CSE 473

Chapter 4

Informed Search

Last Time

Blind Search

BFS
UC-BFS
DFS
DLS
Iterative Deepening
Bidirectional Search
Repeated States

- Failure to detect repeated states can turn a linear problem into an exponential one! (e.g., repeated states in 8 puzzle)

- **Graph search algorithm**: Store expanded nodes in a set called *closed* and only add new nodes to the fringe

---

Graph Search

```mermaid
graph TD
A --> B
B --> C
C --> D
```

**function** `GRAPH-SEARCH(problem, fringe)` returns a solution, or failure

1. `closed` ← an empty set
2. `fringe` ← `INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)`
3. loop do
   1. if `fringe` is empty then return failure
   2. `node` ← `REMOVE-FRONT(fringe)`
   3. if `GOAL-TEST(problem)(STATE[node])` then return `SOLUTION(node)`
   4. if `STATE[node]` is not in `closed` then
      1. add `STATE[node]` to `closed`
      2. `fringe` ← `INSERTALL(EXPAND(node, problem), fringe)`
Can we do better?

All these methods are slow (blind)
Solution ⇒ use problem-specific knowledge to guide search (“heuristic function”) ⇒ “informed search”

Best-first Search
• Generalization of breadth first search
• Priority queue of nodes to be explored
• Evaluation function f(n) used for each node

Insert initial state into priority queue
While queue not empty
  Node = head(queue)
  If goal(node) then return node
  Insert children of node into pr. queue
Who’s on (best) first?

- **Breadth first is best first**
  With \( f(n) = \text{depth}(n) \)

- **Dijkstra’s Algorithm is best first**
  With \( f(n) = g(n) \)
  where \( g(n) = \text{sum of edge costs from start to } n \)

Greedy best-first search

- Evaluation function \( f(n) = h(n) \) (heuristic) = estimate of cost from \( n \) to goal

- e.g., Route finding problems: \( h_{SLD}(n) = \text{straight-line distance from } n \) to destination

- Greedy best-first search expands the node that appears to be closest to goal
Example: Lost in Romania

Example: Greedily Searching for Bucharest
Example: Greedily Searching for Bucharest

```
<table>
<thead>
<tr>
<th></th>
<th>Sibiu</th>
<th>Timisoara</th>
<th>Zerind</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sibiu</td>
<td></td>
<td>253</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>329</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>374</td>
</tr>
</tbody>
</table>
```

Example: Greedily Searching for Bucharest

```
<table>
<thead>
<tr>
<th></th>
<th>Sibiu</th>
<th>Timisoara</th>
<th>Zerind</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sibiu</td>
<td></td>
<td>253</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>329</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>374</td>
</tr>
<tr>
<td></td>
<td></td>
<td>368</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>178</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>390</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>193</td>
<td></td>
</tr>
</tbody>
</table>
```
Example: Greedily Searching for Bucharest

Not optimal!
Arad, Sibiu, Rimnicu Vilcea, Pitesti, Bucharest shorter

Properties of Greedy Best-First Search

- **Complete?** No – can get stuck in loops, e.g.,
  Iasi → Neamt → Iasi → Neamt →
- **Time?** $O(b^m)$, but a good heuristic can give dramatic improvement
- **Space?** $O(b^m)$ -- keeps all nodes in memory
- **Optimal?** No
A* Search
(Hart, Nilsson & Rafael 1968)

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

$g(n) =$ sum of edge costs from start to $n$
+ heuristic function $h(n) =$ estimate of lowest cost path from $n$ to goal

If $h(n)$ is “admissible” then search will be optimal

Underestimates cost of any solution which can be reached from node

Back in Romania Again

Aici noi energie iar!
A* Example

- Arad
  - Sibiu: 393 = 140 + 253
  - Timisoara: 447 = 118 + 329
  - Zerind: 449 = 75 + 374
A* Example

1. Arad
2. Sibiu
3. Timisoara
4. Zerind

Arad:
- Fagaras: 640+280+360, 415+230+170, 671+291+380, 413+220+193

Sibiu:
- Arad: 640+280+360, 415+230+170, 671+291+380, 413+220+193

Timisoara:
- Arad: 640+280+360, 415+230+170, 671+291+380, 413+220+193

Zerind:
- Arad: 640+280+360, 415+230+170, 671+291+380, 413+220+193

A* Example

1. Arad
2. Sibiu
3. Timisoara
4. Zerind

Arad:
- Fagaras: 526+306+100, 417+317+100, 553+300+253

Sibiu:
- Arad: 526+306+100, 417+317+100, 553+300+253

Timisoara:
- Arad: 526+306+100, 417+317+100, 553+300+253

Zerind:
- Arad: 526+306+100, 417+317+100, 553+300+253
A* Example
Admissible heuristics

• A heuristic $h(n)$ is **admissible** if for every node $n$,
  $$h(n) \leq h^*(n)$$
  where $h^*(n)$ is the true cost to reach the goal state from $n$.

• An admissible heuristic never overestimates the cost to reach the goal, i.e., it is **optimistic**

Admissible Heuristics

• Example: Straight Line Distance heuristic $h_{SLD}(n)$ is **admissible** (never overestimates the actual road distance)

• Theorem: If $h(n)$ is admissible, A* using TREE-SEARCH is optimal.
Optimality of $A^*$ (proof)

- Suppose some suboptimal goal $G_2$ has been generated and is in the fringe. Let $n$ be an unexpanded node in the fringe such that $n$ is on a shortest path to an optimal goal $G$.

  - $f(G_2) = g(G_2)$ since $h(G_2) = 0$
  - $> g(G)$ since $G_2$ is suboptimal
  - $f(G) = g(G)$ since $h(G) = 0$
  - $f(G_2) > f(G)$ from above

Optimality of $A^*$ (cont.)

- Suppose some suboptimal goal $G_2$ has been generated and is in the fringe. Let $n$ be an unexpanded node in the fringe such that $n$ is on a shortest path to an optimal goal $G$.

  - $f(G_2) > f(G)$ from above
  - $h(n) \leq h^*(n)$ since $h$ is admissible
  - $g(n) + h(n) \leq g(n) + h^*(n)$
  - $f(n) \leq f(\emptyset)$

Hence $f(n) < f(G_2)$, i.e., $A^*$ will never select $G_2$ for expansion.
Optimality of A*

- A* expands nodes in order of increasing $f$ value
- Gradually adds "$f$-contours" of nodes

Okay, proof is done!
Time to wake up...
Properties of A*

- **Complete?** Yes (unless there are infinitely many nodes with $f \leq f(G)$)
- **Time?** Exponential (for most heuristic functions in practice)
- **Space?** Keeps all generated nodes in memory (exponential number of nodes)
- **Optimal?** Yes

Admissible heuristics

E.g., for the 8-puzzle, what are some heuristic functions?

- $h_1(n) =$ ?
- $h_2(n) =$ ?

![Start State](image1)

![Goal State](image2)
Admissible heuristics

E.g., for the 8-puzzle:
- \( h_1(n) \) = number of misplaced tiles
- \( h_2(n) \) = total Manhattan distance (no. of squares from desired location of each tile)

\[
\begin{align*}
&h_1(S) = ? \\
&h_2(S) = ?
\end{align*}
\]
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

Dominance

- E.g., for 8-puzzle heuristics $h_1$ and $h_2$, typical search costs (average number of nodes expanded for solution depth $d$):

  - $d=12$  
    - IDS = 3,644,035 nodes
    - $A^*(h_1) = 227$ nodes
    - $A^*(h_2) = 73$ nodes
  
  - $d=24$  
    - IDS = too many nodes
    - $A^*(h_1) = 39,135$ nodes
    - $A^*(h_2) = 1,641$ nodes
Iterative-Deepening A*

- Like iterative-deepening search, but...
- Depth bound modified to be an \textbf{f-limit}

Start with \textit{limit} = \( h(\text{start}) \)
Prune any node if \( f(\text{node}) > f\text{-limit} \)
Next \( f\text{-limit}=\text{min-cost of any node pruned} \)

Next Time

- How to climb hills
- How to reach the top by annealing
- How to simulate and profit from evolution