

Building Probabilistic Temporal Planning Tools

Doug Aberdeen
With Olivier Buffet and Sylvie Thiébaux

National ICT Australia,
Canberra Node,
NICTA next to Northbourne

13th January 2006



Australian Government

NICTA Members



Department of State and
Regional Development



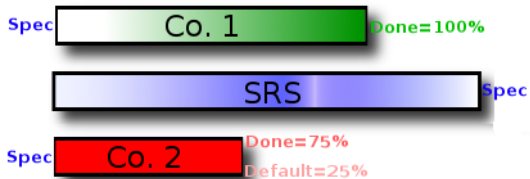
Outline

- 1 The Problem
- 2 Markov Decision Process Formalism
- 3 The MO Planner
- 4 The FPG Planner
- 5 Some Results

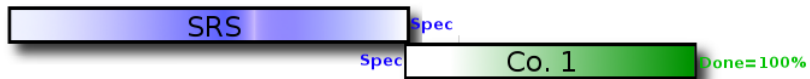
Outline

- 1 The Problem
- 2 Markov Decision Process Formalism
- 3 The MO Planner
- 4 The FPG Planner
- 5 Some Results

Task model



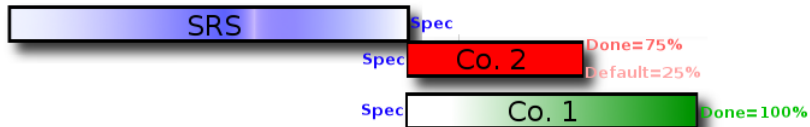
Plan for certainty



Plan for time



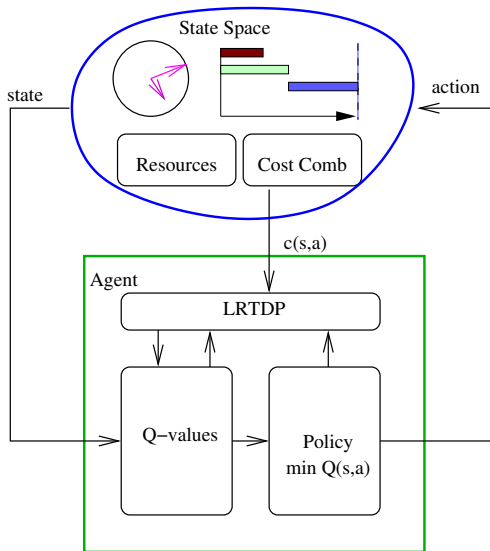
Plan for both



Outline

- 1 The Problem
- 2 Markov Decision Process Formalism**
- 3 The MO Planner
- 4 The FPG Planner
- 5 Some Results

Markov Decision Processes: Q-Values



Operations Planning States

MDP states must contain enough information to act optimally

- Current timestamp
- Available resources & occupied resources
- Current truth value for each state var.
- **Event queue:** future deterministic events

An action **a** is an order to start a subset of *eligible* tasks

Other formalisms need extension to uncertainties:

- Strips operators
- SAT
- Situation Calculus
- Constraint based solvers

Exploring the State Space

- Given current state s and action a , find $\Pr[s'|s, a]$
- For each action $a \in \{\text{powerset of eligible tasks}\}$, get \bar{s}

if terminal state **then** push \bar{s} on stack and **return**
process next event in queue

for each possible outcome o **do**

$s' \leftarrow \text{apply_effects}(o, \bar{s})$

$\Pr[s'] = \Pr[\bar{s}] \times \Pr[o]$

if ≥ 0 ready tasks **then**

push s' on stack

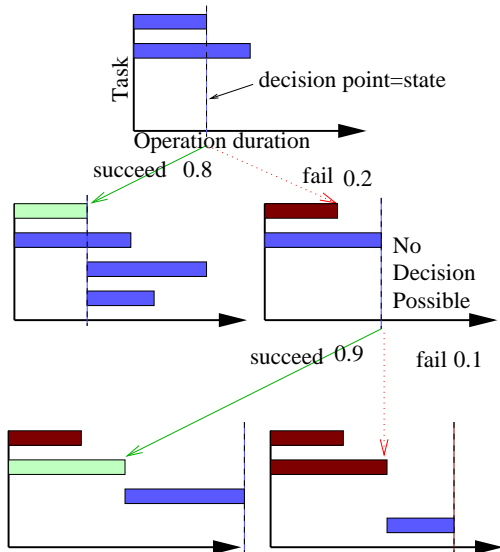
else

recurse with $\bar{s} \leftarrow s'$

end if

end for

Example of State Search



Outline

- 1 The Problem
- 2 Markov Decision Process Formalism
- 3 The MO Planner**
- 4 The FPG Planner
- 5 Some Results

Dynamic Programming

- Each planning state s incurs a cost $c(s)$, e.g., the resources consumed over that state
- Q: Which policy, $\pi(s)$, minimises

$$Q(s, a) = \mathbb{E} \left[\sum_{t=0}^{T: s_T \in \mathcal{T}} c(s_t) \mid s_0 = s, a_0 = a \right] ?$$

- Instead, compute the $Q(s, a)$ obtained with minimising policy:

$$Q_{t+1}(s, a) = c(s) + \min_{a'} \sum_{s' \in \mathcal{S}} \Pr[s' | s, a] Q_t(s', a')$$

(Dynamic Programming, a.k.a Value Iteration)

Dynamic Programming II

- A: The minimising policy is now easy to compute

$$\pi(s) = \arg \min_a Q(s, a)$$

- Asymptotic convergence
- End when $|Q_{t+1}(s, a) - V_t(s, a)| < \epsilon$ for all s over current policy

Real-Time Dynamic Programming

- Value iteration repeatedly loops over all states
 - Requires all states to be enumerated in memory
 - Spends as much time on low probability states as high probability states
- RTDP simulates the problem following the **greedy** policy
- Value updates are performed as states are encountered
- Retain **exploration** because bad states increase in value and leave current policy
- Branches of state space with high values are never fully explored
- Asymptotic convergence to optimal policy

Algorithm Outline

Real-Time Dynamic Programming

- $Q(s, a)$ approaches long-term cost of doing a in state s
- s_0 is the plan start state, time 0
- $c(s)$ is the combined cost incurred in state s
- Updates to $Q(s, a)$ are driven by simulation

while $Q(s_0, \cdot)$ has not converged **do**

$s \leftarrow s_0$

while s not a plan termination state **do**

$a \leftarrow \min_{a'} Q(s, a')$

$Q(s, a) \leftarrow c(s) + \sum_{s'} \Pr[s'|a, s] \min_{a'} Q(s', a')$

$s \leftarrow \sim \Pr[s'|a, s]$

end while

end while

Outline

- 1 The Problem
- 2 Markov Decision Process Formalism
- 3 The MO Planner
- 4 The FPG Planner**
- 5 Some Results

Policy-Gradient Methods

- We all know what gradient descent is?
- Policy-gradient methods learn **policies** $\Pr[\mathbf{a}|s, \mathbf{w}]$, not **values**
- They estimate the gradient of a long-term cost measure
- Gradient is with respect to the parameters \mathbf{w} that describe the policy
- The action parameters and state information is mapped by a function, e.g., a logistic regression function, to probabilities
- The policy is a decision for each eligible task: **run** or **not**
- Policy starts random, and becomes more deterministic

Why Policy-Gradient

Pro's

- Allows us to introduce function approximation and partial observability, *at a cost*
- Policies can be simpler to represent
- Memory use independent of number of states
- Are we trying to learn $Q(0, left) = 0.24$, $Q(0, right) = 0.25$ or $Q(0, left) < Q(0, right)$

Con's

- Lost convergence to the globally optimal policy
- Lost the Bellman constraint \rightarrow larger variance
- Sometimes the values carry meaning

Long-Term Average Cost

- We will minimise the long-term average cost R

$$R(\mathbf{w}, s) = \lim_{T \rightarrow \infty} \frac{1}{T} \mathbb{E}_{\mathbf{w}} \left[\sum_{t=0}^{T-1} c(s_t) | s_0 = s \right]$$

- And if the Markov system is ergodic then $R(\mathbf{w}, s)$ is constant for all s
- We want to minimise $R(\mathbf{w})$ by computing its gradient

$$\nabla R(\mathbf{w}) = \left[\frac{\partial R}{\partial w_1}, \dots, \frac{\partial R}{\partial w_p} \right]$$

and stepping the parameters away.

- For example (but there are better ways to do it):

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla R(\mathbf{w})$$

Experience Based Policy Gradient

Problem: Too many states to compute gradient analytically

Answer: Compute a Monte-Carlo estimate of the gradient ∇R

$$\nabla R(\mathbf{w}) = \lim_{\beta \rightarrow 1} \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=0}^{T-1} \frac{\nabla \Pr[a_t | \mathbf{o}_t, \mathbf{w}_t]}{\Pr[a_t | \mathbf{o}_t, \mathbf{w}_t]} \sum_{\tau=t+1}^T \beta^{\tau-t-1} c_\tau$$

Introduced by Baxter & Bartlett (2001) [1].

Policy Gradient for Planning

- **Problem:** Monolithic policy $\Pr[\mathbf{a}|\mathbf{o}, \mathbf{w}]$ too complex!
- **Instead:** one parameterised policy per task:

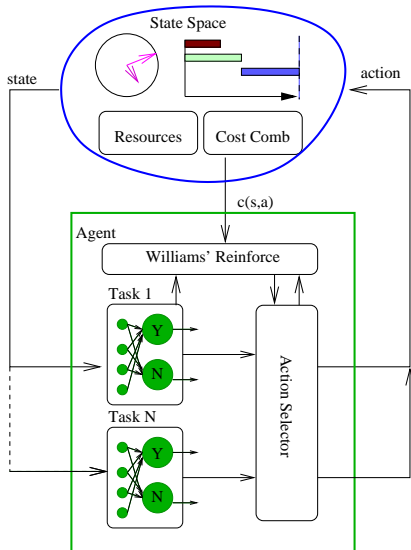
$$\Pr[a_n|\mathbf{o}, \mathbf{w}_n]$$

Analogy

Instead of waiting for word from HQ, local field commander observes situation and decides when to begin the mission

- Overall action probability is $\prod \Pr[a_n|\mathbf{o}, \mathbf{w}_n]$
- Current methods need common cost per planning step
- Theory covered by Peshkin et al. (2000) and Tao & Baxter (2001) [2, 3]

Planning Decision Loop



FPG Algorithm

- 1: **while** $t < T$ **do**
- 2: $\mathbf{e}_t = \beta \mathbf{e}_{t-1}$
- 3: Generate observation \mathbf{o}_t of s_t
- 4: **for** Each eligible task n **do**
- 5: Compute $\Pr[\text{Yes}|\mathbf{o}_t, \mathbf{w}_n]$ and $\Pr[\text{No}|\mathbf{o}_t, \mathbf{w}_n]$
- 6: Sample $a_{tn} = \text{Yes}$ or $a_{tn} = \text{No}$
- 7: Compute $\mathbf{e}_t = \mathbf{e}_t + \nabla \log \Pr[a_{tn}|\mathbf{o}, \mathbf{w}_n]$
- 8: **end for**
- 9: **repeat**
- 10: Try $\mathbf{a}_t = \{a_{t1}, a_{t2}, \dots, a_{tN}\}$
- 11: **if** MUTEX randomly turn off one task start in \mathbf{a}_t
- 12: **until** \mathbf{a}_t possible
- 13: $s_{t+1} = \text{findSuccessor}(s_t, \mathbf{a}_t)$
- 13: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha C_t \mathbf{e}_t$
- 14: $t \leftarrow t + 1$
- 15: **end while**

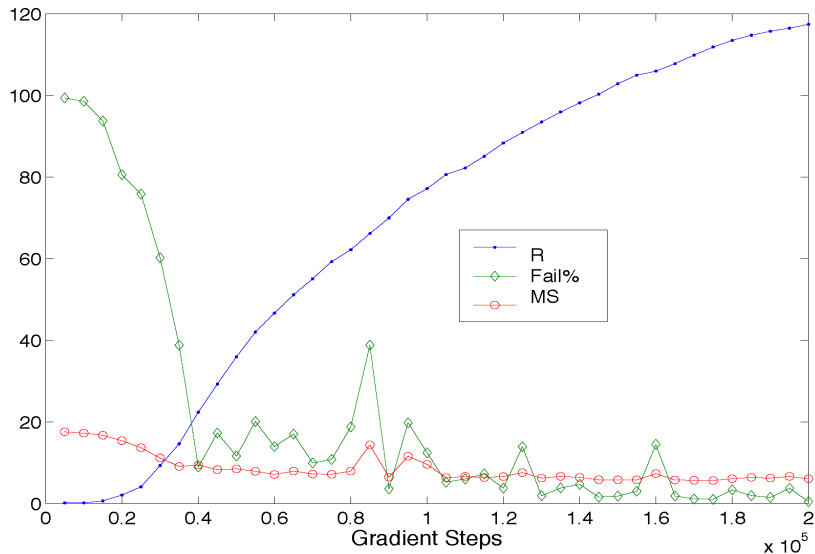
Outline

- 1 The Problem
- 2 Markov Decision Process Formalism
- 3 The MO Planner
- 4 The FPG Planner
- 5 Some Results**

Comparisons

<i>Prob.</i>	<i>Opt.</i>	<i>Fail%</i>	<i>MS</i>	<i>Res.</i>	<i>R</i>	<i>Time</i>
	FPG-L	0.02	5.5		166	60
	Prottle	2.9				272
	MO	Out of memory				
	random	99.3	18		0.1	
	naïve	100	20		0.0	
	FPG-L	14.7	6.9		130	4.4
	Prottle	17.8				10
	MO	7.15	8.2			72
	random	76.5	13		16.4	
	naïve	90.8	16		8.6	

Convergence Example

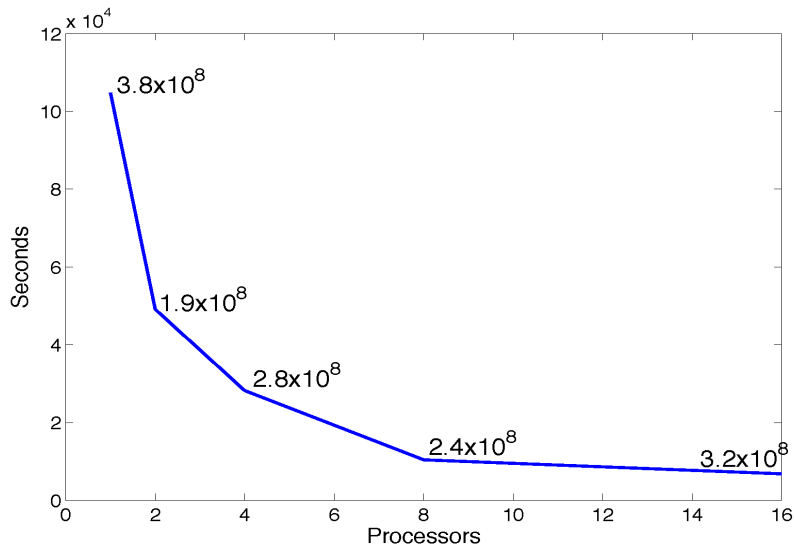


Parallel Results

- No benchmarks big enough
- Desire for real data continues
- 500 task, 250 predicate, success/failure outcome domain synthesised

<i>Prob.</i>	<i>Opt.</i>	<i>Fail%</i>	<i>MS</i>	<i>Res.</i>	<i>R</i>	<i>Time</i>
500	random	76.6	765	6194	0.231	
500	naïve	69.5	736	6359	0.100	
500	16	2.5	158	4276	1.56	3345

Scaling to More Processors



Conclusion

- We are almost able to produce **satisfactory** plans for real world domains
- Many fields, such as machine translation, spell checking, and bio-informatics are relying less on deduction, and more on induction over vast data sets.
- Ideally we will combine induction and deduction, e.g., relational reinforcement learning.

In the near future

- Move to an actor-critic style gradient algorithm
- Incorporate knowledge of the model into monte-carlo gradient estimate
- A new GUI front end to capture these rich domains

- [1] Jonathan Baxter and Peter L. Bartlett.
Infinite-horizon policy-gradient estimation.
Journal of Artificial Intelligence Research, 15:319–350, 2001.
- [2] L. Peshkin, K.-E. Kim, N. Meuleau, and L. P. Kaelbling.
Learning to cooperate via policy search.
In *UAI*, 2000.
- [3] Nigel Tao, Jonathan Baxter, and Lex Weaver.
A multi-agent, policy-gradient approach to network routing.
In *Proc. ICML'01*. Morgan Kaufmann, 2001.