CS 188: Artificial Intelligence

Search

Instructors: Angela Liu and Yanlai Yang

University of California, Berkeley

[slides adapted from Dan Klein, Pieter Abbeel, Stuart Russel, Dawn Song]
Today

- Finish discussion of agents and environments
- Search Problems
- Uninformed Search Methods
  - Depth-First Search
  - Breadth-First Search
  - Uniform-Cost Search
# Environment types

<table>
<thead>
<tr>
<th></th>
<th>Pacman</th>
<th>Backgammon</th>
<th>Diagnosis</th>
<th>Taxi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully or partially observable</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single-agent or multiagent</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deterministic or stochastic</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Static or dynamic</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Discrete or continuous</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Known physics?</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Known perf. measure?</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Agent design

The environment type largely determines the agent design

*Partially observable* => agent requires *memory* (internal state)

*Stochastic* => agent may have to prepare for *contingencies*

*Multi-agent* => agent may need to behave *randomly*

*Static* => agent has time to compute a rational decision

*Continuous time* => continuously operating *controller*

*Unknown physics* => need for *exploration*

*Unknown perf. measure* => observe/interact with *human principal*
Simple reflex agents

Agent

Sensors

What the world is like now

Condition-action rules

What action I should do now

Actuators

Environment
class GoWestAgent(Agent):

def getAction(self, percept):
    if Directions.WEST in percept.getLegalPacmanActions():
        return Directions.WEST
    else:
        return Directions.STOP
Eat adjacent dot, if any
Eat adjacent dot, if any
Can we (in principle) extend this reflex agent to behave well in all standard Pacman environments?

No – Pacman is not quite fully observable (power pellet duration)

Otherwise, yes – we can *in principle* make a lookup table.....

*How large would it be?*
Model-based agents

Agent

- State
- How the world evolves
- What my actions do
- Condition-action rules

What the world is like now

What action I should do now

Sensors

Environment

Actuators
Goal-based agents
Spectrum of representations

(a) Atomic

(b) Factored

(c) Structured
Outline of the course
CS 188: Artificial Intelligence

Search

Instructors: Angela Liu and Yanlai Yang

University of California, Berkeley

[slides adapted from Dan Klein, Pieter Abbeel, Stuart Russel, Dawn Song]
Planning Agents

- Planning agents decide based on evaluating future action sequences
- Search algorithms typically assume
  - Known, deterministic transition model
  - Discrete states and actions
  - Fully observable
  - Atomic representation
- Usually have a definite goal
- Optimal: Achieve goal at least cost
Move to nearest dot and eat it
Precompute optimal plan, execute it
Search Problems
Search Problems

- A search problem consists of:
  - A state space $S$
  - An initial state $s_0$
  - Actions $A(s)$ in each state
  - Transition model $Result(s,a)$
  - A goal test $G(s)$
    - $s$ has no dots left
  - Action cost $c(s,a,s')$
    - +1 per step; -10 food; -500 win; +500 die; -200 eat ghost

- A solution is an action sequence that reaches a goal state
- An optimal solution has least cost among all solutions
Search Problems Are Models
Example: Traveling in Romania

- **State space:**
  - Cities

- **Initial state:**
  - Arad

- **Actions:**
  - Go to adjacent city

- **Transition model:**
  - Reach adjacent city

- **Goal test:**
  - $s = $Bucharest$?

- **Action cost:**
  - Road distance from $s$ to $s'$

- **Solution?**
State Space Sizes

- **World state:**
  - Agent positions: 120
  - Food count: 30
  - Ghost positions: 12
  - Agent facing: NSEW

- **How many**
  - World states?
    \[120 \times (2^{30}) \times (12^2) \times 4\]
  - States for pathing?
    120
  - States for eat-all-dots?
    \[120 \times (2^{30})\]
State Space Graphs and Search Trees
State Space Graphs

- State space graph: A mathematical representation of a search problem
  - Nodes are (abstracted) world configurations
  - Arcs represent transitions (labeled with actions)
  - The goal test is a set of goal nodes (maybe only one)

- In a state space graph, each state occurs only once!

- We can rarely build this full graph in memory (it’s too big), but it’s a useful idea
State Space Graphs

- State space graph: A mathematical representation of a search problem
  - Nodes are (abstracted) world configurations
  - Arcs represent successors (action results)
  - The goal test is a set of goal nodes (maybe only one)

- In a state space graph, each state occurs only once!

- We can rarely build this full graph in memory (it’s too big), but it’s a useful idea
We construct the tree on demand – and we construct as little as possible.

Each NODE in the search tree is an entire PATH in the state space graph.
Consider this 4-state graph: How big is its search tree (from S)?
Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:

How big is its search tree (from S)?

Important: Those who don’t know history are doomed to repeat it!
Quiz: State Space Graphs vs. Search Trees

Consider a rectangular grid:

How many states within $d$ steps of start?

How many states in search tree of depth $d$?
Tree Search
Search Example: Romania
Creating the search tree
Creating the search tree
Creating the search tree
function \textsc{Tree-Search} (\textit{problem}, \textit{strategy}) \textbf{returns} a solution, or failure
initialize the search tree using the initial state of \textit{problem}

\textbf{loop do}
  \textbf{if} there are no candidates for expansion \textbf{then return} failure
  choose a leaf node for expansion according to \textit{strategy}
  \textbf{if} the node contains a goal state \textbf{then return} the corresponding solution
  \textbf{else} expand the node and add the resulting nodes to the search tree
\textbf{end}

- \textbf{Main variations:}
  - Which leaf node to expand next
  - Whether to check for repeated states
  - Data structures for frontier, expanded nodes
Depth-First Search
**Strategy:** expand a deepest node first

**Implementation:** Frontier is a LIFO stack
Search Algorithm Properties
Search Algorithm Properties

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

**Cartoon of search tree:**
- $b$ is the branching factor
- $m$ is the maximum depth
- Solutions at various depths

**Number of nodes in entire tree?**
- $1 + b + b^2 + \ldots + b^m = O(b^m)$
Depth-First Search (DFS) Properties

- **What nodes does DFS expand?**
  - Some left prefix of the tree down to depth $m$. 
  - Could process the whole tree!
  - If $m$ is finite, takes time $O(b^m)$

- **How much space does the frontier take?**
  - Only has siblings on path to root, so $O(bm)$

- **Is it complete?**
  - $m$ could be infinite
  - preventing cycles may help

- **Is it optimal?**
  - No, it finds the “leftmost” solution, regardless of depth or cost
Breadth-First Search
Breadth-First Search

Strategy: expand a shallowest node first

Implementation: Frontier is a FIFO queue
Breadth-First Search (BFS) Properties

- What nodes does BFS expand?
  - Processes all nodes above shallowest solution
  - Let depth of shallowest solution be $s$
  - Search takes time $O(b^s)$

- How much space does the frontier take?
  - Has roughly the last tier, so $O(b^s)$

- Is it complete?
  - $s$ must be finite if a solution exists, so yes!

- Is it optimal?
  - If costs are equal (e.g., 1)
Quiz: DFS vs BFS
Quiz: DFS vs BFS

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

[Demo: dfs/bfs maze water (L2D6)]
Iterative Deepening

- Idea: get DFS’s space advantage with BFS’s time / shallow-solution advantages
  - Run a DFS with depth limit 1. If no solution...
  - Run a DFS with depth limit 2. If no solution...
  - Run a DFS with depth limit 3. ..... 

- Isn’t that wastefully redundant?
  - Generally most work happens in the lowest level searched, so not so bad!
  - Extra work is $O(b^{s-1})$
Uniform Cost Search
Uniform Cost Search

\[ g(n) = \text{cost from root to } n \]

Strategy: expand lowest \( g(n) \)

Frontier is a priority queue sorted by \( g(n) \)
Uniform Cost Search (UCS) Properties

- **What nodes does UCS expand?**
  - Expands all nodes with cost less than cheapest solution!
  - If that solution costs $C^*$ and arcs cost at least $\varepsilon$, then the “effective depth” is roughly $C^*/\varepsilon$.
  - Takes time $O(b^{C^*/\varepsilon})$ (exponential in effective depth).

- **How much space does the frontier take?**
  - Has roughly the last tier, so $O(b^{C^*/\varepsilon})$.

- **Is it complete?**
  - Assuming $C^*$ is finite and $\varepsilon > 0$, yes!

- **Is it optimal?**
  - Yes! (Proof next lecture via A*)

$g \leq 1$
$g \leq 2$
$g \leq 3$
Assume known, discrete, observable, deterministic, atomic

Search problems defined by $S, s_0, A(s), \text{Result}(s,a), G(s), c(s,a,s')$

Search algorithms find action sequences that reach goal states

- Optimal => minimum-cost

Search algorithm properties:

- Depth-first: incomplete, suboptimal, space-efficient
- Breadth-first: complete, (sub)optimal, space-prohibitive
- Iterative deepening: complete, (sub)optimal, space-efficient
- Uniform-cost: complete, optimal, space-prohibitive
## Bonus Search Algo Summary

<table>
<thead>
<tr>
<th>Search</th>
<th>Frontier</th>
<th>Completeness</th>
<th>Optimality</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS (Depth-First)</td>
<td>Stack</td>
<td>tree search - no</td>
<td>no</td>
<td>$O(b^m)$</td>
<td>$O(bm)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cycle</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>graph search &lt; yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>finite</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>no</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>infinite</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BFS (Breadth-First)</td>
<td>queue</td>
<td>yes</td>
<td>no</td>
<td>$O(b^s)$</td>
<td>$O(b^s)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(except when all</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>edge costs same)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Iterative</td>
<td>Stack</td>
<td>yes</td>
<td>no</td>
<td>$O(b^s)$</td>
<td>$O(b^s)$</td>
</tr>
<tr>
<td>Deepening</td>
<td></td>
<td>(same as DFS)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(BFS result w/</td>
<td></td>
<td>(same as BFS)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>modified DFS algo)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UCS (Uniform Cost)</td>
<td>heap-based PQ</td>
<td>yes</td>
<td>yes</td>
<td>$O(b^{cE})$</td>
<td>$O(b^{cE})$</td>
</tr>
<tr>
<td>(backward cost)</td>
<td></td>
<td>(assuming positive</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>edge costs and $E &gt; 0$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>