Decision Making Under Uncertainty

CMSC 671, Fall 2016

Class #23

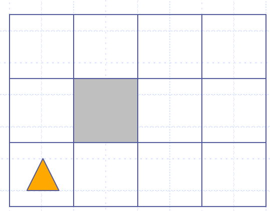
material from Marie desJardin, Lise Getoor, Jean-Claude Latombe, and Daphne Koller

SEQUENTIAL DECISION MAKING UNDER UNCERTAINTY

Sequential Decision Making

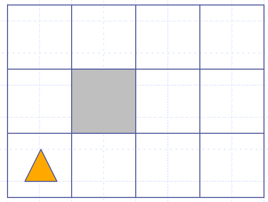
- Finite Horizon
- Infinite Horizon

Simple Robot Navigation Problem



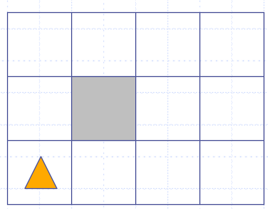
- In each state, the possible actions are U, D, R, and L

Probabilistic Transition Model



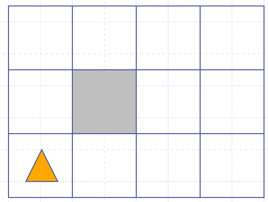
- In each state, the possible actions are U, D, R, and L
- The effect of U is as follows (transition model):
  - With probability 0.8, the robot moves up one square (if the robot is already in the top row, then it does not move)

Probabilistic Transition Model



- In each state, the possible actions are U, D, R, and L
- The effect of U is as follows (transition model):
  - With probability 0.8, the robot moves up one square (if the robot is already in the top row, then it does not move)
  - With probability 0.1, the robot moves right one square (if the robot is already in the rightmost row, then it does not move)

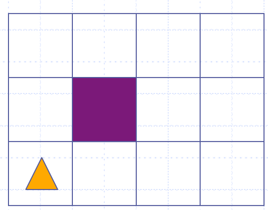
Probabilistic Transition Model



- In each state, the possible actions are U, D, R, and L
- The effect of U is as follows (transition model):
  - With probability 0.8, the robot moves up one square (if the robot is already in the top row, then it does not move)
  - With probability 0.1, the robot moves right one square (if the robot is already in the rightmost row, then it does not move)
  - With probability 0.1, the robot moves left one square (if the robot is already in the leftmost row, then it does not move)

7

Probabilistic Transition Model



- In each state, the possible actions are U, D, R, and L
- The effect of U is as follows (transition model):
  - With probability 0.8, the robot moves up one square (if the robot is already in the top row, then it does not move)
  - With probability 0.1, the robot moves right one square (if the robot is already in the rightmost row, then it does not move)
  - With probability 0.1, the robot moves left one square (if the robot is already in the leftmost row, then it does not move)
- D, R, and L have similar probabilistic effects

8

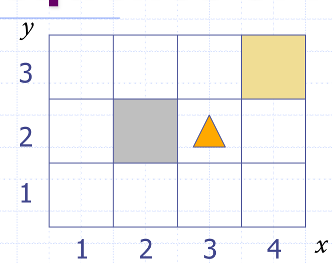
Markov Property

The transition properties depend only on the current state, not on the previous history (how that state was reached)

Markov assumption generally: current state only ever depends on previous state (or finite set of previous states).

9

Sequence of Actions

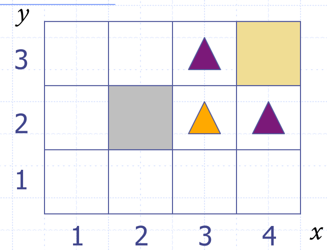


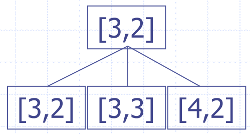
[3,2]

- Planned sequence of actions: (U, R)

10

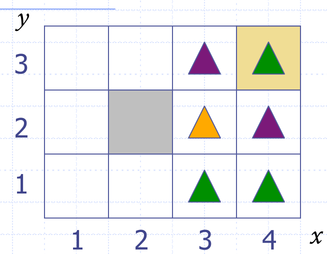
Sequence of Actions

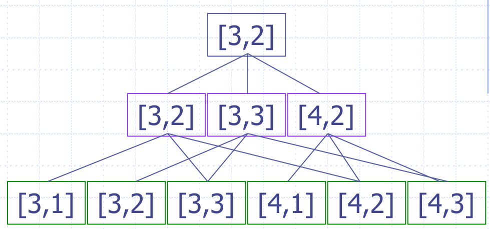




- Planned sequence of actions: (U, R)
- U is executed

Histories

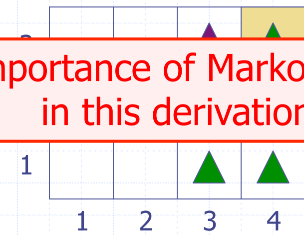




- Planned sequence of actions: (U, R)
- U has been executed
- R is executed

- 9 possible sequences of states – called histories
- 6 possible final states for the robot!

Probability of Reaching the Goal

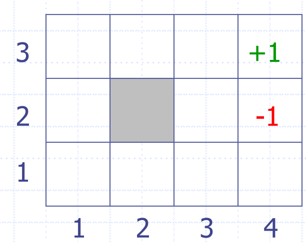


Note importance of Markov property in this derivation

- **P**([4,3] | (U,R).[3,2]) =
  P([4,3] | R.[3,3]) x **P**([3,3] | U.[3,2])
  \+ **P**([4,3] | R.[4,2]) x **P**([4,2] | U.[3,2])
- **P**([4,3] | R.[3,3]) = 0.8
- **P**([4,3] | R.[4,2]) = 0.1
- **P**([3,3] | U.[3,2]) = 0.8
- **P**([4,2] | U.[3,2]) = 0.1
- **P**([4,3] | (U,R).[3,2]) = 0.65

13

Utility Function



- [4,3] provides power supply
- [4,2] is a sand area from which the robot cannot escape

14

Utility Function

3				+1
2				-1
1				
	1	2	3	4

- [4,3] provides power supply
- [4,2] is a sand area from which the robot cannot escape
- The robot needs to recharge its batteries

Utility Function

3				+1
2				-1
1				
	1	2	3	4

- [4,3] provides power supply
- [4,2] is a sand area from which the robot cannot escape
- The robot needs to recharge its batteries
- [4,3] and [4,2] are terminal states

Utility of a History

3				+1
2				-1
1				
	1	2	3	4

- [4,3] provides power supply
- [4,2] is a sand area from which the robot cannot escape
- The robot needs to recharge its batteries
- [4,3] or [4,2] are terminal states
- The utility of a history is defined by the utility of the last state (+1 or −1) minus n/25, where n is the number of moves
  - Many utility functions possible, for many kinds of problems.

17

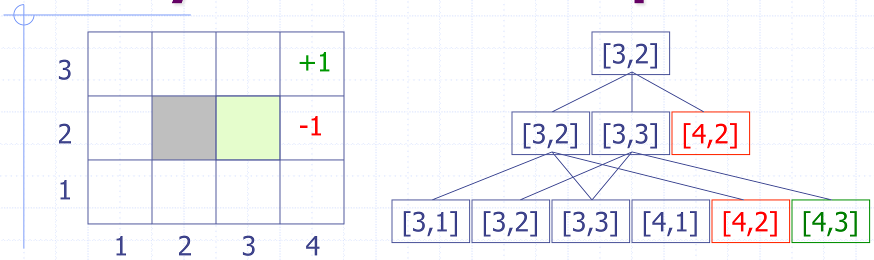
Utility of an Action Sequence

3				+1
2				-1
1				
	1	2	3	4

- Consider the action sequence (U,R) from [3,2]

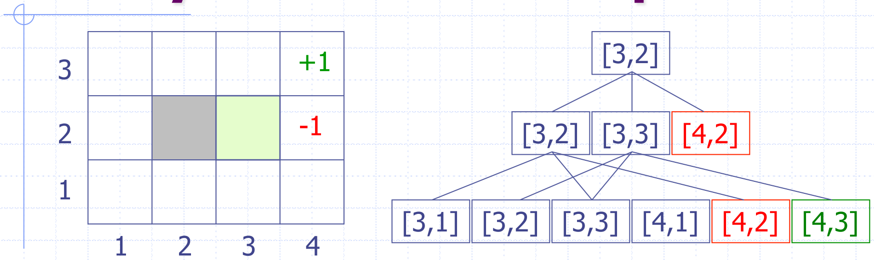
18

Utility of an Action Sequence



- Consider the action sequence (U,R) from [3,2]
- A run produces one of 7 possible histories, each with some probability

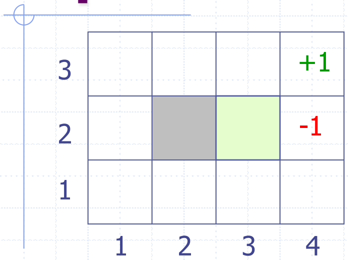
Utility of an Action Sequence

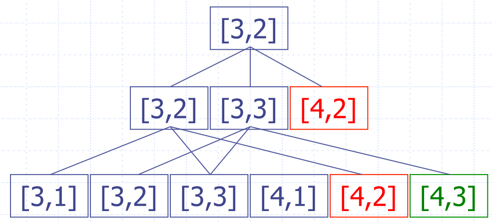


- Consider the action sequence (U,R) from [3,2]
- A run produces one of 7 possible histories, each with some probability
- The utility of the sequence is the expected utility of the histories:

$$\mathbf{U} = \Sigma_h \mathbf{U}_h \mathbf{P}(h)$$

Optimal Action Sequence





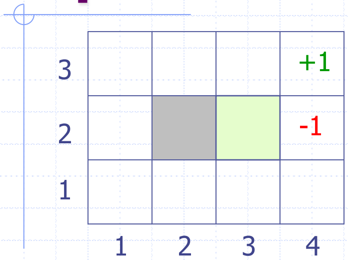
- Consider the action sequence (U,R) from [3,2]
- A run produces one of 7 possible histories, each with some probability
- The utility of the sequence is the expected utility of the histories:

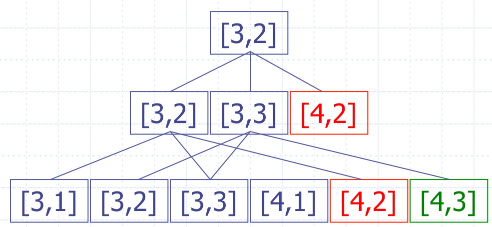
$$\mathbf{U} = \Sigma_h \mathbf{U}_h \, \mathbf{P}(h)$$

- The optimal sequence is the one with maximal utility

21

Optimal Action Sequence





- Consider the action sequence (U,R) from [3,2]
- A run produc ... ability
- The utility of ...

only if the sequence is executed blindly!

- The optimal sequence is the one with maximal utility
- **But is the optimal action sequence what we want to compute?**

22

Reactive Agent Algorithm

Accessible or observable state

Repeat:

- s ← sensed state
- If s is terminal then exit
- a ← choose action (given s)
- Perform a

Policy (Reactive/Closed-Loop Strategy)

3	→	→	→	+1
2	↑		↑	-1
1	↑	←	←	←
	1	2	3	4

- A policy Π is a complete mapping from states to actions

Reactive Agent Algorithm

Repeat:

- s ← sensed state
- If s is terminal then exit
- a ← Π(s)
- Perform a

25

Optimal Policy

3	→	→	→	+1
2	↑		↑	-1
1	↑	←	←	←
	1	2	3	4

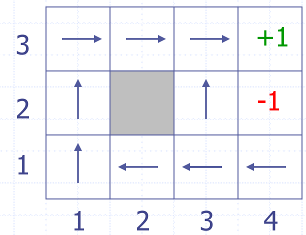
Note that [3,2] is a "dangerous" state that the optimal policy tries to avoid

- A policy Π is a complet
- The optimal policy Π* i
  history (ending at a ter
  expected utility

Makes sense because of Markov property

26

Optimal Policy



- A policy Π is a comp ns
- The optimal policy Π history with maximal expected utility

This problem is called a Markov Decision Problem (MDP)

How to compute Π^*?

Additive Utility

- History $H = (s_0, s_1, \ldots, s_n)$
- The utility of H is additive iff:

$$\mathbf{U}(s_0, s_1, \ldots, s_n) = \mathbf{R}(0) + \mathbf{U}(s_1, \ldots, s_n) = \Sigma\, \mathbf{R}(i)$$

Reward

Additive Utility

- History H = $(s_0, s_1, \ldots, s_n)$
- The utility of H is additive iff:
  $\mathbf{U}(s_0, s_1, \ldots, s_n) = \mathbf{R}(0) + \mathbf{U}(s_1, \ldots, s_n) = \Sigma\, \mathbf{R}(i)$
- Robot navigation example:
  - $\mathbf{R}(n) = +1$ if $s_n = [4,3]$
  - $\mathbf{R}(n) = -1$ if $s_n = [4,2]$
  - $\mathbf{R}(i) = -1/25$ if $i = 0, \ldots, n-1$

Principle of Max Expected Utility

- History H = $(s_0, s_1, \ldots, s_n)$
- Utility of H: $\mathbf{U}(s_0, s_1, \ldots, s_n) = \Sigma\, \mathbf{R}(i)$

→	→	→	+1
↑		↑	-1
↑	←	←	←

reminder! utility of a sequence:
$\mathbf{U} = \Sigma_h \mathbf{U}_h\, \mathbf{P}(h)$

First-step analysis →

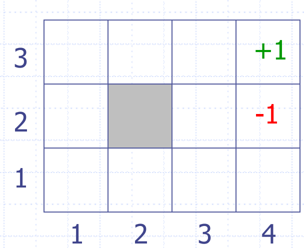
- $\mathbf{U}(i) = \mathbf{R}(i) + \max_a \Sigma_k \mathbf{P}(k \mid a.i)\, \mathbf{U}(k)$
- $\Pi^*(i) = \arg\max_a \Sigma_k \mathbf{P}(k \mid a.i)\, \mathbf{U}(k)$

Defining State Utility

- Problem:
  - When making a decision, we only know the reward so far, and the possible actions
  - We've defined utility retroactively (i.e., the utility of a history is known *once we finish it*)
  - What is the utility of a particular ***state*** in the middle of decision making?
  - Need to compute ***expected utility*** of possible future histories

Value Iteration

- Initialize the utility of each non-terminal state
  s_i to $\mathbf{U}_0(i) = 0$ } *or some uniform or uniformly distributed value*
- For t = 0, 1, 2, …, do:
  $\mathbf{U}_{t+1}(i) \leftarrow \mathbf{R}(i) + \max_a \sum_k \mathbf{P}(k \mid a.i)\, \mathbf{U}_t(k)$



Value Iteration

Note the importance of terminal states and connectivity of the state-transition graph

- Initialize the utility of each
  s_i to $\mathbf{U_0}(i) = 0$
- For t = 0, 1, 2, …, do:

$$\mathbf{U_{t+1}}(i) \leftarrow \mathbf{R}(i) + \max_a \Sigma_k \mathbf{P}(k \mid a.i)\, \mathbf{U_t}(k)$$

	1	2	3	4
3	0.812 →	0.868 →	??? →	+1
2	0.762 ↑		0.660 ↑	-1
1	0.705 ↑	0.655 ←	0.611 ←	0.388 ←

EXERCISE: What is U*([3,3]) (assuming that the other U* are as shown)?

Value Iteration

- Initialize the utility of each non-terminal state
  s_i to $\mathbf{U_0}(i) = 0$
- For t = 0, 1, 2, …, do:

$$\mathbf{U_{t+1}}(i) \leftarrow \mathbf{R}(i) + \max_a \Sigma_k \mathbf{P}(k \mid a.i)\, \mathbf{U_t}(k)$$

	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 ←

$U^*_{3,3} =$
$R_{3,3} +$
$[P_{3,2}\, U^*_{3,2} + P_{3,3}\, U^*_{3,3} + P_{4,3}\, U^*_{4,3}]$

34

Policy Iteration

- Pick a policy Π at random

Policy Iteration

- Pick a policy Π at random
- Repeat:
  - Compute the utility of each state for Π

    $\mathbf{U}_{t+1}(i) \leftarrow \mathbf{R}(i) + \Sigma_k \mathbf{P}(k \mid \Pi(i).i)\ \mathbf{U}_t(k)$

Policy Iteration

- Pick a policy Π at random
- Repeat:
  - Compute the utility of each state for Π

    $U_{t+1}(i) \leftarrow R(i) + \Sigma_k P(k \mid \Pi(i).i)\, U_t(k)$
  - Compute the policy Π' given these utilities

    $\Pi'(i) = \arg\max_a \Sigma_k P(k \mid a.i)\, U(k)$

39

Policy Iteration

- Pick a policy Π at random
- Repeat:
  - Compute the utility of each state for Π

    $\boxed{U_{t+1}(i) \leftarrow R(i) + \Sigma_k P(k \mid \Pi(i).i)\, U_t(k)}$
  - Compute the policy Π' given these utilities

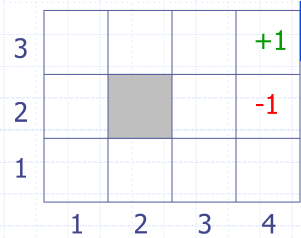
    $\Pi'(i) = \arg\max_a \Sigma_k$
  - If $\Pi' = \Pi$ then ret

Or solve the set of linear equations:
$U(i) = R(i) + \Sigma_k P(k \mid \Pi(i).i)\, U(k)$
(often a sparse system)

40

Infinite Horizon

In many problems, e.g., the robot navigation example, histories are potentially unbounded and the same state can be reached many times



What if the robot lives forever?

One trick:
Use discounting to make an infinite horizon problem mathematically tractable

41

Value Iteration

- Value iteration:
  - Initialize state values (expected utilities) randomly
  - Repeatedly update state values using best action, according to current approximation of state values
  - Terminate when state values stabilize
  - Resulting policy will be the best policy because it' s based on accurate state value estimation
- Policy iteration:
  - Initialize policy randomly
  - Repeatedly update state values using best action, according to current approximation of state values
  - Then update policy based on new state values
  - Terminate when policy stabilizes
  - Resulting policy is the best policy, but state values may not be accurate (may not have converged yet)
  - Policy iteration is often faster (because we don' t have to get the state values right)
- **Both methods have a major weakness: They require us to know the transition function exactly in advance!**

42