

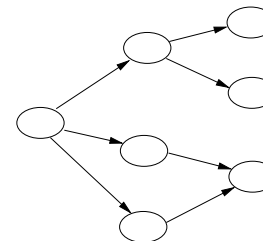
## DECISION-THEORETIC PLANNING

- All the techniques for planning that we have looked at assume deterministic actions.
- This is a BIG assumption.
- (It is certainly not true in general).
- However, it is simple enough in concept to deal with.
- We will cover two closely related approaches to handling non-deterministic actions:
  - Markov decision processes.
  - Partially observable Markov decision processes.
- Both are close in many ways to the kind of planning we looked at in the last class.
- The big change is that actions can have more than one outcome.

### Planning as search

- Let's start by thinking about deterministic actions.
- We can describe a state space search model as:
  - a state space  $S$ ;
  - an initial state  $s_0$ ;
  - a set of actions,  $A(s) \subseteq A$ , applicable in each state  $s \in S$ ;
  - transition function  $f(s, a)$  for  $s \in S$  and  $a \in A$ ;
  - action costs  $c(a, s) > 0$ ; and
  - a set of goal states  $G \subseteq S$

- This gives us a problem space that looks like:



- A solution is a path through this space from initial state to a goal state.

- There are lots of ways of searching this space.
- One simple way is greedy search:
  1. Evaluate each action  $a$  which can be performed in the current state:
 
$$Q(a, s) = c(a, s) + h(s_a)$$
 where  $s_a$  is the next state.
  2. Apply action  $a$  that minimises  $Q(a, s)$ ;
  3. If  $s_a$  is the goal, exit  
else  $s := s_a$ , goto 1.
- This just picks the cheapest move at each point.

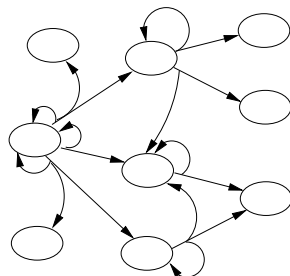
- This is a simple approach that uses little (and constant) memory.
- It can be easily adapted to give a version that we evaluate in real time.
  - Instead of  $s_a$  being the state we expect to get, make it the one we observe.
- Like any depth first approach, it isn't optimal.
- It might not even find solutions.
- (But from the last class we know how to use learning to ensure that it gets better over time).

### Markov decision processes

- So far, there is nothing really new here.
- But it is only a small step to a much better representation.
- In a non-deterministic environment, we don't have a simple transition function.
- Instead an action can lead to one of a number of states.
- When we can tell which state we are in, then we have a Markov decision process (MDP)

- An MDP has the following formal model:
  - a state space  $S$ ;
  - a set of actions,  $A(s) \subseteq A$ , applicable in each state  $s \in S$ ;
  - transition probabilities  $\Pr_a(s'|s)$  for  $s, s' \in S$  and  $a \in A$ ;
  - action costs  $c(a, s) > 0$ ; and
  - a set of goal states  $G \subseteq S$
- Thus for each state we have a set of actions we can apply, and these take us to other states with some probability.
- We don't know which state we will end up in, but we know which one we are in after the action (we have *full observability*).

- This gives us a problem space that looks like:



- A solution is now choice of action in every possible state that the agent might end up in.

- We can think of this solution as a function  $\pi$  which maps states into applicable actions,  $\pi(s_i) = a_i$ .
- This function is called a *policy*.
- What a policy allows us to compute is a probability distribution across all the trajectories from a given initial state.
- This is the product of all the transition probabilities,  $\Pr_{a_i}(s_{i+1}|s_i)$ , along the trajectory.
- Goal states are taken to have no cost, no effects, so that if  $s \in G$ :
  - $c(a, s) = 0$
  - $\Pr(s|s) = 1$

- We can then calculate the expected cost of a policy starting in state  $s_0$ .
- This is just the probability of the policy multiplied by the cost of traversing it:

$$\sum_{i=0}^{\infty} c(\pi(s_i), s_i)$$

- An optimal policy is then a  $\pi^*$  that has minimum expected cost for all states  $s$ .
- As with the search version of the problem, we can solve this by searching, albeit through a much larger space.
- Later we will look at ways to do this search.

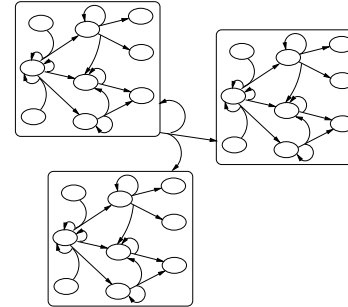
### Partially observable MDPs

- Full observability is a big assumption (it requires an accessible environment). Much more likely is *partial observability*.
- This means that we don't know what state we are in, but instead we have some set of beliefs about which state we are in.
- We represent these beliefs by a probability distribution over the set of possible states.
- These probabilities are obtained by making observations.
- The effect of observations are modelled as probabilities  $\Pr_a(o|s)$ , where  $o$  are observations.

- Formally a POMDP is:

- a state space  $S$ ;
- a set of actions,  $A(s) \subseteq A$ , applicable in each state  $s \in S$ ;
- transition probabilities  $\Pr_a(s'|s)$  for  $s, s' \in S$  and  $a \in A$ ;
- action costs  $c(a, s) > 0$ ;
- a set of goal states,  $G$ ;
- an initial belief state  $b_0$ ;
- a set of final belief states  $b_F$ ;
- observations  $o$  after action  $a$  with probabilities  $\Pr_a(o|s)$

- So we have a situation which looks like:



- This is just an MDP over belief states.

- The goal states of an MDP are just replaced by, for example, states in which we are pretty sure we have reached a goal:

$$\sum_{s \in G} b(s) > 1 - \epsilon$$

- We solve a POMDP by looking for a function which maps belief states into actions, where belief states  $b$  are probability distributions over the set of states  $S$ .
- Given a belief state  $b$ , the effect of carrying out action  $a$  is:

$$b_a(s) = \sum_{s' \in S} \Pr_a(s|s')b(s')$$

- If we carry out  $a$  in  $b$  and then observe  $o$ , we get to state  $b_a^o$ :

$$b_a^o(s) = \frac{\Pr_a(o|s)b_a(s)}{\sum_{s' \in S} \Pr_a(o|s')b_a(s')}$$

- The term on the bottom is the probability of observing  $o$  after doing  $a$  in  $b$ .
- Thus actions map between belief states with probability:

$$b_a(o) = \sum_{s' \in S} \Pr_a(o|s')b_a(s')$$

and we want to find a trajectory from  $b_0$  to  $b_F$  at minimum cost.

## Dynamic programming

- Again we could use greedy search (or any other search technique) to solve POMDPs.
- However, there are more efficient techniques from *dynamic programming* for both MDPs and POMDPs.
- We start from Bellman's *principle of optimality*:  
If  $a$  is the best action in  $s$  to reach the goal, and  $s_a$  is the resulting state, then the optimal cost from  $s$  is the optimal cost from  $s$  plus the cost of doing  $a$ 

$$V^*(s) = \min_{a \in A(s)} [c(a, s) + V^*(s_a)]$$
- This gives us a recursive definition of the optimal cost.

- This can easily be extended to handle MDPs:

$$V^*(s) = \min_{a \in A(s)} [c(a, s) + \sum_{s' \in S} \Pr(s'|s) V^*(s')]$$

- replacing the cost of the path from  $s_a$  with the expected cost across all states that might result from  $a$ .
- This search depends upon the heuristic estimate for the expected cost.
  - The optimal cost is just  $V^*(s)$ , so the greedy policy:  

$$\pi^*(s) = \arg\min_{a \in A(s)} [c(a, s) + \sum_{s' \in S} \Pr(s'|s) V^*(s')]$$
is the optimal policy.

- The problem then is to find  $V^*(\cdot)$ .
- We do this by *value iteration*, solving the recursive equation:

$$V^*(s) = \min_{a \in A(s)} [c(a, s) + \sum_{s' \in S} \Pr_a(s'|s) V^*(s')]$$

for  $V^*(\cdot)$  iteratively.

- So:
  - $V_0(s) = 0$ ;
  - $V_{i+1}(s) = \min_{a \in A(s)} [c(a, s) + \sum_{s' \in S} \Pr_a(s'|s) V_i(s')]$

- Value iteration converges on  $V^*(\cdot)$ .
- In other words:

$$\lim_{i \rightarrow \infty} V_i(s) = V^*(s)$$

- So, if we run the algorithm for long enough, it will give us the optimal value function, and from this we can recover the optimal policy.
- Value iteration can solve MDPs with up to  $10^7$  states.
- This is enough for many purposes.

- We can combine greedy search with value iteration.
- The algorithm is:
  1. Evaluate each action  $a$  applicable in current state  $s$  as:

$$Q(s, a) = c(s, a) + \sum_{s' \in \mathcal{S}} \Pr_a(s'|s) V_i(s')$$

2. Apply  $a$  that minimises  $Q(s, a)$
3. Update  $V(s)$  to  $Q(s, a)$ .
4. Observe resulting state  $s'$
5. Exit if  $s'$  is goal, else with  $s := s'$  go to 1.

- This process is known as *real-time dynamic programming*.
- $V(s)$  is initialized to  $h(s)$
- If  $h$  is admissible, and after repeated trials, this greedy policy eventually becomes optimal.
- This is just like the *reinforcement learning* we saw before for learning a heuristic, but adapted for a more realistic environment.
- If  $h$  is good, very large problems can be solved this way.

- The same approach can be adopted for POMDPs.
- As we already mentioned, a POMDP is an MDP over belief states:
  - An action  $a$  transforms a belief state  $b$  into  $b_a$
  - An action  $a$  and an observation  $o$  map  $b$  into  $b_a^o$  with probability  $b_a(o)$ .
- This makes it easy to come up with a RTDP algorithm.

- We have:
  1. Evaluate each action  $a$  applicable in current state  $b$  as:
 
$$Q(b, a) = c(b, a) + \sum_{o \in \mathcal{O}} b_a(o) V(b_a^o)$$
  2. Apply  $a$  that minimises  $Q(b, a)$
  3. Update  $V(b)$  to  $Q(b, a)$ .
  4. Observe  $o$
  5. Compute new belief state  $b_a^o$
  6. Exit if  $b_a^o$  is final belief state, else with  $b := b_a^o$  go to 1.
- POMDPs are much less tractable than MDPs — the state space is way larger.
- Currently POMDPs with  $\sim 1000$  states are unsolvable (lots of work on *factoring* state spaces).

## Summary

- In this lecture, we have looked at a more sophisticated model of planning than STRIPS.
- Starting from the notion of planning as search, we introduced the Markov decision process representation.
- A solution to an MDP is a *policy*, a choice of what action to take in *every* state.
- We looked at the use of dynamic programming to solve MDPs.