CS 188: Artificial Intelligence

Search

Instructors: Stuart Russell and Dawn Song

University of California, Berkeley

[slides adapted from Dan Klein, Pieter Abbeel]
Today

- Agents that Plan Ahead
- Search Problems
- Uninformed Search Methods
  - Depth-First Search
  - Breadth-First Search
  - Uniform-Cost Search
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 \( \mathcal{A}(s) \) in each state
- Transition model \( \text{Result}(s,a) \)
- 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
Bucharest to London

British Airways • Fri, Jan 21

9:45am - 11:35am
3h 50m (Nonstop)
WiFi

Show details ▼
Example: Traveling in Romania

- **State space:**
  - Cities

- **Initial state:**
  - Arad

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

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

- **Goal test:**
  - \( s = \text{Bucharest?} \)

- **Action cost:**
  - Road distance from \( s \) to \( s' \)

- **Solution?**
Models are almost always wrong
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)?
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
Creating the search tree
Creating the search tree
Creating the search tree
General Tree Search

function **Tree-Search**(  \textit{problem, strategy} \textit{returns} a solution, or failure
initialize the search tree using the initial state of \textit{problem}
loop do
  if there are no candidates for expansion then return failure
  choose a leaf node for expansion according to \textit{strategy}
  if the node contains a goal state then return the corresponding solution
  else expand the node and add the resulting nodes to the search tree
end

- **Main variations:**
  - Which leaf node to expand next
  - Whether to check for repeated states
  - Data structures for frontier, expanded nodes
1. Frontier separates expanded from unexplored region of state-space graph
2. Expanding a frontier node:
   a. Moves a node from frontier into expanded
   b. Adds nodes from unexplored into frontier, maintaining property 1
Depth-First Search
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 (more later)

- **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)]
Example: Maze Water DFS/BFS (part 1)
Example: Maze Water DFS/BFS (part 2)
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!
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*)
Video of Demo Maze with Deep/Shallow Water --- BFS or UCS? (part 1)
Video of Demo Maze with Deep/Shallow Water --- BFS or UCS? (part 2)
Summary

- 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