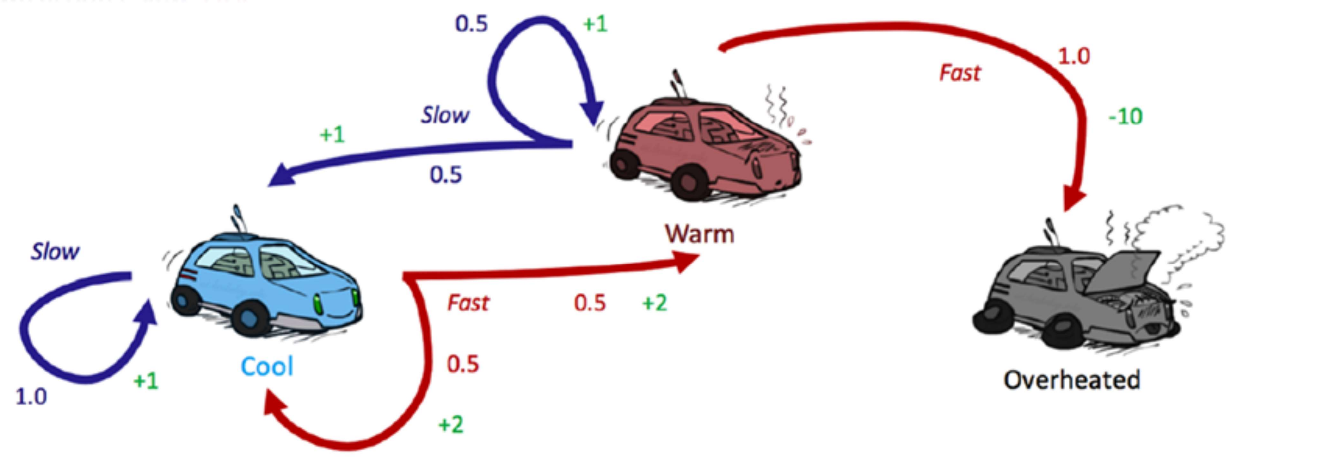
Complex Decisions

Chapter 17



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

States={cool, warm, overheated}
Actions={fast, slow}

γ=0.5
Initially: U_0(cool)=0, U_0(warm)=0, U_0(overheated)=0
Run two iterations of the value iteration algorithm.

U_{t+1}(cool) = max(
P(cool | cool, slow) * (R(cool, slow, cool) + γ U_t(cool)),
P(cool | cool, fast) * (R(cool, fast, cool) + γ U_t(cool)) + P(warm | cool, fast) * (R(cool, fast, warm) + γ U_t(warm)
) = max(1 * (1 + 0.5U_t(cool)), 0.5 * (2 + 0.5U_t(cool)) + 0.5 * (2 + 0.5U_t(warm))) = max(1 + 0.5U_t(cool), 2 + 0.25U_t(cool) + 0.25U_t(warm))
U_{t+1}(warm) = max(
P(cool | warm, slow) * (R(warm, slow, cool) + γ U_t(cool)) + P(warm | warm, slow) * (R(warm, slow, warm) + γ U_t(warm),
P(overheated | warm, slow) * (R(warm, slow, overheated) + γ U_t(overheated))
) = max(0.5 * (1 + 0.5U_t(cool)) + 0.5 * (1 + 0.5U_t(warm)), 1* (-10 + 0.5U_t(overheated))) = max(1 + 0.25U_t(cool) + 0.25U_t(warm), -10 + 0.5U_t(overheated))
U_{t+1}(overheated) = 0

Iteration 1:
U_1(cool) = max(1 + 0.5U_0(cool), 2 + 0.25U_0(cool) + 0.25U_0(warm)) = max(1 + 0.5*0, 2 + 0.25*0 + 0.25*0) = max(1, 2) = 2
U_1(warm) = max(1 + 0.25U_0(cool) + 0.25U_0(warm), -10 + 0.5U_0(overheated)) = max(1 + 0.25* + 0.25*0, -10 + 0.5*0) = max(1, -10) = 1
U_1(overheated) = 0

Iteration 2:
U_2(cool) = max(1 + 0.5U_1(cool), 2 + 0.25U_1(cool) + 0.25U_1(warm)) = max(1 + 0.5*2, 2 + 0.25*2 + 0.25*1) = max(2, 2.75) = 2.75
U_2(warm) = max(1 + 0.25U_1(cool) + 0.25U_1(warm), -10 + 0.5U_1(overheated)) = max(1 + 0.25*2 + 0.25*1, -10 + 0.5*0) = max(1.75, -10) = 1.75

U_2(overheated) = 0

Policy extraction:
π(cool) = argmax(slow: 1 + 0.5U_1(cool), fast: 2 + 0.25U_1(cool) + 0.25U_1(warm)) = argmax(slow: 1 + 0.5*2.75, fast 2 + 0.25*2.75 + 0.25*1.75) = argmax(slow: 2.375, fast: 3.125) = fast
π(warm) = argmax(slow: 1 + 0.25U_1(cool) + 0.25U_1(warm), fast: -10 + 0.5U_1(overheated)) = argmax(slow: 1 + 0.25*2.75 + 0.25*1.75, fast: -10 + 0.5*0) = argmax(slow: 1.375, fast: -10) = slow

17.4 Sometimes MDPs are formulated with a reward function $R(s, a)$ that depends on the action taken or with a reward function $R(s, a, s')$ that also depends on the outcome state.

a. Write the Bellman equations for these formulations.

b. Show how an MDP with reward function $R(s, a, s')$ can be transformed into a different MDP with reward function $R(s, a)$, such that optimal policies in the new MDP correspond exactly to optimal policies in the original MDP.

c. Now do the same to convert MDPs with $R(s, a)$ into MDPs with $R(s)$.

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

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

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

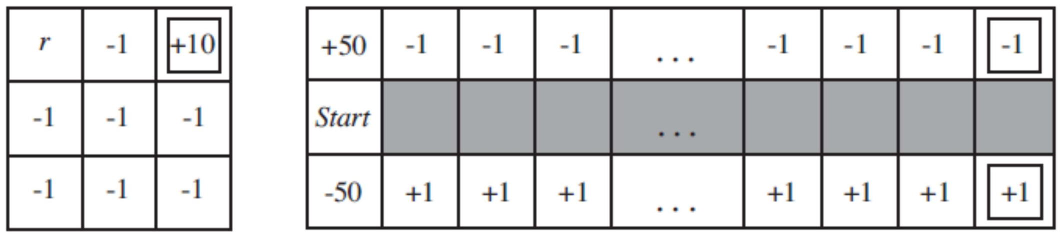
a. The key here is to get the max and summation in the right place.
For R(s), we use the first formula above
For R(s, a), we use the second formula above
For R(s, a, s′), we use the third formula above.

b. There are a variety of solutions here. One is to create a “pre-state” pre(s, a, s′) for every s, a, s′, such that executing a in s leads not to s′ but to pre(s, a, s′). In this state is encoded the fact that the agent came from s and did a to get here. From the pre-state, there is just one action b that always leads to s′. Let the new MDP have transition P′, reward R′, and discount γ′. Then

P′(pre(s, a, s′) | s, a) = P(s′ | s, a)
P′(s′ | pre(s, a, s′), b) = 1
R′(s, a) = 0
R′(pre(s, a, s′), b) = R(s, a, s′) / sqrt(γ)
γ′ = sqrt(γ)

c. In keeping with the idea of part (b), we can create states post(s, a) for every s, a, such that

P′(post(s, a, s′) | s, a) = 1
P′(s′ | post(s, a, s′), b) = P(s′ | s, a)
R′(s) = 0
R′(post(s, a, s′)) = R(s, a) / sqrt(γ)
γ′ = sqrt(γ)



(a) (b)

17.8 Consider the 3 × 3 world shown in Figure 17.14(a). The transition model is the same as in the 4 × 3 Figure 17.1: 80% of the time the agent goes in the direction it selects; the rest of the time it moves at right angles to the intended direction.

Implement value iteration for this world for each value of r below. Use discounted rewards with a discount factor of 0.99. Show the policy obtained in each case. Explain intuitively why the value of r leads to each policy.

a. $r = 100$

b. $r = -3$

c. $r = 0$

d. $r = +3$

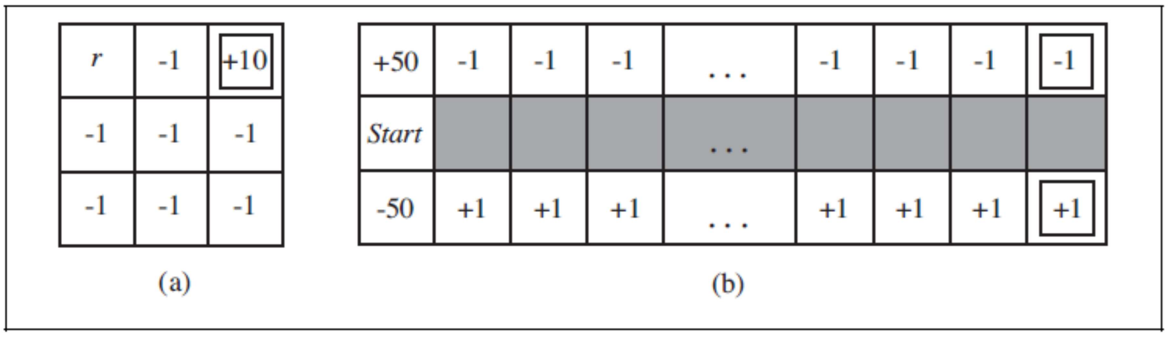
Instead of actually implementing value iteration, try to predict the policy by intuition

a.
LL_
ULD
ULL
Instead of seeking the goal, the agent would prefer to go the r=100 cell. Since the discount factor is high, the agent wouldn't mind paying some losses to reach a larger reward later on. So the agent will avoid the terminal cell (+10) and will do whatever it can to reach the +100 cell. At the cell below the +10, it would take the action "Down" to avoid entering the terminal cell by accident. At the cells in the middle column, it will alway pick "Left" to make sure it never goes right by accident.
At some cells, there are multiple optimal actions. For example, at the "+100" cell, it can pick "Up" or "Left" and the utility shouldn't differ.

b.
RR_
RRU
RRU
The agent will try to avoid the "-3" cell and try to reach the terminal state "+10" as soon as possible.

c.
RR_
UUU
UUU
The agent will try to reach the terminal state "+10" as soon as possible and it will prefer the path going through the "0"-reward cell (if there is no other path that is shorter).

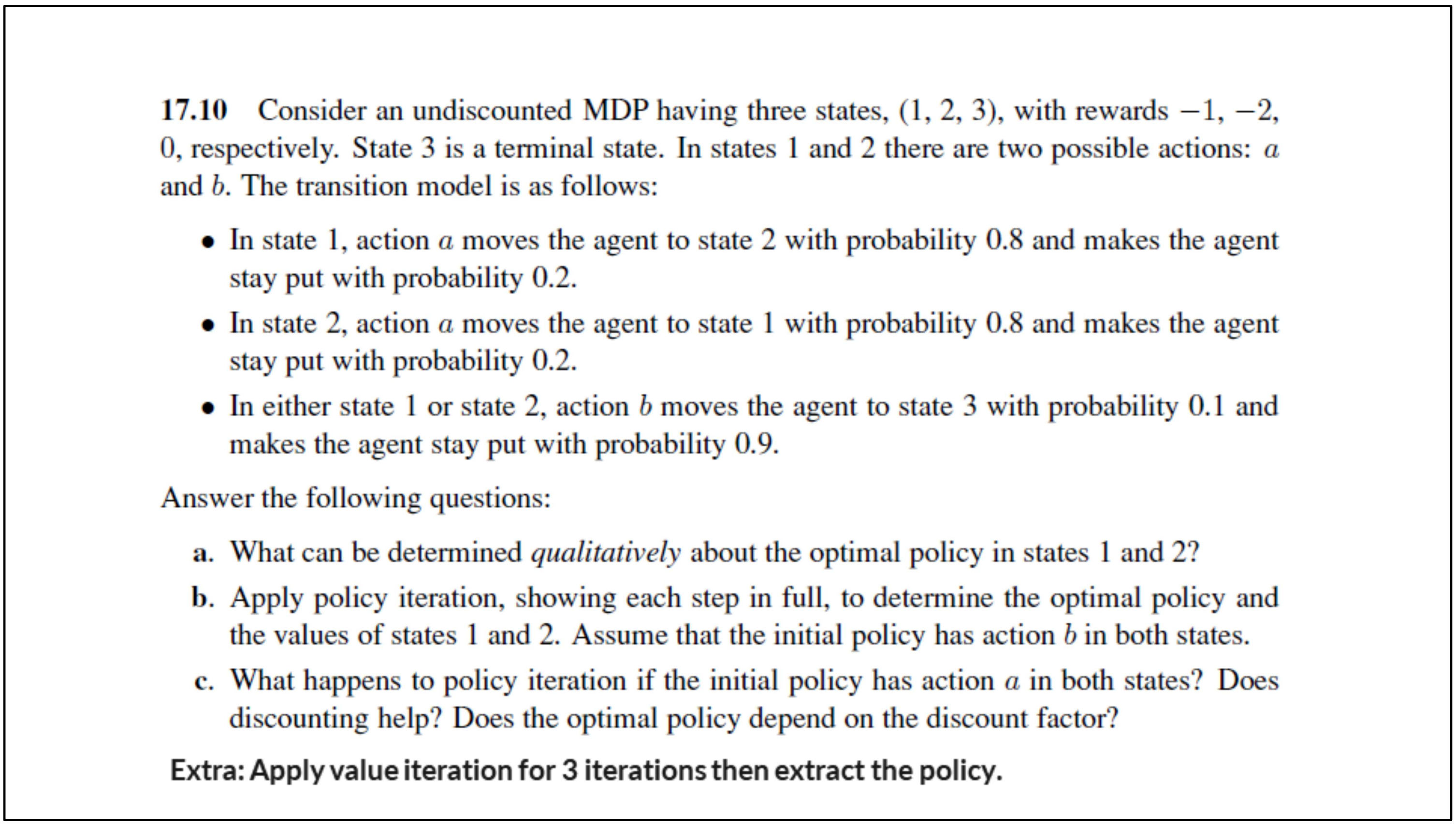
d.
LL_
ULD
ULL
This is similar to the first one. It may seem surprising that the agent will prefer the “+3” cell over the terminal “+10” cell but remember that the “+3” cell is not a terminal state. Thus the agent can keep getting rewards from it as long as it stays on it. So the agent can gain much more rewards by staying on the “+3” cell as long as possible than what it would get from reach the “+10” state which would return +10 once then end the episode. This is only true because the discount factor is high. If it was low, the agent would prefer getting the “+10” early than waiting to get multiple “+3”s in the future.



17.9 Consider the 101 × 3 world shown in Figure 17.14(b). In the start state the agent has a choice of two deterministic actions, *Up* or *Down*, but in the other states the agent has one deterministic action, *Right*. Assuming a discounted reward function, for what values of the discount γ should the agent choose *Up* and for which *Down*? Compute the utility of each action as a function of γ. (Note that this simple example actually reflects many real-world situations in which one must weigh the value of an immediate action versus the potential continual long-term consequences, such as choosing to dump pollutants into a lake.)

$U((1,1)) = -50 + \gamma^1 + \gamma^2 + \gamma^3 + .. + \gamma^{100} = -50 + \gamma(1 - \gamma^{100})/(1 - \gamma)$
$U((1,3)) = 50 - \gamma^1 - \gamma^2 - \gamma^3 - .. - \gamma^{100} = 50 - \gamma(1 - \gamma^{100})/(1 - \gamma)$

To choose up, it must be $U((1,1)) < U((1,3))$
$-50 + \gamma(1 - \gamma^{100})/(1 - \gamma) < 50 - \gamma(1 - \gamma^{100})/(1 - \gamma)$
$2\gamma(1 - \gamma^{100})/(1 - \gamma) < 2(50)$
$\gamma(1 - \gamma^{100}) < 50(1 - \gamma)$ since $1 - \gamma > 0$
This can be solved using numerical methods.
$\gamma < 0.9843976692$
So if $\gamma < 0.9843976692$, we choose up
Otherwise, we choose down



17.10 Consider an undiscounted MDP having three states, (1, 2, 3), with rewards -1, -2, 0, respectively. State 3 is a terminal state. In states 1 and 2 there are two possible actions: *a* and *b*. The transition model is as follows:

- In state 1, action *a* moves the agent to state 2 with probability 0.8 and makes the agent stay put with probability 0.2.
- In state 2, action *a* moves the agent to state 1 with probability 0.8 and makes the agent stay put with probability 0.2.
- In either state 1 or state 2, action *b* moves the agent to state 3 with probability 0.1 and makes the agent stay put with probability 0.9.

Answer the following questions:

a. What can be determined *qualitatively* about the optimal policy in states 1 and 2?

b. Apply policy iteration, showing each step in full, to determine the optimal policy and the values of states 1 and 2. Assume that the initial policy has action *b* in both states.

c. What happens to policy iteration if the initial policy has action *a* in both states? Does discounting help? Does the optimal policy depend on the discount factor?

Extra: Apply value iteration for 3 iterations then extract the policy.

a. The policy should be (b, a, _)
Because, the agent will try reach the terminal state (state 3) as soon as possible to stop getting penalties from being in state 1 or 2.
The only action to reach state 3 is “b” which only has a probability of 0.1 to actually move the agent to state 3. So the agent should prefer to go to state 1 first since it has a lower penalty then try to go to state 3 from there. Therefore, the action to pick in state 2 is “a” while the action to pick in state 1 is “b”.

b. Since π(1) = b and π(2) = b, the values for this policy can be computed from using the following equation:
U(1) = -1 + 0.1*U(3) + 0.9*U(1)
U(2) = -2 + 0.1*U(3) + 0.9*U(2)
U(3) = 0
By solving this equation, we get U(1) = -10, U(2) = -20, U(3) = 0
Now, we compute the new policy
π(1) = argmax(a: 0.8*U(2)+0.2*U(1), b: 0.1*U(3) + 0.9*U(1)) = argmax(a: 0.8*-20+0.2*-10, b: 0.1*0 + 0.9*-10) = argmax(a: -18, b: -9) = b
π(2) = argmax(a: 0.8*U(1)+0.2*U(2), b: 0.1*U(3) + 0.9*U(2)) = argmax(a: 0.8*-10+0.2*-20, b: 0.1*0 + 0.9*-20) = argmax(a: -12, b: -18) = a
Then we get the new equations:
U(1) = -1 + 0.1*U(3) + 0.9*U(1)
U(2) = -2 + 0.8*U(1) + 0.2*U(2)
U(3) = 0
So U(1) = -10, U(2) = -12.5, U(3) = 0
Now, we compute the new policy
π(1) = argmax(a: 0.8*U(2)+0.2*U(1), b: 0.1*U(3) + 0.9*U(1)) = argmax(a: 0.8*-12.5+0.2*-

10, b: 0.1*0 + 0.9*-10) = argmax(a: -12, b: -9) = b
π(2) = argmax(a: 0.8*U(1)+0.2*U(2), b: 0.1*U(3) + 0.9*U(2)) = argmax(a: 0.8*-10+0.2*-12.5, b: 0.1*0 + 0.9*-12.5) = argmax(a: -10.5, b: -11.25) = a
Since the policy did not change from the last iteration, we stop now.
The optimal policy is (b, a, _).

c. Since π(1) = a and π(2) = a, the values for this policy can be computed from using the following equation:
U(1) = -1 + 0.8*U(2) + 0.2*U(1)
U(2) = -2 + 0.8*U(1) + 0.2*U(2)
U(3) = 0
These equations has no solution.
If we add a discount factor γ, the equations will become
U(1) = -1 + γ(0.8*U(2) + 0.2*U(1))
U(2) = -2 + γ(0.8*U(1) + 0.2*U(2))
U(3) = 0
Which are solvable if γ < 1
And yes, the optimal policy depends on the discount factor.

Extra:
Assume that initially, U(1)=U(2)=U(3)=0
Iteration 1:
U(1) = -1 + max(0.8*U(2)+0.2*U(1), 0.1*U(3) + 0.9*U(1)) = -1 + max(0.8*0+0.2*0, 0.1*0 + 0.9*0) = -1 + max(0, 0) = -1 + 0 = -1
U(2) = -2 + max(0.8*U(1)+0.2*U(2), 0.1*U(3) + 0.9*U(2)) = -2 + max(0.8*0+0.2*0, 0.1*0 + 0.9*0) = -2 + max(0, 0) = -2 + 0 = -2
U(3) = 0
Iteration 2:
U(1) = -1 + max(0.8*U(2)+0.2*U(1), 0.1*U(3) + 0.9*U(1)) = -1 + max(0.8*-2+0.2*-1, 0.1*0 + 0.9*-1) = -1 + max(-1.8, -0.9) = -1 + -0.9 = -1.9
U(2) = -2 + max(0.8*U(1)+0.2*U(2), 0.1*U(3) + 0.9*U(2)) = -2 + max(0.8*-1+0.2*-2, 0.1*0 + 0.9*-2) = -2 + max(-1.2, -1.8) = -2 + -1.2 = -3.2
U(3) = 0
Iteration 3:
U(1) = -1 + max(0.8*U(2)+0.2*U(1), 0.1*U(3) + 0.9*U(1)) = -1 + max(0.8*-3.2+0.2*-1.9, 0.1*0 + 0.9*-1.9) = -1 + max(-2.94, -1.71) = -1 + -1.71 = -2.71
U(2) = -2 + max(0.8*U(1)+0.2*U(2), 0.1*U(3) + 0.9*U(2)) = -2 + max(0.8*-1.9+0.2*-3.2, 0.1*0 + 0.9*-3.2) = -2 + max(-2.16, -2.38) = -2 + -2.16 = -4.16
U(3) = 0
Now we extract the policy:
π(1) = argmax(a: 0.8*U(2)+0.2*U(1), b: 0.1*U(3) + 0.9*U(1)) = argmax(a: 0.8*-4.16+0.2*-2.71, b: 0.1*0 + 0.9*-2.71) = argmax(a: -3.37, b: -2.439) = b
π(2) = argmax(a: 0.8*U(1)+0.2*U(2), b: 0.1*U(3) + 0.9*U(2)) = argmax(a: 0.8*-2.71+0.2*-4.16, b: 0.1*0 + 0.9*-4.16) = argmax(a: -3, b: -3.744) = a