FF+FPG: Guiding a Policy-Gradient Planner

Olivier Buffet
LAAS-CNRS
University of Toulouse
Toulouse, France
firstname.lastname@laas.fr

Douglas Aberdeen
National ICT australia &
The Australian National University
Canberra, Australia
firstname.lastname@anu.edu.au

Abstract

The Factored Policy-Gradient planner (FPG) (Buffet & Aberdeen 2006) was a successful competitor in the probabilistic track of the 2006 International Planning Competition (IPC). FPG is innovative because it scales to large planning domains through the use of Reinforcement Learning. It essentially performs a stochastic local search in policy space. FPG's weakness is potentially long learning times, as it initially acts randomly and progressively improves its policy each time the goal is reached. This paper shows how to use an external teacher to guide FPG's exploration. While any teacher can be used, we concentrate on the actions suggested by FF's heuristic (Hoffmann 2001), as FF-replan has proved efficient for probabilistic re-planning. To achieve this, FPG must learn its own policy while following another. We thus extend FPG to off-policy learning using importance sampling (Glynn & Iglehart 1989; Peshkin & Shelton 2002). The resulting algorithm is presented and evaluated on IPC benchmarks.

Introduction

The Factored Policy-Gradient planner (FPG) (Buffet & Aberdeen 2006; Aberdeen & Buffet 2007) was an innovative and successful competitor in the 2006 probabilistic track of the International Planning Competition (IPC). FPG's approach is to learn a parameterized policy –such as a neural network– by reinforcement learning (RL), reminiscent of stochastic local search algorithms for SAT problems. Other probabilistic planners rely either on a search algorithm (Little 2006) or on dynamic programming (Sanner & Boutilier 2006; Teichteil-Königsbuch & Fabiani 2006).

Because FPG uses policy-gradient RL (Williams 1992; Baxter, Bartlett, & Weaver 2001), its space complexity is not related to the size of the state-space but to the small number of parameters in its policy. Yet, a problem's hardness becomes evident in FPG's learning time (speaking of sample complexity). The algorithm follows an initially random policy and slowly improves its policy each time a goal is reached. This works well if a random policy eventually reaches a goal in a short time frame. But, in domains such as blocksworld, the average time before reaching the goal "by chance" grows exponentially with the number of blocks considered.

An efficient solution for probabilistic planning is to use a classical planner based on a determinized version of the problem, and replan when a state that has not been planned for is encountered. This is how FF-replan works (Yoon, Fern, & Givan 2004), relying on the Fast Forward (FF) planner (Hoffmann & Nebel 2001; Hoffmann 2001). FF-replan can perform poorly on domains where low-probability events can either be a key or give non-reliable solutions. FF-replan still proved more efficient than other probabilistic planners, somewhat because many of the domains were simple modifications of deterministic domains.

This paper shows how to combine a stochastic local search RL planners –developed in a machine learning context– with advanced heuristic search planners developed by the AI planning community. Namely, we combine FPG and FF to create a planner that scales well in domains such as blocksworld, while still reasoning about the domain in a probabilistic way. The key to this combination is the use of importance sampling (Glynn & Iglehart 1989; Peshkin & Shelton 2002) to create an off-policy RL planner initially guided by FF.

The paper starts with background knowledge on probabilistic planning, policy-gradient and FF-replan. The following section explains our approach through its two major aspects: the use of importance sampling on one hand and the integration of FF's help on the other hand. Then come experiments on some competition benchmarks and their analysis before a conclusion.

Background

Probabilistic Planning

A probabilistic planning *domain* is defined by a finite set of boolean variables $\mathcal{B} = \{b_1, \dots, b_n\}$ –a *state* $s \in \mathcal{S}$ being described by an assignment of these variables, and often represented as a vector $\mathbf{s}$ of 0s and 1s– and a finite set of *actions* $\mathcal{A} = \{a_1, \dots, a_m\}$. An action a can be executed if its *precondition* $pre(a)$ –a logic formula on $\mathcal{B}$– is satisfied. If a is executed, a probability distribution $\mathbb{P}(\cdot|a)$ is used to sample one of its K *outcomes* $out_k(a)$. An outcome is a set of truth value assignments on $\mathcal{B}$ which is then applied to change the current state.

A probabilistic planning *problem* is defined by a planning domain, an initial state s_o and a *goal* G –a formula on $\mathcal{B}$ that needs to be satisfied. The aim is to find the plan that maximizes the probability of reaching the goal, and possibly minimizes the expected number of actions required. This takes the form of a policy $\mathbb{P}[a|\mathbf{s}]$ specifying the probability of picking action a in state s. In the remainder of this section, we see how FPG solves this with RL, and how FF-replan uses classical planning.

FPG

FPG addresses probabilistic planning as a Markov Decision Process (MDP): a reward function r is defined, taking value 1000 in any goal state, and 0 otherwise; a transition matrix $\mathbb{P}[s'|s,a]$ is naturally derived from the actions; the system resets to the initial state each time the goal is reached; and FPG tries to maximize the expected average reward. But rather than dynamic programming –which is costly when it comes to enumerating reachable states–, FPG computes gradients of a stochastic policy, implemented as a policy $\mathbb{P}[a|\mathbf{s};\boldsymbol{\theta}]$ depending on a parameter vector $\boldsymbol{\theta} \in \mathbb{R}^n$. We now present the learning algorithm, then the policy parameterization.

On-Line POMDP The On-Line POMDP policy-gradient algorithm (OLPOMDP) (Baxter, Bartlett, & Weaver 2001), and many similar algorithms (Williams 1992), maximize 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 the transition matrix and the policy. To maximize $R(\boldsymbol{\theta})$, *goal* states must be reached as frequently as possible. This has the desired property of simultaneously minimizing plan duration and maximizing the probability of reaching the goal (failure states achieve no reward).

A typical gradient ascent algorithm would repeatedly compute the gradient $\nabla_{\boldsymbol{\theta}} R$ and follow its direction. Because an exact computation of the gradient is very expensive in our setting, OLPOMDP relies on Monte-Carlo estimates generated by simulating the problem. At each time step of the simulation loop, it computes a one-step gradient $\mathbf{g}_t = r_t \mathbf{e}_t$ and immediately updates the parameters in the direction of $\mathbf{g}_t$. The eligibility vector $\mathbf{e}_t$ contains the discounted sum of normalized action probability gradients. At each step, r_t indicates whether to move the parameters in the direction of $\mathbf{e}_t$ to promote recent actions, or away from $\mathbf{e}_t$ to deter recent actions (Algorithm 1).

OLPOMDP is "on-line" because it updates parameters for every non-zero reward. It is also "on-policy" in the RL sense of requiring trajectories to be generated according to $\mathbb{P}[\cdot|\mathbf{s}_t;\boldsymbol{\theta}_t]$. Convergence to a (possibly poor) locally optimal policy is still guaranteed even if some state information (e.g., resource levels) is omitted from $\mathbf{s}_t$ for the purposes of simplifying the policy representation.

Linear-Network Factored Policy The policy used by FPG is *factored* because it is made of one linear network per action, each of them taking the same vector $\mathbf{s}$ as input (plus a constant 1 bit to provide bias to the perceptron) and outputting a real value $f_i(\mathbf{s}_t;\boldsymbol{\theta}_i)$. In a given state, a probability distribution over eligible actions is computed as a Gibbs[1] distribution

$$\mathbb{P}[a_t = i|\mathbf{s}_t;\boldsymbol{\theta}] = \frac{exp(f_i(\mathbf{s}_t;\boldsymbol{\theta}_i))}{\sum_{j\in\mathcal{A}} exp(f_j(\mathbf{s}_t;\boldsymbol{\theta}_j))}.$$

The interaction loop connecting the policy and the problem is represented in Figure 1.

Initially, the parameters are set to 0, giving a uniform random policy; encouraging exploration of the action space. Each gradient step typically moves the parameters closer to a deterministic policy.

Due to the availability of benchmarks and compatibility with FF we focus on the non-temporal IPC version of FPG. The temporal version extension simply gives each action a separate Gibbs distribution to determine if it will be executed, independently of other actions (mutexes are resolved by the simulator).

Algorithm 1 OLPOMDP FPG Gradient Estimator

Set s_0 to initial state, $t = 0$, $\mathbf{e}_t = [0]$, init $\boldsymbol{\theta}_0$ randomly
while R not converged **do**
  Compute distribution $\mathbb{P}[a_t = i|\mathbf{s}_t;\boldsymbol{\theta}_t]$
  Sample action i with probability $\mathbb{P}[a_t = i|\mathbf{s}_t;\boldsymbol{\theta}_t]$
  $\mathbf{e}_t = \beta\mathbf{e}_{t-1} + \nabla \log \mathbb{P}[a_t|\mathbf{s}_t;\boldsymbol{\theta}_t]$
  s_{t+1} = next(s_t, i)
  $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha r_t \mathbf{e}_t$
  if s_{t+1}.isTerminalState **then** $s_{t+1} = s_0$
  $t \leftarrow t+1$

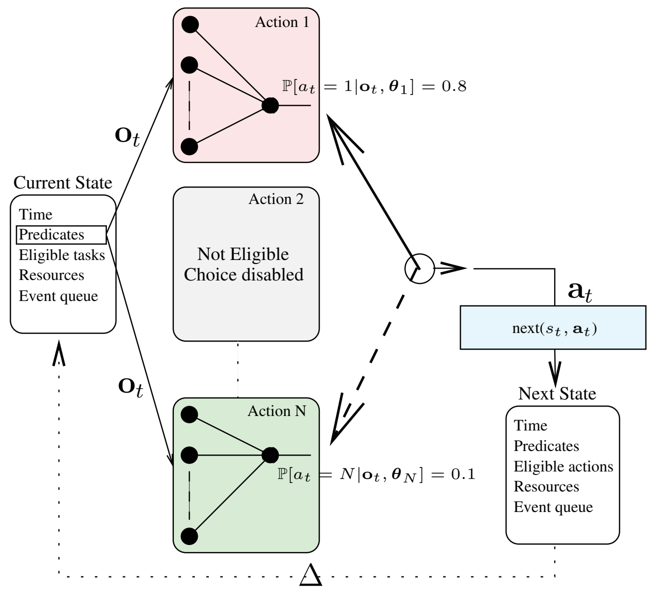

Figure 1: Individual action-policies make independent decisions.

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

Fast Forward (FF) and FF-replan

Fast Forward A detailed description of the Fast Forward planner (FF) can be found in Hoffmann & Nebel (2001) and Hoffmann (2001). FF is a forward chaining heuristic state space planner for deterministic domains. Its heuristic is based on solving –with a graphplan algorithm– a relaxation of the problem where negative effects are removed, which provides a lower bound on each state's distance to the goal. This estimate guides a local search strategy, *enforced hill-climbing* (EHC), in which one step of the algorithm looks for a sequence of actions ending in a strictly better state (better according to the heuristic). Because there is no backtracking in this process, it can get trapped in dead-ends. In this case, a complete best-first search (BFS) is performed.

FF-replan Variants of FF include metric-FF, conformant-FF and contingent-FF. But FF has also been successfully used in the probabilistic track of the international planning competition in a version called FF-replan (Yoon, Fern, & Givan 2004; 2007). FF-replan works by planning in a determinized version of the domain, executing its plan as long as no unexpected transition is met. In such a situation, FF is called for replanning from current state. One choice is how to turn the original probabilistic domain into a deterministic one. Two possibilities have been tried:

- in IPC4 (FF-replan-4): for each probabilistic action, keep its most probable outcome as the deterministic outcome; a drawback is that the goal may not be reachable anymore; and
- in IPC5 (FF-replan-5, not officially competing): for each probabilistic action, create one deterministic action per possible outcome; a drawback is that the number of actions grows quickly.

Both approaches are potentially interesting: the former should give more efficient plans if it is not necessary to rely on low-probability outcomes of actions (it is necessary in Zenotravel), but will otherwise get stuck in some situations.

Simple experiments with the blocksworld show that FF-replan-5 can prefer to execute actions that, with a low probability, achieve the goal very fast. E.g., the use of `put-on-block ?b1 ?b2` when `put-down ?b1` would be equivalent –to put a block `?b1` on the table– and safer from the point of view of reaching the goal with the highest probability. This illustrates the drawback of the FF approach of determinizing the domain. Good translations somewhat avoid this by removing action B in cases where

- actions A and B have the same effects,
- action A's preconditions are less or equally restrictive as action B's preconditions, and
- action A is more probable than action B.

Note: at the time of this work, no details about FF-replan were published, which is now fixed (Yoon, Fern, & Givan 2007).

Off-Policy FPG

FPG relies on OLPOMDP, which assumes that the policy being learned is the one used to draw actions while learning. As we intend to also take FF's decisions into account while learning, OLPOMDP has to be turned into an off-policy algorithm by the use of importance sampling.

Importance Sampling

Importance sampling (IS) is typically presented as a method for reducing the variance of the estimate of an expectation by carefully choosing a sampling distribution (Rubinstein 1981). For a random variable X distributed according to p, $\mathbb{E}_p[f(X)]$ is estimated by $\frac{1}{n}\sum_i f(x_i)$ with i.i.d samples $x_i \sim p(x)$. But a lower variance estimate can be obtained with a sampling distribution q having higher density where $\|f(x)\|$ is larger. Drawing $x_i \sim q(x)$, the new estimate is $\frac{1}{n}\sum_i f(x_i)K(x_i)$, where $K(x_i) = \frac{p(x_i)}{q(x_i)}$ is the *importance coefficient* for sample x_i.

IS for OLPomdp: Theory

Unlike Shelton (2001), Meuleau, Peshkin, & Kim (2001) and Peshkin & Shelton (2002), we do not want to estimate $R(\boldsymbol{\theta})$ but its gradient. Rewriting the gradient estimation given by Baxter, Bartlett, & Weaver (2001), we get:

$$\hat{\nabla}R(\boldsymbol{\theta}) = \sum_X r(X)\frac{\nabla p(X)}{p(X)}\frac{p(X)}{q(X)}q(X),$$

where the random variable X is sampled according to distribution q rather than its real distribution p. In our setting, a sample X is a sequence of states $s_0, \ldots, s_t$ obtained while drawing a sequence of actions $a_0, \ldots, a_t$ from $\mathbb{Q}[a|s;\boldsymbol{\theta}]$ (the teacher's policy). This leads to:

$$\hat{\nabla}R(\boldsymbol{\theta}) = \sum_{t=0}^{T} r(s_t)\frac{\nabla p(s_t)}{p(s_t)}\frac{p(s_t)}{q(s_t)},$$

where

$$p(s_t) = \prod_{t'=0}^{t-1}\mathbb{P}[a_{t'}|s_{t'};\boldsymbol{\theta}]\,\mathbb{P}[s_{t'+1}|s_{t'},a_{t'}],$$

$$q(s_t) = \prod_{t'=0}^{t-1}\mathbb{Q}[a_{t'}|s_{t'};\boldsymbol{\theta}]\,\mathbb{P}[s_{t'+1}|s_{t'},a_{t'}], \text{ and}$$

$$\frac{\nabla p(s_t)}{p(s_t)} = \sum_{t'=0}^{t-1}\frac{\nabla\left(\mathbb{P}[a_{t'}|s_{t'};\boldsymbol{\theta}]\,\mathbb{P}[s_{t'+1}|s_{t'},a_{t'}]\right)}{\mathbb{P}[a_{t'}|s_{t'};\boldsymbol{\theta}]\,\mathbb{P}[s_{t'+1}|s_{t'},a_{t'}]}$$

$$= \sum_{t'=0}^{t-1}\frac{\nabla\,\mathbb{P}[a_{t'}|s_{t'};\boldsymbol{\theta}]}{\mathbb{P}[a_{t'}|s_{t'};\boldsymbol{\theta}]}, \text{ hence}$$

$$\frac{\nabla p(s_t)}{p(s_t)}\frac{p(s_t)}{q(s_t)} = \frac{\prod_{t'=0}^{t-1}\mathbb{P}[a_{t'}|s_{t'};\boldsymbol{\theta}]}{\prod_{t'=0}^{t-1}\mathbb{Q}[a_{t'}|s_{t'};\boldsymbol{\theta}]}\sum_{t'=0}^{t-1}\frac{\nabla\,\mathbb{P}[a_{t'}|s_{t'};\boldsymbol{\theta}]}{\mathbb{P}[a_{t'}|s_{t'};\boldsymbol{\theta}]}.$$

The off-policy update of the eligibility trace is then:

$$\mathbf{e}_{t+1} = \mathbf{e}_t + K_{t+1}\nabla\log\mathbb{P}[a_t|s_t;\boldsymbol{\theta}], \text{ where}$$

$$K_{t+1} = \frac{\prod_{t'=0}^{t}\mathbb{P}[a_{t'}|s_{t'};\boldsymbol{\theta}]}{\prod_{t'=0}^{t}\mathbb{Q}[a_{t'}|s_{t'};\boldsymbol{\theta}]}, = K_t\frac{\mathbb{P}[a_t|s_t;\boldsymbol{\theta}]}{\mathbb{Q}[a_t|s_t;\boldsymbol{\theta}]}.$$

IS for OLPomdp: Practice

It is known that there are possible instabilities if the true distribution differs a lot from the one used for sampling, which is the case in our setting. Indeed, K_t is the probability of a trajectory if generated by $\mathbb{P}$ divided by the probability of the same trajectory if generated by $\mathbb{Q}$. This typically converges to 0 when the horizon increases. *Weighted* importance sampling solves this by normalizing each IS sample by the *average* importance co-efficient. This is normally performed in a batch setting, where the gradient is estimated from several runs before following its direction. With our online policy-gradient ascent we use an equivalent batch size of 1. The update becomes:

$$K_{t+1} = \frac{1}{t}\sum_{t'=1}^{t} k_{t'}, \text{ and } k_t = \frac{\mathbb{P}[a_t|s_t;\boldsymbol{\theta}]}{\mathbb{Q}[a_t|s_t;\boldsymbol{\theta}]},$$
$$\mathbf{e}_{t+1} = \mathbf{e}_t + K_{t+1}\nabla \log \mathbb{P}[a_t|s_t;\boldsymbol{\theta}],$$
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \frac{1}{K_{t+1}} r\mathbf{e}_{t+1}.$$

Learning from FF

We have turned FF into a library (LIBFF) that makes it possible to ask for FF's action in a given state. There are two versions:

- EHC: use enforced hill climbing only, or
- EHC+BFS: do a breadth first search if EHC fails.

Often, the current state appears in the last plan found, so that the corresponding action is already in memory. Plus, to make LIBFF more efficient, we cache frequently encountered state-action suggestions.

Choice of the Sampling Distribution

Off-policy learning requires that each trajectory possible under the true distribution be possible under the sampling distribution. Because FF acts deterministically in any state, the sampling distribution cannot be based on FF's decisions alone. Two candidate sampling distributions are:

1. FF(ϵ)+uni($1-\epsilon$): use FF with probability ϵ, a uniform distribution with probability $1-\epsilon$; and
2. FF(ϵ)+FPG($1-\epsilon$): use FF with probability ϵ, FPG's distribution with probability $1-\epsilon$.

As long as $\epsilon \neq 1$, the resulting sampling distribution has the same support as FPG. The first distribution favors a small constant degree of uniform exploration. The second distribution mixes the FF suggested action with FPG's distribution, and for high ϵ we expect FPG to learn to mimic FF's action choice closely. Apart from the expense of evaluating the teacher's suggestion, the additional computational complexity of using importance sampling is negligible. An obvious idea is to reduce ϵ over time, so that FPG takes over completely, however the rate of this reduction is highly domain dependent, so we chose a fixed ϵ for the majority of optimization, reverting to standard FPG towards the end of optimization.

FF+FPG in Practice

Both FF and FPG accept the language considered in the competition (with minor exceptions), i.e., PDDL with extensions for probabilistic effects (Younes *et al.* 2005). Note that source code is available for FF[2], FPG[3] and libPG[4] (the policy-gradient library used by FPG). Excluding parameters specific to FF or FPG, one has to choose:

1. whether to translate the domain in either an IPC4 or IPC5 type deterministic domain for FF;
2. whether to use EHC or EHC+BFS;
3. $\epsilon \in (0, 1)$; and
4. how long to learn with and without a teacher.

Experiments

The aim is to let FF help FPG. Thus the experiments will focus on problems from the 5th international planning competition for which FF clearly outperformed FPG, in particular the Blocksworld and Pitchcatch domains. In the other 6 IPC domains FPG was close to, or better, than the version of FF we implemented. However, we begin by analyzing the behavior of FF+FPG.

Simulation speed The speed of the simulation+learning loop in FPG (without FF) essentially depends on the time taken for simple matrix computation. FF, on the other hand, enters a complete planning cycle for each new state, slowing down planning dramatically in order to help FPG reach a goal state. Caching FF's answers greatly reduced the slowdown due to FF. Thus, an interesting reference measure is the number of simulation steps performed in 10 minutes while not learning –FPG's default behavior being a random walk– as it helps evaluating how time-consuming the teacher is.

Various settings are considered: the teacher can be EHC, EHC+BFS or none; and the type of deterministic domain is IPC4 (most probable effects) or IPC5 (all effects). Table 1 gives results for the blocksworld[5] problems p05 and p10 (involving respectively 5 and 10 blocks), with different ϵ values. Having no teacher is here equivalent to no learning at all as there are very few successes.

Considering the number of simulation steps, we observe that EHC is faster than EHC+BFS only for p05, with $\epsilon = 0.5$. Indeed, if one run of EHC+BFS is more time-consuming, it usually caches more future states, which are only likely to be re-encountered if $\epsilon = 1$. With p05, the score of the fastest teacher (4072.10^3) is close to the score of FPG alone (4426.10^3), which reflects the predominance of matrix computations compared to FF's reasoning. But this changes with p10, where the teacher becomes necessary to get FPG to the goal in a reasonable number of steps. Finally, we clearly observed that the simulation speeds up as the cache fills up.

[2] `http://members.deri.at/~joergh/ff.html`

[3] `http://fpg.loria.fr/`

[4] `http://sml.nicta.com.au/~daa/software.htm`

[5] Errors appear in this blocksworld domain, but we use it as a

Table 1: Number of simulation steps ($\times 10^3$), [number of successes ($\times 10^3$)] and (average reward) in 10 minutes in the Blocksworld

problem	p05, $\epsilon = 0.5$		p05, $\epsilon = 1$		p10, $\epsilon = 1$	
domain	IPC4	IPC5	IPC4	IPC5	IPC4	IPC5
no teacher	4426 [0.022] (4.96e-3)				275*5=1375 [0*5] (0*5)	
EHC	3525 [1.9] (0.5)	3171 [7.9] (2.4)	3547 [5.2] (1.5)	4072 [26.5] (6.5)	495 [0.05] (0.1)	595 [1.0] (1.7)
EHC + BFS	2156 [7.6] (3.6)	1518 [10.1] (6.7)	3610 [199.2] (55.2)	2136 [65.5] (30.7)	514 [10.3] (20.0)	559 [5.0] (9.0)

Note: FPG with no teacher stopped after 2 minutes in p10, because of its lack of success. (Experiments performed on a P4-2.8GHz.)

Success Frequency Another important aspect of the choice of a teacher is how efficiently it achieves rewards. Two interesting measures are: 1) the number of successes that shows how rewarding these 10 minutes have been; and 2) the average reward per time step (which is what FPG optimizes directly).

As can be expected, both measures increase with ϵ ($\epsilon = 0$ implies no teacher) and decrease with the size of the problem. With a larger problem, there is a cumulative effect of FF's reasoning being slower and the number of steps to the goal getting larger. Unsurprisingly, EHC+BFS is more efficient than EHC alone when wall-clock time is not important. Also, unsurprisingly in blocksworld, IPC4 determinizations are better than IPC5, due to the fact that blocksworld is made probabilistic by giving the highest probability (0.75) to the normal deterministic effect.

Learning Dynamics

We look now at the dynamics of FPG while learning, focusing on two difficult but still accessible problems: Blocksworld/p10 and Pitchcatch/p07. EHC+BFS was applied in both cases. Pitchcatch/p07 required an IPC5-type domain, while IPC4 was used for blocksworld/p10. Figures 2 and 3 show the average number of successes per time step when using FPG alone or FPG+FF. But, as can be observed on Table 1, FPG's original random walk does not initially find the goal by chance. To overcome this problem, the competition version of FPG implemented a simple progress estimator –counting how many facts from the goal are gained or lost in a transition– to modify the reward function, i.e., reward shaping. This leads us to also consider results with and without the progress estimator (the measured average reward not taking it into account).

In the experiments –performed on a P4-2.8GHz– the teacher is always used during the first 60 seconds (for a total learning time of 900 seconds, as in the competition). The settings include two learning step sizes: α and α_{tea} (a specific step size while teaching). If a progress estimator is used, each goal fact made true (respectively false) brings a reward of +100 (resp. -100). Note that we used our own simple implementation of FF-replan. Based on published results, the IPC FF-Replan (Yoon, Fern, & Givan 2004) performs slightly better. The curves appearing on Fig. 2 and 3 are over a single run, in a view to exhibit typical behaviors –which have been observed repeatedly–. No accurate comparison between the various settings should be done.

On Fig. 2, it appears that the progress estimator is not sufficient for Blocksworld/p10, so that no teacher-free approach starts learning. With the teacher used for 60 seconds, a first high-reward phase is observed before a sudden fall when teaching stops. Yet, this is followed by a progressive growth up to higher rewards than with just the teacher. Here, ϵ is high to ensure that the goal is met frequently. Combining the teacher and the progress estimator led to quickly saturating parameters $\boldsymbol{\theta}$, causing numerical problems.

In Pitchcatch/p07, vanilla FPG fails, but the progress estimator makes learning possible, as shown on Fig. 3. Using the teacher or a combination of the progress estimator and the teacher also works. The three approaches give similar results. As with blocksworld, a decrease is observed when teaching ends, but the first phase is much lower than the optimum, essentially because ϵ is set to a relatively low 0.5.

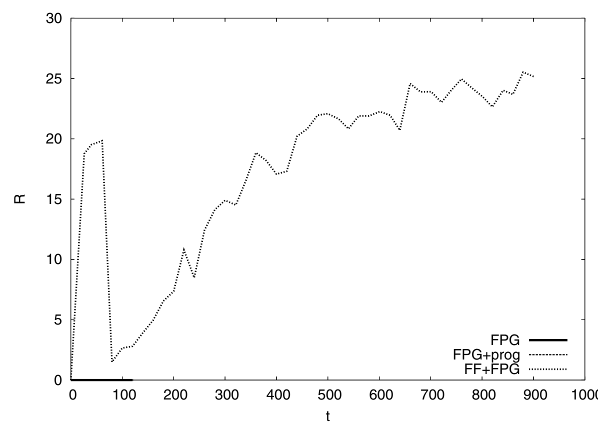

Figure 2: Average reward per time step on Blocksworld/p10 $\epsilon = 0.95$, $\alpha = 5.10^{-4}$, $\alpha_{tea} = 10^{-5}$, $\beta = 0.95$

Blocksworld Competition Results

We recreated the competition environment for the 6 hardest blocksworld problems, which the original IPC FPG planner struggled with despite the use of progress rewards. Optimization was limited to 900 seconds. The EHC+BFS teacher was used throughout the 900 seconds with $\epsilon = 0.9$ and discount factor $\beta = 1$ (the eligibility trace is reset after reaching the goal). The progress reward was *not* used. P10 contains 10 blocks, and the remainder contain 15. As in the competition, evaluation was over 30 trials of the problem. FF was *not* used during evaluation.

Table 2 shows the results. The IPC results were taken from the 2006 competition results. The FF row shows our implementation of the FF-based replanner without FPG, using the faster IPC-4 determinization of domains, hence the discrepancy with the IPC5-FF row. The results demonstrate

reference from the competition.

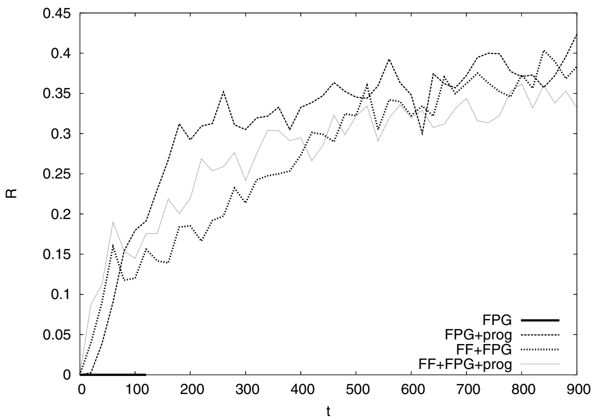

Figure 3: Average reward per time step on Pitchcatch/p07 $\epsilon = 0.5$, $\alpha = 5.10^{-4}$, $\alpha_{tea} = 10^{-5}$, $\beta = 0.85$

Table 2: Number of success out of 30 for the hardest probabilistic IPC5 blocksworld problems.

Planner	p10	p11	p12	p13	p14	p15
FF+FPG	30	29	0	3	30	30
FF	30	30	2	13	30	11
IPC5-FPG	13	0	0	0	0	0
IPC5-FF	30	30	27	1	0	30

that FPG is at least learning to imitate FF well, and particularly in the case of Blocksworld-P15 FPG bootstraps from FF to find a better policy. This is a very positive result considering how difficult these problems are.

Where FPG Fails: A XOR Problem

We present here some experiments on a toy problem whose optimal solution cannot be represented with the usual linear networks. In this "XOR" problem, the state is represented by two predicates A and B (randomly initialised), and the only two actions are α and β. Applying α if $A \neq B$ leads to a success, as well as applying β if $\neg(A \neq B)$. Any other decision leads to a failure.

Table 3 shows results for various planners, two function approximators being used within FPG: the usual linear network (noted "2L" because it is a 2-layer perceptron) and a 3-layer perceptron "3L" (with two hidden units). The observed results can be interpreted as follows:

- FPG(2L) finds the best policy it can express: it picks one action in 3 cases out of 4, and the other in the last case; there is a misclassification in only a quarter of all situations;
- FPG(3L) but usually falls in a local optimum achieving the same result as FPG(2L);
- FF always finds the best policy;
- with FF+FPG(2L), FPG tries –with no success– to learn the true optimal policy, as exhibited by FF; the result is a stochastic policy finding the appropriate action only half of the time;
- with FF+FPG(3L), FPG really learn FF's behavior, i.e. the optimal policy.

Table 3: Success probability on the XOR problem

FPG(2L)	FPG(3L)	FF	FF+FPG(2L)	FF+FPG(3L)
74%	81%	100%	44%	100%

Discussion

Because classical planners like FF return a plan quickly compared to probabilistic planners, using them as a heuristic input to probabilistic planners makes sense. Our experiments demonstrate that this is feasible in practice, and makes it possible for FPG to solve new problems efficiently, such as 15 block probabilistic blocksworld problems.

Choosing ϵ well for a large range of problems is difficult. Showing too much of a teacher's policy ($\epsilon \simeq 1$) will lead to copying this policy (provided it does reach the goal). This is close to supervised learning where one tries to map states to actions exactly as proposed by the teacher, which may be a local optimum. Avoiding local optima is made possible by more exploration ($\epsilon \simeq 0$), but at the expense of losing the teacher's guidance.

Another difficulty is finding an appropriate teacher. As we use it, FF proposes only one action (no heuristic value for each action), making it a poor choice for sampling distribution without mixing it with another. Computation times can be expensive, however this is more than offset by its ability to initially guide FPG to the goal in combinatorial domains. And the choice between IPC-4 and IPC-5 determinization of domains is not straightforward. There is space to improve FF which may result in FF being an even more competitive stand-alone planner, as well as assisting stochastic local search based planners. In particular, recently published details on the original implementation of FF-rePlan (Yoon, Fern, & Givan 2007) should help us develop a better replanner than the version we are using. In many situations, the best teacher would be a human expert. But importance sampling cannot be used straightforwardly in this situation.

In similar approach to ours, Mausam, Bertoli, & Weld (2007) use a *non-deterministic* planner to find potentially useful actions, whereas our approach exploits a heuristic borrowed from a classical planner.

Another interesting comparison is with Fern, Yoon, & Givan (2003) and Xu, Fern, & Yoon (2007). Here, the relationship between heuristics and learning is inverted, as the heuristics are learned rather than used for learning. Given a fixed planning domain, this can be an efficient way to gain knowledge from some planning problems and reuse it in more difficult situations.

Conclusion

FPG's benefits are that it learns a compact and factored representation of the final plan, represented as a set of parameters; and the per step learning algorithm complexity does not depend on the complexity of the problem. However FPG suffers in problems where the goal is difficult to achieve via initial random exploration. We have shown how to use a non-optimal planner to help FPG to find the goal, while still allowing FPG to learn a better policy than the original teacher, with initial success on IPC planning problems that FPG could not previously solve.

Acknowledgments

We thank Sungwook Yoon for his help on FF-replan. This work has been supported in part via the DPOLP project at NICTA.

References

Aberdeen, D., and Buffet, O. 2007. Temporal probabilistic planning with policy-gradients. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS'07)*.

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

Buffet, O., and Aberdeen, D. 2006. The factored policy gradient planner (ipc'06 version). In *Proceedings of the Fifth International Planning Competition (IPC-5)*.

Fern, A.; Yoon, S.; and Givan, R. 2003. Approximate policy iteration with a policy language bias. In *Advances in Neural Information Processing Systems 15 (NIPS'03)*.

Glynn, P., and Iglehart, D. 1989. Importance sampling for stochastic simulations. *Management Science* 35(11):1367–1392.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Hoffmann, J. 2001. FF: The fast-forward planning system. *AI Magazine* 22(3):57–62.

Little, I. 2006. Paragraph: A graphplan-based probabilistic planner. In *Proceedings of the Fifth International Planning Competition (IPC-5)*.

Mausam; Bertoli, P.; and Weld, D. S. 2007. A hybridized planner for stochastic domains. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*.

Meuleau, N.; Peshkin, L.; and Kim, K. 2001. Exploration in gradient-based reinforcement learning. Technical Report AI Memo 2001-003, MIT - AI lab.

Peshkin, L., and Shelton, C. 2002. Learning from scarce experience. In *Proceedings of the Nineteenth International Conference on Machine Learning (ICML'02)*.

Rubinstein, R. 1981. *Simulation and the Monte Carlo Method*. John Wiley & Sons, Inc. New York, NY, USA.

Sanner, S., and Boutilier, C. 2006. Probabilistic planning via linear value-approximation of first-order MDPs. In *Proceedings of the Fifth International Planning Competition (IPC-5)*.

Shelton, C. 2001. Importance sampling for reinforcement learning with multiple objectives. Technical Report AI Memo 2001-003, MIT AI Lab.

Teichteil-Königsbuch, F., and Fabiani, P. 2006. Symbolic stochastic focused dynamic programming with decision diagrams. In *Proceedings of the Fifth International Planning Competition (IPC-5)*.

Williams, R. 1992. Simple statistical gradient-following algorithms for connectionnist reinforcement learning. *Machine Learning* 8(3):229–256.

Xu, Y.; Fern, A.; and Yoon, S. 2007. Discriminative learning of beam-search heuristics for planning. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*.

Yoon, S.; Fern, A.; and Givan, R. 2004. FF-rePlan. http://www.ecn.purdue.edu/ sy/ffreplan.html.

Yoon, S.; Fern, A.; and Givan, B. 2007. FF-Replan: a baseline for probabilistic planning. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS'07)*.

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.