Lectures on Search

- Formulation of search problems.
  - State Spaces
- Uninformed (blind) search algorithms.
- Informed (heuristic) search algorithms.
- Constraint Satisfaction Problems.
- Game Playing Problems.
Review

State spaces: formulating problems as graph search

• States
• Operators
• Start and goal states
• Costs
  – Cost of search
  – Cost of path from start to goal
  – Cost of goal

Review

• Some classical problems
  – As examples of formulation
  – Because they are often referred to

• Problems
  – 8 puzzle (also 15 puzzle)
  – N-queens
  – Water jugs
  – Towers of Hanoi
Review

• The same problem can be formalized in different ways.
  – Better if fewer nodes in space
  – Better if fewer paths to same node
• A non-classical problem: vacuum cleaner
  – An example of a non-observable environment

Lectures on Search

• Formulation of search problems.
• Uninformed (blind) search algorithms.
• Informed (heuristic) search algorithms.
• Constraint Satisfaction Problems.
• Game Playing Problems.
Searching a Graph

- Simple tree search
  - Depth first, breadth first, constant cost
- Problem: infinite trees
  - If this path is infinite, DFS never gets here.
- Problem: cycles look like infinite trees

Iterative Deepening

- Completeness and shortest-first order of BFS
- Space cost of DFS
- Time cost not much more than B/DFS
Uniform Cost

- Keep a list of open nodes (aka frontier)
- \( g(n) \) = distance from root
- Loop expanding node with minimal \( g \)

![Diagram of Uniform Cost](image)

Lectures on Search

- Formulation of search problems.
  - State Spaces
- Uninformed (blind) search algorithms.
- Informed (heuristic) search algorithms.
- Constraint Satisfaction Problems.
- Game Playing Problems.
Informed Search

- Incorporate problem-specific knowledge into the search strategy.
- Only useful if extra knowledge exists.
- One common form:
  \[ f(n) = \text{estimate of distance to goal} \]

Best-First Search

- Expand the nodes in order of their \( f \) value.
- Also known as “Greedy” search

Initialize open to contain root
Loop while not(found or empty(open))
  new = expand(head(open))
  open =
    sort-by-value(append(rest(open), new))
Example

The Sixteen Puzzle

- Number of misplaced tiles: $h(n) = 9$
The Sixteen Puzzle

1 10 3 8
15 6 7 9
4 2 5
13 14 12 11

5 steps

- Sum of Manhattan distances to correct place, $h(n)=26$

A Maze Problem

Entrée

Exit
Heuristic for a Maze Problem

Manhattan Distance: \( h(s) = \text{North/South distance to exit} + \text{East/West distance to exit} \)

Euclidean Distance: \( h(s) = \text{Straight line distance to exit "as the crow flies"} \)
Notes:

- not optimal (best path through R.V. and Pitesti)
- prone to false starts (consider Iasi to Fagaras)
- not complete (if repeated states are not checked)
**A* idea**

Heuristic Evaluation Functions

- **Initial State**
- **Current State** $s$
- **Goal State**

$g(s)$: Cost of shortest path from initial state to current state $s$.

$h(s)$: Estimate of cost of shortest path from current state $s$ to a goal state.

$f(s) = g(s) + h(s)$

---

**Monotonic Heuristic Function**

- A heuristic evaluation function $h(s)$ is said to be "monotonic" if $f(s) = g(s) + h(s)$ does not go down along any path in the state space.
Monotone $f$

- $f(s_{i+1}) \geq f(s_i)$
- $g(s_{i+1}) + h(s_{i+1}) \geq g(s_i) + h(s_i)$
- $g(s_i) + \text{cost}(O(s_i)) + h(s_{i+1}) \geq g(s_i) + h(s_i)$
- $\text{cost}(O(s_i)) + h(s_{i+1}) \geq h(s_i)$
- $h(s_{i+1}) \geq h(s_i) - \text{cost}(O(s_i))$

- How to define a monotonic heuristic evaluation function?
  
  $h_{\text{monotonic}}(s_{i+1}) = \max[h(s_{i+1}), h(s_i) - \text{cost}(O(s_i))]$

A* example

```
g, h, f
```

```
0, 8, 8
```

```
3, 5, 8
```

```
1, 7, 8
```

```
6, 5, 11
```

```
4, 4, 8
```

```
6, 3, 9
```

```
6, 0, 6
```
A* search contours

- If \( f^* \) is the cost of the optimal solution path, then:
  - A* expands all nodes with \( f(n) < f^* \).
  - A* may expand some of the nodes on the "goal contour" for which \( f(n) = f^* \) before selecting a goal node.
  - A* is optimally efficient for a given heuristic (no other algorithm is guaranteed to expand fewer nodes).

Topographical Interpretation of A* Algorithm

Uniform Cost Search:

A* Search:
**Admissible Heuristic Function**

- A heuristic evaluation function \( h(s) \) is said to be "admissible" if \( h(s) \) is always less than or equal to the cost of the shortest path from \( s \) to a goal state, i.e., \( h(s) \) underestimates the cost of reaching a solution from state \( s \).

- Therefore:
  - The value of \( h(\text{GoalState}) \) is zero.
  - The value of \( f(\text{GoalState}) \) is the exact cost of a shortest path
  - from the initial state to a goal state.

**A* Finds Optimal Solutions if \( h(s) \) is Admissible**
How complete is A*

• A* will expand nodes in order of increasing f and thus it will eventually reach a goal state with cost f*.
• Unless there is an infinite number of states with f(n) ≤ f*.

Heuristic Performance

• If N nodes are expanded and the solution depth is d, then the effective branching factor, b*, is the branching factor that a uniform tree of depth d would need to contain N nodes.
• \[ N = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d \]
• \( h_2 \) dominates \( h_1 \) if, for every node n,
  \[ h_2 (n) \geq h_1 (n) \]
• Larger heuristics have smaller branching factors.
• If several heuristics exist for a problem, the best way is to use a composite heuristic:
• \( h(n) = \max(h_1 (n), \ldots, h_m (n)) \)
Comparison of performance for 8-puzzle

<table>
<thead>
<tr>
<th></th>
<th>Search Cost</th>
<th>Effective Branching Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IDS</td>
<td>$A^*(h_1)$</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>112</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>680</td>
<td>20</td>
</tr>
<tr>
<td>8</td>
<td>6384</td>
<td>39</td>
</tr>
<tr>
<td>10</td>
<td>47127</td>
<td>93</td>
</tr>
<tr>
<td>12</td>
<td>364404</td>
<td>227</td>
</tr>
<tr>
<td>14</td>
<td>3473941</td>
<td>539</td>
</tr>
<tr>
<td>16</td>
<td>–</td>
<td>1301</td>
</tr>
<tr>
<td>18</td>
<td>–</td>
<td>3056</td>
</tr>
<tr>
<td>20</td>
<td>–</td>
<td>7276</td>
</tr>
<tr>
<td>22</td>
<td>–</td>
<td>18094</td>
</tr>
<tr>
<td>24</td>
<td>–</td>
<td>39135</td>
</tr>
</tbody>
</table>

Where do Heuristics Come From?

- Convert the original problem into a simpler one.
  - E.g., A decomposable problem.
- Solve the simple problem using a specialized method.
  - E.g., By decomposition/recomposition.
- Use the solution of the simple problem as a guide to solving the original problem.
  - E.g., Let heuristic evaluation function $h(s)$ compute the length of the solution to the simple problem.