UNIVERSITY OF LEEDS

Class: Machine Learning

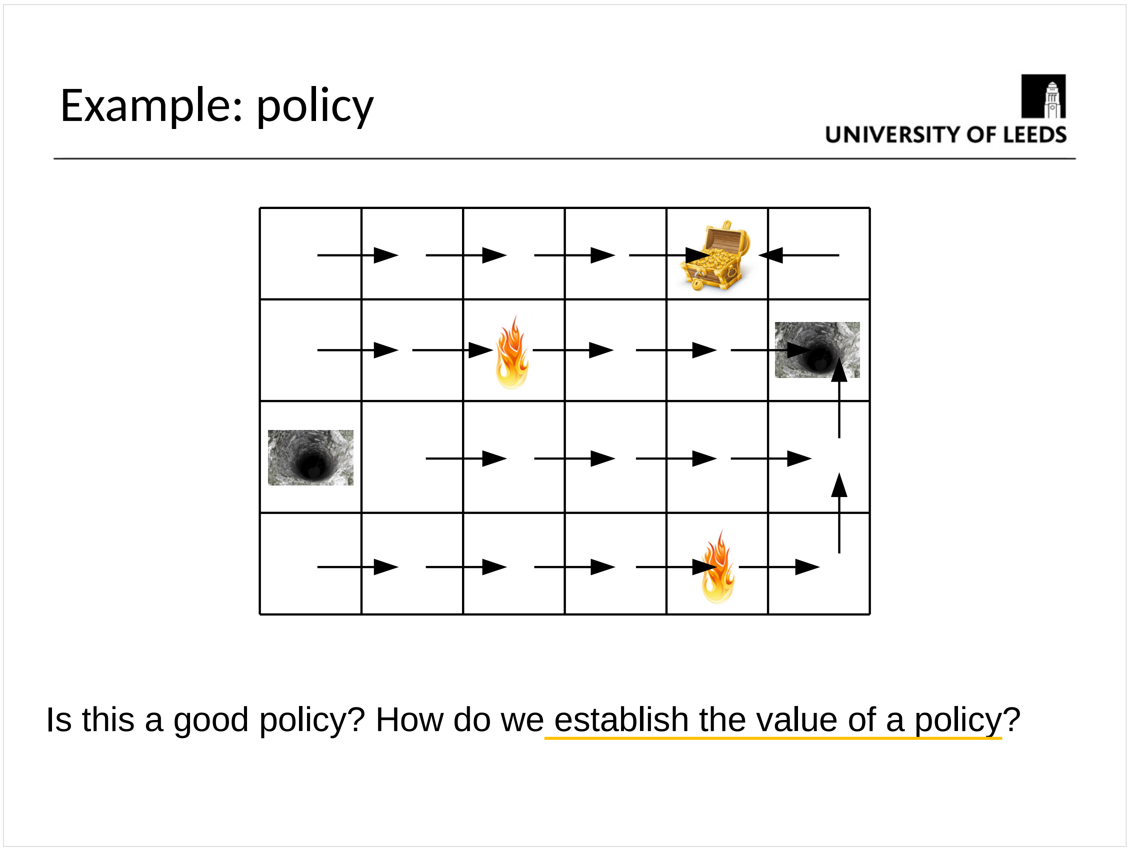
Policy Iteration

Instructor: Matteo Leonetti

Learning outcomes

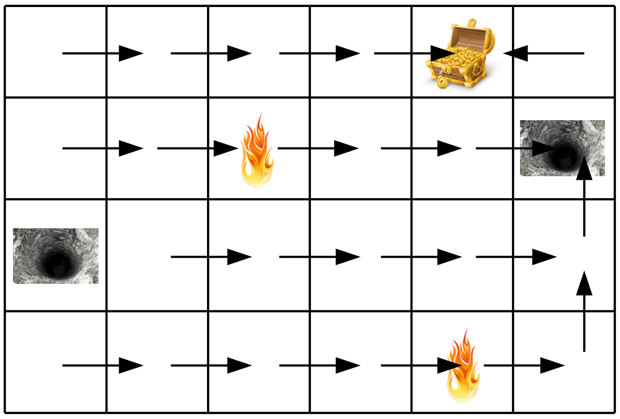
UNIVERSITY OF LEEDS

- Define and compute the value and action value function for a given policy
- Perform a step of generalized policy iteration



Example: policy

UNIVERSITY OF LEEDS



Is this a good policy? How do we establish the value of a policy?

This would be an example of a deterministic policy. For each state, we have a single action represented by the arrows.

Up to this point we can describe the environment with *states*, *actions*, *transitions*, and *rewards*, the goal with the *return*, and behaviours with *policies*.

We can start asking the question: if we start with an initial policy (even just random) how can we get a better one?

Improving the policy goes through being able to evaluate it.

Our next step will be the definition of a *value* function.

Value

UNIVERSITY OF LEEDS

We define the *value* of a state under a given policy, as the expected return from that state while following the policy:

$$G_t \equiv \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} = R_{t+1} + \gamma G_{t+1}$$

$$v_\pi(s) \equiv E_\pi[G_t | S_t = s] = E_\pi[R_{t+1}] + \gamma E_\pi[G_{t+1}]$$

$$= E_\pi[R_t] + \gamma v(s_{t+1})$$

This is called the *Bellman* equation

The value function of a policy is a function of the states. For each state, the value function gives us the expected *return* starting in the input state and following the policy.

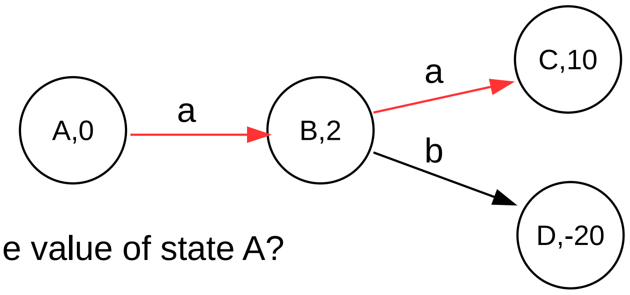
So, for any time step t, the value of a state under a given policy is defined as the expected return G_t form the current state S_t = s at time t.

G_t, S_t, and R_t are upper case because they are *random* variables. The state the agent is in at time t, and the reward at time t, are determined by the initial state, the transition function, the reward function, and the policy. Each one of these can be a probability distribution, so even knowing the initial state S_0 exactly, we can't be sure about which state the agent will be in at time t. However, there is a certain probability of being in any state that depends on the policy and the domain dynamics (the transition function).

If you don't remember random variables, probability distributions, and expectations, this is a good time to go revise those! For instance
Random variables: https://www.youtube.com/watch?v=3v9w79NhsfI
https://www.youtube.com/watch?v=dOr0NKyD31Q
Expectation: https://www.youtube.com/watch?v=j__Kredt7vY

So, $v_\pi(s) = E\pi[G_t | S_t = s]$ means that the value of a state under policy π is the expected value of the return from time step t, given that the current state s_t is indeed s (the input of the value function). Let's now look at examples of how to compute the value of a state under a given policy. The function v has a convenient recurse formulation.

Understanding the value



What is the value of state A?

It depends on what the agent is going to do in B (the policy)

Assuming:

$\gamma = 0.5$

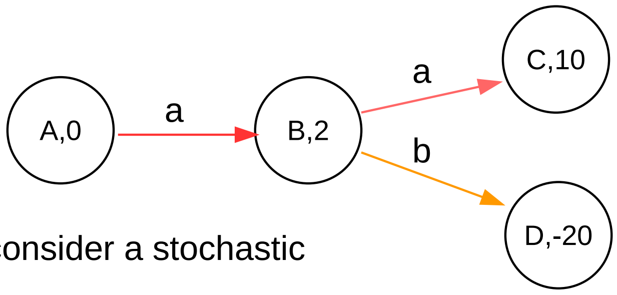
$\pi(B) = a$

$v_\pi(B) = 10$

$v_\pi(A) = 2 + \gamma v_\pi(B) = 7$

Understanding the value

UNIVERSITY OF LEEDS



Now let's consider a stochastic policy:

$\gamma = 0.5$

$\pi(a|B) = 0.8$

$\pi(b|B) = 0.2$

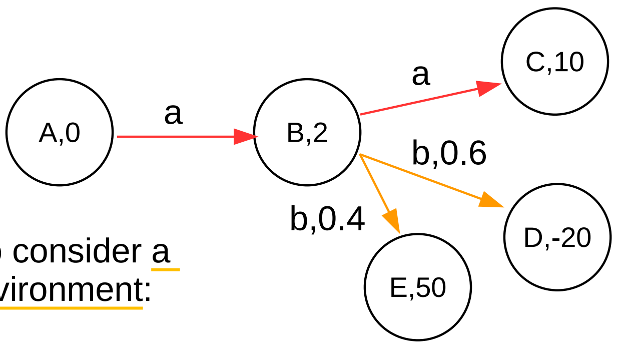
$v_\pi(B) = 0.8 \cdot 10 + 0.2 \cdot (-20) = 4$

$v_\pi(A) = 2 + \gamma\, v_\pi(B) = 4$

A and B are less valuable under this policy, because the agent may go to the bad state.

Understanding the value

UNIVERSITY OF LEEDS



考

Now let's also consider a stochastic environment:

$\gamma = 0.5$

$\pi(a|B) = 0.8$

$\pi(b|B) = 0.2$

$T(B, b, D) = 0.6$

$T(B, b, E) = 0.4$

$$v_\pi(B) = 0.8 \cdot 10 + 0.2 \cdot (0.6 \cdot (-20) + 0.4 \cdot 50) = 9.6$$

$$v_\pi(A) = 2 + \gamma v_\pi(B) = 6.8$$

Taking b in B may end up in a good state, and the values reflect that.

Understanding the value

$$v_\pi(B)=0.8\cdot 10+0.2\cdot\boxed{(0.6\cdot(-20)+0.4\cdot 50)}=9.6$$

Probability of taking an action

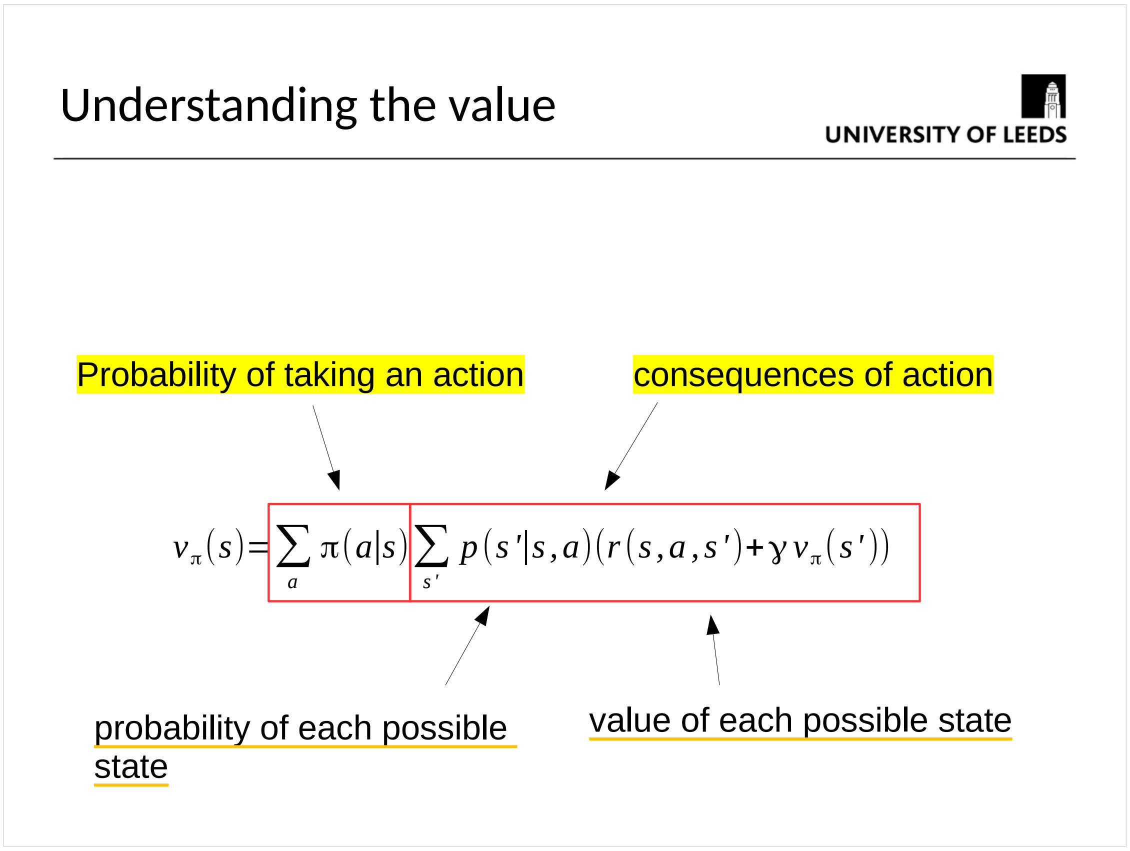
consequences of action

Breaking down action consequences:

$$(0.6\cdot(-20+0.5\cdot 0)+0.4\cdot(50+0.5\cdot 0))$$

$$(p(D|B,b)\cdot(r(B,b,D)+\gamma v_\pi(D))+p(E|B,b)\cdot r(B,b,E)+\gamma v_\pi(D))$$

Having chosen an action (b, in this case), the value is the probability of ending up in each possible state, times the value of that state.



Understanding the value

UNIVERSITY OF LEEDS

Probability of taking an action

consequences of action

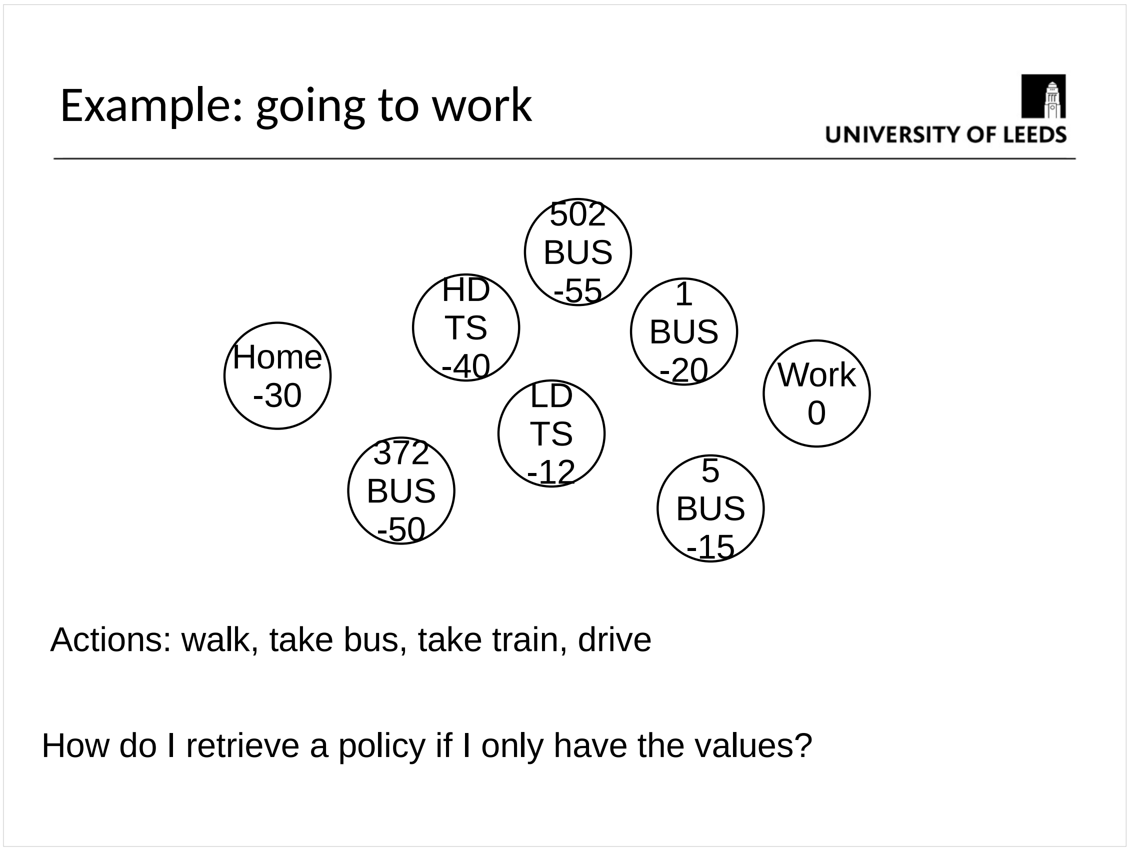
$$v_\pi(s)=\boxed{\sum_a \pi(a|s)}\boxed{\sum_{s'} p(s'|s,a)(r(s,a,s')+\gamma v_\pi(s'))}$$

probability of each possible state

value of each possible state

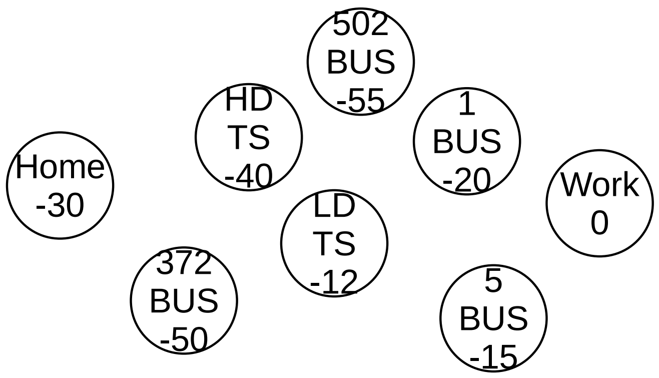
Here the expectation has been made explicit with respect to all the probabilities involved.

Since this is an MDP, that is, is markovian, all the distributions depend only on the current state, and not also on the previous states.



Example: going to work

UNIVERSITY OF LEEDS

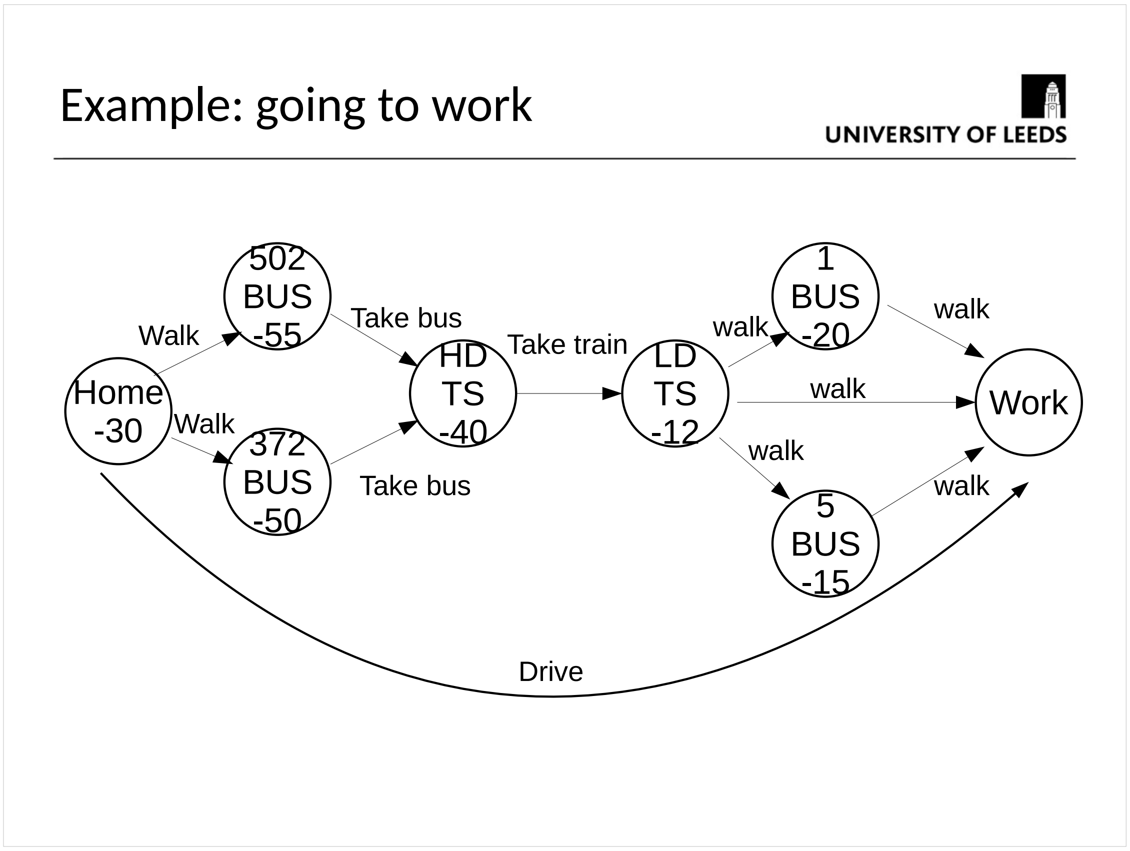


Actions: walk, take bus, take train, drive

How do I retrieve a policy if I only have the values?

Here you have the value of each state, which is telling you that there is a sequence of actions that takes me from home to work in 30 minutes (in *expectation*, that is, on average).

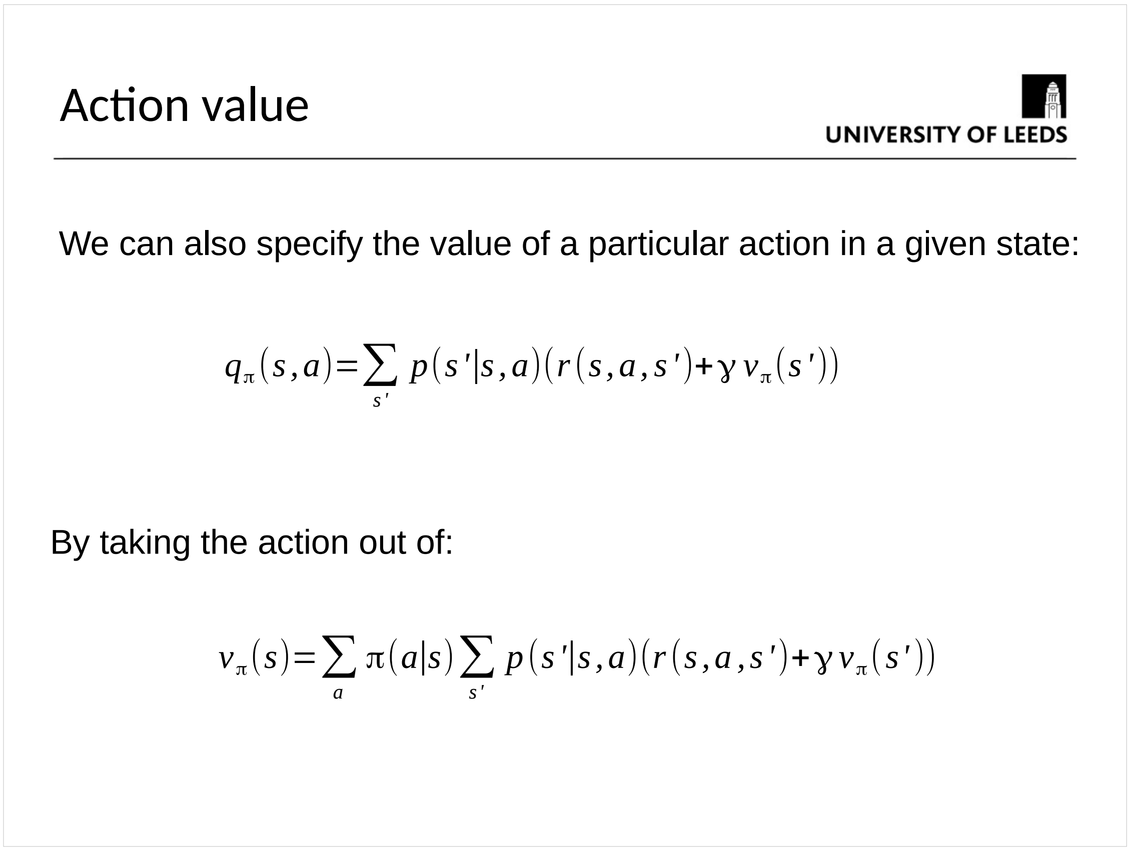
What is the first action I should take then? You cannot know, because you don’t know what the actions do, and therefore what the next state would be if you chose each action!



Example: going to work

On the contrary, if I showed you this model, it’s obvious that the action I should take is to drive, since it takes me directly to the best state, which is the goal.

So, having the value of the actions you can only make a decision if you also have the dynamics of the system, that is the transition function. This means that you have to know what each action does when applied in a certain state.



Action value

UNIVERSITY OF LEEDS

We can also specify the value of a particular action in a given state:

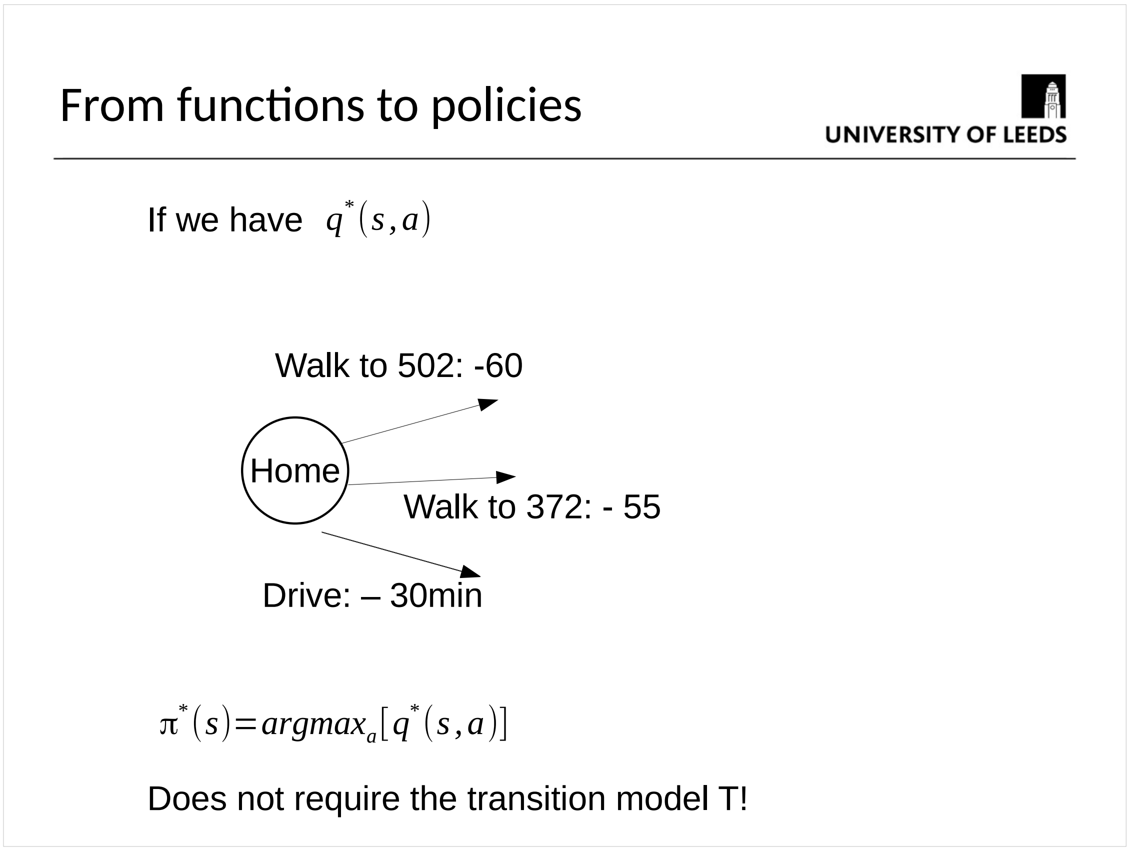
$$q_\pi(s,a)=\sum_{s'} p(s'|s,a)(r(s,a,s')+\gamma v_\pi(s'))$$

By taking the action out of:

$$v_\pi(s)=\sum_{a} \pi(a|s)\sum_{s'} p(s'|s,a)(r(s,a,s')+\gamma v_\pi(s'))$$

The previous function, v, defines the value of a state under a certain policy.

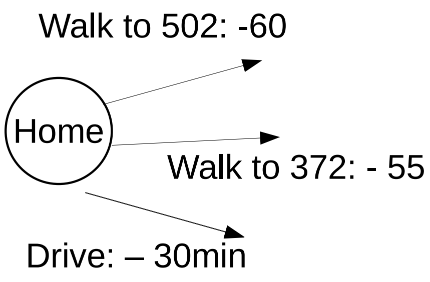
This function, q, called an “action value function” defines the value of a state-action pair, under a certain policy, That is, $q_\pi(s,a)$ is the expected return for taking action a in state s, and following π thereafter.



From functions to policies

UNIVERSITY OF LEEDS

If we have $q^*(s,a)$



$$\pi^*(s) = argmax_a[q^*(s,a)]$$

Does not require the transition model T!

However, if we have the action value function, we don’t need the action model ,we just choose the action with the highest action value function in every state.

We don’t even need to know where the action will take the agent. The agent knows that by starting with each action, and following the optimal policy, it can reach the goal with the expected return given by the action value function.

The agent can simply blindly trust the action value function, without thinking about the consequences of its actions.

Optimal value functions

UNIVERSITY OF LEEDS

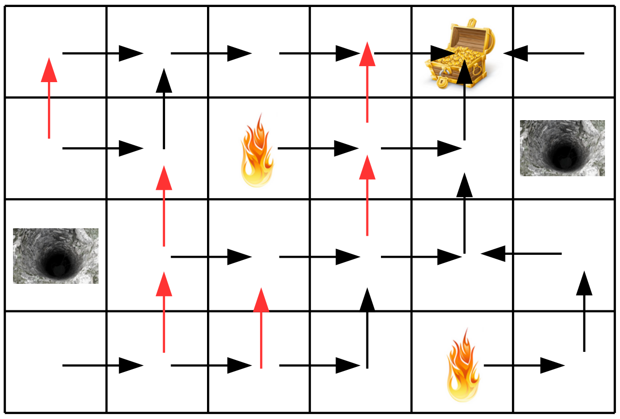
We have now the idea of an *optimal* value function, that is a value function that is the highest it can be on every state, and corresponds to optimal policies.

$$v^*(s) \quad q*(s,a)$$

We now see how to compute the optimal value function

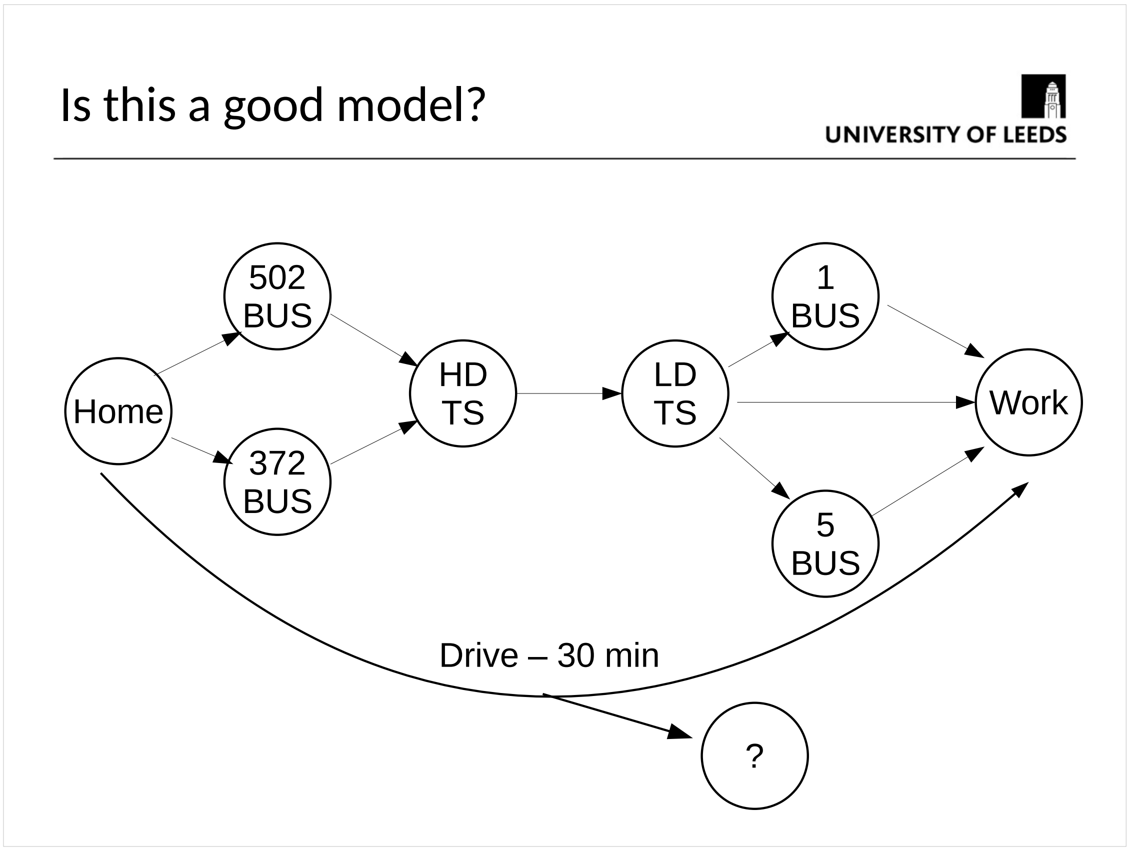
Example: optimal policy

UNIVERSITY OF LEEDS



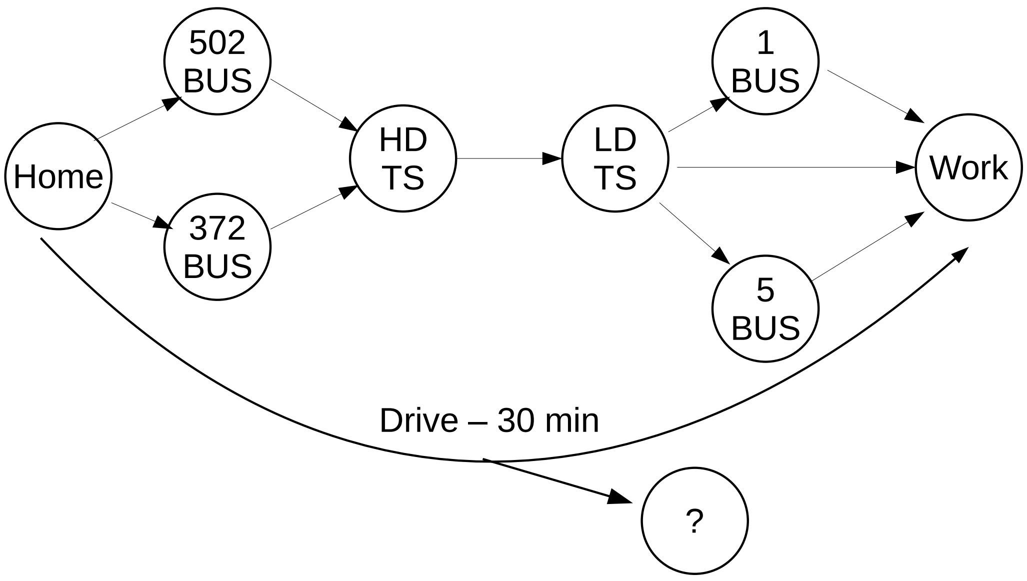
Why is the optimal policy not unique? You can brake ties between equally valuable actions in many ways (other possible optimal actions in red)

The red actions are alternative to the black actions, but they are nonetheless optimal, since the value for both actions in every state is the same.



Is this a good model?

UNIVERSITY OF LEEDS

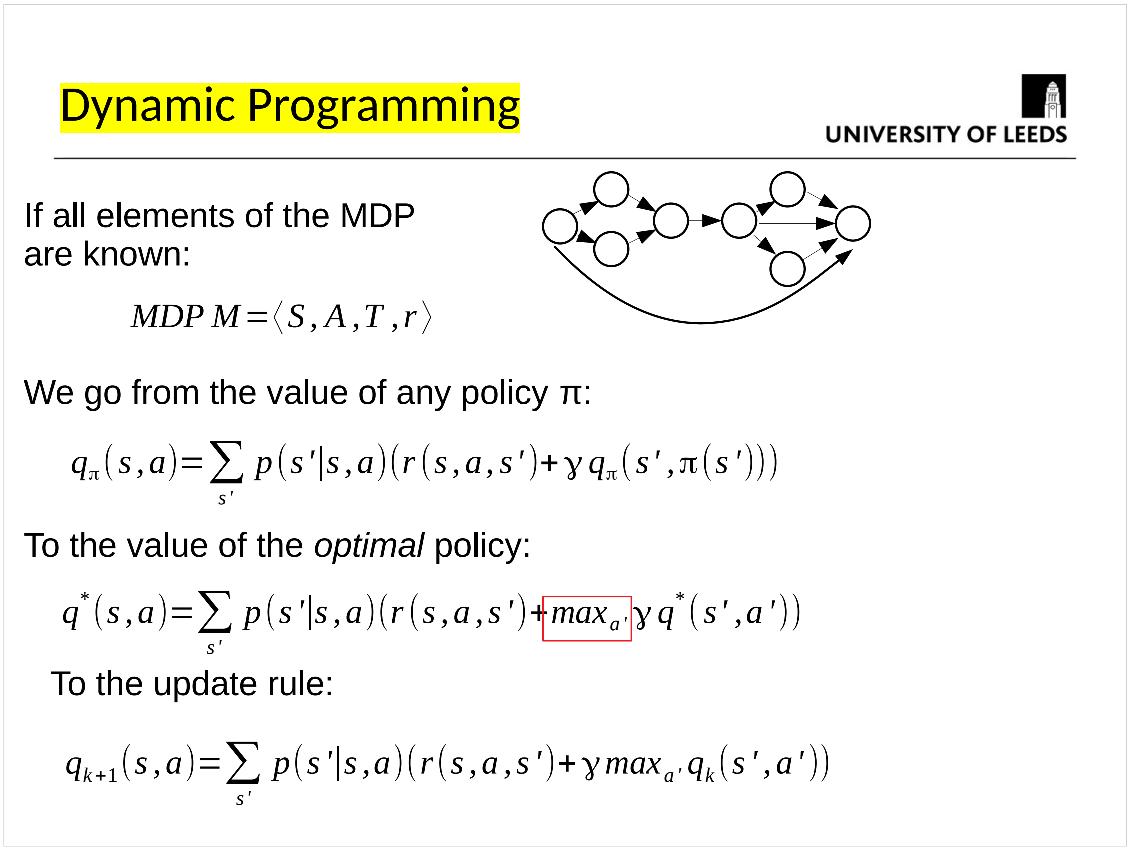


Every model is an approximation of the reality.

When we plan we have to simplify our view of the world because otherwise planning for all possible outcomes of an action would be overwhelming.

For instance, the action "drive to work from home" normally results in the agent being at work. However, the car may break down, or there may be an accident and the agent ends up in an hospital, or the road may be closed, or…

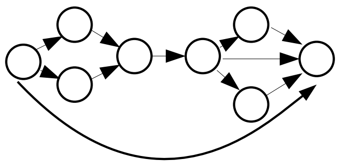
Even if we could list all possible outcomes of an action, planning with such a model would be extremely computationally complex.



Dynamic Programming

UNIVERSITY OF LEEDS

If all elements of the MDP are known:



$$MDP\ M = \langle S, A, T, r \rangle$$

We go from the value of any policy π:

$$q_\pi(s,a) = \sum_{s'} p(s'|s,a)\left(r(s,a,s') + \gamma\, q_\pi(s', \pi(s'))\right)$$

To the value of the *optimal* policy:

$$q^*(s,a) = \sum_{s'} p(s'|s,a)\left(r(s,a,s') + \boxed{max_{a'}}\, \gamma\, q^*(s',a')\right)$$

To the update rule:

$$q_{k+1}(s,a) = \sum_{s'} p(s'|s,a)\left(r(s,a,s') + \gamma\, max_{a'}\, q_k(s',a')\right)$$

If we have the full model of the system, that is the state space, the actions, the transition function and the reward function, we can *plan*.

Planning has been studied in AI for a very long time and takes many forms. Planning in an MDP amounts to computing the optimal action value function for each state.

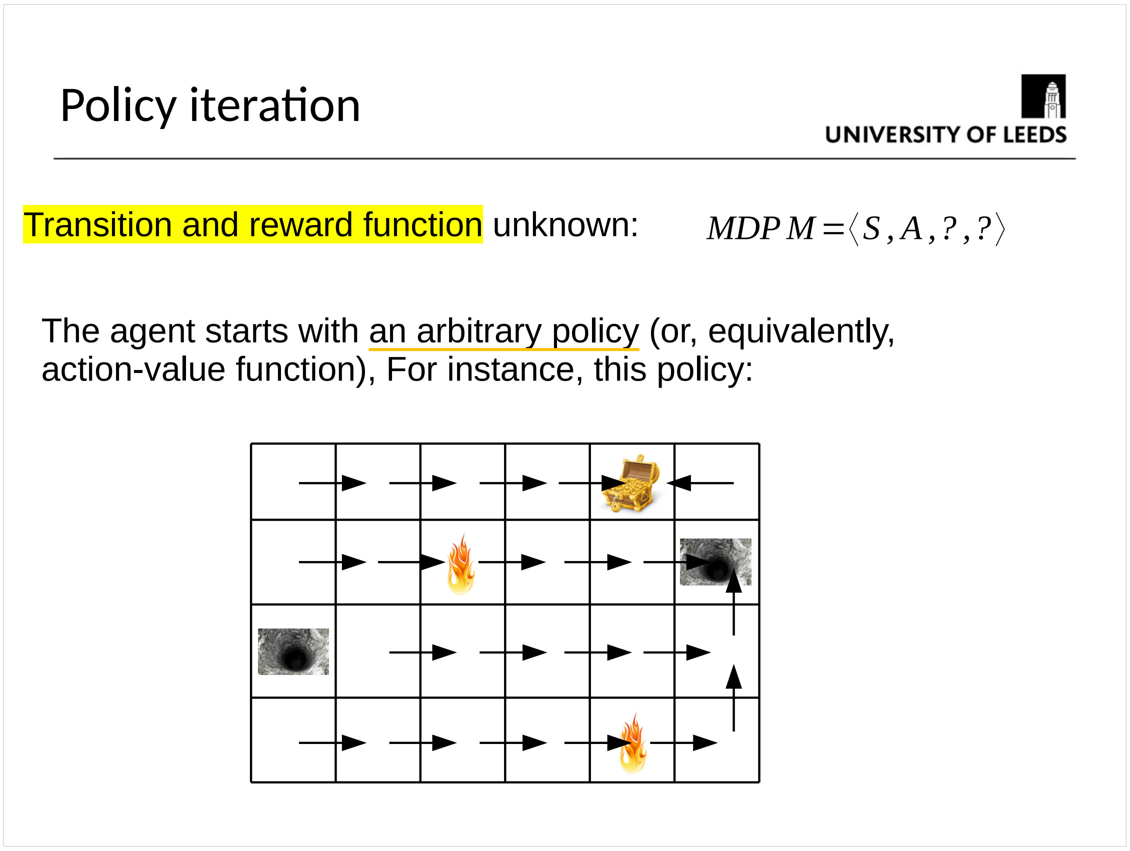
The equation that computes the optimal action value function starting from any action value function is obtained directly from the equation from the value of a policy.

The first step consists in substituting the policy we are estimating with the optimal policy. Since the optimal policy always chooses the action with the highest value, we introduce a max operator over the next actions a’. This is called the Bellman *optimality* equation. For the more mathematically inclined, the action value function of the optimal policy is a fixed point of the Bellman optimality equation.

Then, the Bellman optimality equation can be turned into an update rule, and it has been shown that it is possible to start with any initial value function, and by applying the update rule repeatedly for every state and action pair it will eventually not modify the value function anymore: this is the fixed point.

This planning method is called *Dynamic Programming* and is at the heart of reinforcement learning.

However, since it is not learning (but planning), we do not need to know more about it.

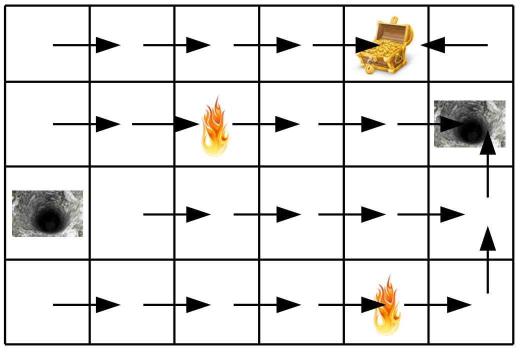


Policy iteration

UNIVERSITY OF LEEDS

Transition and reward function unknown: $MDP\ M = \langle S, A, ?, ? \rangle$

The agent starts with an arbitrary policy (or, equivalently, action-value function), For instance, this policy:

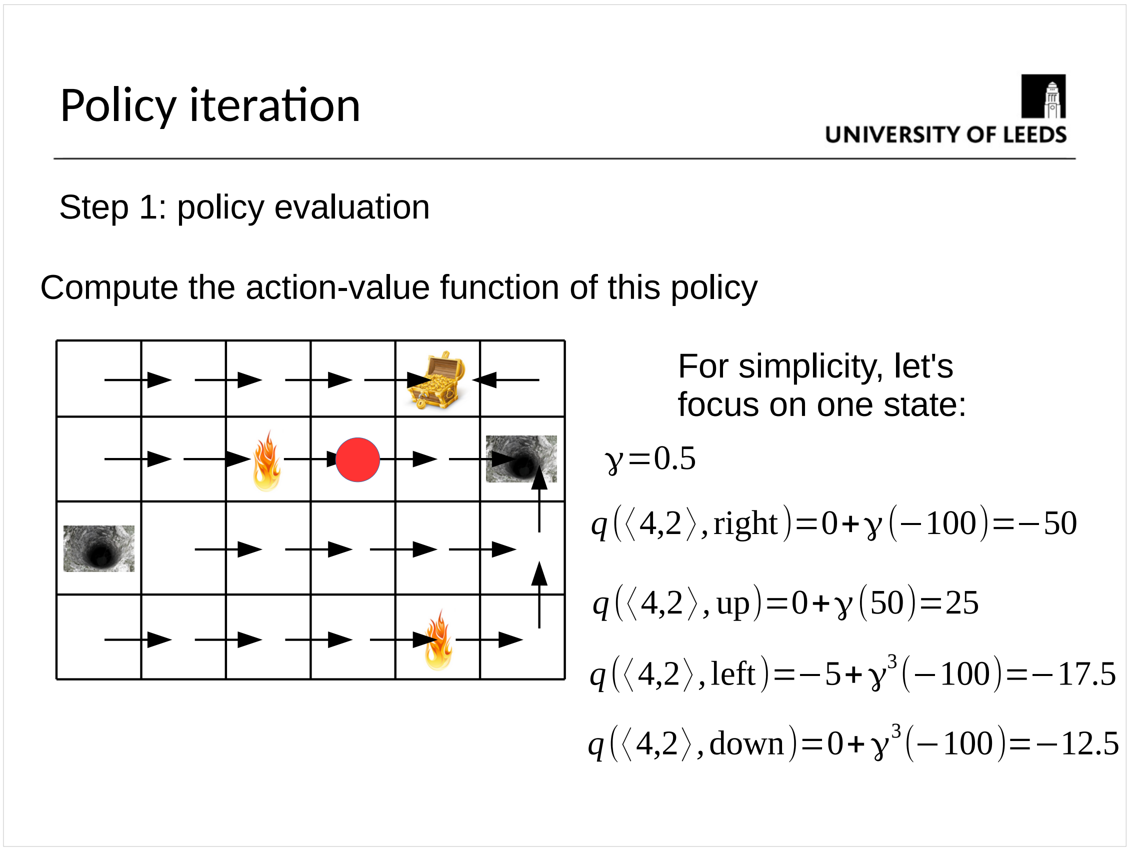


The need for learning arises from the fact that there is something we don’t know. In particular we don’t know the transition function and the reward function, so we can’t apply dynamic programming directly.

The previous method for dynamic programming is also called value iteration, because it iterates over successive value functions, until it converges to the optimal one.

We are now going to look at what is called *policy iteration* which is more suitable for learning.

Policy iteration starts by having an initial policy. Any policy will do. The one in the picture is just a possible policy.

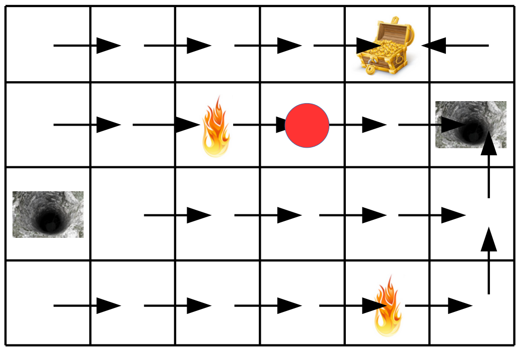


Policy iteration

UNIVERSITY OF LEEDS

Step 1: policy evaluation

Compute the action-value function of this policy



For simplicity, let's focus on one state:

$\gamma = 0.5$

$q(\langle 4,2\rangle, \text{right}) = 0 + \gamma(-100) = -50$

$q(\langle 4,2\rangle, \text{up}) = 0 + \gamma(50) = 25$

$q(\langle 4,2\rangle, \text{left}) = -5 + \gamma^3(-100) = -17.5$

$q(\langle 4,2\rangle, \text{down}) = 0 + \gamma^3(-100) = -12.5$

The first step is a policy evaluation step. We'll see later how to *learn* the value of the current policy, but for now let's just evaluate the current policy by looking at the transition function and the reward.

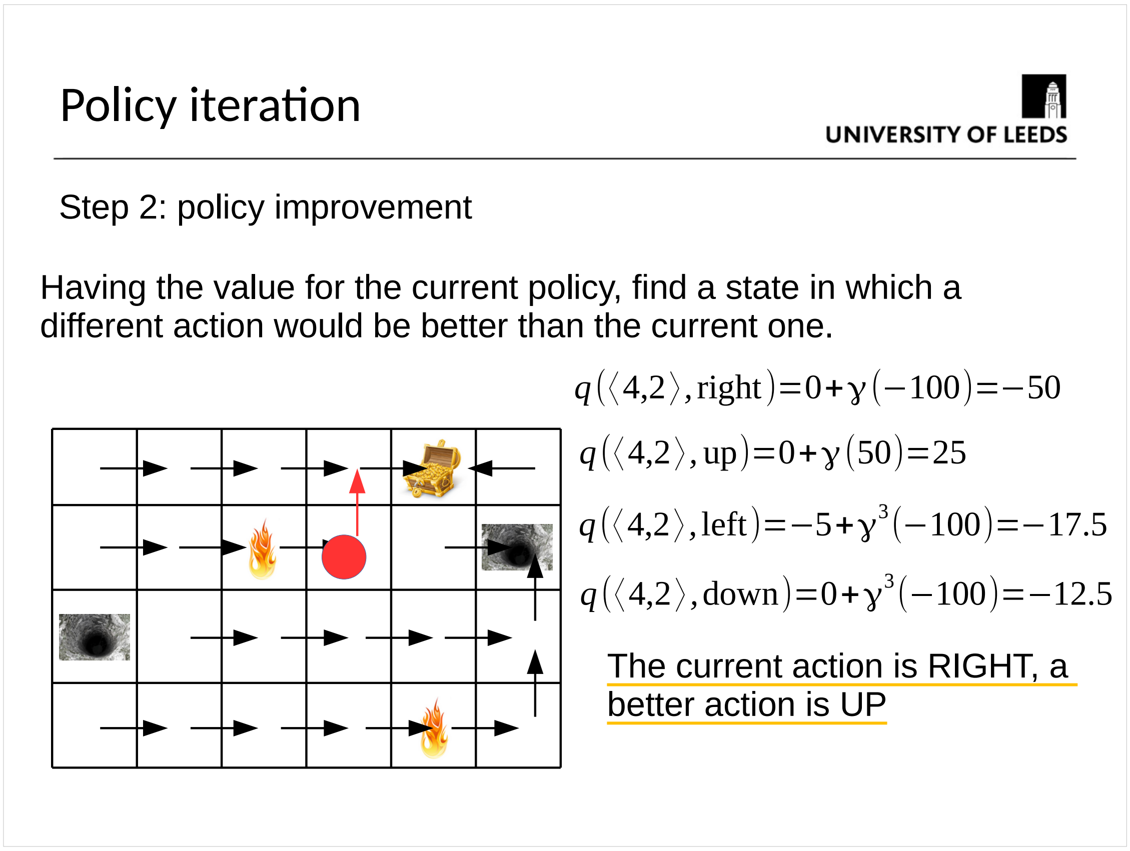
This must be done for every state, but for brevity we focus on a single state.

The current policy goes right in this state. Going right causes the agent to fall into the pit in the second step, so it's value is 0 for the first step, and -100 for the second step. The value of the second step is discounted by γ.

The expected value of going up and then follow the current policy is 0 for the first step, and then 50 for the second. So the return, that is the cumulative discounted reward, is 25.

The expected value of going left is an immediate -5 for the fire, and then falling into the pit in three steps, so γ is elevated to the power of 3. This value is a little higher than going right directly into the pit because the agent lives a little longer, but only marginally better.

Lastly, the expected value of going down is 0 immediately, and then -100 in 3 steps. This is a little better than going straight into the fire, but again results in the agent jumping into the pit, so still not great.



Policy iteration

UNIVERSITY OF LEEDS

Step 2: policy improvement

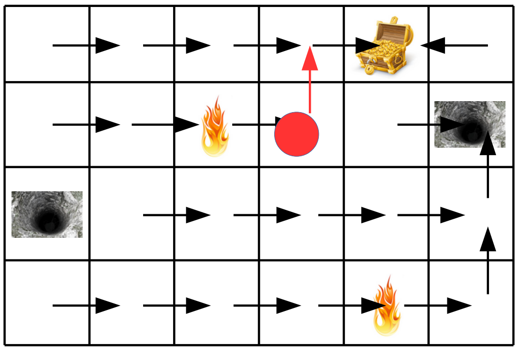
Having the value for the current policy, find a state in which a different action would be better than the current one.

$$q(\langle 4,2\rangle, \text{right}) = 0 + \gamma(-100) = -50$$

$$q(\langle 4,2\rangle, \text{up}) = 0 + \gamma(50) = 25$$

$$q(\langle 4,2\rangle, \text{left}) = -5 + \gamma^3(-100) = -17.5$$

$$q(\langle 4,2\rangle, \text{down}) = 0 + \gamma^3(-100) = -12.5$$

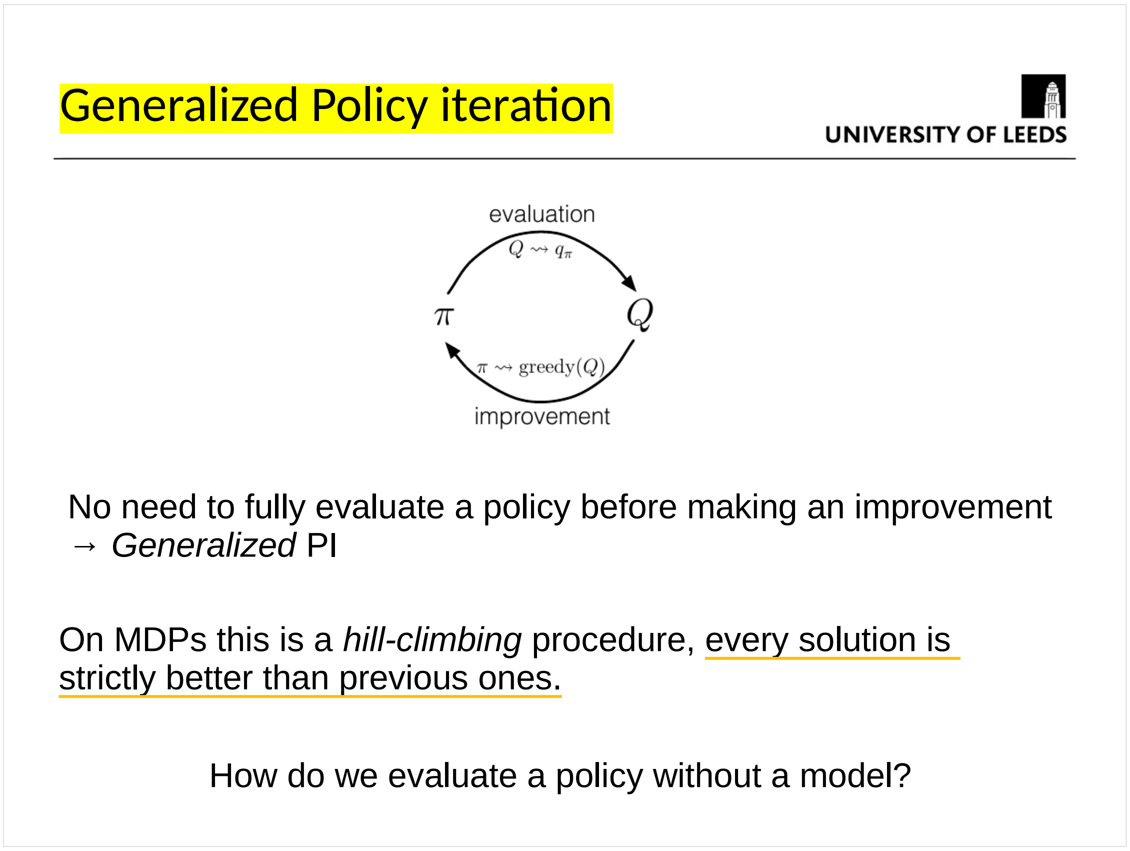


The current action is RIGHT, a better action is UP

The next step is to look at the evaluation of every action in every state (in our example we only look at one state), and change the policy to any action that achieves a higher expected return.

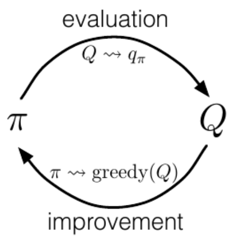
Now that the policy has changed, we can evaluate the new policy, and keep this loop until the policy does not change any more.

This loop is called: policy iteration.



Generalized Policy iteration

UNIVERSITY OF LEEDS


evaluation
$Q \rightsquigarrow q_\pi$
π
Q
$\pi \rightsquigarrow \mathrm{greedy}(Q)$
improvement

No need to fully evaluate a policy before making an improvement → *Generalized* PI

On MDPs this is a *hill-climbing* procedure, every solution is strictly better than previous ones.

How do we evaluate a policy without a model?

In practice, we don't have to evaluate the policy in every state before we can make an improvement, but as soon as we find one state in which a change of the action would improve the policy we can make the change, and start evaluating the new policy.

This loop is called *generalized policy iteration*, and is what we are going to use.

So far we computed the action-value function by hand, looking at the grid. This means we have access to the transition function (we know that taking action RIGHT moves the agent one step to the right), and the reward function (we know that the fire is -5, the pit -100, and the gold +50).

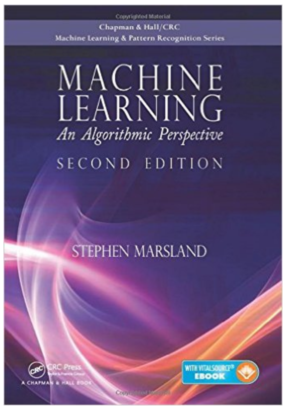
UNIVERSITY OF LEEDS

Conclusion

Learning outcomes

- Define and compute the value and action value function for a given policy
- Perform a step of generalized policy iteration

UNIVERSITY OF LEEDS



Chapter 11