CSE 573: Artificial Intelligence
Autumn 2012

Introduction & Search
Dan Weld

With slides from
Dan Klein, Stuart Russell, Andrew Moore, Luke Zettlemoyer

Course Logistics

Textbook:
Artificial Intelligence: A Modern Approach, Russell and Norvig (3rd ed)

Prerequisites:
- Data Structures (CSE 326 or CSE 322) or equivalent
- Understanding of probability, logic, algorithms, complexity

Work:
Readings (text & papers), Programming assignment (40%), Written assignments (10%), Final project (30%), Class participation (10%)

Topics
- Introduction
- Search Methods & Heuristic Construction
- Game Playing (minimax, alpha beta, expectimax)
- Markov Decision Processes & POMDPs
- Reinforcement Learning
- Knowledge Representation & Reasoning
  - Logic & Planning
  - Constraint Satisfaction
  - Uncertainty, Bayesian Networks, HMMs
- Supervised Machine Learning
- Natural Language Processing
- Mixed Human / Machine Computation

Prehistory
- Logical Reasoning: (4th C BC+) Aristotle, George Boole, Gottlob Frege, Alfred Tarski
- Probabilistic Reasoning: (16th C+) Gerolamo Cardano, Pierre Fermat, James Bernoulli, Thomas Bayes

1940-1950: Early Days

1942: Asimov: Positronic Brain; Three Laws of Robotics
1. A robot may not injure a human being or, through inaction, allow a human being to come to harm.
2. A robot must obey the orders given to it by human beings, except where such orders would conflict with the First Law.
3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Laws.

1943: McCulloch & Pitts: Boolean circuit model of brain

1946: First digital computer - ENIAC

The Turing Test

Turing (1950) “Computing machinery and intelligence”
- “Can machines think?”
- “Can machines behave intelligently?”
- The Imitation Game:

- Suggested major components of AI: knowledge, reasoning, language understanding, learning
1950-1970: Excitement

- 1950s: Early AI programs, including
  - Samuel's checkers program,
  - Newell & Simon's Logic Theorist,
  - Gelernter's Geometry Engine
- 1956: Dartmouth meeting: "Artificial Intelligence" adopted
- 1965: Robinson's complete algorithm for logical reasoning
  "Over Christmas, Allen Newell and I created a thinking machine."
  -Herbert Simon

1970-1980: Knowledge Based Systems

- 1969-79: Early development of knowledge-based systems
- 1980-88: Expert systems industry booms
- 1988-93: Expert systems industry busts
  "AI Winter"
  The knowledge engineer practices the art of bringing the principles and tools of AI research to bear on difficult applications problems requiring experts' knowledge for their solution.
  - Edward Felgenbaum in "The Art of Artificial Intelligence"

1988--: Statistical Approaches

- 1985-1990: Rise of Probability and Decision Theory
  - Eg, Bayes Nets
  - Judea Pearl - ACM Turing Award 2011
- 1990-2000: Machine learning takes over subfields:
  - Vision, Natural Language, etc.
  "Every time I fire a linguist, the performance of the speech recognizer goes up"
  - Fred Jelinek, IBM Speech Team

What is AI?

The science of making machines that:

<table>
<thead>
<tr>
<th>Think like humans</th>
<th>Think rationally</th>
</tr>
</thead>
<tbody>
<tr>
<td>Act like humans</td>
<td>Act rationally</td>
</tr>
</tbody>
</table>

What is a rational agent?

We'll use the term rational in a particular way:

- Rational: maximally achieving pre-defined goals
- Rational only concerns what decisions are made (not the thought process behind them)
- Goals are expressed in terms of the utility of outcomes
- Being rational means maximizing your expected utility

A better title for this course might be:

Computational Rationality
Can We Build It?

10^11 neurons
10^14 synapses
cycle time: 10^{-3} sec

vs.

10^9 transistors
10^12 bits of RAM
cycle time: 10^{-9} sec

State of the Art
May 1997

“lf I could feel –
I could smell – a
new kind of
intelligence
across the
table”
-Gary Kasparov

Saying Deep Blue
doesn’t really think
about chess is like
saying an airplane
doesn’t really fly
because it doesn’t
flap its wings.
– Drew McDermott

Agents

http://www.youtube.com/watch?v=WFR3I0m_xhE

Recommendations

More Top Picks for You

Agents

Stanford Car
DARPA Grand Challenge

Agents

Berkeley Autonomous Helicopter
What Can AI Do?

Quiz: Which of the following can be done at present?

- Play a decent game of Soccer?
- Play a winning game of Chess, Go, Jeopardy?
- Drive safely along a curving mountain road? University Way?
- Buy a week’s worth of groceries on the Web? At QFC?
- Make a car? Make a cake?
- Discover and prove a new mathematical theorem?
- Perform a complex surgical operation?
- Unload a dishwasher and put everything away?
- Translate Chinese into English in real time?

Brownies Anyone?

PS1: Search

Goal:
- Help Pac-man find his way through the maze

Techniques:
- Search: breadth-first, depth-first, etc.
- Heuristic Search: Best-first, A*, etc.

PS2: Game Playing

Goal:
- Play Pac-man!

Techniques:
- Adversarial Search: minimax, alpha-beta, expectimax, etc.

Mathematical Calculation

\[
\begin{align*}
\delta^2 u &= \left[ b^3 - \frac{(l + 1)}{2} \right] u(s) \\
\delta^3 b &= \left[ b^3 - \frac{(l + 1)}{2} \right] u(s) \\
\delta^3 b &= \left[ b^3 - \frac{(l + 1)}{2} \right] u(s) \\
\delta^3 b &= \left[ b^3 - \frac{(l + 1)}{2} \right] u(s) \\
\delta^3 b &= \left[ b^3 - \frac{(l + 1)}{2} \right] u(s)
\end{align*}
\]

Pacman as an Agent

Utility Function?

Implementation?

Originally developed at UC Berkeley:
[http://www.inst.eecs.berkeley.edu/~cs188/pacman/pacman.html](http://www.inst.eecs.berkeley.edu/~cs188/pacman/pacman.html)
PS3: Planning and Learning

Goal:
• Help Pac-man learn about the world

Techniques:
• Planning: MDPs, Value Iterations
• Learning: Reinforcement Learning

PS4: Ghostbusters

Goal:
• Help Pac-man hunt down the ghosts

Techniques:
• Probabilistic models: HMMS, Bayes Nets
• Inference: State estimation and particle filtering

Final Project

Your choice
(No final exam)

Advanced topics
• Partially-observable MDPs
• Supervised learning
• Natural language processing
• Mixed human/autonomous computation

Starting… Now!

• Assign 0: Python Tutorial
  • Online, but not graded

• Assign 1: Search
  • On the web.
  • Due Thurs 10/11
  • Start early and ask questions. It’s longer than most!

Outline

• Agents that Plan Ahead

• Search Problems

• Uninformed Search Methods (part review for some)
  • Depth-First Search
  • Breadth-First Search
  • Uniform-Cost Search

• Heuristic Search Methods (new for all)
  • Best First / Greedy Search

Agent vs. Environment

• An agent is an entity that perceives and acts.

• A rational agent selects actions that maximize its utility function.

• Characteristics of the percepts, environment, and action space dictate techniques for selecting rational actions.
Types of Environments

- Fully observable vs. partially observable
- Single agent vs. multiagent
- Deterministic vs. stochastic
- Episodic vs. sequential
- Discrete vs. continuous

Fully observable vs. Partially observable
Can the agent observe the complete state of the environment?

Single agent vs. Multiagent
Is the agent the only thing acting in the world?

Deterministic vs. Stochastic
Is there uncertainty in how the world works?

Episodic vs. Sequential
Does the agent take more than one action?

Discrete vs. Continuous
Is there a finite (or countable) number of possible environment states?
Types of Agent

- An agent is an entity that perceives and acts.
- A rational agent selects actions that maximize its utility function.
- Characteristics of the percepts, environment, and action space dictate techniques for selecting rational actions.

Reflex Agents

- Reflex agents:
  - Choose action based on current percept (and maybe memory)
  - Do not consider the future consequences of their actions
  - Act on how the world is
  - Can a reflex agent be rational?
  - Can a non-rational agent achieve goals?

Famous Reflex Agents

- Goal Based Agents
  - Plan ahead
  - Ask “what if”
  - Decisions based on (hypothesized) consequences of actions
  - Must have a model of how the world evolves in response to actions
  - Act on how the world would be

Search thru a Problem Space / State Space

- Input:
  - Set of states
  - Operators [and costs]
  - Start state
  - Goal state [test]
- Output:
  - Path: start $\Rightarrow$ a state satisfying goal test
  - [May require shortest path]
  - [Sometimes just need state passing test]

Example: Simplified Pac-Man

- Input:
  - A state space
  - A successor function
  - A start state
  - A goal test
- Output:
Ex: Route Planning: Romania → Bucharest

- **Input:**
  - Set of states
  - Operators [and costs]
  - Start state
  - Goal state (test)

- **Output:**

Example: N Queens

- **Input:**
  - Set of states
  - Operators [and costs]
  - Start state
  - Goal state (test)

- **Output**

Ex: Blocks World

- **Input:**
  - Set of states
  - Operators [and costs]
  - Start state
  - Goal state (test)

- **Output:**

Multiple Problem Spaces

- **Real World**
  - States of the world (e.g. block configurations)
  - Actions (take one world-state to another)

- **Robot’s Head**

  - **Problem Space 1**
    - PS states:
      - partially spec. plan
    - Operators:
      - models of actions

  - **Problem Space 2**
    - PS states:
      - partially spec. plan
    - Operators:
      - plan modificn ops

Algebraic Simplification

\[
\begin{align*}
\Delta_s &= \{ s \rightarrow \{ s, (s' \neq s) \} \} \\
\Delta_g &= \{ g \rightarrow \{ g, (g' \neq g) \} \} \\
\Delta_w &= \{ w \rightarrow \{ w, (w' \neq w) \} \} \\
\Delta_r &= \{ r \rightarrow \{ r, (r' \neq r) \} \} \\
\Delta_a &= \{ a \rightarrow \{ a, (a' \neq a) \} \}
\end{align*}
\]

- **Input:**
  - Set of states
  - Operators [and costs]
  - Start state
  - Goal state (test)

- **Output:**

State Space Graphs

- **State space graph:**
  - Each node is a state
  - The successor function is represented by arcs
  - Edges may be labeled with costs
  - We can rarely build this graph in memory (so we don’t)

- **Output:**

Ridiculously tiny search graph for a tiny search problem
State Space Sizes?

- Search Problem: Eat all of the food
- Pacman positions: 10 x 12 = 120
- Pacman facing: up, down, left, right
- Food Count: 30
- Ghost positions: 12

Search Methods

- Blind Search
  - Depth first search
  - Breadth first search
  - Iterative deepening search
  - Uniform cost search
- Local Search
- Informed Search
- Constraint Satisfaction
- Adversary Search

Search Trees

- A search tree:
  - Start state at the root node
  - Children correspond to successors
  - Nodes contain states, correspond to PLANS to those states
  - Edges are labeled with actions and costs
  - For most problems, we can never actually build the whole tree

Example: Tree Search

State Graph:

What is the search tree?

States vs. Nodes

- Nodes in state space graphs are problem states
  - Represent an abstracted state of the world
  - Have successors, can be goal / non-goal, have multiple predecessors
- Nodes in search trees are plans
  - Represent a plan (sequence of actions) which results in the node's state
  - Have a problem state and one parent, a path length, a depth & a cost
  - The same problem state may be achieved by multiple search tree nodes
Building Search Trees

- Search:
  - Expand out possible plans
  - Maintain a fringe of unexpanded plans
  - Try to expand as few tree nodes as possible

General Tree Search

- Important ideas:
  - Fringe
  - Expansion
  - Exploration strategy
- Main question: which fringe nodes to explore?

Review: Depth First Search

- **Strategy**: expand deepest node first
- **Implementation**:
  Fringe is a LIFO queue (a stack)

Review: Breadth First Search

- **Strategy**: expand shallowest node first
- **Implementation**:
  Fringe is a FIFO queue

Detailed pseudocode is in the book!
Search Algorithm Properties

- **Complete?** Guaranteed to find a solution if one exists?
- **Optimal?** Guaranteed to find the least cost path?
- **Time complexity?**
- **Space complexity?**

### Variables:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>Number of states in the problem</td>
</tr>
<tr>
<td>$b$</td>
<td>The maximum branching factor $B$ (the maximum number of successors for a state)</td>
</tr>
<tr>
<td>$C^*$</td>
<td>Cost of least cost solution</td>
</tr>
<tr>
<td>$d$</td>
<td>Depth of the shallowest solution</td>
</tr>
<tr>
<td>$m$</td>
<td>Max depth of the search tree</td>
</tr>
</tbody>
</table>

### DFS

- Infinite paths make DFS incomplete…
- How can we fix this?
- Check new nodes against path from $S$
- Infinite search spaces still a problem

### BFS

When will BFS outperform DFS?
When will DFS outperform BFS?

### Memory a Limitation?

- **Suppose:**
  - 4 GHz CPU
  - 6 GB main memory
  - 100 instructions / expansion
  - 5 bytes / node
  - 400,000 expansions / sec
  - Memory filled in 300 sec ... 5 min

### Comparisons
Iterative Deepening

Iterative deepening uses DFS as a subroutine:

1. Do a DFS which only searches for paths of length 1 or less.
2. If “1” failed, do a DFS which only searches paths of length 2 or less.
3. If “2” failed, do a DFS which only searches paths of length 3 or less.

…and so on.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS</td>
<td>Y</td>
<td>N</td>
<td>(O(b^d))</td>
<td>(O(bw))</td>
</tr>
<tr>
<td>BFS</td>
<td>Y</td>
<td>Y</td>
<td>(O(b^{d/2}))</td>
<td>(O(b^{d/2}))</td>
</tr>
<tr>
<td>Iterative Deepen</td>
<td>Y</td>
<td>Y</td>
<td>(O(b^{d/2}))</td>
<td>(O(b^{d/2}))</td>
</tr>
</tbody>
</table>

Cost of Iterative Deepening

<table>
<thead>
<tr>
<th>b</th>
<th>ratio ID to DFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>1.5</td>
</tr>
<tr>
<td>10</td>
<td>1.2</td>
</tr>
<tr>
<td>25</td>
<td>1.08</td>
</tr>
<tr>
<td>100</td>
<td>1.02</td>
</tr>
</tbody>
</table>

Speed

Assuming 10M nodes/sec & sufficient memory

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8 Puzzle</td>
<td>(10^5)</td>
<td>.01 sec</td>
<td>(10^5)</td>
<td>.01 sec</td>
</tr>
<tr>
<td>2x2x2 Rubik’s</td>
<td>(10^6)</td>
<td>.2 sec</td>
<td>(10^6)</td>
<td>.2 sec</td>
</tr>
<tr>
<td>15 Puzzle</td>
<td>(10^{13})</td>
<td>6 days</td>
<td>(10^{17})</td>
<td>20k yrs</td>
</tr>
<tr>
<td>3x3x3 Rubik’s</td>
<td>(10^{19})</td>
<td>68k yrs</td>
<td>(10^{20})</td>
<td>574k yrs</td>
</tr>
<tr>
<td>24 Puzzle</td>
<td>(10^{25})</td>
<td>12B yrs</td>
<td>(10^{37})</td>
<td>(10^{23}) yrs</td>
</tr>
</tbody>
</table>

Why the difference?
- Rubik has higher branching factor
- 15 puzzle has greater depth

When to Use Iterative Deepening

- N Queens?

Costs on Actions

Notice that BFS finds the shortest path in terms of number of transitions. It does not find the least-cost path.

Uniform Cost Search

Expand cheapest node first:
Fringe is a priority queue
Uniform Cost Search

Expansion order:
(S, p, d, b, e, a, r, f, e, G)

Cost contours

Priority Queue Refresher

- A priority queue is a data structure in which you can insert and retrieve (key, value) pairs with the following operations:
  - `pq.push(key, value)` inserts (key, value) into the queue.
  - `pq.pop()` returns the key with the lowest value, and removes it from the queue.
  - You can decrease a key’s priority by pushing it again
  - Unlike a regular queue, insertions aren’t constant time, usually O(log n)
  - We’ll need priority queues for cost-sensitive search methods

Uniform Cost Search

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS</td>
<td>Y</td>
<td>N</td>
<td>O(m)</td>
<td>O(mn)</td>
</tr>
<tr>
<td>BFS</td>
<td>Y</td>
<td>Y</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>UCS</td>
<td>Y*</td>
<td>Y</td>
<td>O(n^3)</td>
<td>O(n^3)</td>
</tr>
</tbody>
</table>

Uniform Cost Issues

- Remember: explores increasing cost contours
- The good: UCS is complete and optimal!
- The bad:
  - Explores options in every "direction"
  - No information about goal location

Uniform Cost: Pac-Man

- Cost of 1 for each action
- Explores all of the states, but one

Search Heuristics

- Any estimate of how close a state is to a goal
- Designed for a particular search problem

- Examples: Manhattan distance, Euclidean distance
Heuristics

Expand closest node first: Fringe is a priority queue

Best First / Greedy Search

Expand the node that seems closest...

What can go wrong?

A common case:
- Best-first takes you straight to the (wrong) goal
- Worst-case: like a badly-guided DFS in the worst case
  - Can explore everything
  - Can get stuck in loops if no cycle checking
- Like DFS in completeness (finite states w/ cycle checking)

To Do:
- Look at the course website:
  - http://www.cs.washington.edu/cse473/12sp
- Do the readings (Ch 3)
- Do PS0 if new to Python
- Start PS1, when it is posted

Extra Work?

- Failure to detect repeated states can cause exponentially more work (why?)
Graph Search

- In BFS, for example, we shouldn’t bother expanding the circled nodes (why?)

Some Hints

- Graph search is almost always better than tree search (when not?)
- Implement your closed list as a dict or set!
- Nodes are conceptually paths, but better to represent with a state, cost, last action, and reference to the parent node

Uniform Cost Search

- What will UCS do for this graph?
- What does this mean for completeness?

Best First / Greedy Search

- Strategy: expand the closest node to the goal

Best First Greedy Search

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform Cost Search</td>
<td>Y</td>
<td>N</td>
<td>O(b)</td>
<td>O(b)</td>
</tr>
</tbody>
</table>

Graph Search

- Very simple fix: never expand a state type twice
- Can this wreck completeness? Why or why not?
- How about optimality? Why or why not?
Example: Tree Search

5 Minute Break