Fundamentals of Artificial Intelligence

Sequential Decision Problems

Shyamanta M Hazarika
Mechanical Engineering
Indian Institute of Technology Guwahati
s.m.hazarika@iitg.ac.in

http://www.iitg.ac.in/s.m.hazarika/

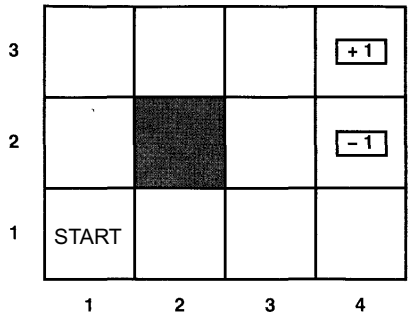
Sequential Decision Problems

- Our discussion so far, we were concerned with single decision problems, in which the utility of each action's outcome was well-known.
- In this lecture we will be concerned with sequential decision problems, where the agent's utility depends on a sequence of decisions.
  - Recall the search algorithms - looking for a sequence of actions that leads to a good state. Difference is that what is returned here - not the fixed sequence of actions; rather a policy - a set of situation-action rules for each state - arrived at by calculating utilities for each state
- Sequential decision problems, which include utilities, uncertainty, and sensing, generalize the search and planning problems.

Example: Sequential Decision Problem

Problem

- Beginning at the start state, choose an action at each time step.
- Problem terminates when either goal state is reached.



Execute a sequence of actions. Terminates when the agent reaches one of the states marked +1 or -1.

- Possible actions are Up, Down, Left, and Right
- Assume that the environment is fully observable, i.e., the agent always knows where it is.

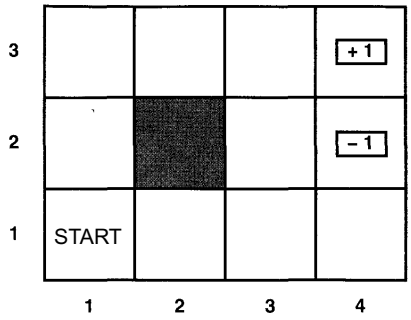
Stuart J. Russel and Peter Norvig: Artificial Intelligence – A Modern Approach, Chapter 17, Pages 498-507.

Example: Sequential Decision Problem

Deterministic Version

- If the environment is deterministic and the objective is to get the maximum reward:

With knowledge of the initial state and the effects of actions, the problem can be solved directly by the search algorithms described before. This is true regardless of whether the environment is accessible or inaccessible. The agent knows exactly which state it will be in after any given action, so there is no need for sensing



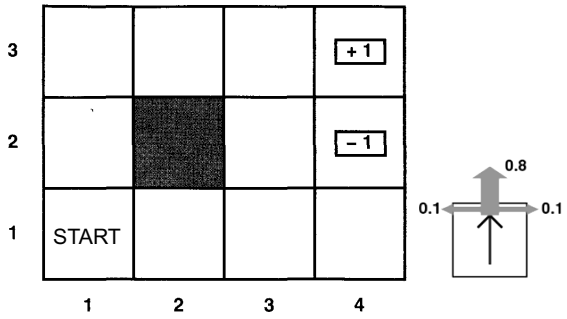
In the deterministic version of the problem, each action reliably moves one square in the intended direction, except that moving into a wall results in no change in position

The solution: (Up, Up, Right, Right, Right)

Example: Sequential Decision Problem

Stochastic Version

- Suppose that there is a 0.8 probability to go to intended cell, but rest of the time it goes to cells at right angles of intended cell with probability 0.1.
- If bumps into wall – stays in place.
- Cannot create a plan ahead of time!



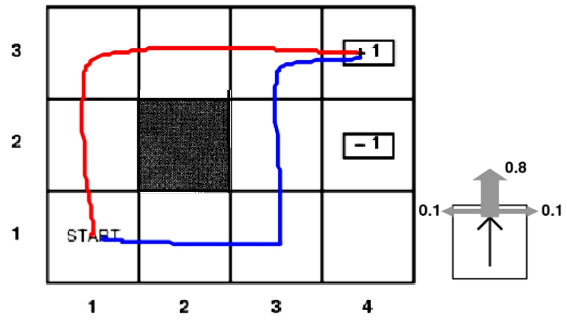
What is the probability that the old plan succeeds?

Example: Sequential Decision Problem

Stochastic Version

Probability to succeed: 0.32776

$$0.8^5 + (0.1^4 \cdot 0.8) = 0.32776$$



In the stochastic version, the actions are unreliable. Each action achieves the intended effect with probability 0.8, but the rest of the time, the action moves the agent at right angles to the intended direction with probability 0.1.

Transition Model

In the stochastic version, the actions are unreliable. Each action achieves the intended effect with probability 0.8, but the rest of the time, the action moves the agent at right angles to the intended direction.

- We will use the term transition model to refer to the set of probabilities associated with the possible transitions between states after any given action.
  - A transition model is a specification of the outcome probabilities for each action in each possible state.
- T(s,a,s′) denotes the probability of reaching state s′ if action a is done in state s.

Rewards and Utilities

- A utility function must be specified for the agent in order to determined the value of an action.
  - The problem is sequential, the utility function depends on a sequence of states.
- The tricky part is the utility function. Other than the terminal states (the ones marked +1 and -1), there is no indication of a state's utility. Base the utility function on a sequence of states - an environment history - rather than on a single state.
- Rewards are assigned to states, i.e., R(s) returns the reward of the state.
- For this example, assume the following:
  - The reward for all states, except for the goal states, is -0.04.
  - The utility function is the sum of all the states visited.
    - E.g., if the agent reaches (4,3) in 10 steps, the total utility is 1 + (10 x -0.04) = 0.6.
  - The negative reward - incentive to stop interacting as quickly as possible.

Dealing with action sequences

- Action sequences could be to consider sequences as long actions. Then one could simply apply the basic Maximum Expected Utility principle to sequences.
  - The rational action would then be the first action of the optimal sequence.
- Closely related to the way that search algorithms work - it has a fundamental flaw – for an agent in a stochastic environment.
  - Assumes that the agent is required to commit to an entire sequence of actions before executing it.

    If the agent has no sensors, then this is the best it can do!
  - But if the agent can acquire new sensory information - then committing to an entire sequence is irrational.

Dealing with action sequences

- In reality, the agent will have the opportunity to choose a new action after each step, given whatever additional information its sensors provide.
- We therefore need an approach much more like the conditional planning algorithms, rather than the search algorithms.
  - Of course, these will have to be extended to handle probabilities and utilities.
  - Deal with the fact that the "conditional plan" for a stochastic environment may have to be of infinite size, because it is possible, although unlikely, for the agent to get stuck in one place (or in a loop) no matter how hard it tries not to.

Policy

- In an accessible environment, the agent's percept at each step will identify the state it is in.
  - If it can calculate the optimal action for each state, then that will completely determine its behavior.
- A complete mapping from states to actions is called a policy.
  - Given a policy, it is possible to calculate expected utility of the possible environment histories generated by that policy.
- The problem, is not to calculate the optimal action sequence, but to calculate the optimal policy - the policy that results in the highest expected utility.

Markov Decision Process

- The problem of calculating an optimal policy in an accessible, stochastic environment with a known transition model is called a Markov decision problem (MDP).
  - Markov's work is so closely associated with the assumption of accessibility, that decision problems are often divided into "Markov" and "non-Markov."
- Markov property holds if the transition probabilities from any given state depend only on the state and not on previous history.
  - T depends only on the previous state s and not the rest of the history

Markov Decision Process

- The specification of a sequential decision problem for a fully observable environment that satisfies the Markov Assumption and yields additive rewards.

- Defined as a tuple: <S, A, P, R>
  - S: State
  - A: Action
  - T: Transition model
    - Table T(s,a,s'), probability of s' given action a in state s
  - R: Reward
    - R(s) = cost or reward being in state s

Example: Sequential Decision Problem

- S: State of the agent on the grid
  - E.g. (4,3)
  - Note that cell denoted by (x,y)
- A: Actions of the agent
  - i.e. Up, Down, Left, Right
- T: Transition Model
  - Table T(s, a,s'), probability of s' given action "a" in state "s"
  - E.g., P((4,3) | (3,3), Up) = 0.1
  - E.g., P((3, 2) | (3,3), Up) = 0.8
- R: Reward
  - R(3, 3) = -0.04
  - R (4, 3) = +1

Solution for an MDP

- Since outcomes of actions are not deterministic, a fixed set of actions cannot be a solution.
- A solution must specify what an a agent should do for any state that the agent might reach.
- A policy, denoted by π, recommends an action for a given state, i.e.,
  - π(s) is the action recommended by policy π for state s.
  - Environment is stochastic, each time a given policy is executed, there can be different environment histories.
- Quality of a policy is determined by the *expected* utility of the possible environment histories generated by that policy.

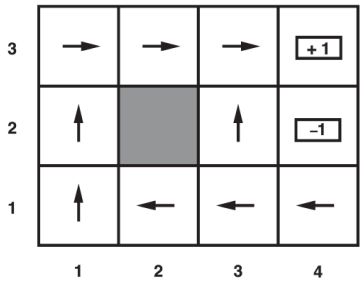
Optimal Policy

- An Optimal policy is a policy that yields the highest expected utility.
- Optimal policy is denoted by π^*.
- Once a π^* is computed for a problem, then the agent, once identifying the state (s) that it is in, consults π^*(s) for the next action to execute.

Optimal Policy for Example

With γ =1 and R(s) = -0.04 for non-terminal states:

Note that at (3,1), the policy goes back towards the initial state.
Because the cost of taking a step is fairly small compared to the penalty for ending up in (4,2) by accident, the optimal policy for the state (3,1) is conservative.
Long route rather than a SHORT one!



Assignment of Utility to State Sequences

- Utility function for environment histories (sequences of states) is denoted as $U_h([S_0,S_1,\ldots,S_n])$.
- Two methods:
  - **Additive rewards** – Sum up rewards of states, i.e., $U_h([S_0,S_1,\ldots]) = R(s_0) + R(s_1) + R(s_2) + \ldots$
  - **Discounted Rewards** – Sum of progressively discounted rewards of states, i.e.,
    $U_h([S_0,S_1,\ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots$
    where **discount factor** γ is a number between 0 and 1.
    - Closer γ to 0, the less future rewards count.
    - When γ is 1, the same as Additive Rewards.

Choosing between Policies

- The value of a policy is the *expected* sum of discounted rewards obtained, where the expectation is taken over all possible state sequences that could occur, give that the policy is executed.

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi\right]$$

Many state sequences to compare; As before, maximize expected utility.

Value Iteration

- Value Iteration is an algorithm for computing an optimal policy.
  - Key insight: Utility of a state is immediate reward plus discounted expected utility of next states

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s')U(s')$$

Basic idea:

1. Calculate the utility of each state U(s) and
2. Then use the state utilities to select an optimal action in each state.

Utility of States

- Utility of a state is the expected utility of the state sequences that might follow it, which are determined by a policy.
- Let $U^{\pi}(s)$ be the utility of a state and s_t be the state the agent is in after executing π for t steps, then

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi, s_0 = s\right]$$

- Let U(s) be a shorthand for $U^{\pi^*}(s)$; R(s) is the short-term reward for being in s and U(s) is the long-term total reward from s onwards.

Utilities for Example Problem

The utilities of the states in our 4x3 world

For non−terminal states:
γ =1 and R (s)= - 0.04

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

Note that utilities closer to (4,3) are higher because fewer steps are required to reach the exit.

Bellman Equation

- π^* selects the action that maximizes the expected utility of the subsequent state.

$$\pi^*(s) = \operatorname*{argmax}_a \sum_{s'} T(s,a,s')U(s')$$

- The Bellman equation defines U(s) as the utility of s plus the discounted utility of the next state, assuming the optimal action, i.e.,

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s')U(s')$$

Bellman Equation on Example Problem

The equation for state (1,1) is

$$U(1,1) = -0.04 + \gamma \max[\ 0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1), \quad \text{Up}$$
$$0.9U(1,1) + 0.1U(1,2), \quad \text{Left}$$
$$0.9U(1,1) + 0.1U(2,1), \quad \text{Down}$$
$$0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1)\ ]. \quad \text{Right}$$

When we plug in the numbers from Utilities for the Optimal Policy (discussed before), we find that Up is the optimal action in (1,1).

Using Bellman equations for solving MDPs.

- If there are n possible states, then there are n Bellman equations (one for each state).
- To compute the n utilities, we would like to solve simultaneously the n Bellman equations.
  - Problematic because max is not a linear operator.

Apply an iterative approach in which we replace the equality of the bellman equation by an assignment:

- Use iteration applying Bellman update:

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} T(s,a,s')U_i(s')$$

  - Start with the utilities of all states initialized to 0.
  - Guaranteed to converge.

Policy Iteration

- Alternative to value iteration to find a policy
  - Searches policy space.
- Basic idea:
  - **Policy Evaluation**: start with a random policy π_0 and calculate utilities based on if that policy were executed.

$$V(s) = \sum_{s' \in \mathcal{S}} T(s, \pi(s), s') \left[R(s, \pi(s), s') + \gamma V(s') \right], \forall s \in \mathcal{S}$$

  - **Policy Improvement**: Calculate a new MEU policy π_{i+1} based on computed utilities.

$$\pi(s) \leftarrow \arg\max_a \sum_{s' \in \mathcal{S}} T(s, a, s') \left[R(s, a, s') + \gamma V(s') \right]$$

  - Iterate until the policy does not change.

POMDP

- In an inaccessible environment, the percept does not provide enough information to determine the state or the associated transition probabilities; such problems are called Partially Observable Markov Decision Problems, or POMDP.
- The Markov property does not hold for percepts (as opposed to states)
  - The next percept does not depend just on the current percept and the action taken.
- Methods used for Markov Decision Problems are not directly applicable to POMDPs.

POMDP

- The correct approach for POMDPs is to calculate a probability distribution over the possible states given all previous percepts, and to base decisions on this distribution.
  - This seems simple enough.
- Unfortunately, in POMDPs, calculating the utility of an action in a state is made more difficult by the fact that actions will cause the agent to obtain new percepts, which will cause the agent's beliefs to change in complex ways!
  - POMDPs therefore include the value of information as one component of the decision problem.

POMDP

- The standard method for solving a POMDP is to construct a new MDP in which this probability distribution plays the role of the state variable.
  - Resulting MDP is not easy to solve! State space is characterized by real-valued probabilities; therefore infinite.
- Exact solution methods for POMDPs require some fairly advanced tools; Beyond the scope of this course.
- Instead of exact solutions, one can often obtain a good approximation using a limited lookahead.
  - We shall revisit Decision Network to see how this approach can be realized for POMDPs.