Artificial Intelligence Programming

Markov Decision Processes

Cindi Thompson

Department of Computer Science

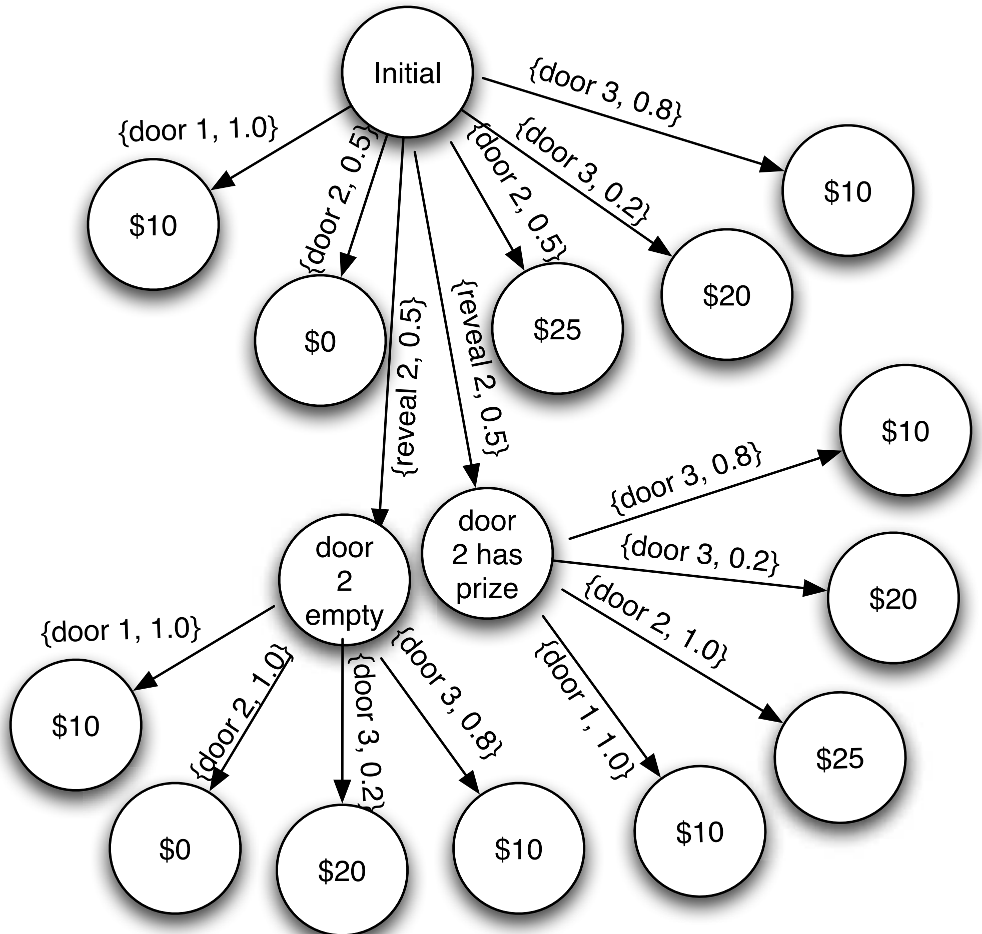
University of San Francisco

Making Sequential Decisions

- Previously, we've talked about:
  - Making one-shot decisions in a deterministic environment
  - Making sequential decisions in a deterministic environment
    - Search
    - Inference
  - Making one-shot decisions in a stochastic environment
    - Probabilities
    - Expected Utility
- What about sequential decisions in a stochastic environment?

Sequential Decisions

- We've thought a little bit about this in terms of value of information.
- We can model this as a state-space problem.
- We can even use a minimax-style approach to determine the optimal actions to take.



Expected Utility

- Recall that the expected utility of an action is the utility of each possible outcome, weighted by the probability of that outcome occurring; last week we wrote this $\sum s \in SP(s)U(s)$
- Let's write this a little differently:
  - from state s, an agent may take actions $a_1, a_2, ..., a_n$.
  - Each action a_i can lead to states $s_{i1}, s_{i2}, ..., s_{im}$, with probability $p_{i1}, p_{i2}, ..., p_{im}$

$$EU(a_i) = \sum p_{ij} U(s_{ij})$$

- We call the set of probabilities and associated states the state transition model.
- The agent should choose the action a' that maximizes EU.

Markovian environments

- We can extend this idea to sequential environments.
- Problem: How to determine transition probabilities?
  - The probability of reaching state s given action a might depend on previous actions that were taken.
  - Reasoning about long chains of probabilities can be complex and expensive.
- The Markov assumption says that state transition probabilities depend only on a finite number of parents.
- Simplest: a first-order Markov process. State transition probabilities depend only on the previous state.
  - This is what we'll focus on.

Stationary Distributions

- We'll also assume a *stationary distribution*
- This says that the probability of reaching a state s' given action a from state s with history H does not change.
- Different histories may produce different probabilities
- Given identical histories, the state transitions will be the same.
- We'll also assume that the utility of a state does not change throughout the course of the problem.
  - In other words, our model of the world does not change while we are solving the problem.

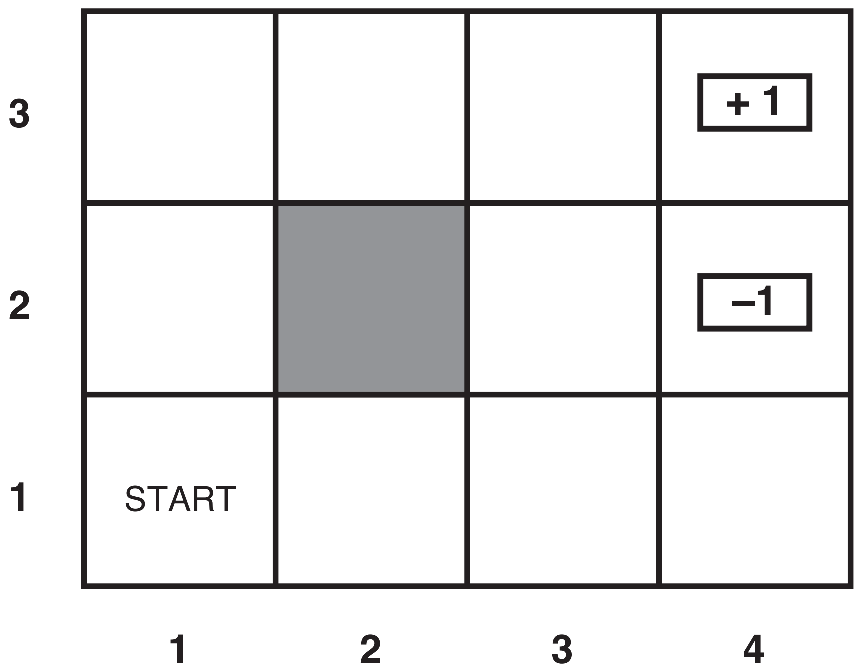
Assumptions restated

- The state transition function depends only on the current state.
- Probability distributions do not change while we are solving the problem.
- We can assign utilities to outcomes

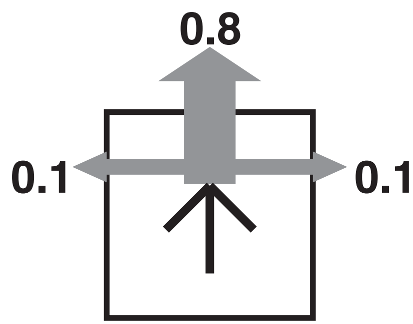
Solving sequential problems

- If we have to solve a sequential problem, the total utility will depend on a sequence of states $s_1, s_2, ..., s_n$.
- Let's assign each state a utility or *reward* $R(s_i)$.
- Agent wants to maximize the sum of rewards.
- We call this formulation a Markov decision process.
  - Formally:
  - An initial state s_0
  - A discrete set of states and actions
  - A Transition model: $p(s'|a, s)$ that indicates the probability of reaching state s' from s when taking action a.
  - A reward function: $R(s)$

Example grid problem



(a)



(b)

- Agent moves in the "intended" direction with probability 0.8, and at a right angle with probability 0.2
- What should an agent do at each state to maximize reward?

MDP solutions

- Since the environment is stochastic, a solution will not be an action sequence.
- Instead, we must specify what an agent should do in any reachable state.
- We call this specification a *policy*
    - “If you’re below the goal, move up.”
    - “If you’re in the left-most column, move right.”
- We denote a policy with π, and $\pi(s)$ indicates the policy for state s.

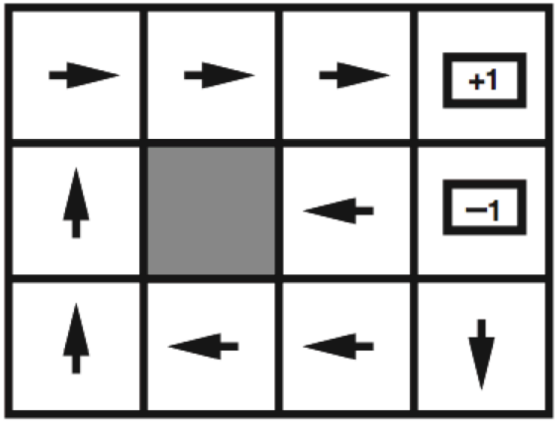
MDP solutions

- Things to note:
  - We've wrapped the goal formulation into the problem
    - Different goals will require different policies.
  - We are assuming a great deal of (correct) knowledge about the world.
    - State transition models, rewards
    - We'll see how to learn these without a model.

Comparing policies

- We can compare policies according to the expected utility of the histories they produce.
- The policy with the highest expected utility is the *optimal policy*.
- Once an optimal policy is found, the agent can just look up the best action for any state.

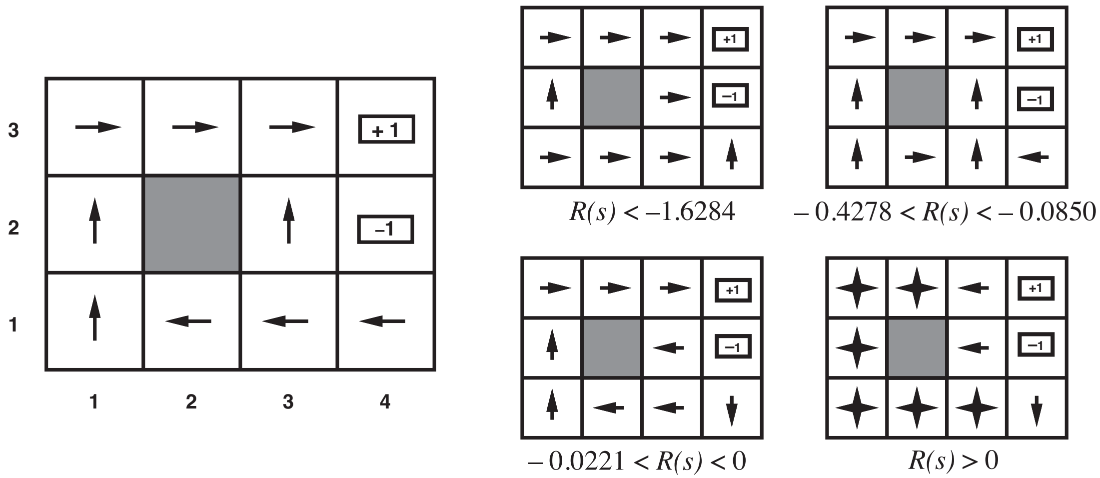
Example Grid Policy



Non-goal Costs

- Spending unlimited time trying to find the best solution is not always the best idea.
- We can give a cost (negative utility) to each non-goal state
- Agent is penalized for taking too long to find the goal state

Example grid problem



(a) (b)

Left figure: R(s) = -0.04; R(S)= “Reward” for non-goal state

- As the costs for nonterminal states change, so does the optimal policy.
- Very high cost: Agent tries to exit immediately
- Middle ground: Agent tries to avoid bad exit
- Positive reward: Agent doesn’t try to exit.

More on reward functions

- In solving an MDP, an agent must consider the value of future actions.
- There are different types of problems to consider:
- Horizon - does the world go on forever?
  - Finite horizon: after N actions, the world stops and no more reward can be earned.
  - Infinite horizon; World goes on indefinitely, or we don't know when it stops.
    - Infinite horizon is simpler to deal with, as policies don't change over time.

More on reward functions

- We also need to think about how to value future reward.
- \$100 is worth more to me today than in a year.
- We model this by *discounting* future rewards.
  - γ is a *discount factor*
- $U(s_0, s_1, s_2, s_3, ...) =$
  $R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \gamma^3 R(s_3) + ..., \gamma \in [0, 1]$
- If γ is large, we value future states
- if γ is low, we focus on near-term reward
- In monetary terms, a discount factor of γ is equivalent to an interest rate of $(1/\gamma) - 1$

More on reward functions

- Discounting lets us deal sensibly with infinite horizon problems.
  - Otherwise, all EUs would approach infinity.
- Expected utilities will be finite if rewards are finite and bounded and $\gamma < 1$.
- We can now describe the optimal policy π^* as:

$$\pi^* = argmax_\pi EU(\sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi)$$

Value iteration

- How to find an optimal policy?
- We'll begin by calculating the expected utility of each state and then selecting actions that maximize expected utility.
- In a sequential problem, the utility of a state is the expected utility of all the state sequences that follow from it.
- This depends on the policy π being executed.
- Essentially, $U(s)$ is the expected utility of executing an optimal policy from state s.

Utilities of States

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

- Notice that utilities are highest for states close to the +1 exit.

Utilities of States

- 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.
- $$U(s) = R(s) + \gamma max_a \sum_{s'} P(s'|s,a)U(s')$$
- This is called the Bellman equation
- Example:

$$\begin{aligned} U(1,1) = -0.04 + \gamma max(0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1) \\ 0.9U(1,1) + 0.1U(1,2), \\ 0.9U(1,1) + 0.1U(2,1), \\ 0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1)) \end{aligned}$$

Dynamic Programming

- Solving he Bellman equation is a dynamic programming problem.
- In an acyclic transition graph, you can solve these recursively by working backward from the final state to the initial states.
- Can't do this directly for transition graphs with loops.

Value Iteration

- Since state utilities are defined in terms of other state utilities, how to find a closed-form solution?
- We can use an iterative approach:
  - Give each state random initial utilities.
  - Calculate the new left-hand side for a state based on its neighbors' values.
  - Propagate this to update the right-hand-side for other states,
  - Update rule:
    $U_{i+1}(s) = R(s) + \gamma max_a \sum_{s'} P(s'|s,a)U_i(s')$
- This is guaranteed to converge to the solutions to the Bellman equations.

Markov Decision Problem

First, let's formally define a Markov Decision Problem (MDP):

- States, S
- Actions, $A(s)$
- Transition model, $P(s'|s, a)$
- Rewards, $R(s)$
- Discount, γ

Value Iteration algorithm

Given an MDP and ϵ, the max error allowed in the utility of any state

```
Assign random utilities to each state
do
   for s in states
      U(s) = R(s) + gamma * max P(s'|s,a) U(s')
until
     all utilities change by less then delta
```

- where $\delta = \epsilon * (1 - \gamma)/\gamma$

Example

1 0.1	2 -0.1	3 0.05	**+1**
4 -0.02		5 0.15	**-1**
6 0.0	7 0.1	8 -0.1	9 0.15

- Start by assigning random utilities to each state.
- Assume $\gamma = 0.8$
- Assume time cost is -0.04: $(R(s) = -0.04)$
- Assume $\epsilon = 0.01$: $\delta = 0.0025$

Example

1 0.03	2 -0.02	3 0.62	**+1**
4 0.02		5 0.05	**-1**
6 0.02	7 0.02	8 0.08	9 0.06

- After one iteration, here are the estimated values.

Example

1 -0.02	2 0.35	3 0.65	**+1**
4 -0.02		5 0.28	**-1**
6 -0.02	7 0.01	8 0.02	9 0.01

- After two iterations, here are the estimated values.

Example

1 0.19	2 0.43	3 0.69	+1
4 -0.06		5 0.32	-1
6 -0.04	7 -0.03	8 0.14	9 -0.03

- After three iterations, here are the estimated values.

Example

1 0.25	2 0.47	3 0.68	**+1**
4 0.07		5 0.34	**-1**
6 -0.07	7 0.04	8 0.16	9 -0.03

- After four iterations, here are the estimated values.

Example

1 0.27	2 0.47	3 0.68	**+1**
4 0.13		5 0.34	**-1**
6 0.0	7 0.07	8 0.18	9 -0.02

- After five iterations, here are the estimated values.

Example

1 0.29	2 0.47	3 0.68	**+1**
4 0.15		5 0.34	**-1**
6 0.04	7 0.08	8 0.18	9 -0.01

- After six iterations, here are the estimated values.
- At this point, we are close to converging.

Discussion

- Strengths of Value iteration
  - Guaranteed to converge to correct solution
  - Simple iterative algorithm
- Weaknesses:
  - Convergence can be slow
  - We really don’t need all this information
  - Just need *what to do* at each state.

Policy iteration

- Policy iteration helps address these weaknesses.
- Searches directly for optimal policies, rather than state utilities.
- Same idea: iteratively update policies for each state.
- Two steps:
  - Given a policy, compute the utilities for each state.
  - Compute a new policy based on these new utilities.

Policy iteration algorithm

Initialize all state utilities to zero
Choose a random policy π_0
$i = 0$
do
    Perform "Policy evaluation":
    Evaluate $U_\pi(s)$ values if we were to follow π_i
    For $s \in S$
        If expected utility for any action, a, from s $> U_\pi(s)$:
            i++
            $\pi_i[s] = a$
While any updates to π

Policy evaluation

How does policy evaluation work?

- We don't need full value iteration (phew!): action in each state is fixed by the policy
- Simplified version of Bellman's:

$$U_i(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s))U_i(s')$$

- For example, if $\pi_i(1,1) =$Up, then

$$U_i(1,1) = -0.04 + 0.8U_i(1,2) + 0.1U_i(1,1) + 0.1U_i(2,1)$$

- No more max! But still $O(n^3)$, where n is the number of states

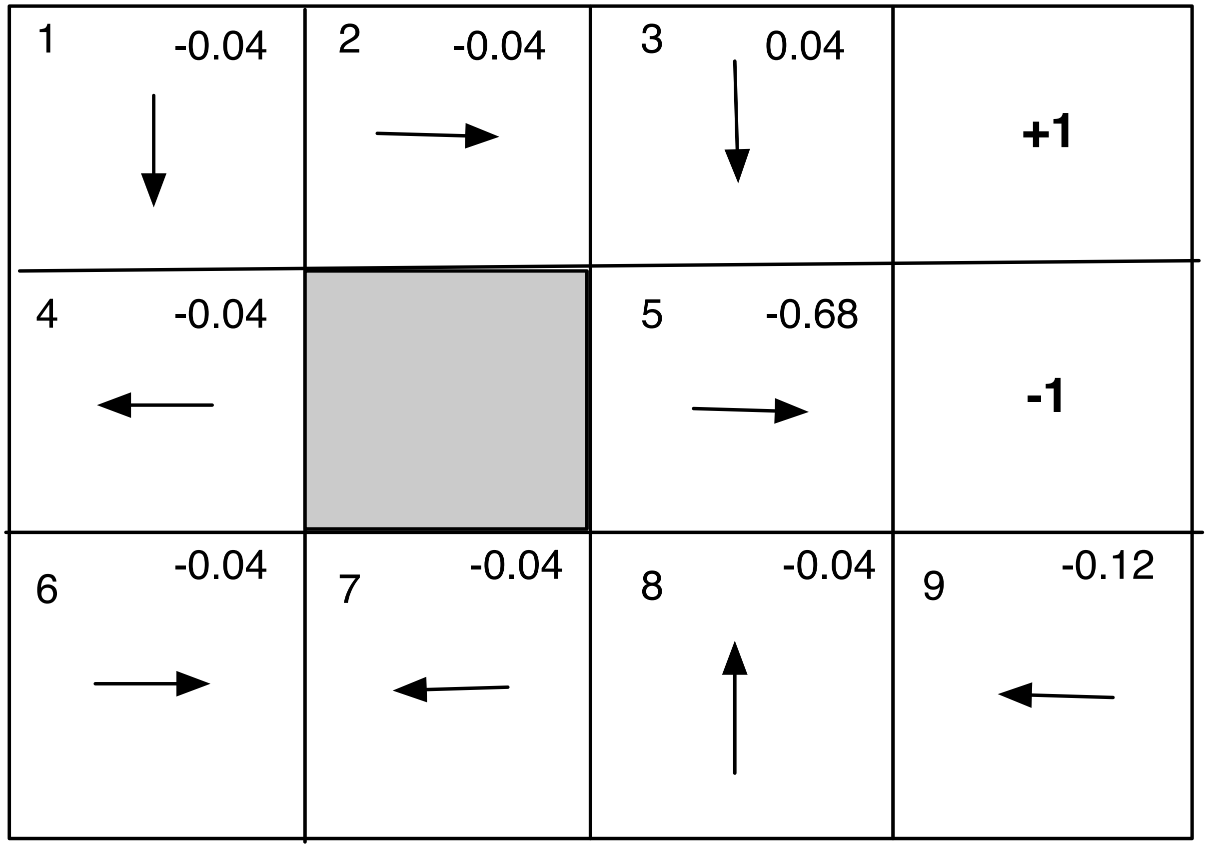
Modified Policy iteration

Alternative to previous slide: simplified Bellman update *estimates* the policy value:

$$U_{i+1}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s))U_i(s')$$

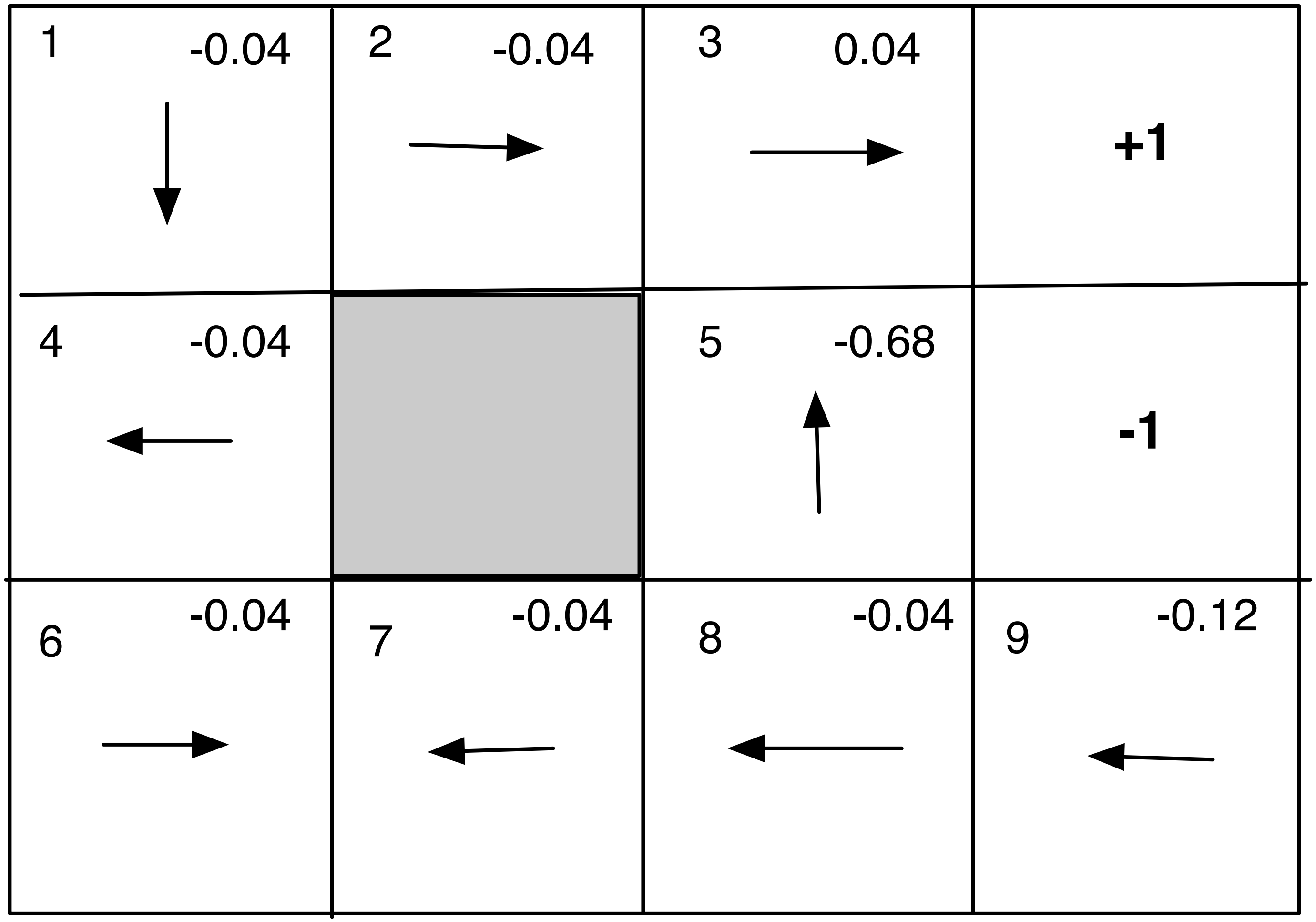
This is only $O(n * b)$ where b is the branching factor of the space.

Example



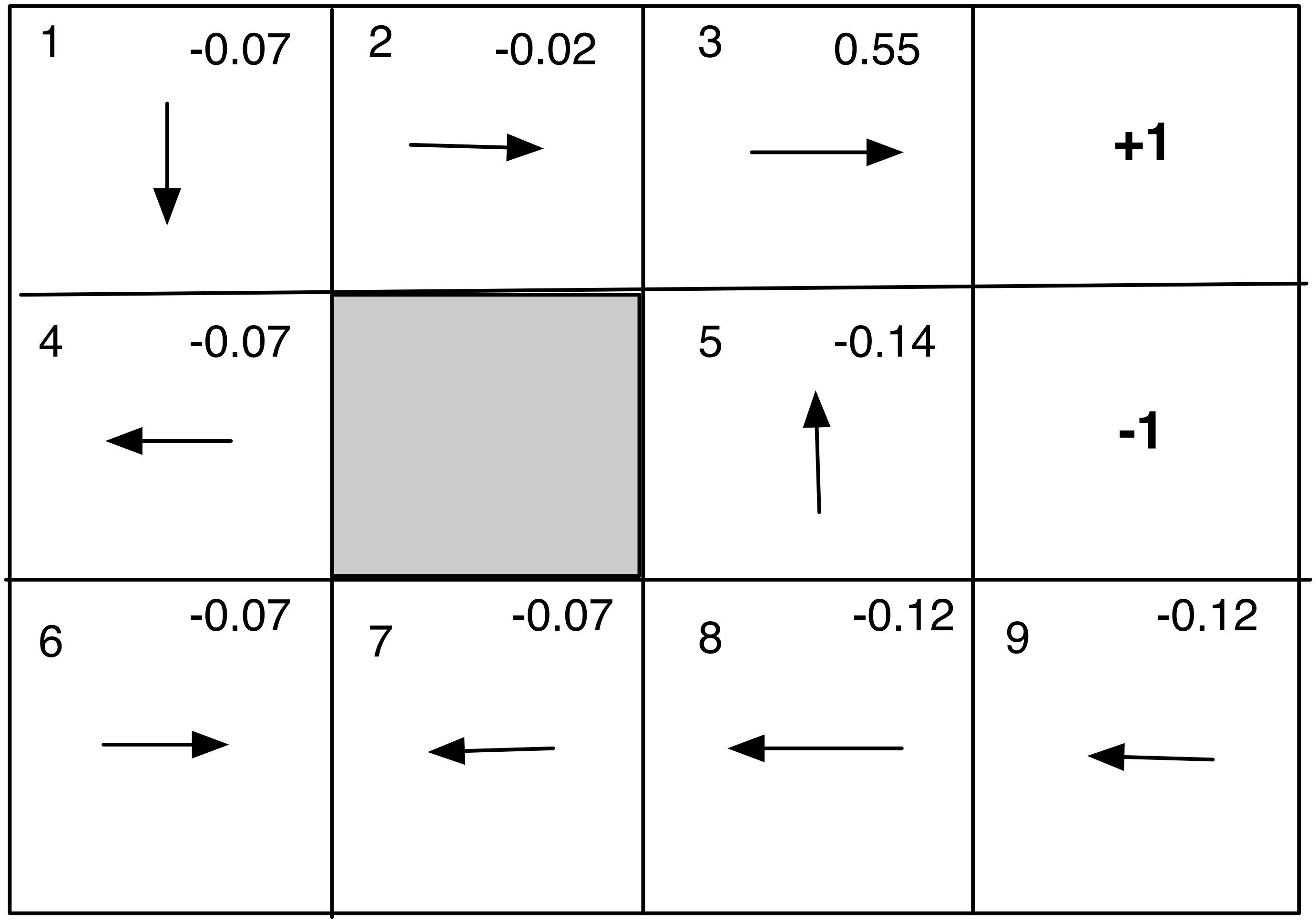
- Assign random policies
- Evaluate state utilities based on these policies

Policy Iteration Example



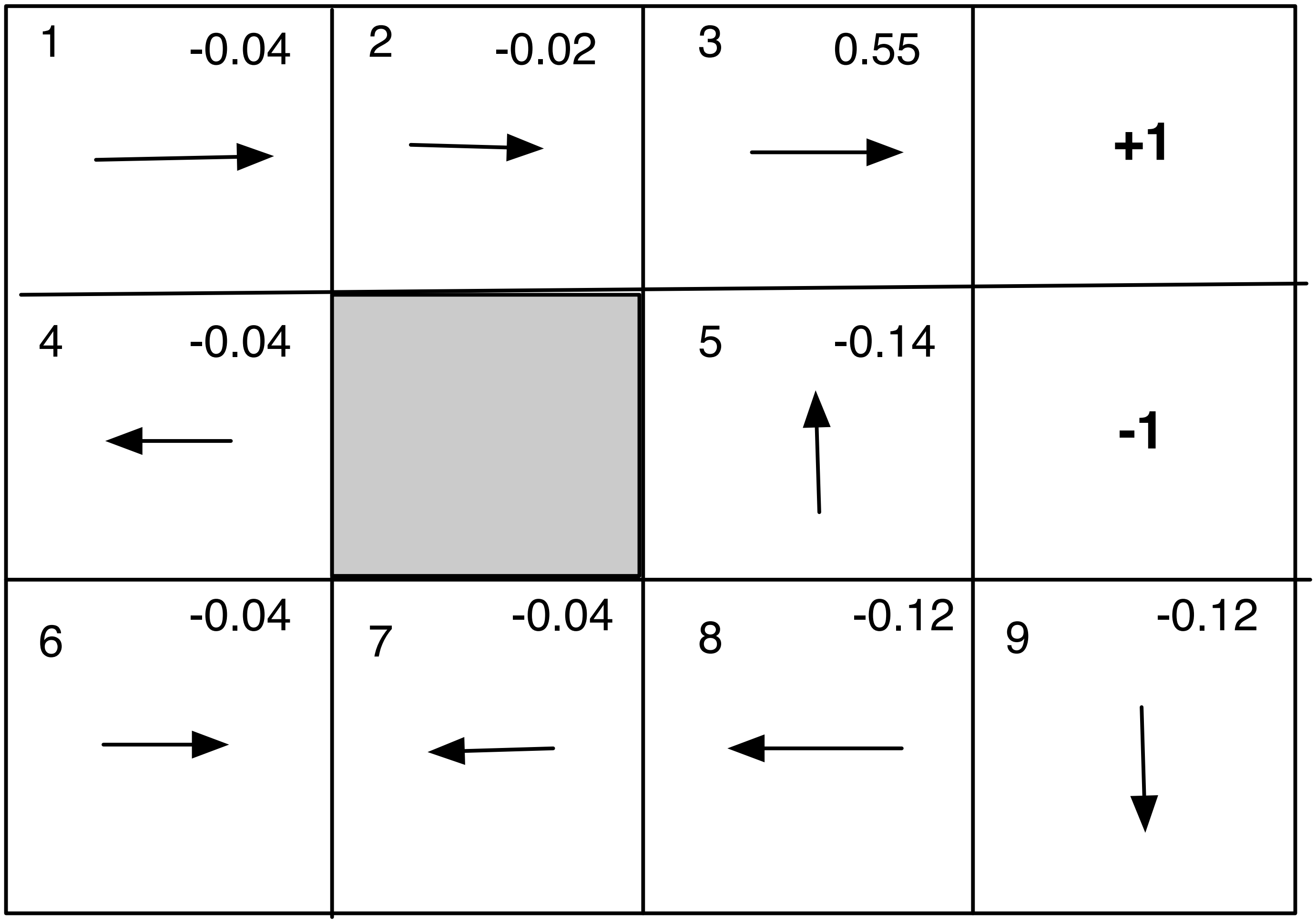
- Select optimal policies given these utilities

Example



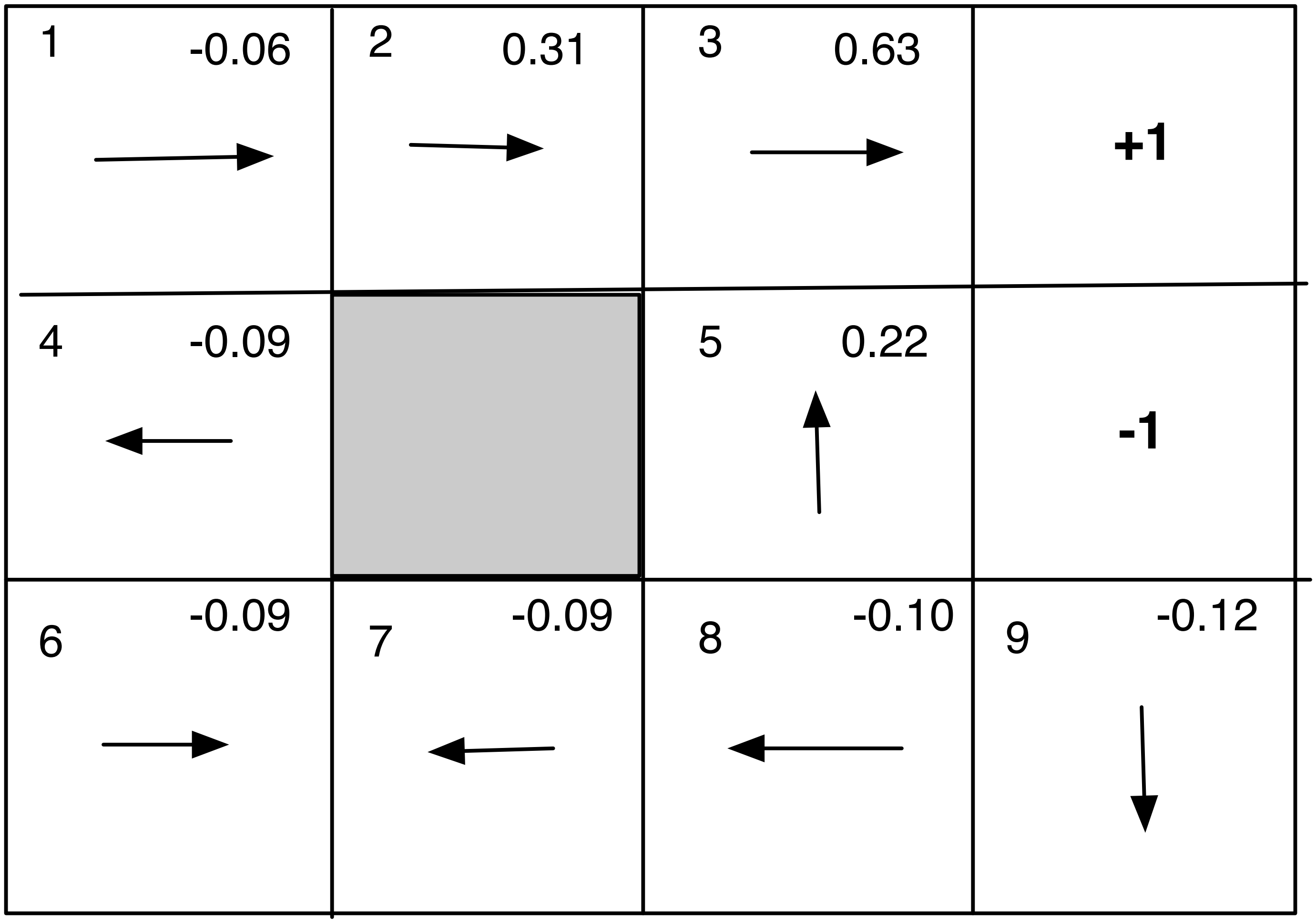
- Based on these new policies, estimate new utilities for each state

Example



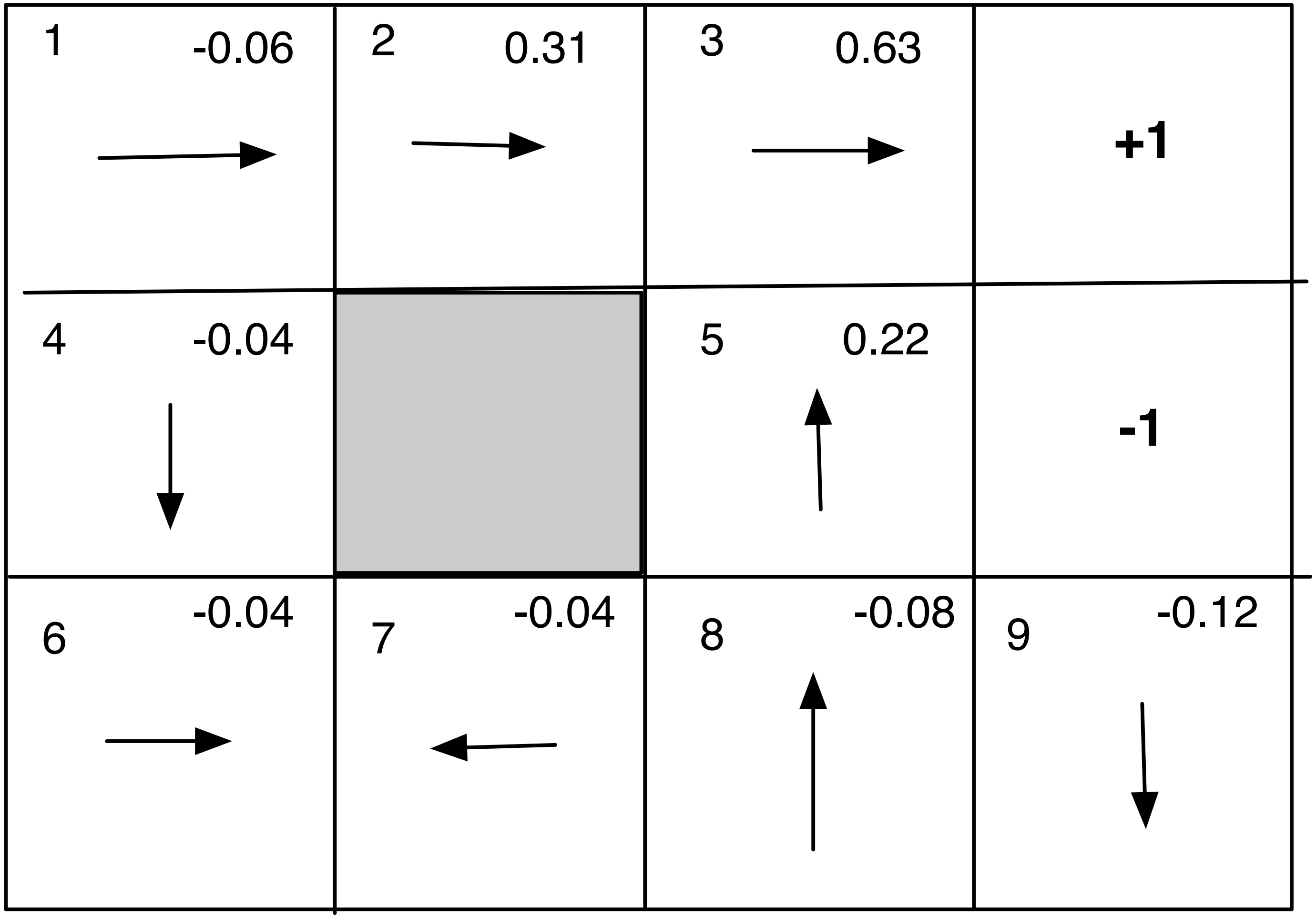
- Based on these new utilities, select optimal policies

Example



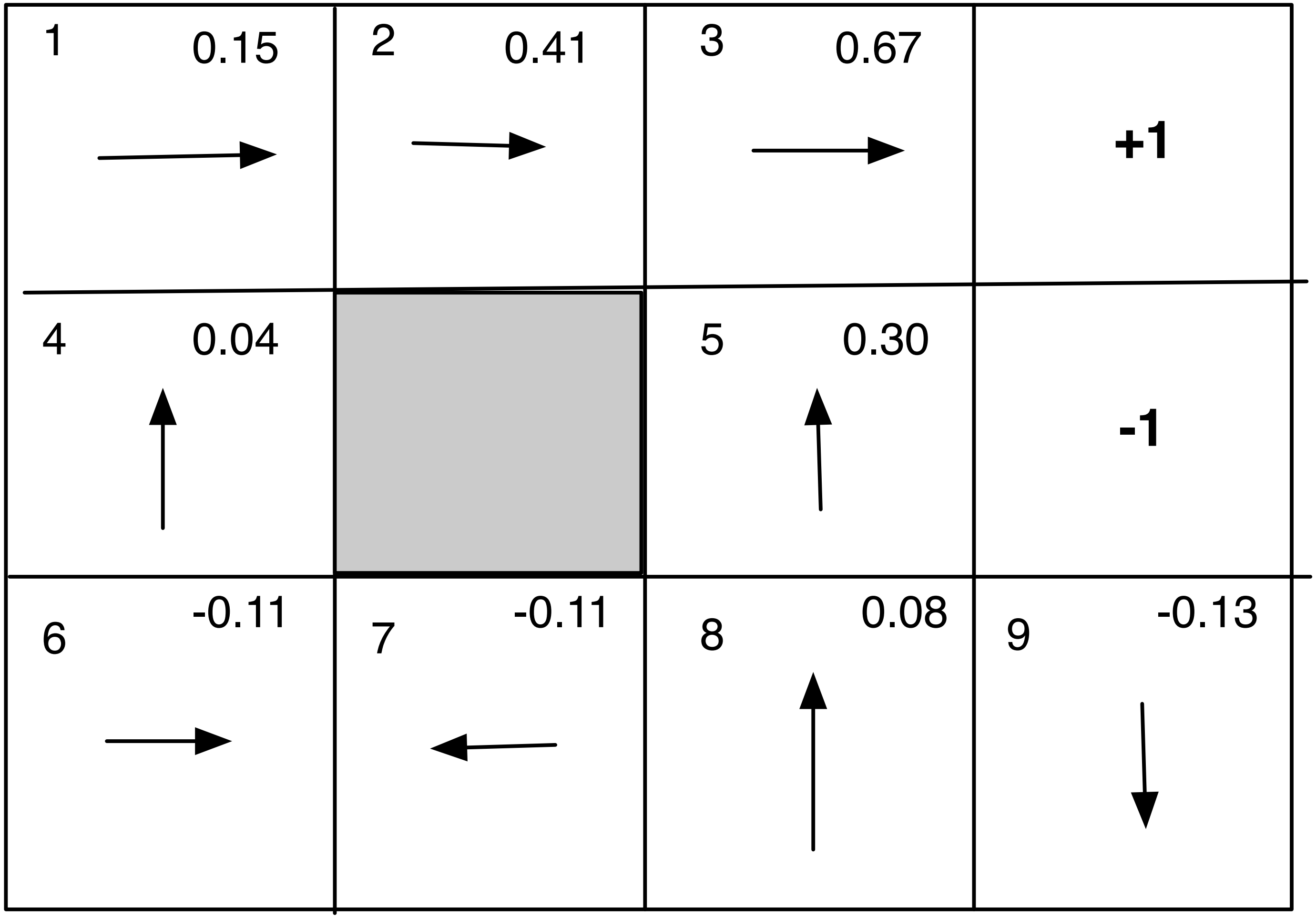
- Use these new optimal policies to re-estimate utilities

Example



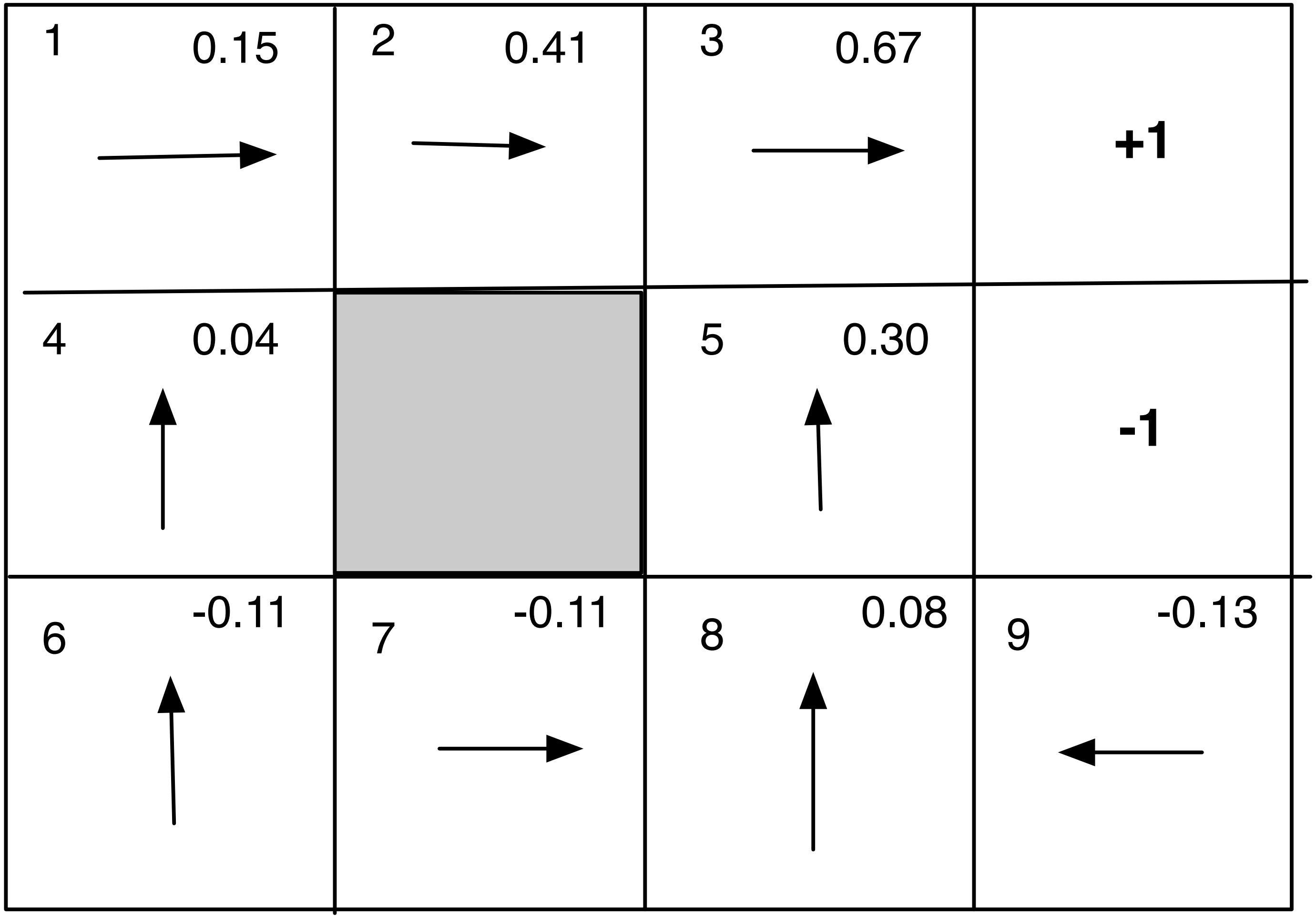
- And use these new utility estimates to construct optimal policies

Example



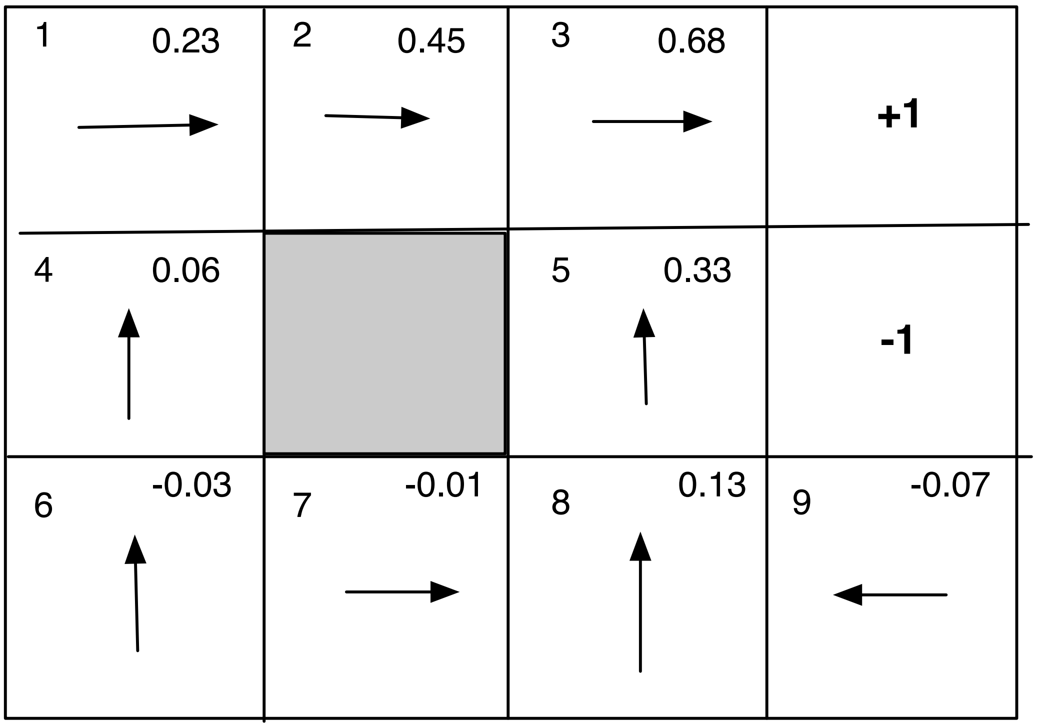
- Again, use the policies to re-estimate utilities

Example



- And then use the utilities to update your optimal policies.

Example



- And once more update your utility estimates based on the new policy.
- Once we update our policy based on these new estimates, we see that it doesn't change, so we're done.

Discussion

- Advantages:
  - Faster convergence.
  - Solves the actual problem we're interested in. We don't really care about utility estimates except as a way to construct a policy.

Learning a Policy

- MDPs assume that we know a model of the world
  - Specifically, the transition function
- We can also learn a policy through interaction with the environment.
- This is known as *reinforcement learning*.
- We'll talk about this next class.

Summary

- Markov decision policies provide an agent with a description of *how to act optimally* for any state in a problem.
  - Must know state space, have a fixed goal.
- Value iteration and policy iteration can be applied to solve this.