CSEP 573: Artificial Intelligence

Search

slides adapted from Stuart Russel, Dan Klein, Pieter Abbeel from ai.berkeley.edu
And Hanna Hajishirzi, Jared Moore, Dan Weld
Today

- Agents that Plan Ahead
  - goal-based

- 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
- Must have a model of how the world evolves in response to actions
- Usually have a definite goal
- *Optimal*: Achieve goal at least cost
Optimal?
Precompute optimal plan, execute it
Search Problems
A search problem consists of:

- A state space $S$
- An initial state $s_0$
- Actions $A(s)$ 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
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
What’s in a State Space?

The **world state** includes every last detail of the environment

![Pacman board](image)

A **search state** keeps only the details needed for planning (abstraction)

- **Problem: Pathing** (= path finding)
  - States: (x,y); location
  - Actions: NSEW
  - Transition: update x,y value
  - Goal test: is (x,y)=destination

- **Problem: Eat-All-Dots**
  - States: pacman location, boolean for each food
  - Actions: NSEW
  - Transition: update x,y and possibly a dot Boolean
  - Goal test: dots all false
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 (path finding)?
    \[120\]
  - 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 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
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.
Quiz: State Space Graphs vs. Search Trees

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)?
Quiz: State Space Graphs vs. Search Trees

Consider a rectangular grid:

How many unique states within $d$ steps of start?

How many states in search tree of depth $d$?
Quiz: State Space Graphs vs. Search Trees

Consider a rectangular grid:

How many unique states within \( d \) steps of start?

*Enumerate after step 1:*

\[ \{4, 4 + 8, 4 + 8 + 12, \ldots \} \]

How many states in search tree of depth \( d \)?

\[ = O(4^d) \]
Tree Search
Search Example: Romania
Creating the search tree
Creating the search tree
Creating the search tree

[Diagram showing a search tree with nodes labeled Arad, Fagaras, Oradea, Rimnicu Vilcea, Timisoara, Zerind, and connections to other nodes like Arad, Lugoj, and Oradea.]
General Tree Search

function TREE-SEARCH(problem, strategy) returns a solution, or failure
initialize the search tree using the initial state of problem
loop do
    if there are no candidates for expansion then return failure
    choose a leaf node for expansion according to 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
Strategy: expand a deepest node first

Implementation: Frontier is a LIFO stack

(last in first out)
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?
  - $1 + b + b^2 + \ldots + b^m = O(b^m)$

*Remember $O(\ldots)$ is the upper bound of the function*
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
(first in first out)
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

(In terms of $S$, the depth of the shallowest solution and $M$, the maximum depth)

- When will BFS outperform DFS?
- When will DFS outperform BFS?
Quiz: DFS vs BFS

(In terms of $S$, the depth of the shallowest solution and $M$, the maximum depth)

- When will BFS outperform DFS?
  - $S << M$

- When will DFS outperform BFS?
  - $S \sim= M$
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!
  - Also useful for the meta data
BFS finds the shortest path in terms of number of actions. It does not find the least-cost path. We will now cover a similar algorithm which does find the least-cost path.
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?**
  - Processes all nodes with cost less than cheapest solution!
  - If solution costs $C^*$ and arcs cost at least $\epsilon$, then $C^*/\epsilon$ is effective depth (upper bound on depth of solution)
  - Takes time $O(b^{C^*/\epsilon})$ (exponential in effective depth)

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

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

- **Is it optimal?**
  - Yes! (Proof next lecture via A*)
Video of Demo Empty UCS
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)