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.

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 ⇒ a state satisfying goal test
  - [May require shortest path]
  - [Sometimes just need state passing test]

Example: N Queens

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

Machine Learning: predict fuel efficiency

Discrete Data

Predict MPG

Need to find "Hypothesis": $f : X \rightarrow Y$
**Hypotheses:** decision trees $f : X \rightarrow Y$

- Each internal node tests an attribute $x_i$.
- Each branch assigns an attribute value $x_i = v$.
- Each leaf assigns a class $y$.
- To classify input $x$, traverse the tree from root to leaf, output the labeled $y$.

**Search Methods**

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

**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)

**State Space Sizes?**

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

**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: Depth First Search

Expansion ordering:
(d,h,a,c,e,h,p,q,q,f,c,a,G)

Review: Breadth First Search

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

Review: Breadth First Search

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

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

<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>No</td>
<td>No</td>
<td>Infinite</td>
<td>Infinite</td>
</tr>
</tbody>
</table>

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

DFS

<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 if finite</td>
<td>N</td>
<td>O(h)</td>
<td>O(few)</td>
</tr>
</tbody>
</table>

* Or graph search – next lecture.
BFS

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td>Y</td>
<td>N</td>
<td>O(b^m)</td>
<td>O(bm)</td>
</tr>
<tr>
<td>BFS</td>
<td>Y</td>
<td>Y</td>
<td>O(b^d)</td>
<td>O(bd)</td>
</tr>
</tbody>
</table>

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?)

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?

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

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

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

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.

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>BFS Time</th>
<th>Iter. Deep. Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 Puzzle</td>
<td>.01 sec</td>
<td>.01 sec</td>
</tr>
<tr>
<td>2x2x2 Rubik’s</td>
<td>.2 sec</td>
<td>.2 sec</td>
</tr>
<tr>
<td>15 Puzzle</td>
<td>6 days</td>
<td>20k yrs</td>
</tr>
<tr>
<td>3x3x3 Rubik’s</td>
<td>68k yrs</td>
<td>574k yrs</td>
</tr>
<tr>
<td>24 Puzzle</td>
<td>6 days</td>
<td>574k yrs</td>
</tr>
</tbody>
</table>

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

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

Priority Queue Refresher

- A priority queue is a data structure in which you can insert and retrieve (key, value) pairs with the following operations:
  - \( \text{pq.push}(\text{key}, \text{value}) \) inserts (key, value) into the queue.
  - \( \text{pq.pop()} \) 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 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
Exponentials Everywhere

“I think we’re going to need a stronger donkey…”

Search Heuristics

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

- Examples: Manhattan distance, Euclidean distance

Best First / Greedy Search

Expand closest node first: Fringe is a priority queue

- Expand the node that seems closest...

What can go wrong?
Greedy Search

Expand the node that seems closest...

What can go wrong?

Best First / Greedy Search

- 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)

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>Greedy Best-First Search</td>
<td>Y*</td>
<td>N</td>
<td>O(bm)</td>
<td>O(bm)</td>
</tr>
</tbody>
</table>

- What do we need to do to make it complete?
- Can we make it optimal?

A* Search

Hart, Nilsson & Rafael 1968

Best first search with \( f(n) = g(n) + h(n) \)

- \( g(n) = \) sum of costs from start to \( n \)
- \( h(n) = \) estimate of lowest cost path \( n \rightarrow \text{goal} \)

\( h(\text{goal}) = 0 \)

If \( h(n) \) is admissible and monotonic, then A* is optimal

Undestimates cost of reaching goal from node

f values increase from node to descendants (triangle inequality)

Graph Search Detail

When do we check for goals?

- When adding to queue?
- When removing from queue?

European Example
Optimality of A*
Suppose some suboptimal goal $G_2$ has been generated and is in the queue.
Let $n$ be an unexpanded node on a shortest path to an optimal goal $G_1$.

\[
\begin{align*}
  f(G_2) &= g(G_2) + h(G_2) \\
  &> g(G_1) + h(G_1) \\
  &\geq h(n) \\
\end{align*}
\]

Since $f(G_2) > f(n)$, A* will never select $G_2$ for expansion.

A* Summary
- Pros
- Cons

IDA* Analysis
- Complete & Optimal (ala A*)
- Space usage $\propto$ depth of solution
- Each iteration is DFS - no priority queue!
- # nodes expanded relative to A*
  - Depends on # unique values of heuristic function
  - In 8 puzzle: few values $\Rightarrow$ close to # A* expands
  - In traveling salesman: each f value often unique
    $\Rightarrow 1 + 2 + \ldots + n = O(n^2)$ where $n$=nodes A* expands
    if $n$ is too big for main memory, $n^2$ is too long to wait!
- Generates duplicate nodes in cyclic graphs

Optimality Continued
Lemma: A* expands nodes in order of increasing $f$ value
Gradually adds "f-contours" of nodes (cf. breadth-first adds layers)
Contour $i$ has all nodes with $f = f_i$, where $f_i < f_{i+1}$

Iterative-Deepening A*
- Like iterative-deepening depth-first, but...
- Depth bound modified to be an f-limit
  - Start with f-limit = $h$(start)
  - Prune any node if $f$(node) > f-limit
  - Next f-limit = min-cost of any node pruned

Forgetfulness
- A* used exponential memory
- How much does IDA* use?
  - During a run?
  - In between runs?
SMA*

- Use all available memory
- Start like A*
- When memory is full…
  - Erase node with highest f-value
  - First, backup parent with this f-value
  - So… parent knows cost-bound on best child

Alternative Approach to Finite Memory…

- Optimality is nice to have, but…

Depth-First Branch & Bound

- Single DF search
  - \(\rightarrow\) uses linear space
- Keep track of best solution so far
- If \(f(n) = g(n) + h(n) \geq \text{cost(best-soln)}\)
  - Then prune \(n\)

- Requires
  - Finite search tree, or
  - Good upper bound on solution cost
- Generates duplicate nodes in cyclic graphs

Beam Search

- Idea
  - Best first but only keep \(N\) best items on priority queue
- Evaluation
  - Complete?
  - Time Complexity?
  - Space Complexity?

Hill Climbing

- Idea
  - Always choose best child; no backtracking
  - Beam search with |queue| = 1

- Problems?
  - Local maxima
  - Plateaus
  - Diagonal ridges

Randomizing Hill Climbing

- Randomly disobeying heuristic
- Random restarts
  - (heavy tailed distributions)

\(\rightarrow\) Local Search
Simulated Annealing

- **Objective**: avoid local minima
- **Technique**:
  - For the most part use hill climbing
  - When no improvement possible
    - Choose random neighbor
    - Let $\Delta$ be the decrease in quality
    - Move to neighbor with probability $e^{-\Delta/T}$
  - Reduce “temperature” $(T)$ over time
- **Optimal?**
  - If $T$ decreased slowly enough, will reach optimal state
- **Widely used**
  - See also WalkSAT