic. The evolution of the state s is Markovian, governed by an $|\mathcal{S}| \times |\mathcal{S}|$ transition probability matrix $\boldsymbol{P}(\boldsymbol{\theta}) = [\mathbb{P}[s'|\boldsymbol{\theta},s]]$ with entries given by

$$\mathbb{P}[s'|\boldsymbol{\theta},s] = \sum_{a\in\mathcal{A}} \mathbb{P}[a|\mathbf{o},\boldsymbol{\theta}]\, \mathbb{P}[s'|s,a]\,. \qquad (2)$$

GPOMDP is an infinite-horizon policy-gradient method to compute the gradient of the *long-term average reward* (1) with respect the policy parameters $\boldsymbol{\theta}$. In this abstract we give only the gradient estimator customised for planning. For the derivation of the gradient estimator, and proofs of convergence, please refer to Baxter, Bartlett, & Weaver (2001).

Policy-Gradient for Planning

The desirability of some eligible action i in the set of actions with satisfied preconditions E_t is a real value $d(i)$ computed by a multi-layer perceptron. This preceptron usually has at most one hidden layer, and its weight vector $\boldsymbol{\theta}_i$ is part of the complete vector of parameters $\boldsymbol{\theta}$ learned by reinforcement. With input vector $\mathbf{o}$, the perceptron computes $d(i) = f_i(\mathbf{o}_t;\boldsymbol{\theta}_i)$.

Action a_t is sampled from a probability distribution obtained by computing a Gibbs[2] distribution over $d(i)$'s of eligible actions as follows:

$$\mathbb{P}[a_t = i|\mathbf{o}_t,\boldsymbol{\theta}] = \frac{exp(f_i(\mathbf{o}_t;\boldsymbol{\theta}_i))}{\sum_{j\in E_t} exp(f_j(\mathbf{o}_t;\boldsymbol{\theta}_j))}. \qquad (3)$$

Initially, the parameters are set to small random values giving a near uniform random policy; encouraging exploration of the action space. Each gradient step typically moves the parameters closer to a deterministic policy. After some experimentation we chose an observation vector that is a binary description of predicates current truth values plus a constant 1 bit to provide bias to the preceptron.

The observations are simply the truth value of the predicates for the current state.

[1] This concatenation trick is simply used to formulate the SSP planning in the framework used in Baxter, Bartlett, & Weaver (2001). In practice we can take advantage of the episodic nature of the problem.

[2] Essentially the same as a Boltzmann or soft-max distribution.

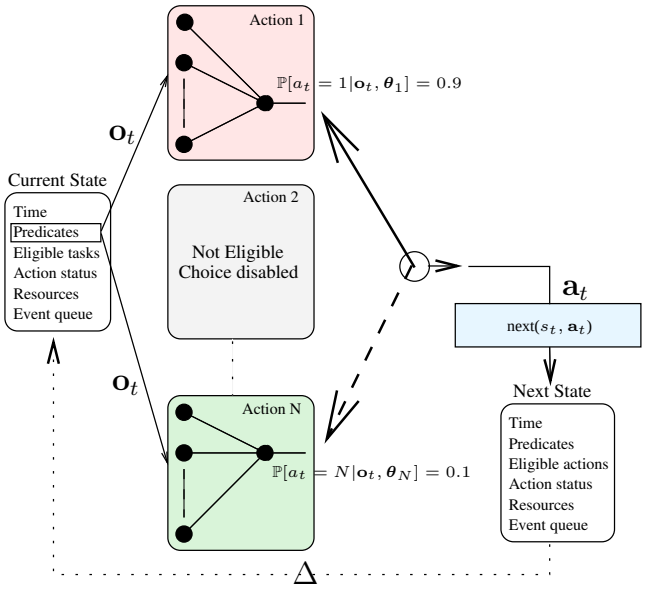

Fig. 1: Individual action-policies make independent decisions.

The FPG Gradient Estimator

Alg. 1 completes our description of FPG by showing how to implement OLPOMDP for planning with factored action policies. The algorithm works by sampling a single long trajectory through the state space: 1) the first state represents time 0 in the plan; 2) the perceptrons attached to eligible actions all receive the vector observation $\mathbf{o}_t$ of the current state s_t; 3) each network computes the desirability of its action; 4) a planning action is sampled; 5) the state transition is sampled; 6) the planner receives the global reward for the new state action and produces an instantaneous gradient $\mathbf{g}_t = r_t \mathbf{e}_t$; 7) all parameters are immediately updated in the direction of $\mathbf{g}_t$.

Algorithm 1 OLPOMDP FPG Gradient Estimator

1: Set s_0 to initial state, $t = 0, \mathbf{e}_t = [0]$, init $\boldsymbol{\theta}_0$ randomly
2: **while** R not converged **do**
3: $\quad \mathbf{e}_t = \beta \mathbf{e}_{t-1}$
4: $\quad$ Extract predicate values as observation $\mathbf{o}_t$ of s_t
5: $\quad$ **for** Each eligible action i **do**
6: $\quad\quad$ Evaluate desirability $d(i) = f_i(\mathbf{o}_t; \boldsymbol{\theta}_{ti})$
7: $\quad$ Sample action i with probability $\mathbb{P}[a_t = i|\mathbf{o}_t; \boldsymbol{\theta}_t]$
8: $\quad \mathbf{e}_t = \mathbf{e}_{t-1} + \nabla \log \mathbb{P}[a_t|\mathbf{o}_t; \boldsymbol{\theta}_t]$
9: $\quad s_{t+1}$ = next(s_t, $\mathbf{a}_t$)
10: $\quad \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha r_t \mathbf{e}_{t+1}$
11: $\quad$ **if** s_{t+1}.isTerminalState **then** $s_{t+1} = s_0$
12: $\quad t \leftarrow t + 1$

Because planning is inherently episodic we could alternatively set $\beta = 1$ and reset $\mathbf{e}_t$ every time a terminal state is encountered. However, empirically, setting $\beta = 0.9$ performed better than resetting $\mathbf{e}_t$.[3] The gradient for parameters not relating to eligible actions is 0. We do not compute $f_i(\mathbf{o}_t; \boldsymbol{\theta}_i)$ or gradients for actions with unsatisfied preconditions. Line 11 resets to the initial planning state upon encountering a terminal state.

[3]Perhaps because $\beta < 1$ can reduce the variance of gradient estimates, even in the episodic case.

Conclusion

FPG diverges from traditional planning approaches in two key ways: we search for plans directly, using a local optimisation procedure (an on-line gradient ascent); and we simplify the plan representation by factoring the plan into a function approximator for each action, each of which observes only a stripped down version of the current state.

The drawback of our approach is that local optimisation, simplified parameterisations, and reduced observability can all lead to sub-optimal plans; but this sacrifice is deliberate in order to achieve scalability through memory use and *per step* computation times that grow linearly with the domain. However, the total number of gradient steps is a function of the *mixing time* of the underlying POMDP, which can grow exponentially with the state variables. How to compute the mixing time of an arbitrary MDP is an open problem. This hints at the hardness of assessing in advance the difficulty of general planning problems.

FPG is a planner with great potential to produce *good* policies in large domains, especially considering the version handling concurrency. Further work will refine our parameterised action policies, apply more sophisticated stochastic gradient ascent methods, and attempt to characterise possible local minima.

Acknowledgments

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

Barto, A.; Bradtke, S.; and Singh, S. 1995. Learning to act using real-time dynamic programming. *Artificial Intelligence* 72.

Baxter, J.; Bartlett, P.; and Weaver, L. 2001. Experiments with infinite-horizon, policy-gradient estimation. *JAIR* 15:351–381.

Little, I.; Aberdeen, D.; and Thiébaux, S. 2005. Prottle: A probabilistic temporal planner. In *Proc. AAAI'05*.

Mausam, and Weld, D. S. 2005. Concurrent probabilistic temporal planning. In *Proc. International Conference on Automated Planning and Scheduling*. Moneteray, CA: AAAI.

Singh, S.; Jaakkola, T.; and Jordan, M. 1994. Learning without state-estimation in partially observable Markovian decision processes. In *Proceedings of ICML 1994*, number 11.

Younes, H. L. S., and Littman, M. L. 2004. PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects. Technical Report CMU-CS-04-167, Carnegie Mellon University.

Younes, H. L. S.; Littman, M. L.; Weissman, D.; and Asmuth, J. 2005. The first probabilistic track of the international planning competition. *Journal of Artificial Intelligence Research* 24:851–887.