Planning and Automated Decision Making: Introduction to AI Planning

Prof. Ronen Brafman

Introduction & Fully Observable MDP
Instructor: Prof. Ronen Brafman
  - Office: 209, building 37
  - Office Hours: Sunday & Wednesday 13-14
  - Email: brafman@cs.bgu.ac.il

Site: www.cs.bgu.ac.il/~aip181
Course Goals

Understand models and algorithms developed, mostly in AI, to support autonomous systems, and to automate and support decision making processes.
Applications

- System that can figure out how to achieve a goal
  - Autonomous systems (robots, game-playing agents, web agents)
  - Automated composition of web-services (e.g., planning a vacation)
  - Solve complicated planning problems (e.g., logistics, construction)

- Decision-support systems and decision analysis
  - Making decisions under uncertainty and complex objectives
  - Strategic decisions such as whether to invest in some technology, where to build a new plant, etc.

- Probabilistic expert systems (diagnosis of illness, machine problems, business problems)

- Preference elicitation, especially for e-commerce (not covered this year)
  - Helping users choose among a set of complex alternatives (camera, pc, vacation package)
NASA’s Mars Exploration Rovers

- Galileo’s Jupiter or Cassini’s Saturn missions:
  - $1G budget
  - Ground crew of 100-300
- Mars micro-rover Sojourne
  - $100M budget
  - Much smaller ground team
  - Sojourne was designed to operate for two months, but lasted much longer
  - Its level of autonomy is still low compared to what we can achieve
Autonomous Vehicle for Border Patrol

- Continuously monitor longer borders (fences)
- Able to recognize bombs planted and safely remove them
- Either fully autonomously, or operated remotely
- Saves lives!
- Current project!
Service Robot

- Respond to use requests
- Use planning algorithms to figure out what to do
- Voice interaction
- An ongoing project

Demo
Workload and Grade

- Two written assignments (10% each)
- Two programming assignments (13% each)
- Exam (50%).
- Online quizzes (4% total)
- Attendance Mandatory. May miss up to 6 classes
Main Topics

- Planning under uncertainty: How to act well in a stochastic environment or when we have only partial information
- Classical AI planning: Compact intuitive models of deterministic systems, search and heuristics
- Bayesian networks and influence diagrams: How to represent and reason about uncertain events
- Reinforcement learning: How to improve your performance in an unknown environment
An Example

- $S = (s_1...s_{11})$
- $A = \{\uparrow, \downarrow, \rightarrow, \leftarrow\}$
- Uncertainty on movement
  - For example: when moving up: $0.1 \leftarrow \uparrow \rightarrow 0.1$
- Attempt to move into a wall: stay in place
- 4&7 are sink states
- Reward for all states except 4&7: $-0.04$
- Goal: accumulate as much reward as possible
The optimal policy for $R(s) = -0.04$

Optimal policies for other choice for $R(s)$
Markov Decision Process (MDP)

\[ \langle S, A, Tr, R \rangle \]

- **S**: The set of States.
- **A**: The set of actions.
- **Tr**: \( S \times A \rightarrow \prod[S] \) – probabilistic transition function
  - \( Tr(s, a, s') = p \) implies that the probability of reaching \( s' \) from \( s \) with \( a \) is \( p \).
- **R**: \( S \times A \rightarrow \mathbb{R} \). \( R(s, a) \) : The reward for doing \( a \) in state \( s \)
We differentiate between different horizons:

- **Finite horizon**
  - Agent takes $n$ actions for some fixed $n$

- **Unbounded Horizon**
  - Agent takes finite number of actions, but apriori unbounded number of actions
    - In the maze domain, with probability 1, finite number of actions are executed

- **Infinite horizon**
  - Agent acts "forever" – infinitely many steps
    - The border patrol robot is designed to work "forever"
What is the value (utility) of the (possibly infinite) stream of rewards \(r_0, r_1, r_2, \ldots\) obtained by an agent?
- For example, is 100,99,98,...1, the same as 1,2,...,100?
- **Sum:** \(r_0 + r_1 + r_2 + \ldots\)
- **Discounted Sum:** \(r_0 + \gamma^1 r_1 + \gamma^2 r_2 + \ldots\)
- **Average (finite)** \(1/n[r_0 + r_1 + r_2 + \ldots r_n]\)
- **Average (infinite)** \(\lim_{n \to \infty} \{1/n[r_0 + r_1 + r_2 + \ldots r_n]\}\)
What does a solution look like?

- I should decide what to do based on all the information I have.
- Solution:
  - History \( h \): the sequence of all past states and actions:
    \[ s_0, a_1, s_1, a_2, s_2, \ldots, a_n, s_n \]
  - Policy \( p : h \rightarrow A \): What action to do as a function of the entire history.
Finite vs. Infinite horizon: Do we really care about the entire history?

- Markovian model: next state depends on current state only.
- So why care about anything except the current state?
- Consider the finite horizon case:
  - I have 100$ and only one step (think of days) to go, or
  - I have 100$ and two steps to go.
- Finite horizon: we care about current state and number of steps left.
- Infinite horizon: time is meaningless
  - We always have infinite steps to go
  - Same state, same possible futures
- Infinite horizon: past does not matter, future identical with identical states

→ Focus on policies of the form: \( p : S \rightarrow A \)
  - Called **stationary** policies

Isomorphic trees
Infinite horizon: Utility of a history

- To compare policies, we evaluate the histories/trajectories they induce.

- History $h : s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \ldots$

- $u(h)$ – the utility of history $h$
  - Could be discounted sum of rewards (what we will use)
  - Or, average reward

- What is the utility of a policy?
  - Each history occurs with probability $\prod \text{tr}(s_i, a_i, s_{i+1})$
  - Policy value = expected utility of all possible trajectories.
Infinite horizon with Discounted Sum of Rewards

The Value Function of Policy: $V_\rho : S \rightarrow \mathbb{R}$

$V_\rho(s) :$ The expected sum of discounted reward for policy $\rho$ when we start in state $s$

Optimal Policy

- **Claim:** There exists a policy $\rho^*$ such that for all $s \in S$ and for every policy $\rho$ we have that: $V_{\rho^*}(s) \geq V_\rho(s)$

- **Intuitive explanation:** Consider policies $\rho_1$ and $\rho_2$. Let $S_1$ be the states on which $V_{\rho_1} \geq V_{\rho_2}$. Let $S_2 = S \setminus S_1$ be the states on which $V_{\rho_2} > V_{\rho_1}$. Define $\rho$ to be identical to $\rho_1$ on $S_1$ and identical to $\rho_2$ on $S_2$. $V_\rho$ will be at least as high as $\max\{V_{\rho_1}, V_{\rho_2}\}$. 
Computing the Value Function $V_\rho$

- We solve a set of $n (= |S|)$ linear equations:
  - $V_\rho(s_1) = R(s_1, \rho(s_1)) + \gamma \sum_{s' \in S} Tr(s_1, \rho(s_1), s') \cdot V_\rho(s')$
  - $V_\rho(s_2) = R(s_2, \rho(s_2)) + \gamma \sum_{s' \in S} Tr(s_2, \rho(s_2), s') \cdot V_\rho(s')$
  - $\ldots$
  - $V_\rho(s_n) = R(s_n, \rho(s_n)) + \gamma \sum_{s' \in S} Tr(s_n, \rho(s_n), s') \cdot V_\rho(s')$
An Example

- We will consider a smaller 2x2 grid
- Actions are as before: u/d/l/r
- Actions succeed 80% of the time move to each side 10% of the time
- Rewards: see figure
- Policy $\rho$: see figure
Imagine we somehow know the value function of the optimal policy:

- Denote the optimal policy by $\rho^*$ and its value function by $v^*$

$$
\rho^*(s) = \arg\max_{a \in A} [R(s, a) + \gamma \cdot \sum_{s' \in S} Tr(s, a, s') \cdot v^*(s')] 
$$

- Given $v^*$ we can derive $\rho^*(s)$ for every $s \in S$.
Computing $v^*$ without knowing $\rho^*$

Bellman’s equation:

$$v^*(s) = \max_{a \in A} [R(s, a) + \gamma \cdot \sum_{s' \in S} Tr(s, a, s') \cdot v^*(s')]$$

$n$ non-linear equations with $n$ unknown

Value iteration is an iterative algorithm for computing the unknowns.

Initialization: $v_0^*(s) = \max_{a \in A} R(s, a)$ (or any other value)

Update: For all $s$, $v_{i+1}^*(s) = \max_{a \in A} [R(s, a) + \gamma \cdot \sum_{s' \in S} Tr(s, a, s') \cdot v_i^*(s')]$

Claim: $v_i^* \rightarrow v^*$ when $i \rightarrow \infty$
Value Iteration Key Points

- Given $v^*$ it is easy to find $\rho^*$.
- Bellman's equations define $v^*$ directly.
- Using iterative algorithm we can estimate $v^*$ as accurately as desired.

Demo
Evolution of utilities as a function of \# of iterations for various states during execution of value iteration
The graph shows the number of iterations required to obtain an error of $c \times R_{\text{max}}$ as a function of the discount factor $\gamma$. The curves represent different values of $c$: $c = 0.0001$, $c = 0.001$, $c = 0.01$, and $c = 0.1$. As the discount factor $\gamma$ increases, the number of iterations needed for a given error decreases.
Value Iteration: Convergence

- A mapping $f$ from a metric space to itself is a contraction if for some $c < 1$:
  $$\|f(x) - f(x')\| < c\|x - x'\|$$
- Every contraction on a complete metric space has a unique fixpoint (i.e. $f(x) = x$).
- A value function $U$ can be viewed as a vector of dimension $|S|$.
- Define $\|U\| = \max_S U(s)$.
- Define $B(U)$ to be the Bellman update of $U$:
  $$B(U(s)) = \max_{a \in A}[R(s, a) + \gamma \cdot \sum_{s' \in S} T(s, a, s') \cdot U(s')]$$
- $B$ is contraction on $R^{|S|}$
  $$\|BU - BU'\| < \gamma\|U - U'\|$$
Maximum error vs. policy loss for value iteration as function of # of iterations
Policy Iteration (Howard)

Start with an arbitrary policy and improve it.

Algorithm:
1. (Initial policy) Choose initial policy $\rho$.
2. (Policy evaluation) Compute $v_\rho$.
3. (Policy improvement) If there exists a state $s$ action $a$ s.t.:
   
   $a \neq \rho(s)$ and $v_\rho(s) < R(s, a) + \gamma \cdot \sum_{s' \in S} Tr(s, a, s') \cdot v_\rho(s')$

   3.1 $\rho(s) \leftarrow a$
   3.2 goto 2.
4. End.
Policy Iteration: Correctness

1. Correct termination: The algorithm terminates when
   - $\forall s \in S, \forall a \in A : v_\rho(s) \geq R(s, a) + \gamma \cdot \sum_{s' \in S} Tr(s, a, s') \cdot v_\rho(s')$
   - That is: $v_\rho(s) = \max_{a \in A} [R(s, a) + \gamma \cdot \sum_{s' \in S} Tr(s, a, s') \cdot v_\rho(s')]$
   - This means that $v_\rho$ is a solution to Bellman’s optimality conditions
   - Which implies: $\rho = \rho^*$

2. Termination:
   - There is a fine number of policies: $|A|^{|S|}$
   - (exercise) If the policy changed, the value of the new policy is as good as the old policy on all states, and strictly higher on at least one state
Linear Programming

- $x_1, \ldots, x_n$ is a set of real-valued variables
- **Linear objective function:** $\sum_{i=1,\ldots,n} c_i x_i$
- **Set of linear constraints:**
  \[
  a_1 x_1 + a_2 x_2 + \cdots + a_n x_n \geq c \\
  b_1 x_1 + b_2 x_2 + \cdots + b_n x_n \geq d
  \]
- $x_i \geq 0$
- Goal: find an assignment to $x_1 \cdots x_n$ that maximizes the objective function while satisfying all constraints
- Can be solved in polynomial time in the number of variables and constraints
A Linear Program for Finding an Optimal Policy

- Variables: \( v^*(s_i) \) for every \( s_i \in S \)
- Objective function: \( \min \sum_i v^*(s_i) \)
- Constraints:
  - \( v^*(s_1) \geq R(s_1, a_1) + \gamma \cdot \sum_{s' \in S} Tr(s_1, a_1, s') \cdot v^*(s') \)
  - \( v^*(s_1) \geq R(s_1, a_2) + \gamma \cdot \sum_{s' \in S} Tr(s_1, a_2, s') \cdot v^*(s') \)
  - \( \ldots \)
  - \( v^*(s_2) \geq R(s_2, a_1) + \gamma \cdot \sum_{s' \in S} Tr(s_2, a_1, s') \cdot v^*(s') \)
  - \( v^*(s_2) \geq R(s_2, a_2) + \gamma \cdot \sum_{s' \in S} Tr(s_2, a_2, s') \cdot v^*(s') \)
  - \( \ldots \)
  - \( \ldots \)

- Conclusion: we can find an optimal policy for an MDP in time polynomial in \( |V||A| \)
Finite Horizon

- Recall previous example:
- \( S = (s_1 \cdots s_{11}) \)
- \( A = \{ \leftarrow, \uparrow, \rightarrow, \downarrow \} \)
- \( R(s, a) = \)
  - \(-0.04 + 1, s = s_4 \)
  - \(-0.04 - 1, s = s_7 \)
  - \(-0.04 + 0, \text{otherwise} \)
- Uncertainty on movement - \(0.1 \leftarrow \uparrow \rightarrow 0.1\)
- Attempt to move to a water square keeps you in place.
- Assume horizon 20.
Solving Finite Horizon MDP using Backwards Induction

- **Search And/Or Tree:**
  - Root node is the initial state. It is an OR node.
  - Every OR node has one AND node child for every action.
  - Every AND node an OR node child for every possible effect of the parent action.

- **Node values:**
  - Value of leaf node is 0.
  - Value of an OR node (state) = maximum over the value of its children
  - Value of an AND node (action) = expected value of its children + immediate reward $R(s, a)$

- Backwards Induction evaluate node values bottom-up

- Improvement: Use a graph, not a tree
  - If two nodes denote same state at a level, use the same node.
  - Node value depends only on its sub-tree (future) not its history.
AND-OR Tree

At(Start)

Navigate(Start, R1)

At(R1)

TakePic(R1)

At(R1)
HavePic(R1)

Navigate(R1, R3)

At(R3)
HavePic(R1)

TakePic(R3)

At(R3)
HavePic(R1)
HavePic(R3)

TakePic(R3)

At(R3)
HavePic(R3)

At(R3)
HavePic(R1)
HavePic(R3)

Navigate(R1, R3)

TakePic(R2)

At(R2)

At(R2)
HavePic(R2)

Lost
Backward Induction
Backward Induction
Backward Induction

- **Navigate(Start, R1)**: $V = 16$, $Q = 12$
  - $V = 16$ (with $Q = 16$)
  - $V = 6$ (with $Q = 6$)
- **TakePic(R1)**: $V = 6$, $Q = 6$
  - $V = 6$ (with $Q = 6$)
  - $V = 10$
- **Navigate(R1, R3)**: $V = 8$, $Q = 8$
  - $V = 8$ (with $Q = 8$)
  - $V = 0$ (with $Q = 8$)
- **TakePic(R3)**: $V = 8$, $Q = 8$
  - $V = 8$ (with $Q = 8$)
  - $V = 0$
- **Navigate(Start, R2)**: $V = 20$, $Q = 15$
  - $V = 20$ (with $Q = 20$)
  - $V = 0$ (with $Q = 20$)
- **TakePic(R2)**: $V = 20$, $Q = 20$
  - $V = 20$ (with $Q = 20$)
  - $V = 0$ (with $Q = 20$)
AO*—A* for And/Or Trees

- Backward induction evaluates the entire tree
- Many path are irrelevant
- AO* uses a **heuristic** function $h : S \rightarrow R$ to focus on interesting parts of the tree
  - $h$ associates a value with internal nodes
  - $h(s)$ is an estimate of the true value of the node
  - $h$ is called **admissible** if it is optimistic (overestimates reward or underestimates cost)
  - We’ll discuss methods for generating $h$ later on
- Using forward heuristic search, we can focus on relevant parts of the tree
- We build the tree incrementally
  - Start with the root node
  - Expand (add children) the node with highest value
  - When $h$ optimistic (**admissible**), we can prune parts of the tree without sacrificing optimality
AO*
Pruning

“Currently Optimal”

Became “Optimal”

Greatly Reduced X

Caused the expansion Of this sub tree
AO*
AO*

- **Navigate(Start, R1)**
  - $V = 18$
  - $Q = 18$
  - $H = 24$
  - 0.75

- **Navigate(Start, R2)**
  - $V = 15$
  - $Q = 15$
  - $H = 20$
  - 0.75

- **V = 0**

Legend:
- open
- closed
- terminal
AO*

$Q = 15$

$H = 20$

$V = 0$

- **open**
- **closed**
- **terminal**
AO*

Diagram showing decision-making process with nodes and edges. Nodes include:
- Navigate(Start, R1)
- Navigate(Start, R2)
- TakePic(R1)
- TakePic(R2)
- Navigate(R1, R3)

Arrows indicate transitions with probabilities and costs:
- Q = 12 from Navigate(Start, R1)
- V = 16 from Navigate(Start, R1) with probability 0.75
- Q = 16 from TakePic(R1)
- H = 6 from TakePic(R1) with cost $10
- Q = 6 from Navigate(R1, R3)
- H = 8 from Navigate(R1, R3) with probability 0.75
- Q = 20 from TakePic(R2)
- H = 8 from TakePic(R2) with probability 0.25

Legend:
- : open
- : closed
- : terminal

Automated Planning and Decision Making 2017
AO*
AO*