COMP90054 — AI Planning for Autonomy

12. Extensive-form games: Solving Games with Reinforcement Learning

Adrian Pearce

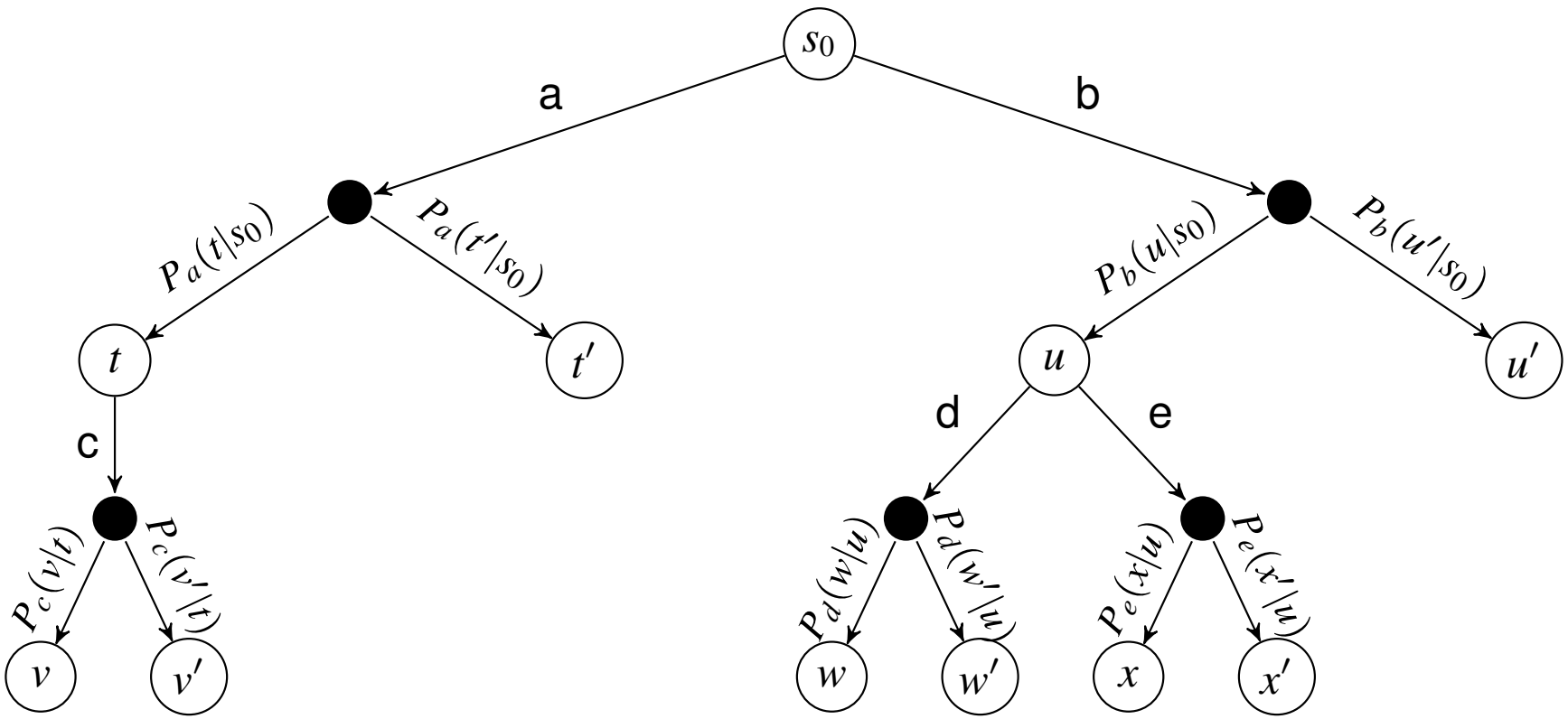
THE UNIVERSITY OF MELBOURNE

Semester 2, 2020

Agenda

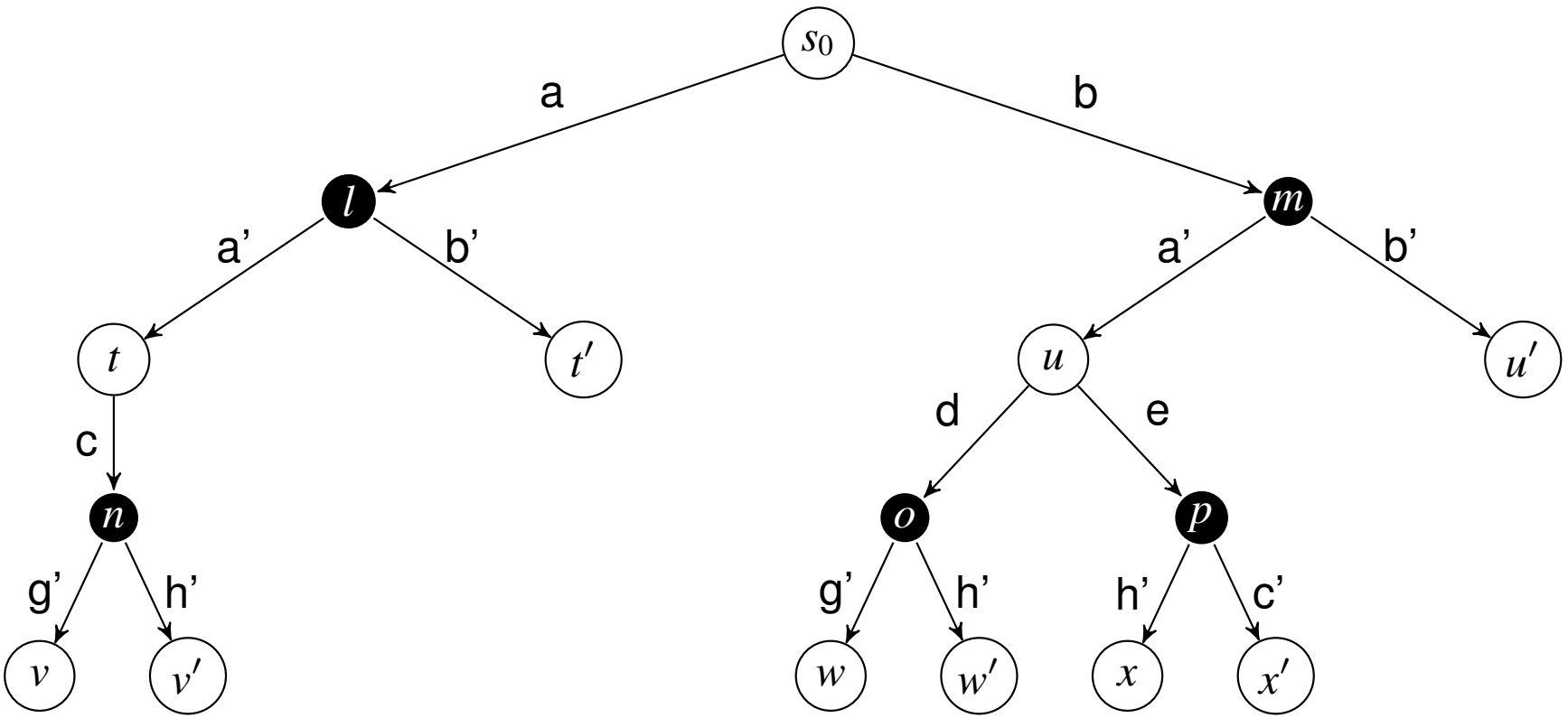
ExpectiMax Trees (revision)

Recall the idea of ExpectiMax trees, which we first encountered in the MCTS module. ExpectiMax trees are representations of MDPs. Recall that the white nodes are states and the black nodes are what we consider choices by the environment:



Game trees

An extensive-form game tree can be thought of as a slight modification to an ExpectiMax tree, except the choice at the black node is no longer left up to ‘nature’, but is made by another agent:



Rewards are only received at the end of games.

MCTS for games

Using MCTS for games is very similar to using MCTS on simulators (week 8).
To use MCTS in games, we simply modify the basic MCTS algorithm as follows:

1. For 'our' moves, we run *selection* as before, however, we also need to select models for our opponents (more on this in the next slide).
2. In the *expansion* step, instead of expanding all child nodes of an action, we run the simulation forward one step. How do we choose the outcome? We simply choose an action for our opponents!
3. We then simulate as before, and we learn the rewards when we receive them.
4. In the *backpropagation* step, instead of using the Bellman equation to calculate the expected return, we simply use the *average* return. If we simulate each step enough times, the average will converge to the expected return.

As with using MCTS for simulation, we have to do repeated simulations for the average to converge.

Choosing actions for opponents

One question that remains is: how should we choose actions for opponents in the selection and expansion step?s There are a few ways to do this.

1. *Random*: Select a random action. This is easy, but it means that we may end up exploring a lot of actions that will never be taken by a good opponent.
2. *Using a fixed policy*: Hand-code a simple stochastic policy that gives reasonable behaviour of the opponent.
3. *Simultaneously learn a policy*: We *learn* a policy for ourselves and our opponents, and we choose actions for our opponents based on the learnt policies.

Simultaneously learning opponent policies

If we want to learn policies for our opponents, we turn this into a *multi-agent reinforcement learning problem*. We are not going to go into the details of multi-agent reinforcement learning in this subject, but to give some intuition that may help on your final project:

1. The payoffs for each agent are their own reward function.
2. We learn n Q-functions — one for each of the n players. The function $Q_i(s, a)$ is the Q-function for agent i.
3. When we have a selection or expansion in the tree, we use the Q-function for the respective agent.
4. Similarly, when we back propagate values back, we update the right agent at the right node.

Model-free reinforcement learning

Similarly, we can apply model-free techniques for multi-agent problems. In a truly model-free problem, we cannot simulate our opponent necessarily, however, our opponent will be executing their own moves, so we don't need to.

A simple way to use model-free techniques in a multi-agent setting is simply to delay the update until our opponents moves have been selected.

If we consider the Q-learning algorithm, the inner loop is:

Choose a from s using policy derived from Q (e.g. ϵ-greedy)
Take action a, observe r [, observe s']
Wait until opponents have executed and observe final outcome s'
$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$

Here, the light grey text is removed from the original Q-learning algorithm, and the bold is added.

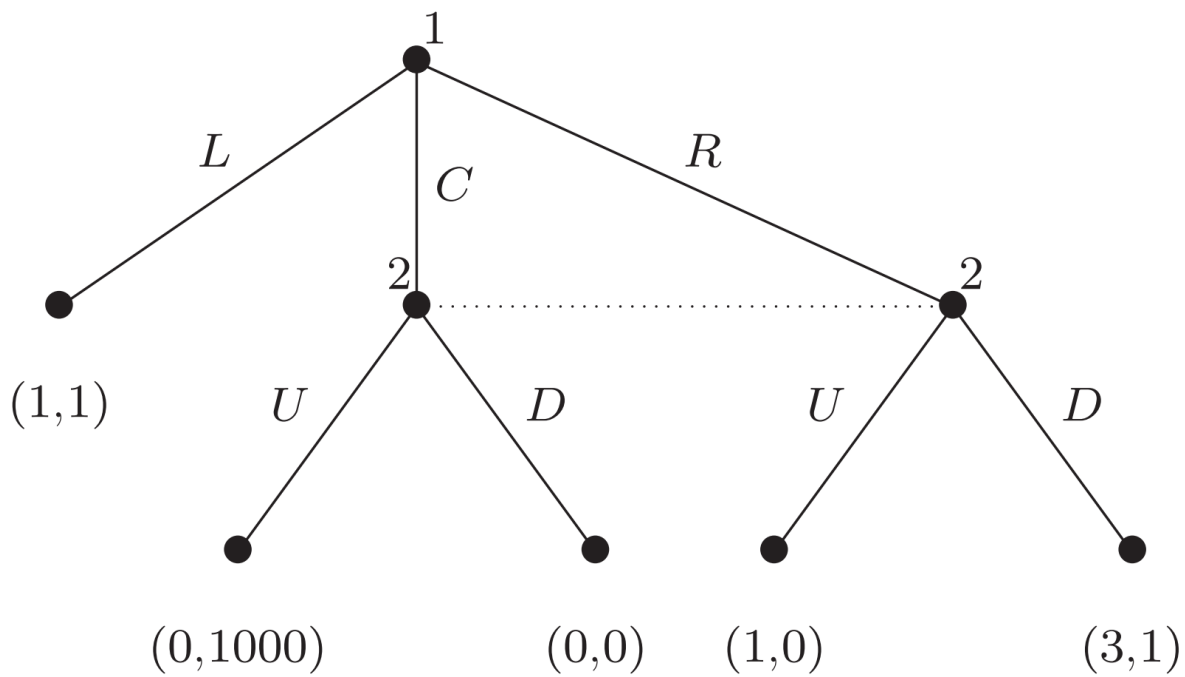
Poker

Image from

https://www.cs.cmu.edu/~sandholm/Solving%20games.Science-2015.pdf

Poker — Imperfect Information Games

Imperfect information games are a generalisation of extensive-form games in which players do not necessarily know what moves their opponent has made in the past, so when we select an action we are unsure which part of the tree we are in. For example, the following game tree, player 2 does not see whether player 1 has played C or R. The dotted line indicates cannot tell the difference between the two states:



Poker — Scaling with Abstraction

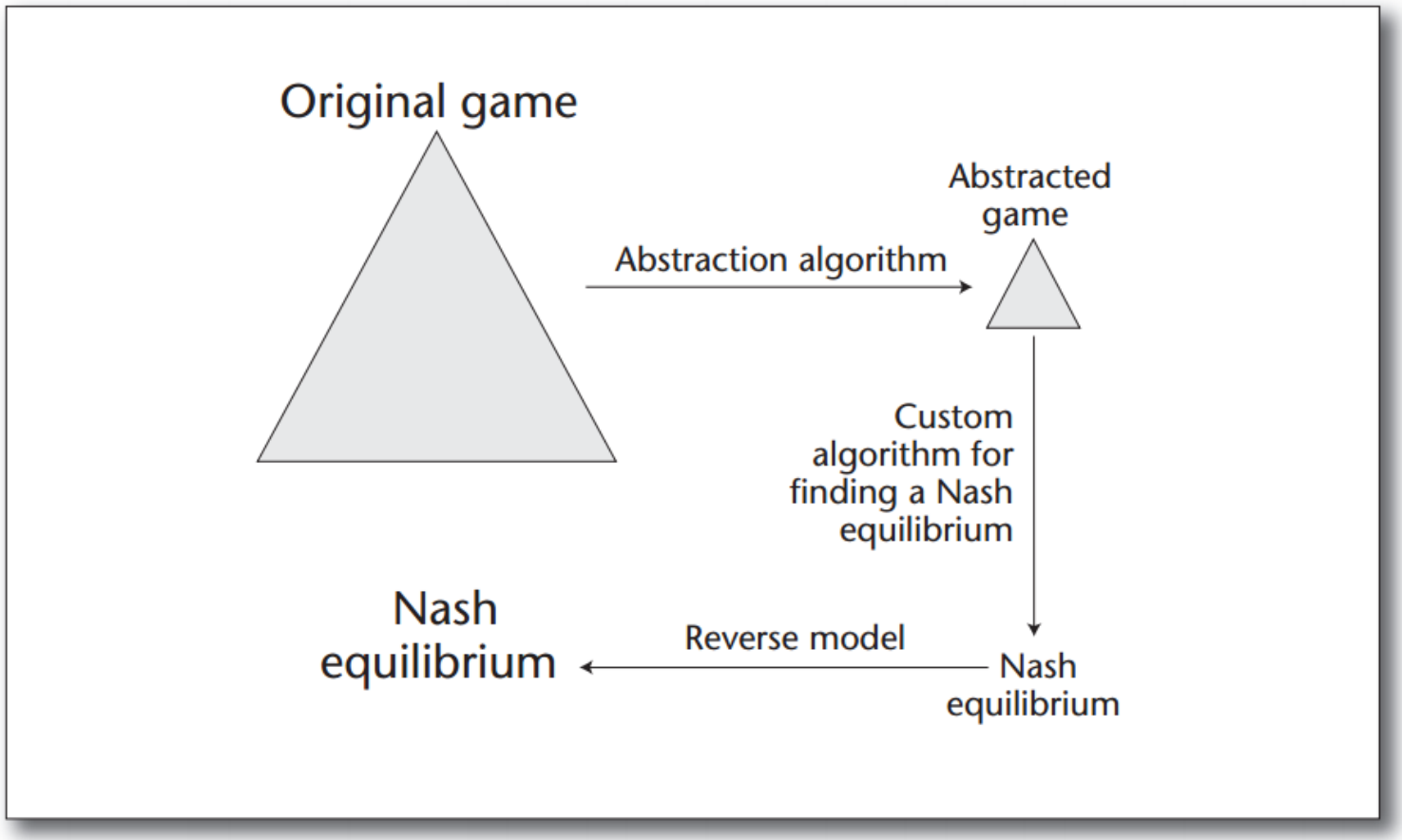


Image from

https://www.cs.cmu.edu/~sandholm/Solving%20games.Science-2015.pdf

Security games — Bayesian Stackelberg Games

A Bayesian game is one in which the is incomplete information about the game; in particular, we do not know the pay-offs of our opponent. Instead, we know about different *types* of player. For example, we can model uncertainty in our opponent's pay-off by modelling that there are two possible attackers with differing pay-offs:

Adversary (P=0.3)

Defender	T1	T2
T1	5, -3	-1, 1
T2	-5, 5	2, -1

Adversary (P=0.7)

Defender	T1	T2
T1	5, -2	-1, 3
T2	-5, 2	2, -2

From this, we can assign e.g. a probability of 0.3 to attacker type I and 0.7 to attacker type 2. The problem is then to select a strategy that maximises expected utility over both attackers.

Security games – PAWS: Stopping animal poaching

Check out this brilliant application of game theory (and other AI techniques) to help prevent animal poaching by using mixed strategies to randomise strategies about where to place anti-poaching officers:

Save the Wildlife, Save the Planet: Protection Assistant for Wildlife Security (PAWS)

Or go to the URL: `https://youtu.be/ai6yhbx5iGw?t=94`