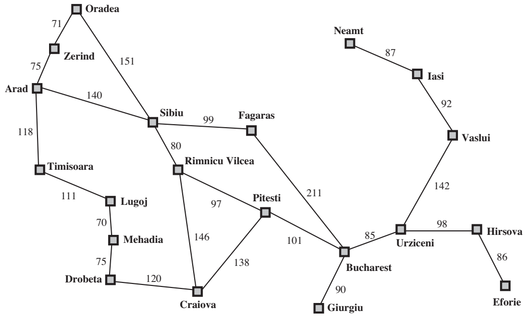
Markov Decision Processes

3007/7059 Artificial Intelligence

School of Computer Science
The University of Adelaide

Back to Romania...



In problem solving by searching, given a start state the objective is to find a path that leads to the goal state.

At any given state, the list of available actions are known, e.g., in Fagaras, we can go to Sibiu or Bucharest.

Moreover, it is assumed that any action taken will definitely lead to the expected outcome, e.g., go to Sibiu will land you in Sibiu.

Back to Romania... (cont.)

Making the assumption above means we adopt a **deterministic transition model**, e.g.,

$$RESULT(In(Fagaras), Go(Sibiu)) = In(Sibiu)$$

This may not be totally realistic, e.g., many roads lead to Sibiu and elsewhere and we are not familiar with Romanian traffic signs.

A **stochastic transition model** encodes the probability of reaching outcome s' after executing action a in state s, or

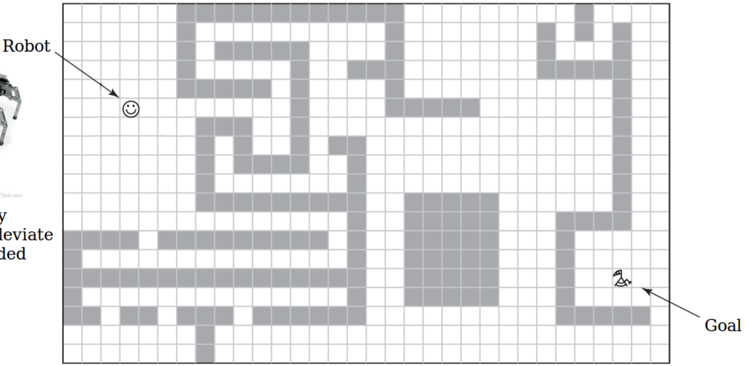
$$P(s'|s,a)$$

For example

$$P(In(Sibiu)|In(Fagaras), Go(Sibiu)) = 0.7$$
$$P(In(Bucharest)|In(Fagaras), Go(Sibiu)) = 0.2$$
$$P(In(Elsewhere)|In(Fagaras), Go(Sibiu)) = 0.1$$

Robotic path planning

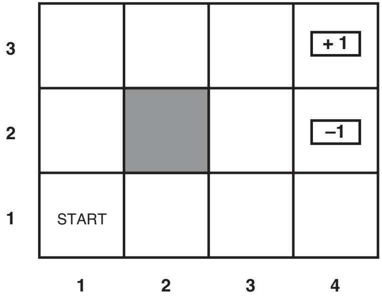
Legs not totally reliable. May deviate from the intended direction/path.



Markov Decision Processes (MDP)

MDPs are about learning to how to make **sequential decisions** to reach a goal in a **non-deterministic environment**.

Consider a simplified environment where the objective is to reach any of the goal states (+1,-1) from the start state:



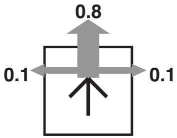
The available actions at each state (a position on the map) are Up, $Down$, $Left$, and $Right$ (no diagonal moves).

What sequence of actions should be taken?

The transition model

If the environment is fully deterministic, the solution is simple: $[Up, Up, Right, Right, Right]$.
But the actions are unreliable, so we may not get there.

The **transition model** $P(s'|s, a)$ adopted is as follows:
Each action achieves its intended effect with probability 0.8, but the rest of the time it may move along an orthogonal direction. If the agent bumps into a wall, it stays in the same position.



This is a **Markovian** model since the probability of reaching s' depends only on s and not the other states prior to s.

What is the probability of success of the action sequence $[Up, Up, Right, Right, Right]$?

The utility function

A **utility function** measures the “goodness” of the sequence of states visited due to a sequence of actions, e.g.,

$$U([s_0, s_1, \ldots, s_N]) = \sum_{t=0}^{N} R(s_t)$$

where $R(s)$ is the **reward** for being in state s.

Utility = long-term goodness, reward = short term goodness

Example:
$R(s) = -0.04$ for all non-terminal states s, while for the two terminal states $R(4,3) = 1$ and $R(4,2) = -1$.

What is the utility if the agent reaches $(4,3)$ after 10 steps? What about after 100 steps?

Solution to an MDP

Given $P(s'|s,a)$ and $U([s_0, s_1, \dots])$ what does a solution look like?

We need to know what action to take given the current state. This is given by a **policy**

$$\pi(s)$$

where $\pi(s)$ returns the action to take at state s.

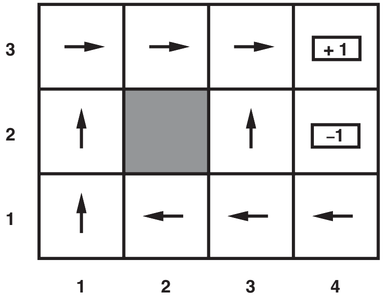
Given $\pi(s)$, we will always know what to do no matter what the outcome of any action is. This also amounts to saying that **the starting state does not matter** — we only need to figure out what to do given the current state.

A good policy guides the agent to reach the terminal state quickly (high utility), while a bad policy may cause the agent to be stuck for a long time (low utility).

The optimal policy

The **optimal policy** $\pi^*(s)$ yields the **highest expected utility**.

Example:
The optimal policy if $R(s) = -0.04$ for the non-terminal states.

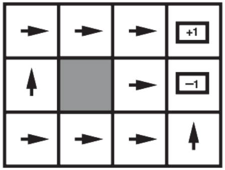


Notice that at $s = (3, 1)$ the policy recommends going left (take the long way round) instead of up (take the shortcut).

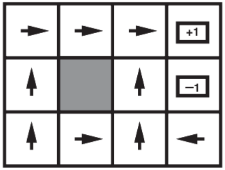
How does changing $R(s)$ for the non-terminal states change the optimal policy? (Look at the utility function)

The optimal policy (cont.)

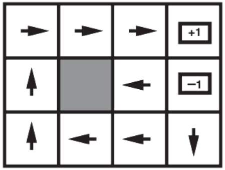
Different values of $R(s)$ require different optimal policies:



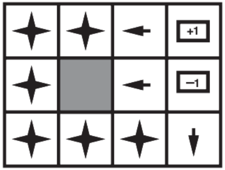
$R(s) < -1.6284$



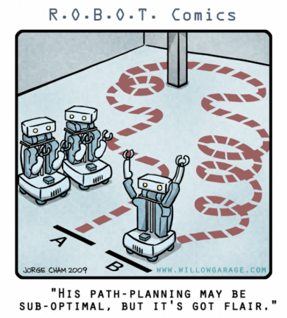
$-0.4278 < R(s) < -0.0850$



$-0.0221 < R(s) < 0$



$R(s) > 0$



The careful **balancing of risk and reward** is a characteristic of MDP not found in basic search problems.

Discounted rewards

There is an issue with how the previous utility function works

$$U([s_0, s_1, \ldots, s_N]) = \sum_{t=0}^{N} R(s_t)$$

If $R(s) > 0$, the solution is just to remain in the current position forever to get infinite utility! (see 4th example in previous page)

Fix this by using **discounted rewards**

$$U([s_0, s_1, \ldots, s_N]) = \sum_{t=0}^{N} \gamma^t R(s_t)$$

where $0 < \gamma < 1$ is the **discount factor**.

With discounted rewards utility is bounded

$$U([s_0, s_1, \ldots, s_\infty]) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \sum_{t=0}^{\infty} \gamma^t R_{\max} = R_{\max}/(1-\gamma)$$

where $R_{\max}$ is the highest reward possible.

Calculating the optimal policy

Our goal is to compute the optimal policy for the MDP.

The **expected utility** of executing π given that we are in s is

$$U^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right]$$

where the expectation $E[\cdot]$ is taken over the distribution of all possible state sequences $S_0, S_1, S_2, \ldots$ starting from $S_0 = s$, **if we execute π from s onwards**.

The optimal policy $\pi^*(s)$ **maximises the expected utility**

$$\pi^*(s) = \arg\max_\pi U^\pi(s)$$

Calculating the optimal policy (cont.)

The **utility of a state** s is the expected utility if we execute the optimal policy starting from s

$$U(s) = U^{\pi^*}(s)$$

Note: Previously we had the **utility of a sequence of states**.

Example (this is just a preview. We haven't learned how to calculate them yet):
State utilities when $R(s) = -0.04$ for non-terminal states:

3	0.812	0.868	0.918	$\boxed{+1}$
2	0.762	▨	0.660	$\boxed{-1}$
1	0.705	0.655	0.611	0.388
	1	2	3	4

Notice that states closer to the terminals have higher utility.

Calculating the optimal policy (cont.)

Given the utility of the states, the optimal policy becomes

$$\pi^*(s) = \arg\max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s')$$

In other words, choose the action a that gives the **maximum expected utility (MEU)** (no difference from before actually, but now we get to factor in the transition model).

If this all sounds circular, it's because it actually <u>is circular</u>:

- If we know $U(s)$, we know how to compute $\pi^*(s)$ via MEU.
- If we know $\pi^*(s)$, we know in principle how to compute $U(s)$.

$\implies$ classic **chicken-and-egg** problem.

A flash of brilliance

The utility of a state is the immediate reward for that state plus the expected discounted utility of the next state, assuming that the agent chooses the optimal action.

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U(s')$$

This is called the **Bellman Equation** (aft. Richard Bellman, 1975).

Example:
The Bellman Equation for $s = (1,1)$ is

$$\begin{aligned} U(1,1) = R(1,1) + \gamma \max[&0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1), (Up)\\ &0.9U(1,1) + 0.1U(1,2), \qquad (Left)\\ &0.9U(1,1) + 0.1U(2,1), \qquad (Down)\\ &0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1)].(Right) \end{aligned}$$

If we plug in the state utilities (from 2 pages back), we find that Up is the best action.

The value iteration algorithm

If there are n states, we have exactly n Bellman Equations. The problem becomes **solving n simultaneous equations**.

One problem: there is a "$\max$" in each equation. If it isn't there the equations are all linear and we can solve easily in $\mathcal{O}(n^3)$.

The solution is what we usually do when presented with a chicken-and-egg problem: **start somewhere and iterate**.

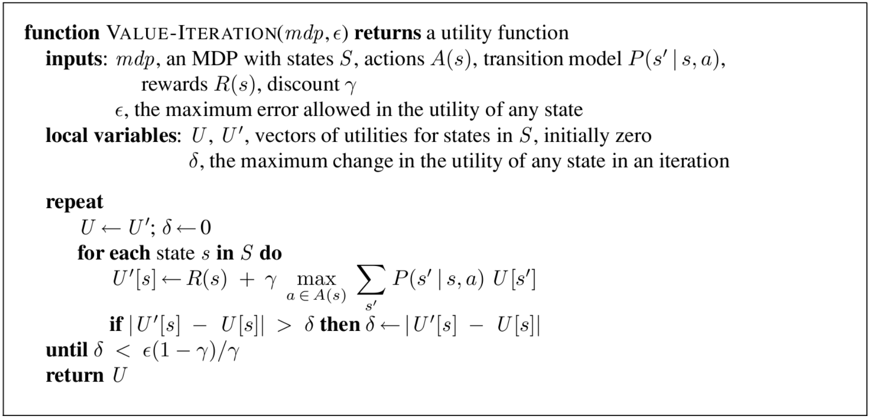
Specifically, we initialise the $U(s)$'s randomly, then compute

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U_i(s')$$

where $U_i(s)$ is the value of $U(s)$ after the i-th update. This is called the **Bellman update**.

The value iteration algorithm (cont.)

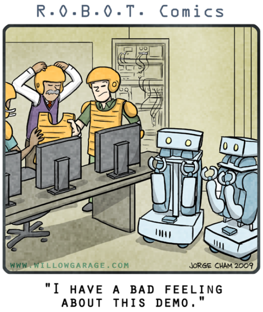
The overall algorithm is called **Value Iteration**.



```
function VALUE-ITERATION($mdp, \epsilon$) returns a utility function
  inputs: $mdp$, an MDP with states $S$, actions $A(s)$, transition model $P(s' \mid s, a)$,
              rewards $R(s)$, discount $\gamma$
          $\epsilon$, the maximum error allowed in the utility of any state
  local variables: $U$, $U'$, vectors of utilities for states in $S$, initially zero
                   $\delta$, the maximum change in the utility of any state in an iteration

  repeat
      $U \leftarrow U'$; $\delta \leftarrow 0$
      for each state $s$ in $S$ do
          $U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U[s']$
          if $|U'[s] - U[s]| > \delta$ then $\delta \leftarrow |U'[s] - U[s]|$
  until $\delta < \epsilon(1-\gamma)/\gamma$
  return $U$
```

Does it converge?



If we update infinitely often, we are **guaranteed to converge**.

In fact, we will **always reach a unique solution** (regardless of the initialisation), and the corresponding policy is the optimal one.

The proof is somewhat complicated but very interesting. See Section 17.2.3 in Russell and Norvig.

Does it converge? (cont.)

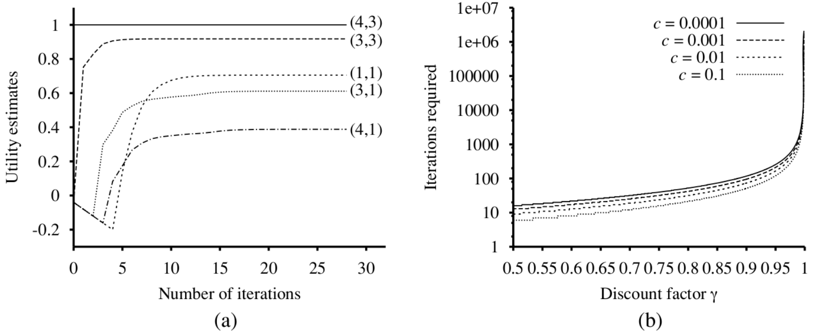


Figure 17.5 (a) Graph showing the evolution of the utilities of selected states using value iteration. (b) The number of value iterations k required to guarantee an error of at most $\epsilon = c \cdot R_{\max}$, for different values of c, as a function of the discount factor γ.