Reminder!

- Project pre-proposals due Friday
Overview

- Uninformed search
  - BFS, DFS, Uniform-Cost, Graph-Search
- Informed search
  - Heuristics, $A^*$, admissibility, consistency, memory-bounded $A^*$
- Paper Discussions
Why search?

• Agents with several immediate options of unknown value can choose actions by:
  • examining different *sequences* of actions
  • choosing the best sequence
  • Examples?
Tree Search

- Explore state space by generating successors of explored states

- Need
  - Initial State
  - Successor function
  - Goal test
  - Path cost

How to get from Arad to Bucharest?
Tree Search

- Initial State: In(Arad)
- Successor function
  - <action, successor>
  - (Go(Sibiu), In(Sibiu))
  - Go(Timisoara), In(Timisoara)
  - Go(Zerind), In(Zerind)
- Goal test: In(Bucharest)
- Path cost: sum of step costs (in km)

How to get from Arad to Bucharest?
Class Exercise

- You have three jugs: 12 gal, 8 gal, and 3 gal and a water faucet.
- You can fill the jugs up or pour them out from one into another or onto the ground.
- You need to measure out exactly one gal.
- In groups, determine the following precisely enough to implement:
  - Initial state
  - Goal test
  - Successor function
  - Cost function
States vs. Nodes

• State is a property of the environment (location, configuration, etc)

• Node is data structure including (state, parent node, action, path cost, depth)

• A state can have multiple nodes
Applications
The Mother of All Search Algorithms

- current_state <= start_state
- loop:
  - is current_state = goal_state? if yes, output path and terminate
  - generate all next states of current_state and add to fringe set
  - current_state <= choose from fringe set
Complications

• How to choose next state from fringe set?

• Answer has significant implications on:
  • Time complexity
    • Number of loops
  • Space complexity
    • Size of fringe set
  • Completeness
    • Does loop terminate?
  • Optimality
    • Quality of solution path
Measuring complexity

- \( b \): maximum branching factor (largest number of next states of any state)
- \( d \): steps to goal
- \( m \): maximum possible path length
- not always finite
Breadth-first search

- Expand shallowest unexpanded node
- Implementation: FIFO queue is data structure for fringe set
Breadth-first search

• Expand shallowest unexpanded node

• Implementation: FIFO queue is data structure for fringe set
Breadth-first search

- Expand shallowest unexpanded node
- Implementation: FIFO queue is data structure for fringe set
Breadth-first search

- Expand shallowest unexpanded node
- Implementation: FIFO queue is data structure for fringe set
Properties of BFS

• Complete?
• Optimal?
• Space?
  • Size of fringe set? $O(b^{d+1})$
• Time?
Depth-first search

- Expand deepest unexpanded node
- Implementation: LIFO (stack) is data structure for fringe set
Depth-first search

- Expand deepest unexpanded node
- Implementation: LIFO (stack) is data structure for fringe set
Depth-first search

- Expand deepest unexpanded node
- Implementation: LIFO (stack) is data structure for fringe set
Depth-first search

- Expand deepest unexpanded node
- Implementation: LIFO (stack) is data structure for fringe set
Depth-first search

- Expand deepest unexpanded node
- Implementation: LIFO (stack) is data structure for fringe set
Depth-first search

• Expand deepest unexpanded node

• Implementation: LIFO (stack) is data structure for fringe set
Depth-first search

• Expand deepest unexpanded node

• Implementation: LIFO (stack) is data structure for fringe set

![Depth-first search tree](image-url)
Depth-first search

- Expand deepest unexpanded node
- Implementation: LIFO (stack) is data structure for fringe set
Depth-first search

- Expand deepest unexpanded node
- Implementation: LIFO (stack) is data structure for fringe set

![Diagram of a depth-first search tree with nodes A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, with A as the root and C as the selected node.]
Depth-first search

- Expand deepest unexpanded node
- Implementation: LIFO (stack) is data structure for fringe set
Properties of DFS

- Space?
  - \( bm+1 \)
- Time?
- Complete?
  - No. Why?
Properties of DFS

- Complete?
- Spaces with loops
  - Modify to avoid repeated states along path
- Infinite-depth spaces
  - What if we cut off after depth limit \( L \)
- Depth-Limited DFS
Guaranteeing Completeness

• Idea: Invoke Depth-Limited DFS with iteratively increasing values of L

• This is called Iterative Deepening Search
Iterative Deepening

L=0
Iterative Deepening

$L = 1$
Iterative Deepening

$L=2$
Iterative Deepening

$L = 3$

Etc.
Iterative Deepening Search--how efficient?

- Re-expanding all those nodes seems bad

- # of nodes in depth-limited search $L=d$
  
  $N = b^0 + b^1 + b^2 + \ldots + b^{d-2} + b^{d-1} + b^d$

- # of nodes in iterated deepening search to $L=d$

  $N = (d+1)b^0 + db^1 + (d-1)b^2 + \ldots + 3b^{d-2} + 2b^{d-1} + b^d$

- For $b=10, d=5$

  $N= 1+10+100+1,000+10,000+100,000=111,111$

  $N=6+50+400+3,000+20,000+100,000=123,456$

  Overhead = $(123,456-111,111)/111,111=12,345/111,111=11.11\%$
Bi-directional Search

- Search gets more efficient with depth
- Minimize depth by doing two searches
  - Forward from the Start State
  - Backward from the Goal
- Catch: sometimes it’s hard to go backward from the goal
What if not all paths are equal?
Uniform-cost search

- Expand least-cost unexpanded node
- Implementation: fringe is priority queue ordered by path cost
- Same as breath first search if all steps equal

![Search Diagram]

1. A
2. B: 1, 3
3. C: 0, 4
4. D: 3
5. E: 4
6. F: 0, 4
7. G
Uniform-cost search

- Expand least-cost unexpanded node
- Implementation: fringe is priority queue ordered by path cost
- Same as breath first search if all steps equal
Uniform-cost search

- Expand least-cost unexpanded node
- Implementation: fringe is priority queue ordered by path cost
- Same as breadth first search if all steps equal
Uniform-cost search

- Expand least-cost unexpanded node
- Implementation: fringe is priority queue ordered by path cost
- Same as breadth first search if all steps equal
Notation: $g(n)$

- $g(n)$: cost of path from start to node $n$
- Fringe is priority queue ordered by $g(n)$
Uniform-cost Search

• Complete? If step cost $\geq$ Epsilon
• Time and Space?
  • Cost of Optimal Soln = $C^*$
  • Cost = $O(b^{1+\text{floor}(C^*/E)})$
Graph Search

\[
\text{function} \ \text{GRAPH-SEARCH}(\text{problem}, \text{fringe}) \ \text{returns a solution, or failure}
\]

\[
\begin{align*}
\text{closed} & \leftarrow \text{an empty set} \\
\text{fringe} & \leftarrow \text{INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)} \\
\text{loop do} \\
& \quad \text{if fringe is empty then return failure} \\
& \quad \text{node} \leftarrow \text{REMOVE-FRONT(fringe)} \\
& \quad \text{if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)} \\
& \quad \text{if STATE[node] is not in closed then} \\
& \quad \quad \text{add STATE[node] to closed} \\
& \quad \quad \text{fringe} \leftarrow \text{INSERTALL(EXPAND(node, problem), fringe)}
\end{align*}
\]

- Remove repeated states
### 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>$O(b^{d+1})$</td>
<td>$O(b^{C^*/\varepsilon})$</td>
<td>$O(b^m)$</td>
<td>$O(b^l)$</td>
<td>$O(b^d)$</td>
</tr>
<tr>
<td>Space</td>
<td>$O(b^{d+1})$</td>
<td>$O(b^{[C^*/\varepsilon]}$</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: should be floor not ceiling in Uniform-Cost search
What about complexity?
Heuristics

• NP-Complete problems: impossible to guarantee optimality and sub-exponential run-time

• A heuristic algorithm gives up one or both

• Gives good average run-time or “good” (but not guaranteed optimal) solutions
Heuristic in Search

- Heuristic: a “rule of thumb”
- Formally: function that maps a state onto an estimate of the cost to the goal from that state
- Define $h(n) =$ estimated cost of path from node $n$ to goal
  - if $n$ is goal then $h(n) = 0$.

![Diagram of a search graph with nodes S, n', a, b, c, and G, and heuristic values h(a)=10, h(b)=18, h(c)=7, h(G)=0.](image)
Using Heuristics in Informed Search

- Assume $h(n)$ is given. How can we use it to do search?
- Data structure for the fringe set?

```
g(n') h(a) = 10
S -> n' -> a -> b -> c -> G
h(b) = 18
h(c) = 7
h(G) = 0
```
Greedy Best-First Search

- Priority queue using $h(n)$
- Pick the node closest to the goal
- Example fringe $(b, 18)$ $(a, 10)$ $(c, 7)$

```plaintext
S  n'  a  b  c  G
 g(n')  h(a)=10  h(b)=18  h(c)=7
 h(G)=0
```

Diagram shows:
- Start node $S$ connected to node $n'$
- Node $n'$ connected to nodes $a$, $b$, and $c$
- Nodes $a$, $b$, and $c$ leading to goal $G$

Nodes $a$, $b$, and $c$ with associated heuristic values $h(a)=10$, $h(b)=18$, and $h(c)=7$.

Goal $G$ has a heuristic value of $h(G)=0$. 

Diagram illustrates the best-first search approach where the priority queue is updated based on the heuristic function $h(n)$, selecting the node closest to the goal.
When Greedy Search Goes Bad

- Expand successors of \( n' \)
- Fringe set \[ (b, h(b)=3), (a, h(a)=1) \]
- Expand successors of a
  - Fringe set \[ (b, h(b)=3), (G, h(G)=0) \]
- Expand G...found goal...solution path: \( S \ldots n' \rightarrow a \rightarrow G \)
  - Cost = \( g(n') + 100 + 1 \)
  - Not optimal, \( g(n') + 4 \) exists
How to fix?

• So we have
  • $g(n)$: Cost from start to $n$
  • uniform-cost search
  • $h(n)$: Estimated cost from $n$ to goal
  • greedy best-first search
• Ideas?
A* Search

- Take into account total path cost
- Fringe set as a priority queue ordered by $f(n) = g(n) + h(n)$
  - $f(b) = ?$
  - $f(a) = ?$
- A* is really important
A* Search Example

- In state $n'$
  - Two successors, $a$ and $b$
    - $f(a) = g(a) + h(a) = 10 + 100 + 1 = 111$
    - $f(b) = g(b) + h(b) = 10 + 1 + 3 = 14$
  - Fringe set = $\{(b,14), (a,111)\}$
  - Choose to expand $b$
  - Next fringe = $\{(f,14),(a,111)\}$
Is A* Guaranteed Optimal?

- Nope, see $h(b)$
  - A* will return S...n'-> a -> G
- Heuristics can be wrong
“Good” Heuristics: Admissibility

Let $h^*(n)$ be the real cost of cheapest path from $n$ to goal

Definition: $h(n)$ is admissible iff

- for all $n$, $h(n)$ is less than or equal $h^*(n)$
  - Is $h(n') = 99$ admissible?
  - Is $h(n') = 4$ admissible?
  - Is $h(b) = 4$ admissible?
  - Is $h(b) = 0$ admissible?
  - Straight line distance on a map?
Theorem: If \( h(n) \) is admissible, \( A^* \) is optimal (if no repeated states)

- Suppose \( G_2 \) is suboptimal and path \((S...n'...G_2)\) is returned by \( A^* \)
  - Let \( n \) be the first unexpanded node in the optimal path to \( G_1 \)
  - In \( A^* \), the only way \( G_2 \) is expanded before \( n \) is if \( f(n) > f(G_2) \)
    - This is how priority queues work
  - Assume of \( f(n) > f(G_2) \): This is a contradiction, why?
Proof:

- $f(n) > f(G_2)$
- $f(G_2) = g(G_2) + h(G_2)$
- $f(G_2) = g(G_2)$
- $f(G_2) > g(G_1)$
- $f(G_2) > g(n) + h^*(n)$
- $f(G_2) > g(n) + h(n)$
- $f(G_2) > f(n)$ (Contradiction)

Assumption:

- $G_2$ is suboptimal
- $n$ is on opt path to $G_1$
- $h$ is admissible
When Admissibility Fails

- Is $h$ admissible?

$$g(n') = 10$$
$$h(n) = 7$$
$$h(a) = 5$$
$$h(b) = 1$$
$$h(G) = 0$$
When Admissibility Fails

- Expand node n:
  - \(f(a) = 12 + 5 = 17\)
  - \(f(b) = 14 + 1 = 15\)
  - Fringe set \(\{(a,17), (b,15)\}\)
- Expand b:
  - \(f(G) = 14 + 4 = 18\)
  - Fringe set \(\{(G,18),(a,17)\}\)
- Expand a: **Ignore b since b already visited**
- Expand G: test for goal, done. But suboptimal
Still not optimal using Graph-Search

- Graph-Search avoids repeated states by ignoring previously seen states
- What if the second path to the state is optimal?
  - Graph-Search discards it
  - This is why the proof breaks down if repeated states
- Need yet another requirement...
Consistency

- What is wrong with $h$?
• Require that
• $h(a) \leq h(b) + \text{cost}(a,b)$
• Make the heuristic obey the triangle inequality
Consistent Heuristics

• A heuristic is consistent if
  • For every node \( n \)
  • And every successor \( n' \) of
  • Generated by any action \( a \)
  • \( \text{cost}(n,a,n') + h(n') \geq h(n) \)
Consistent heuristics

- If \( h \) is consistent, \( f(n) \) is non-decreasing along any path

- \( f(n') = g(n') + h(n') \)
  - \( = g(n) + c(n,a,n') + h(n') \)
  - \( \geq g(n) + h(n) \)
  - \( f(n') \geq f(n) \)
Consistent Implies Admissible

• Thm: \( h(n) \leq h^*(n) \) for all \( n \)

• Proof by induction:
  • Base case: \( h^*(n) = 0 \); \( n \) is a goal state so \( h(n) = 0 \) by definition
  • Inductive hypothesis: For all \( n' \) after \( n \) on shortest path to Goal
    • Assume \( h(n') \leq h^*(n') \)
    • Lemma: \( h(n) \leq h^*(n) \)
    • \( h(n) \leq c(n,n') + h(n') \) by consistency
    • \( \leq c(n,n') + h^*(n') \) by inductive hypothesis
    • \( h(n) \leq h^*(n) \) by definition
Properties of A*

• Complete: yes (unless infinite nodes where \( f \leq f(G) \) )

• Time: exponential

• Space: All nodes in memory

• Optimal: Yes (if using consistent heuristic and Graph-Search)
Time: find a good admissible heuristic

• What kind of heuristics?
Time: find a good admissible heuristic

- $h_1(n) = \#$ of misplaced tiles
- $h_2(n) = \text{manhattan distance to goal (total number of squares from goal state)}$

$h_1(S) = 8$

$h_2(S) = 3+1+2+2+2+3+3+2=18$
Dominance

- if $h_2(n) \geq h_1(n)$ for all $n$ (both admissible)
- then $h_2$ dominates $h_1$
- $h_2$ is better for search
- Average search costs

<table>
<thead>
<tr>
<th></th>
<th>IDS</th>
<th>$A^*(h_1)$</th>
<th>$A^*(h_2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-puzzle</td>
<td>6384</td>
<td>39</td>
<td>25</td>
</tr>
<tr>
<td>12-puzzle</td>
<td>3644035</td>
<td>227</td>
<td>73</td>
</tr>
<tr>
<td>24-puzzle</td>
<td>too big</td>
<td>39135</td>
<td>1641</td>
</tr>
</tbody>
</table>
Complexity

- So time might be manageable with good heuristics, but space?
- Memory-bounded A*
Simplified Memory-Bounded A*

- Use all available memory
  - Expand best leaves until you can’t
  - Remove worst leaf (highest f-value)
  - Solve ties by expanding newest best leaf and removing oldest worst leaf
- Complete if soln reachable, optimal if optimal soln reachable
Iterative Deepening A* (IDA*)

• Use iterative deepening to progressively expand the memory usage based on f-cost

• Runs into the same problems as Uniform-Cost if paths/heuristics are real-valued
Recursive Best First Search (RBFS)

- Keep track of the best alternative path (f-limit)
- When current f-values exceed f-limit
  - Backtrack and
  - Store best f-value of children at backtracked nodes
Readings this week:
