Today’s Tutorial

- Agents that Plan Ahead
- Search Problems
- Uninformed Search Methods
  - Depth-First Search
  - Breadth-First Search
  - Uniform-Cost Search
Agents that Plan
Reflex Agents

- Reflex agents:
  - Choose action based on current percept (and maybe memory)
  - May have memory or a model of the world’s current state
  - Do not consider the future consequences of their actions

- Can a reflex agent be rational?
  - Can an agent jump to get the apple and land safely on the ground?
Searching/Planning Agents

- Searching/Planning agents:
  - Ask “what if”
  - Decisions based on (hypothesized or if I try this then what happens) consequences of actions
  - Must have a model of how the world evolves in response to actions
  - Must formulate a goal (test)
  - Consider how the world WOULD BE
Search Problems
Search Problems

- A search problem consists of:
  - A state space
  - A successor function (with actions, costs)
  - A start state and a goal test

- A solution is a sequence of actions (a plan) which transforms the start state to a goal state
Search Problem → Model (Representation of the World)

Is a model good or useful for my problem-solving?
Example: Traveling in Romania

- **State space:**
  - Cities
- **Successor function:**
  - Roads: Go to adjacent city with cost = distance
- **Start state:**
  - Arad
- **Goal test:**
  - Is state == Bucharest?
- **Solution?**
What’s in a State Space?

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

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

- **Problem: Pathing**
  - States: \((x,y)\) location
  - Actions: NSEW
  - Successor: update location only
  - Goal test: is \((x,y)\)=END

- **Problem: Eat-All-Dots**
  - States: \(\{(x,y),\text{ dot booleans}\}\)
  - Actions: NSEW
  - Successor: update location and possibly a dot boolean
  - Goal test: dots all false

**NSEW** – action to move one step to North, South, East, West
State Space Sizes?

- **World state:**
  - Agent positions: 120 (=4*30)
  - Food count: 30 (=5*6)
  - Ghost positions: 12
  - Agent facing: NSEW

- **How many**
  - World states?
    \[ = 120 \times (2^{30}) \times (12^2) \times 4 \]
    Position of pacman=120
    Food eaten or not= 2^{30}
    2-Ghost positions= 12^2
    Pacman facing = 4
  - States for pathing? 120
  - States for eat-all-dots? 120 \times (2^{30})
Quiz: Safe Passage

- Problem: eat all dots while keeping the ghosts scared (away from pacman)
- What does the state space have to specify?
  - (agent position, dot booleans, power pellet booleans, remaining scared time)
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!
Search Trees

- A search tree:
  - A “what if” tree of plans and their outcomes
  - The start state is the root node
  - Children correspond to successors
  - Nodes show states, but correspond to PLANS that achieve those states
State Space Graphs vs. Search Trees

Each NODE in the search tree is an entire PATH in the state space graph.

We construct both on demand – and we construct as little as possible.
State Space Graphs vs. Search Trees

Consider this 4-state graph:

How big is its search tree (from S)?

Important: Lots of repeated structure in the search tree!
Tree Search
Search Example: Romania
Searching with a Search Tree

- **Search:**
  - Expand out potential plans (tree nodes)
  - Maintain a fringe (frontier) of partial plans under consideration
  - 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?

```plaintext
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
```
Example: Tree Search
Depth-First Search
Depth-First Search

Strategy: expand a deepest node first

Implementation:
Fringe 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 DFS expand?
  - Some left prefix of the tree.
  - Could process the whole tree!
  - If m is finite, takes time $O(b^m)$

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

- Is it complete?
  - m could be infinite, so only if we prevent cycles (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: Fringe 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 fringe 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?
  - Only if costs are all 1 (more on costs later)
Quiz: DFS vs BFS
Search Tree (stack LIFO vs queue FIFO)

Depth-first search is recommended when all paths reach dead ends, or reach the goal in reasonable number of steps.
- d—depth of tree is small
- FILO stack

Breadth-first search is better for trees that are deep.
- Not good for large b.
- FIFO queue
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!
Cost-Sensitive Search

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

Strategy: expand a cheapest node first:

Fringe is a priority queue (priority: cumulative cost)
Uniform Cost Search (UCS) Properties

- What nodes does UCS expand?
  - Processes 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 fringe take?
  - Has roughly the last tier, so $O(b^{C^*/\varepsilon})$

- Is it complete?
  - Assuming best solution has a finite cost and minimum arc cost is positive, yes!

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

- Remember: UCS explores increasing cost contours

- The good: UCS is complete and optimal!

- The bad:
  - Explores options in every “direction”
  - No information about goal location

- We’ll fix that soon!
The One Queue

- All these search algorithms are the same except for fringe strategies
  - Conceptually, all fringes are priority queues (i.e. collections of nodes with attached priorities)
  - Practically, for DFS and BFS, you can avoid the log(n) overhead from an actual priority queue, by using stacks and queues
  - Can even code one implementation that takes a variable queuing object
Search and Models

- Search operates over models of the world
  - The agent doesn’t actually try all the plans out in the real world!
  - Planning is all “in simulation”
  - Your search is only as good as your models...
Search Gone Wrong?
Example 1 (Water jug puzzle)

There are two empty jugs, one of 4 gallons, one of 3 gallons.
Fill the 4-gallon jug with 2 gallons of water.

(a) Problem formulation
- Decide how to represent states
  - $s_0$: (initial state) (0,0)
  - $s_i$: (state i) $(x_i, y_i)$
    - where: $x_i$—content of 4-gallon jug
    - $y_i$—content of 3-gallon jug
  - $s_G$: (goal state) (2,0)
- Actions are to fill or empty the jugs

(b) Search space

Solution Path:
0 0
0 3
3 0
3 3
4 2
0 2
2 0
## Production Rules for the Water Jug Puzzle

<table>
<thead>
<tr>
<th>Rule</th>
<th>Condition</th>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(x, y) (x &lt; 4)</td>
<td>(4, y)</td>
<td>Fill the 4-gallon jug</td>
</tr>
<tr>
<td>2</td>
<td>(x, y) (y &lt; 3)</td>
<td>(x, 3)</td>
<td>Fill the 3-gallon jug</td>
</tr>
<tr>
<td>3</td>
<td>(x, y) (x &gt; 0)</td>
<td>(x-d, y)</td>
<td>Pour some water out of the 4-gallon jug</td>
</tr>
<tr>
<td>4</td>
<td>(x, y) (y &gt; 0)</td>
<td>(x, y-d)</td>
<td>Pour some water out of the 3-gallon jug</td>
</tr>
<tr>
<td>5</td>
<td>(x, y) (x &gt; 0)</td>
<td>(0, y)</td>
<td>Empty the 4-gallon jug on the ground</td>
</tr>
<tr>
<td>6</td>
<td>(x, y) (y &gt; 0)</td>
<td>(x, 0)</td>
<td>Empty the 3-gallon jug on the ground</td>
</tr>
<tr>
<td>7</td>
<td>(x, y) (x + y \geq 4) (y &gt; 0)</td>
<td>(4, y-(4-x))</td>
<td>Pour water from the 3-gallon jug into the 4-gallon jug until the 4-gallon jug is full</td>
</tr>
</tbody>
</table>
### Production Rules: Continuation

<table>
<thead>
<tr>
<th>Rule</th>
<th>Condition</th>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>((x, y)) (x + y \geq 3) and (x &gt; 0)</td>
<td>(\rightarrow (x - (3 - y), 3))</td>
<td>Pour water from the 4-gallon jug into the 3-gallon jug until the 3-gallon jug is full</td>
</tr>
<tr>
<td>9</td>
<td>((x, y)) (x + y \leq 4) and (y &gt; 0)</td>
<td>(\rightarrow (x + y, 0))</td>
<td>Pour all the water from the 3-gallon jug into the 4-gallon jug</td>
</tr>
<tr>
<td>10</td>
<td>((x, y)) (x + y \leq 3) and (x &gt; 0)</td>
<td>(\rightarrow (0, x + y))</td>
<td>Pour all the water from 4-gallon jug into the 3-gallon jug</td>
</tr>
<tr>
<td>11</td>
<td>((0, 2))</td>
<td>(\rightarrow (2, 0))</td>
<td>Pour the 2 gallons from the 3-gallon jug into the 4-gallon jug</td>
</tr>
<tr>
<td>12</td>
<td>((x, 2))</td>
<td>(\rightarrow (0, 2))</td>
<td>Empty the 4-gallon jug on the ground</td>
</tr>
</tbody>
</table>
Example 2 (More complex Water jug puzzle)

There are three empty jugs, one of 100 gallons, one of 4 gallons and one of 3 gallons. You are to design an intelligent agent to fill the 100-gallon jug with 5 gallons of water.

Specifically show

a) Formulation of the problem by deciding how to represent an arbitrary state, the initial state and the goal state.

An arbitrary state is represented as (X, Y, Z) where X,Y,Z are the volumes of water in 100 gallon jug, 4 gallon jug and 3 gallon jug respectively.

- Initial state (0,0,0)
- Goal state (5,X,X) here x is don’t care.

b) Select a set of actions in the form of if-then rules that are used in your solution. Write the rules in plain English.

1) Rule fill jug if empty
   if jug X is empty, fill jug X to the rim.

2) Transfer water from jug to jug
   if jug X is empty and jug Y has water, then move water from jug Y to jug X (condition X > Y)
Example 2: continuation

c) Show the search space of your design and the path to the solution. When showing the search space the first 2 levels are sufficient, but you need to show the complete path to the goal.

The solution is $5 = 4 + (4-3)$

Other possible solutions

a) $5 = (3+3+3)-4$

b) $5 = (4+4)-3$

c) $5 = (4-3)+4$
Example 3 (The Farmer, Fox, Goose, Grain Puzzle)

A farmer wants to move himself, a fox, a goose and some grain across a river. His boat is tiny, he can only take one of his possessions across on one trip. An unattended fox will eat a goose, and an unattended goose will eat the grain. What should he do?

First, we decide what a state is. A difficulty is that the states are not unique; some formulations are better than others. Here we pick a state as

(Farmer, Fox, Goose, Grain) Or (Fa, Fo, Go, Gr)

These binary values are:
0—in one side of the river
1—in the other side of the river
State Space Search

Legal moves from a state to another:
Each move involves the farmer,
& Farmer can carry at most one item,
& Forbidden states
  0X11, 1X00, 011X, 100X
Base of this we construct the operators
and the search space.

Note that the solution path is not unique.
This raises the question of optimality.
Basic Search Methods

- Blind search
  - Breadth-first
  - Uniform-cost – min g(n)
  - Depth-first
  - Depth-limited
  - Iterative Deepening

- Heuristic search
  - Hill climbing
  - Beam search
  - Best-first search
  - A* search

- Basic search
Summary - Search strategies

- A search strategy is defined by picking the order of node expansion.

- Strategies are evaluated along the following dimensions:
  - **completeness**: does it always find a solution if one exists?
  - **time complexity**: number of nodes generated
  - **space complexity**: maximum number of nodes in memory
  - **optimality**: does it always find a least-cost solution?

- Time and space complexity are measured in terms of:
  - $b$: maximum branching factor of the search tree
  - $d$: depth of the least-cost solution
  - $m$: maximum depth of the state space (may be $\infty$)
Properties of breadth-first search

- **Complete?** Yes (if \( b \) is finite)
- **Time?** \( b + b^2 + b^3 + \ldots + b^d = O(b^d) \)
- **Space?** \( O(b^d) \) (keeps every node in memory)
- **Optimal?** Yes (if cost = 1 per step)

- **Space** is the bigger problem (more than time)

Note: \( b \) is the branching factor, \( d \) is the depth of the shallowest solution, \( m \) is the maximum depth of the search tree, \( l \) is the depth limit.
Uniform-cost search

- Expand **least-cost (shortest path so far)** unexpanded node
- **Implementation**: frontier = priority queue ordered by path cost $g(n)$
- Equivalent to breadth-first if step costs all equal
- **Complete?** Yes, if each step cost $\geq \varepsilon$
  - if the cost of each step exceeds some small-positive integer $\varepsilon$
- **Time?** # of nodes with $g \leq$ cost of optimal solution, $O(b^{1+\left\lfloor C^*/\varepsilon \right\rfloor})$ where $C^*$ is the cost of the optimal solution.
- **Space?** # of nodes with $g \leq$ cost of optimal solution, $O(b^{1+\left\lfloor C^*/\varepsilon \right\rfloor})$
- **Optimal?** Yes – nodes expanded in increasing order of $g(n)$

Note: Suppose that the optimal cost is $C^*$

- Complexities are not computed as function of $b$ and $d$, instead of optimal cost $C^*$.
  Uniform-cost search examines all the nodes at the goal’s depth to see if one has lower cost (shortest path so far – to be conservative in search/exploration).

Note: $b$ is the branching factor, $d$ is the depth of the shallowest solution, $m$ is the maximum depth of the search tree, $l$ is the depth limit.
Properties of depth-first search

- **Complete?** No: fails in infinite-depth spaces, spaces with loops
  - Modify to avoid repeated states along path
    → complete in finite spaces
- **Time?** $O(b^m)$: terrible if $m$ is much larger than $d$
  - but if solutions are dense, may be much faster than breadth-first
- **Space?** $O(bm)$, i.e., linear space!
- **Optimal?** No

Note: $b$ is the branching factor, $d$ is the depth of the shallowest solution, $m$ is the maximum depth of the search tree, $l$ is the depth limit.
Iterative deepening search

Iterative deepening search (IDS)
Iterative deepening depth-first search (IDDFS) from wiki

Search strategy in which a depth-limited version of depth-first search is run repeatedly with increasing depth limits until the goal is found. IDDFS is optimal like breadth-first search, but uses much less memory; at each iteration, it visits the nodes in the search tree in the same order as depth-first search, but the cumulative order in which nodes are first visited is effectively breadth-first.
Iterative deepening search

- Number of nodes generated in a depth-limited search to depth $d$ with branching factor $b$:
  \[ N_{\text{DLS}} = b^1 + b^2 + \ldots + b^{d-2} + b^{d-1} + b^d \]
- Number of nodes generated in an iterative deepening search to depth $d$ with branching factor $b$:
  \[ N_{\text{IDS}} = b^1 + b^1 + b^2 + b^1 + b^2 + b^3 + \ldots + b^1 + b^2 + \ldots + b^{d-2} + b^{d-1} + b^d \]
  \[ = db^1 + (d-1)b^2 + \ldots + 3b^{d-2} + 2b^{d-1} + 1b^d \]
  \[ = O(b^d) \]
  \[ N_{\text{IDS}} = d b^1 + (d-1)b^2 + \ldots + 3b^{d-2} + 2b^{d-1} + 1b^d = O(b^d) \]
- For $b = 10$, $d = 5$,
  \[ N_{\text{DLS}} = 10 + 100 + 1,000 + 10,000 + 100,000 = 111,110 \]
  \[ N_{\text{IDS}} = 50 + 400 + 3,000 + 20,000 + 100,000 = 123,450 \]
- Overhead = \( \frac{(123,450 - 111,110)}{111,110} = 11\% \)

Note. IDS is iterative-deepening search. DLS is depth-limited search.

Note: $b$ is the branching factor, $d$ is the depth of the shallowest solution, $m$ is the maximum depth of the search tree, $l$ is the depth limit.
Properties of iterative deepening search

- **Complete?** Yes
- **Time?** $d b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
- **Space?** $O(bd)$
- **Optimal?** Yes, if step cost = 1

Since it is searching only a limited depth for each level, it can be done with DFS with limited level each time. That is, Space is $O(bd)$ or $O(bl)$ for l-level

Note: $b$ is the branching factor, $d$ is the depth of the shallowest solution, $m$ is the maximum depth of the search tree, $l$ is the depth limit.
Summary of algorithms

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Breadth-first</th>
<th>Uniform-Cost</th>
<th>Depth-First</th>
<th>Depth-limited</th>
<th>Iterative Deepening</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete?</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Time</td>
<td>$O(b^d)$</td>
<td>$O(b^{1+\lceil C/\epsilon \rceil})$</td>
<td>$O(b^m)$</td>
<td>$O(b')$</td>
<td>$O(b^d)$</td>
</tr>
<tr>
<td>Space</td>
<td>$O(b^d)$</td>
<td>$O(b^{1+\lceil C/\epsilon \rceil})$</td>
<td>$O(bm)$</td>
<td>$O(bl)$</td>
<td>$O(bd)$</td>
</tr>
<tr>
<td>Optimal?</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Note: $b$ is the branching factor, $d$ is the depth of the shallowest solution, $m$ is the maximum depth of the search tree, $l$ is the depth limit.