Markov Decision Processes: Making Decision in the Presence of Uncertainty

(some of) R&N 16.1-16.6
R&N 17.1-17.4

Decision Processes: General Description

- Suppose that you own a business. At any time, you know exactly the current state of the business (finances, stock, etc.).
- At any time, you have a choice of several possible actions (advertise, introduce new products,…).
- You cannot predict with certainty the consequence of these actions given the current state of the business, but you have a guess as to the likelihood of the possible outcomes.
- How can you define a policy that will guarantee that you always choose the action that maximizes expected future profits?

Note: Russel & Norvig, Chapter 17.
Decision Processes: General Description

• Decide what action to take next, given:
  – A probability to move to different states
  – A way to evaluate the reward of being in different states
  
  Robot path planning
  Travel route planning
  Elevator scheduling
  Aircraft navigation
  Manufacturing processes
  Network switching & routing

Example

• Assume that time is discretized into discrete time steps $t = 1, 2, \ldots$ (for example, fiscal years)
• Suppose that the business can be in one of a finite number of states $s$ (this is a major simplification, but let’s assume….)
• Suppose that for every state $s$, we can anticipate a reward that the business receives for being in that state: $R(s)$ (in this example, $R(s)$ would the profit, possibly negative, generated by the business)
• Assume also that $R(s)$ is bounded ($R(s) < M$ for all $s$), meaning that the business cannot generate more than a certain profit threshold
• Question: What is the total value of the reward for a particular configuration of states $\{s_1, s_2, \ldots\}$ over time?
Example

• Question: What is the total value of the reward for a particular configuration of states \{s_1, s_2, \ldots\} over time?
• It is simply the sum of the rewards (possibly negative) that we will receive in the future:

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

What is wrong with this formula???

Horizon Problem

\[ U(s_0, \ldots, s_N) = R(s_0) + R(s_1) + \ldots + R(s_N) \]

The sum may be arbitrarily large depending on \( N \)

Need to know \( N \), the length of the sequence (finite horizon)
Horizon Problem

• The problem is that we did not put any limit on the “future”, so this sum can be infinite.
• For example: Consider the simple case of computing the total future reward if the business remains forever in the same state:

\[ U(s, s, \ldots, s, \ldots) = R(s) + R(s) + \ldots + R(s) + \ldots \]

is clearly infinite in general!!
• This definition is useless unless we consider a finite time horizon.
• But, in general, we don’t have a good way to define such a time horizon.

Discounting

\[ U(s_0, \ldots) = R(s_0) + \gamma R(s_1) + \ldots + \gamma^N R(s_N) + \ldots \]

The length of the sequence is arbitrary (infinite horizon)

Discount factor \(0 < \gamma < 1\)
Discounting

- $U(s_0, \ldots) = R(s_0) + \gamma R(s_1) + \ldots + \gamma^N R(s_N) + \ldots$
- Always converges if $\gamma < 1$ and $R(.)$ is bounded
- $\gamma$ close to 0 → instant gratification, don’t pay attention to future reward
- $\gamma$ close to 1 → extremely conservative, consider profits/losses no matter how far in the future
- The resulting model is the discounted reward
- Prefers expedient solutions (models impatience)
- Compensates for uncertainty in available time (models mortality)
- Economic example:
  - Being promised $10,000 next year is worth only 90% as much as receiving $10,000 right now.
  - Assuming payment $n$ years in future is worth only $(0.9)^n$ of payment now

Actions

- Assume that we also have a finite set of actions $a$
- An action $a$ causes a transition from a state $s$ to a state $s'$
- In the “business” example, an action may be placing advertising, or selling stock, etc.
The Basic Decision Problem

• Given:
  – Set of states $S = \{s\}$
  – Set of actions $A = \{a\}$
  – Reward function $R(.)$
  – Discount factor $\gamma$
  – Starting state $s_1$

• Find a sequence of actions such that the resulting sequence of states maximizes the total discounted reward:

$$U(s_0, \ldots) = R(s_0) + \gamma R(s_1) + \ldots + \gamma^N R(s_N) + \ldots.$$

Maze Example: Utility

- Define the reward of being in a state:
  - $R(s) = -0.04$ if $s$ is empty state
  - $R(4,3) = +1$ (maximum reward when goal is reached)
  - $R(4,2) = -1$ (avoid $(4,2)$ as much as possible)

- Define the utility of a sequence of states:
  - $U(s_0, \ldots, s_N) = R(s_0) + R(s_1) + \ldots + R(s_N)$
Maze Example: Utility

- Define the reward of being in a state:
  - $R(s) = -0.04$ if $s$ is empty state
  - $R(4,3) = +1$ (maximum reward when goal is reached)
  - $R(4,2) = -1$ (avoid (4,2) as much as possible)
- Define the utility of a sequence of states:
  - $U(s_0, ..., s_N) = R(s_0) + R(s_1) + ... + R(s_N)$

If no uncertainty:
Find the sequence of actions that maximizes the sum of the rewards of the traversed states.

Maze Example: No Uncertainty

- States: locations in maze grid
- Actions: Moves up/left left/right
- If no uncertainty: Find sequence of actions from current state to goal (+1) that maximizes utility
  \[ \text{We know how to do this using earlier search techniques} \]
What we are looking for: Policy

- **Policy** = Mapping from states to action \( \pi(s) = a \)
  - Which action should I take in each state
- In the maze example, \( \pi(s) \) associates a motion to a particular location on the grid
- For any state \( s \), we define the utility \( U(s) \) of \( s \) as the sum of discounted rewards of the sequence of states starting at state \( s \) generated by using the policy \( \pi \)
  \[
  U(s) = R(s) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots.
  \]
- Where we move from \( s \) to \( s_1 \) by action \( \pi(s) \)
- We move from \( s_1 \) to \( s_2 \) by action \( \pi(s_1) \)
- …etc.

Optimal Decision Policy

- **Policy** = Mapping from states to action \( \pi(s) = a \)
  - Which action should I take in each state
- Intuition: \( \pi \) encodes the best action that we can take from any state to maximize future rewards
- In the maze example, \( \pi(s) \) associates a motion to a particular location on the grid
- **Optimal Policy** = The policy \( \pi^* \) that maximizes the expected utility \( U(s) \) of the sequence of states generated by \( \pi^* \), starting at \( s \)
- In the maze example, \( \pi^*(s) \) tells us which motion to choose at every cell of the grid to bring us closer to the goal
Maze Example: No Uncertainty

- $\pi^*((1,1)) = \text{UP}$
- $\pi^*((1,3)) = \text{RIGHT}$
- $\pi^*((4,1)) = \text{LEFT}$

Maze Example: With Uncertainty

- The robot may not execute exactly the action that is commanded → The outcome of an action is no longer deterministic
- Uncertainty:
  - We know in which state we are (fully observable)
  - But we are not sure that the commanded action will be executed exactly
Uncertainty

• No uncertainty:
  – An action \(a\) deterministically causes a transition from a state \(s\) to another state \(s'\)

• With uncertainty:
  – An action \(a\) causes a transition from a state \(s\) to another state \(s'\) with some probability \(T(s,a,s')\)
  – \(T(s,a,s')\) is called the transition probability from state \(s\) to state \(s'\) through action \(a\)
  – In general, we need \(|S|^2 \times |A|\) numbers to store all the transitions probabilities

Maze Example: With Uncertainty

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>+1</td>
</tr>
<tr>
<td>2</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>↑</td>
<td></td>
<td>↑</td>
<td>←</td>
</tr>
</tbody>
</table>

• We can no longer find a unique sequence of actions, but
• Can we find a policy that tells us how to decide which action to take from each state except that now the policy maximizes the expected utility
Maze Example: Utility Revisited

U(s) = Expected reward of future states starting at s

How to compute U after one step?

Suppose s = (1,1) and we choose action Up.

U(1,1) = R(1,1) +
Suppose \( s = (1,1) \) and we choose action \( \text{Up} \).

\[
U(1,1) = R(1,1) + 0.8 \times U(1,2) + \]

\[
0.8 \]

Intended action \( a: \)

\[
T(s,a,s')
\]

0.1

0.1

\[
0.1
\]

\[
0.1
\]

Maze Example: Utility Revisited
Suppose $s = (1,1)$ and we choose action Up.

\[ U(1,1) = R(1,1) + 0.8 \times U(1,2) + 0.1 \times U(2,1) + 0.1 \times R(1,1) \]
Same with Discount

Suppose $s = (1,1)$ and we choose action Up.

$$U(1,1) = R(1,1) + \gamma (0.8 \times U(1,2) + 0.1 \times U(2,1) + 0.1 \times R(1,1))$$

More General Expression

• If we choose action $a$ at state $s$, expected future rewards are:

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

• If we choose action $a$ at state $s$:

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

- $U(s)$: Expected sum of future discounted rewards starting at $s$.
- $R(s)$: Reward at current state $s$.
- $\gamma$: Discount factor.
- $T(s,a,s')$: Probability of moving from state $s$ to state $s'$ with action $a$.

More General Expression

• If we are using policy $\pi$, we choose action $a=\pi(s)$ at state $s$, expected future rewards are:

$$U_\pi(s) = R(s) + \gamma \sum_{s'} T(s,\pi(s),s') U_\pi(s')$$

- $U_\pi(s)$: Expected sum of future discounted rewards starting at $s$ for policy $\pi$.
Formal Definitions

- Finite set of states: $S$
- Finite set of allowed actions: $A$
- Reward function $R(s)$
- Transitions probabilities: $T(s,a,s') = P(s'|a,s)$
- Utility = sum of discounted rewards:
  \[ U(s_0, \ldots) = R(s_0) + \gamma R(s_1) + \ldots + \gamma^N R(s_N) + \ldots \]
- Policy: $\pi : S \rightarrow A$
- Optimal policy: $\pi^*(s) = \text{action that maximizes the } expected \text{ sum of rewards from state } s$

Markov Decision Process (MDP)

- Key property (Markov):
  \[ P(s_{t+1} | a, s_0, \ldots, s_t) = P(s_{t+1} | a, s_t) \]
- In words: The new state reached after applying an action depends only on the previous state and it does not depend on the previous history of the states visited in the past

  $\Rightarrow$ Markov Process
• When applying the action “Right” from state $s_2 = (1,3)$, the new state depends only on the previous state $s_2$, not the entire history $\{s_1, s_0\}$

**Graphical Notations**

$T(s, a_1, s') = 0.8$
$T(s', a_2, s) = 0.6$
$T(s, a_2, s) = 0.2$

- Nodes are states
- Each arc corresponds to a possible transition between two states given an action
- Arrows are labeled by the transition probabilities

$T(s, a_1, s') = 0.8$
$T(s', a_2, s) = 0.6$
Example (Partial)

Example

- I run a company
- I can choose to either save money or spend money on advertising
- If I advertise, I may become famous (50% prob.) but will spend money so I may become poor
- If I save money, I may become rich (50% prob.), but I may also become unknown because I don’t advertise
- What should I do?
Example Policies

<table>
<thead>
<tr>
<th>Policy Number 1</th>
<th>STATE → ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>PU</td>
<td>S</td>
</tr>
<tr>
<td>PF</td>
<td>A</td>
</tr>
<tr>
<td>RU</td>
<td>S</td>
</tr>
<tr>
<td>RF</td>
<td>A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Policy Number 2</th>
<th>STATE → ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>PU</td>
<td>A</td>
</tr>
<tr>
<td>PF</td>
<td>A</td>
</tr>
<tr>
<td>RU</td>
<td>A</td>
</tr>
<tr>
<td>RF</td>
<td>A</td>
</tr>
</tbody>
</table>

- How many policies?
- Which one is the best policy?
- How to compute the optimal policy?
Example: Finance and Business

- **States**: Status of the company (cash reserves, inventory, etc.)
- **Actions**: Business decisions (advertise, acquire other companies, roll out product, etc.)
- **Uncertainty**: Due to all the external uncontrollable factors (economy, shortages, consumer confidence...)
- **Optimal policy**: The policy for making business decisions that maximizes the expected future profits

Note: Ok, this is an overly simplified view of business models. Similar models could be used for investment decisions, etc.

Example: Robotics

- **States**: 2-D positions
- **Actions**: Commanded motions (turn by x degrees, move y meters)
- **Uncertainty**: Comes from the fact that the mechanism is not perfect (slippage, etc.) and does not execute the commands exactly
- **Reward**: When avoiding forbidden regions (for example)
- **Optimal policy**: The policy that minimizes the cost to the goal
Example: Games

• States: Number of white and black checkers at each location
• Note: Number of states is huge, on the order $10^{20}$ states!!!!
• Branching factor prevents direct search
• Actions: Set of legal moves from any state
• Uncertainty comes from the roll of the dice
• Reward computed from number of checkers in the goal quadrant
• Optimal policy: The one that maximizes the probability of winning the game

Interesting example because it is impossible to store explicitly the transition probability tables (or the states, or the values $U(s)$).

Example: Robotics

• Learning how to fly helicopters!
• States: Possible values for the roll, pitch, yaw, elevation of the helicopter
• Actions: Commands to the actuators. The uncertainty comes from the fact that the actuators are imperfect and that there are unknown external effects like wind gusts
• Reward: High reward if it remains in stable flight (low reward if it goes unstable and crashes!)
• Policy: A control law that associates a command to the observed state
• Optimal policy: The policy that maximizes flight stability for a particular maneuver (e.g., hovering)

Note 1: The states are continuous in this case. Although we will cover only MDPs with discrete states, the concepts can be extended to continuous spaces.
Note 2: It is obviously impossible to "try" different policies on the system itself, for obvious reasons (it will crash to the ground on most policies!!).
Key Result

• For every MDP, there exists an optimal policy
• There is no better option (in terms of expected sum of rewards) than to follow this policy

• How to compute the optimal policy?  
  → We cannot evaluate all possible policies (in real problems, the number of states is very large)

Bellman’s Equation

If we choose an action $a$:

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

If we choose an action \( a \):

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

In particular, if we always choose the action \( a \) that maximizes future rewards (optimal policy), \( U(s) \) is the maximum \( U^*(s) \) we can get over all possible choices of actions:

\[
U^*(s) = R(s) + \gamma \max_a (\sum_{s'} T(s, a, s') U^*(s'))
\]

Bellman’s Equation

\[
U^*(s) = R(s) + \gamma \max_a (\sum_{s'} T(s, a, s') U^*(s'))
\]

- The optimal policy (choice of \( a \) that maximizes \( U \)) is:

\[
\pi^*(s) = \arg\max_a (\sum_{s'} T(s, a, s') U^*(s'))
\]
Why it cannot be solved directly

\[ U^*(s) = R(s) + \gamma \max_a \left( \sum_{s'} T(s,a,s') U^*(s') \right) \]

- The optimal policy (choice of \( a \) that maximizes \( U \)) is:

\[ \pi^*(s) = \arg\max_a \left( \sum_{s'} T(s,a,s') U^*(s') \right) \]

Expected sum of rewards using policy \( \pi^* \) → The right-hand depends on the unknown. Cannot solve directly!

First Solution: Value Iteration

- Define \( U_1(s) = \text{best value after one step} \)

\[ U_1(s) = R(s) \]

- Define \( U_2(s) = \text{best possible value after two steps} \)

\[ U_2(s) = R(s) + \gamma \max_a \left( \sum_{s'} T(s,a,s') U_1(s') \right) \]

\[ \cdots \]

- Define \( U_k(s) = \text{best possible value after} \ k \ \text{steps} \)

\[ U_k(s) = R(s) + \gamma \max_a \left( \sum_{s'} T(s,a,s') U_{k-1}(s') \right) \]
First Solution: Value Iteration

• Define $U_1(s) = \text{best value after one step}$
  \[ U_1(s) = R(s) \]

• Define $U_2(s) = \text{best value after two steps}$
  \[ U_2(s) = R(s) + \gamma \max_a \left( \sum_{s'} T(s,a,s') U_1(s') \right) \]

• Define $U_k(s) = \text{best value after } k \text{ steps}$
  \[ U_k(s) = R(s) + \gamma \max_a \left( \sum_{s'} T(s,a,s') U_{k-1}(s') \right) \]

Maximum possible expected sum of discounted rewards that I can get if I start at state $s$ and I survive for $k$ time steps.

Example

• I run a company
• I can choose to either save money or spend money on advertising
• If I advertise, I may become famous (50% prob.) but will spend money so I may become poor
• If I save money, I may become rich (50% prob.), but I may also become unknown because I don’t advertise
• What should I do?
Value Iteration

<table>
<thead>
<tr>
<th></th>
<th>PU</th>
<th>PF</th>
<th>RU</th>
<th>RF</th>
</tr>
</thead>
<tbody>
<tr>
<td>$U_1$</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>$U_2$</td>
<td>0</td>
<td>4.5</td>
<td>14.5</td>
<td>19</td>
</tr>
<tr>
<td>$U_3$</td>
<td>2.03</td>
<td>8.55</td>
<td>16.53</td>
<td>25.08</td>
</tr>
<tr>
<td>$U_4$</td>
<td>4.76</td>
<td>12.2</td>
<td>18.35</td>
<td>28.72</td>
</tr>
<tr>
<td>$U_5$</td>
<td>7.63</td>
<td>15.07</td>
<td>20.40</td>
<td>31.18</td>
</tr>
<tr>
<td>$U_6$</td>
<td>10.21</td>
<td>17.46</td>
<td>22.61</td>
<td>33.21</td>
</tr>
<tr>
<td>$U_7$</td>
<td>12.45</td>
<td>19.54</td>
<td>24.77</td>
<td>35.12</td>
</tr>
</tbody>
</table>

$$U_k(s) = R(s) + \gamma \max_a \left( \sum_{s'} T(s,a,s') U_{k-1}(s') \right)$$
Value Iteration: Facts

- As $k$ increases, $U_k(s)$ converges to a value $U^*(s)$

- The optimal policy is then given by:
  \[ \pi^*(s) = \arg\max_a \left( \sum_{s'} T(s,a,s') U^*(s') \right) \]

- And $U^*$ is the utility under the optimal policy $\pi^*$
  - See convergence proof in R&N
Upon convergence:

\[ \pi^*(s) = \arg\max_a \left( \sum_{s'} T(s,a,s') U^*(s') \right) \]

\[ \pi^*(PU) = A \quad \pi^*(PF) = S \]
\[ \pi^*(RU) = S \quad \pi^*(RF) = S \]

Better to always save except if poor and unknown

Maze Example

![Maze Example Diagram]
Key Convergence Results

- The error on $U$ is reduced by $\gamma$ at each iteration
- Exponentially fast convergence
- Slower convergence as $\gamma$ increases

So far....

- Definition of discounted sum of rewards to measure utility
- Definition of Markov Decision Processes (MDP)
- Assumes observable states and uncertain action outcomes
- Optimal policy = choice of action that results in the maximum expected rewards in the future
- Bellman equation for general formulation of optimal policy in MDP
- Value iteration (dynamic programming) technique for computing the optimal policy
- Next: Other approaches for optimal policy computation + examples and demos.
Another Solution: Policy Iteration

- Start with a randomly chosen policy $\pi_0$
- Iterate until convergence ($\pi_k \sim \pi_{k+1}$):
  1. Compute $U_k(s)$ for every state $s$ using $\pi_k$
  2. Update the policy by choosing the best action given the utility computed at step $k$:

$$\pi_{k+1}(s) = \arg\max_a \left( \sum_{s'} T(s,a,s') \ U_k(s') \right)$$

The sequence of policies $\pi_0, \pi_1, \ldots, \pi_k, \ldots$ converges to $\pi^*$

Evaluating a Policy

1. Compute $U_k(s)$ for every state $s$ using $\pi_k$

$$U_k(s) = R(s) + \gamma \sum_{s'} T(s, \pi_k(s), s') \ U_k(s')$$

*Linear* set of equations can be solved in $O(|S|^3)$

May be too expensive for $|S|$ large, use instead simplified update:

$$U_k(s) \leftarrow R(s) + \gamma \sum_{s'} T(s, \pi_k(s), s') \ U_{k-1}(s')$$

*(modified policy iteration)*
1. Compute $U_k(s)$ for every state $s$ using

$$U_k(s) = R(s) + \gamma \sum_{s'} T(s, \pi_k(s), s') U_k(s')$$

*Linear* set of equations can be solved in $O(|S|^3)$. May be too expensive for $|S|$ large, use instead simplified update:

$$U_k(s) \leftarrow R(s) + \gamma \sum_{s'} T(s, \pi_k(s), s') U_{k-1}(s')$$

*(modified policy iteration)*

---

**Comparison**

- **Value iteration:**
  - *(Relatively)* small number of actions
- **Policy iteration:**
  - Large number of actions
  - Initial guess at a good policy
- **Combined policy/value iteration is possible**
- **Note:** No need to traverse all the states in a fixed order at every iteration
  - Random order ok
  - Predict "most useful" states to visit
  - Prioritized sweeping ➔ Choose the state with largest value to update
  - States can be visited in any order, applying either value or policy iteration ➔ *asynchronous* iteration
Limitations

- We need to represent the values (and policy) for every state in principle
- In real problems, the number of states may be very large
- Leads to untractably large tables (checker-like problem with $N$ cells and $M$ pieces $\rightarrow N(N-1)(N-2)\ldots(N-M)$ states!)
- Need to find a compact way of representing the states

Solutions:
  - Interpolation
  - Memory-based representations
  - Hierarchical representations

<table>
<thead>
<tr>
<th>State $s$</th>
<th>Value $U$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td></td>
</tr>
<tr>
<td>$s_2$</td>
<td></td>
</tr>
<tr>
<td>........</td>
<td></td>
</tr>
<tr>
<td>$s_{100000}$</td>
<td></td>
</tr>
<tr>
<td>$s_{100001}$</td>
<td></td>
</tr>
</tbody>
</table>

Function Approximation

Polynomials/Splines approximation:
Represent $U(s)$ by a polynomial function that can be represented by a small number of parameters.

Economic models, control
Operations Research
Channel routing, Radio therapy

Neural Nets:
Represent $U(s)$ implicitly by a neural net (function interpolation).

Elevator scheduling, Cell phones, Backgammon, etc.
Memory-Based Techniques

States stored in memory

\[ U(s) = ? \]

Replace \( U(s) \) by \( U(\text{closest neighbor to } s) \)

States stored in memory

\[ U(s) = ? \]

Replace \( U(s) \) by weighted average of \( U(K \text{ closest neighbor to } s) \)

Hierarchical Representations

- Split a state into smaller states when necessary
- Hierarchy of states with high-level managers directing lower-level servants

Example from Dayan, “Feudal learning”.
More Difficult Case

Uncertainty on transition from one state to the next as before because of imperfect actuators.

But, now we have also:

Uncertainty on our knowledge of the state we’re in because of imperfect sensors.

The state is only partially observable: Partially Observable Markov Decision Process (POMDP)
POMDP

• As before:
  – States, \( s \)
  – Actions, \( a \)
  – Transitions, \( T(s,a,s') = P(s'|a,s) \)

• New:
  – The state is not directly observable, instead:
    – Observations, \( o \)
    – Observation model, \( O(s,o) = P(o|s) \)
POMDP: What is a “policy”?

- We don’t know for sure which state we’re in, so it does not make sense to talk about the “optimal” choice of action for a state

- All we can define is the probability that we are in any given state:
  \[ b(s) = [P(s_1),...,P(s_N)] \]

- Policy: Choice of action for a given belief state
  \( \pi(b) \): belief state \( b \) to action \( a \)

---

MDP with belief states instead of states
Unfortunately:
- Requires continuous representation
- Untractable in general
- Approximations or special cases

- All we can define is the probability that we are in any given state:
  \[ b(s) = [P(s_1),...,P(s_N)] \]

- Policy: Choice of action for a given belief state
  \( \pi(b) \): belief state \( b \) to action \( a \)

“belief state”
Probability that the agent is in state \( s_k \)
Summary

- Definition of discounted sum of rewards to measure utility
- Definition of Markov Decision Processes (MDP)
- Assumes observable states and uncertain action outcomes
- Optimal policy = choice of action that results in the maximum expected rewards in the future
- Bellman equation for general formulation of optimal policy in MDP
- Value iteration technique for computing the optimal policy
- Policy iteration technique for computing the optimal policy
- MDP = generalization of the deterministic search techniques studied earlier in class
- POMDP = MDP + Uncertainty on the observed states