Sequential Decision Problems

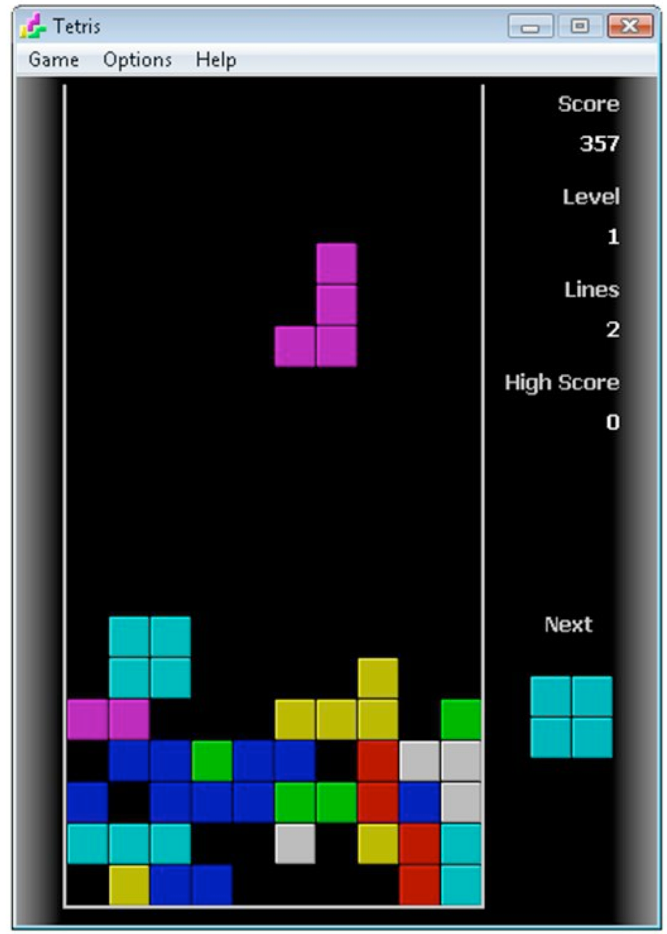
CITS3001 Algorithms, Agents and Artificial Intelligence

Tim French
Department of Computer Science and Software Engineering
The University of Western Australia

2021, Semester 2

Introduction

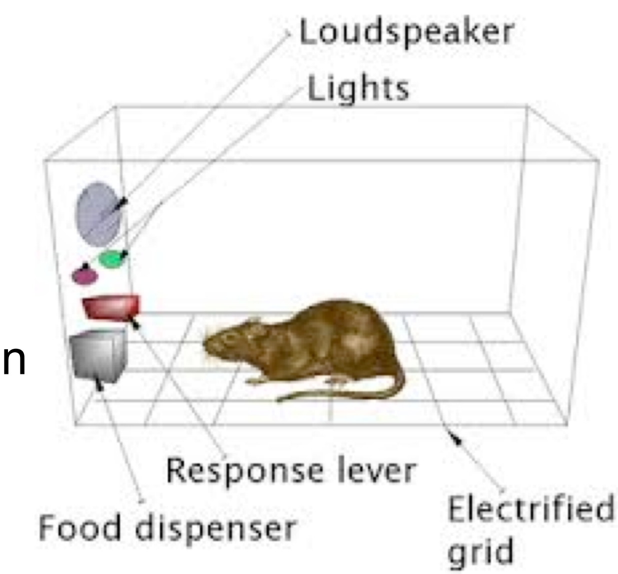
- We will define *sequential decision problems* (SDPs)
- We will discuss two major algorithms for solving SDPs
  - Value iteration:
    - estimate rewards
    - refine rewards, repeatedly
    - use rewards to make plan
  - Policy iteration:
    - make initial plan
    - calculate rewards and re-make plan, repeatedly
- We will discuss the related issues of
  - Delayed rewards
  - Immortal/eternal agents



Sequential decision problems

THE UNIVERSITY OF WESTERN AUSTRALIA

- A *sequential decision problem* (SDP) is a problem where the utility obtained by an agent depends on a sequence of decisions
- SDPs in known, accessible, deterministic domains can be solved using search algorithms that we have already seen
  - The result is a sequence of actions that lead (inevitably) to a "good" state
- But SDPs typically include utilities, uncertainty, sensing issues, *etc.*
  - They generalise the searching and planning problems that we have seen up to now
  - An agent needs to know what action to take in each possible state, allowing for future uncertainties
- A *policy* is a set of state-action rules
  - For each state, which action to take?
  - Providing a policy basically turns a utility-based agent into a simple reflex agent
- We need algorithms that can derive optimal policies for an agent faced with an SDP



An example SDP

- Beginning from the start state of 17.1(a):
  - The agent must select an action at each time step, from the set *{Up, Down, Left, Right}*
    - Each non-terminal state incurs a *step-cost*
  - The agent's interaction finishes when it reaches any terminal state
    - Each terminal state confers a "reward"
  - The agent wants to maximise its overall utility
    - The utility of a sequence of states is the sum of the step-costs, plus the terminal utility
- If actions are deterministic, it's trivial!
  - *[Up, Up, Right, Right, Right]*
- But each action has a pre-defined probability of "failure"
  - Given by the *transition model* in 17.1(b)
  - Non-determinism limits the usefulness of search
- So what's the best policy now?

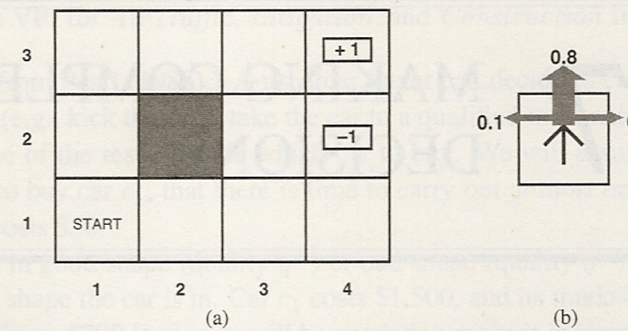


Figure 17.1 (a) A simple 4 × 3 environment that presents the agent with a sequential decision problem. (b) Illustration of the transition model of the environment: the "intended" outcome occurs with probability 0.8, but with probability 0.2 the agent moves at right angles to the intended direction. A collision with a wall results in no movement. The two terminal states have reward +1 and –1, respectively, and all other states have a reward of –0.04.

Optimal Policies

- The optimal policy for this environment depends on many factors
  - Each of the following points assumes “all else being equal”

- It depends on the transition model:
  - Less-certain actions imply a more conservative policy
- It depends on the terminal utilities:
  - A bigger discrepancy between the two implies a more conservative policy
- It depends on the step-cost:
  - A lower step-cost implies a more conservative policy

Optimal policies for various step costs

- 17.2(b1): get to any terminal ASAP!
- 17.2(b2): risk the bad terminal
- 17.2(a): ditto, but less
- 17.2(b3): avoid the bad terminal at all costs
- 17.2(b4): I want to live forever!

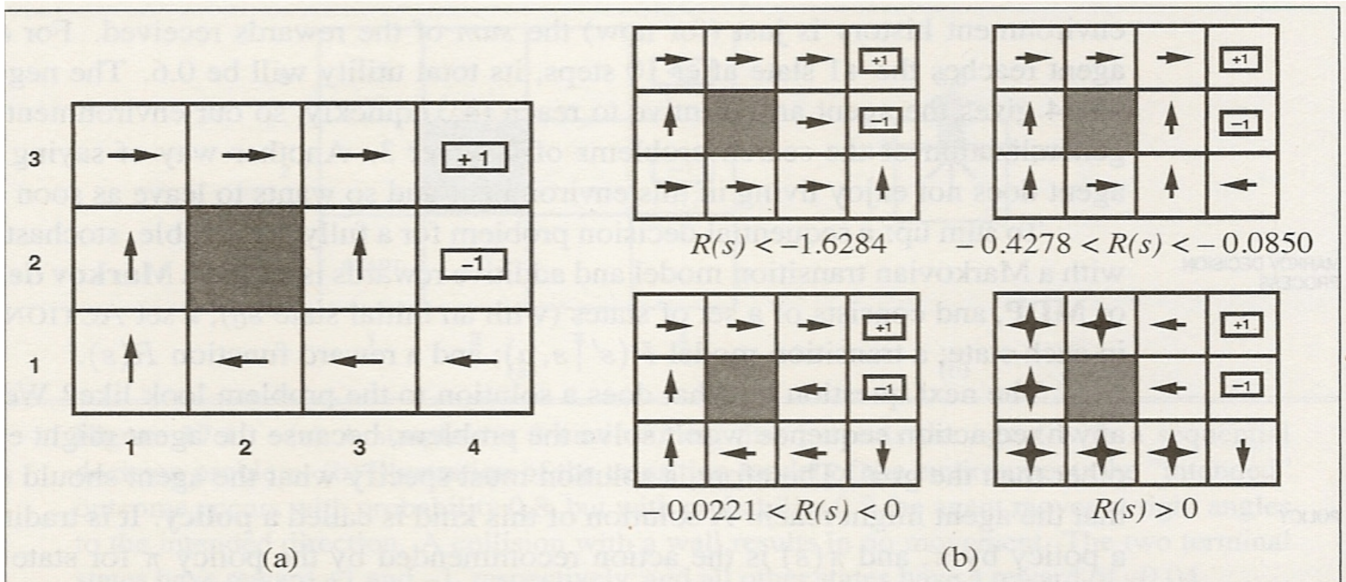


Figure 17.2 (a) An optimal policy for the stochastic environment with $R(s) = -0.04$ in the nonterminal states. (b) Optimal policies for four different ranges of $R(s)$.

Before we describe our two algorithms, we need to describe two fundamental processes that they employ

A policy determines a set of utilities

- Given any policy, we can determine the agent's corresponding utilities *if it follows that policy*
- For each non-terminal state, an equation describes its expected utility as a function of the transition model

e.g. for the policy in 17.2(a):

- $x_{33} = 0.8 \times 1 + 0.1 \times x_{33} + 0.1 \times x_{32} - 0.04$
- $x_{32} = 0.8 \times x_{33} + 0.1 \times x_{32} + 0.1 \times -1 - 0.0$
- $x_{23} = 0.8 \times x_{33} + 0.1 \times x_{23} + 0.1 \times x_{23} - 0.0$
- …

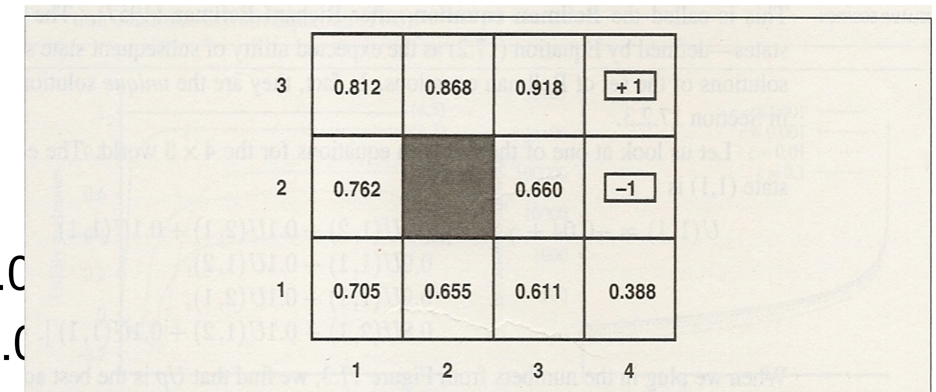


Figure 17.3 The utilities of the states in the 4×3 world, calculated with $\gamma = 1$ and $R(s) = -0.04$ for nonterminal states.

- In general, *n* non-terminal states gives *n* simultaneous linear equations
- Solving with Gaussian elimination gives the utilities
  - But Gaussian elimination is $O(n^3)$…
- This process is often called *value determination*

A set of utilities determines a policy

- Correspondingly: given a utility for each state, we can determine the optimal policy for the agent
- For each state *independently*, calculate the expected outcome for each action, and choose the best action

- *e.g.* for State 3,1 in 17.3:
  - *Up*: $0.8 \times x_{32} + 0.1 \times x_{21} + 0.1 \times x_{41} - 0.04 \approx 0.592$
  - *Down*: $0.8 \times x_{31} + 0.1 \times x_{41} + 0.1 \times x_{21} - 0.04 \approx 0.553$
  - *Right*: $0.8 \times x_{41} + 0.1 \times x_{32} + 0.1 \times x_{31} - 0.04 \approx 0.398$
  - *Left*: $0.8 \times x_{21} + 0.1 \times x_{31} + 0.1 \times x_{32} - 0.04 \approx 0.611$

So the best action in State 3,1 is *Left*
Note that the agent shouldn't just head for the adjacent state with the highest utility…

We shall call this process *action determination*

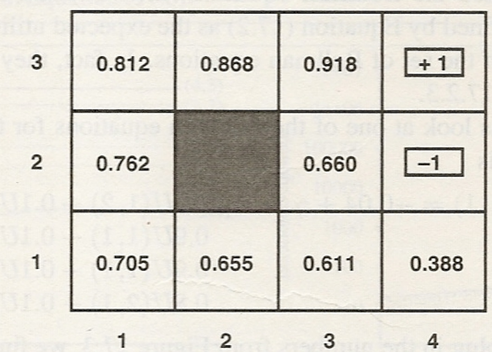


Figure 17.3 The utilities of the states in the 4 × 3 world, calculated with $\gamma = 1$ and $R(s) = -0.04$ for nonterminal states.

The Bellman Equation

THE UNIVERSITY OF WESTERN AUSTRALIA

- The utility of a state is specified formally by the Bellman equation [1957]

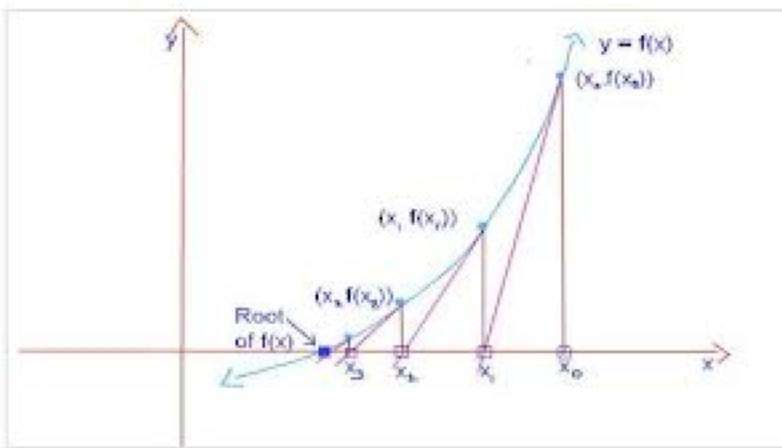
$$U_i = R_i + \max_a \sum_j M_{ij}^a U_j$$

- M_{ij}^a is the probability that doing Action *a* in State *i* leaves the agent in State *j*
  - *i.e.* it represents the transition model
- $\sum_j M_{ij}^a U_j$ is the weighted sum of all possible outcomes of doing Action *a* in State *i*
- $\max_a \sum_j M_{ij}^a U_j$ is the expected outcome of the best action to do in State *i*
- $R_i + \max_a \sum_j M_{ij}^a U_j$ is the cost of being in State *i*, plus the optimal cost from then on.

- The Bellman equation underpins both SDP algorithms
- But it cannot be solved directly because
  - The equations for the states are mutually dependent
  - The use of max_a means the equation is non-linear

Value iteration

- Basic idea:
  - Determine the true utility of each state
  - Then determine the optimal action in each state, by action determination
- To determine the utility of each state, use an iterative approximation algorithm
  - start with arbitrary utilities *U*
  - update *U* to make them *locally consistent* with Bellman
  - repeat until *U* is "close enough"
- This has been proven to converge, under reasonable assumptions



Aside: iterative approximation algorithms

- An iterative approximation algorithm that you may know is Newton's algorithm for finding square roots. Find the square root of *y* by repeatedly improving an initial estimate x_0, using $x_{k+1} = (x_k + y/x_k) / 2$
  - *e.g.* $y = 25$
  - $x_0 = 1$
  - $x_1 = 13$
  - $x_2 = 7.46$
  - $x_3 = 5.41$
  - $x_4 = 5.02$
  - $x_5 = 5.00002$
  - $x_6 = 5.00000000005$
  - *etc.*

- The key point in an iterative approximation algorithm is that the update step *f* is a *contraction*
  i.e. $u \neq v \rightarrow |f(u) - f(v)| < |u - v|$
  e.g. *f* might be "divide by 2"
- Applying *f* brings points closer together
- $f(fix_f) = fix_f$
  - *e.g.* the fixed point of "divide by 2" is 0
  - Therefore *f* brings any point closer to its fixed point
  - And any contraction has only one fixed point

Value iteration approximation

- The key to the algorithm is that in the (iterated) update step, the link between *U* and *U'* is broken
- *U'* (the new set of utilities) is created *under the assumption that U* (the old set of utilities) is correct
  - If *U* is correct, there will be no change and the iteration terminates
  - If *U* is not correct, *U'* will be closer to the correct values than *U*

function VALUE-ITERATION(mdp, ϵ) **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 γ
ϵ, the maximum error allowed in the utility of any state
local variables: U, U', vectors of utilities for states in S, initially zero
δ, 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

Figure 17.4 The value iteration algorithm for calculating utilities of states. The termination condition is from Equation (17.8).

Value iteration performance

- 17.5(a) shows how the utility of each state approaches the correct value as value iteration proceeds
- State 4,3 (a terminal) is immediately correct
- 3,3 achieves correctness early
  - It is "close to" a terminal
  - The other states get worse before they get better, *i.e.* until they are "connected to" a terminal
- As usual with iterative approximation algorithms, diminishing returns applies
  - The utilities approach the correct values asymptotically, and a threshold cut-off must be used

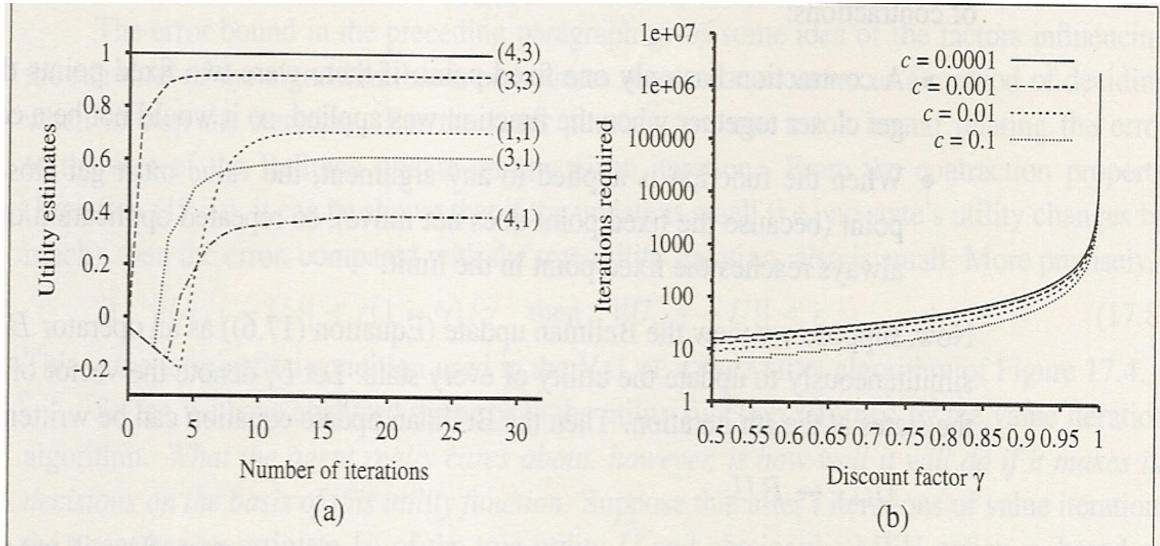


Figure 17.5 (a) Graph showing the evolution of the utilities of selected states using value iteration. (b) The number of value iterations k required to guarantee an error of at most $\epsilon = c \cdot R_{\max}$, for different values of c, as a function of the discount factor γ.

	1	2	3	4
3	0.812	0.868	0.918	+1
2	0.762		0.660	–1
1	0.705	0.655	0.611	0.388

Figure 17.3 The utilities of the states in the 4×3 world, calculated with $\gamma = 1$ and $R(s) = -0.04$ for nonterminal states.

Assessing performance

- But we can derive the optimal policy *without knowing the exact utilities*
- Calculate the *policy loss* at each iteration by using the current value of *U* to derive the “current policy” π
  - Then compare π with the optimal policy
- 17.6 shows, for each iteration, the error in the utilities vs. the policy loss
  - The policy loss is uniformly less than the error in the utilities
  - The optimal policy is derived long before the exact utilities are derived
- Can we use this idea to develop a faster algorithm?

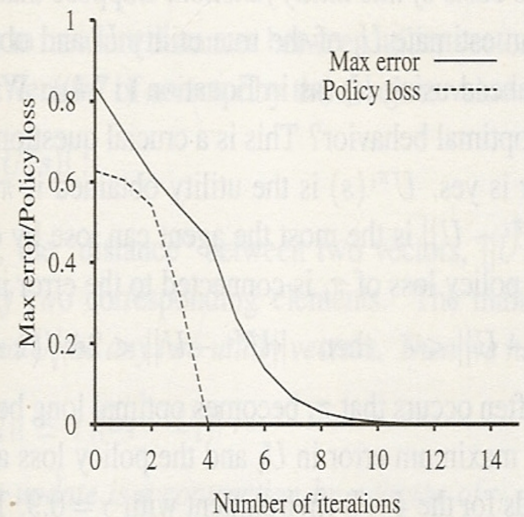
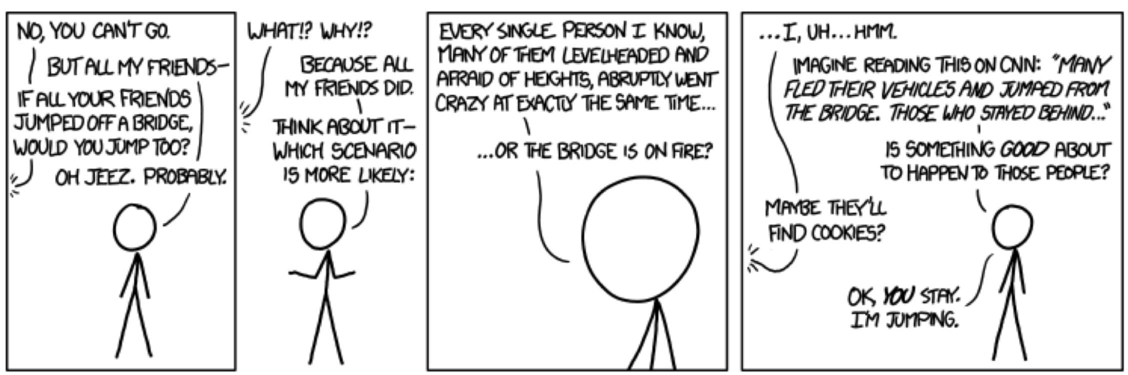


Figure 17.6 The maximum error $||U_i - U||$ of the utility estimates and the policy loss $||U^{\pi_i} - U||$, as a function of the number of iterations of value iteration.

Policy iteration

THE UNIVERSITY OF WESTERN AUSTRALIA

- Basic idea:
  - We (usually) don't need to know exact utilities; we just need to know what to do!
    - *e.g.* is jumping off a bridge –100 or –1,000?
  - Hence iterate on the actual policy, not its utilities
- To determine the optimal policy, use an iterative approximation algorithm
  - start with an arbitrary policy π
  - compute the utilities *U* of π, by value determination
  - update π according to *U*, by action determination
  - repeat until no change in π
- This also has been proven to converge, under reasonable assumptions



Policy iteration operation

- In each iteration
  - Derive the utilities from the current policy, then
  - Check each state to see if its action is optimal
- If there are any updates, iterate again
  - But updating a policy is a much “coarser” operation than updating a utility value
  - Hence convergence is quicker
- Deriving the utilities can be slow
  - Gaussian elimination is cubic in the no. of states
  - For large problems, it may be better to use (a simplified form of) value iteration itself!

function POLICY-ITERATION(mdp) **returns** a policy
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$
  local variables: U, a vector of utilities for states in S, initially zero
    π, a policy vector indexed by state, initially random

  repeat
    $U \leftarrow$ POLICY-EVALUATION(π, U, mdp)
    $unchanged? \leftarrow$ true
    for each state s **in** S **do**
      if $\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s'] > \sum_{s'} P(s' \mid s, \pi[s])\ U[s']$ **then do**
        $\pi[s] \leftarrow \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s']$
        $unchanged? \leftarrow$ false
  until $unchanged?$
  return π

Figure 17.7 The policy iteration algorithm for calculating an optimal policy.

Utilities over time

- In many disciplines where rewards are distributed through time, it is normal to regard present returns as being more valuable than future returns
  - "a bird in the hand is worth two in the bush"
  - From economic theory: *Net Present Value*
- In our context that is usually implemented by *discounting* future rewards
- Our additive rewards for a sequence of states

  $U([s_0, s_1, s_2, \ldots, s_n]) = R(s_0) + R(s_1) + R(s_2) + \ldots$

  becomes

  $U([s_0, s_1, s_2, \ldots, s_n]) = R(s_0) + \gamma\, R(s_1) + \gamma^2\, R(s_2) + \ldots$

- For a constant discount rate γ, this is equivalent to paying an interest rate of $1 / \gamma - 1$

Eternal agents

- This acquires especial importance in the context of eternal agents
  - Some environments have no terminal states
  - Some agents don't want to die!
- If two summations are infinitely long, it becomes difficult to compare them meaningfully without discounting
  - Quite likely they both grow indefinitely
- But with discounting they will be bounded
- Discounting also appeals intuitively to the idea that we cannot look too far ahead
  - cf. limited horizons in game-playing
  - A smaller value of γ implies a shorter horizon