Informed (Heuristic) Search Strategies

A* Search
Outline

• Greedy Best First Search

• A* Search

• Graph Search

• Heuristic Design
Recap: Search

• **Search problem:**
  – States (configurations of the world)
  – Successor function: a function from states to lists of (state, action, cost) triples; drawn as a graph
  – Start state and goal test

• **Search tree:**
  – Nodes: represent plans for reaching states
  – Plans have costs (sum of action costs)

• **Search Algorithm:**
  – Systematically builds a search tree
  – Chooses an ordering of the fringe (unexplored nodes)
  – Optimal: finds least-cost plans
Example: Pancake Problem
Example: Pancake Problem

State space graph with costs as weights

Cost: number of pancakes flipped
General Tree Search

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

Action: Flip top two
Cost: 2

Action: Flip all four
Cost: 4

Path to reach Goal:
Flip four, flip three
Total Cost: 7
Uniform Cost Search

- **Strategy:** expand lowest path cost

- **The good:** UCS is complete and optimal!

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

• Heuristic function $h(n)$ (a function from states to numbers):
  Any estimate of how close a state is to a goal ($h(n)= 0$ for goal node)

• Designed for each particular search problem

• Example: Manhattan distance, Euclidean distance
Example: Heuristic Function

$h(x)$

<table>
<thead>
<tr>
<th>Location</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arad</td>
<td>366</td>
</tr>
<tr>
<td>Bucharest</td>
<td>0</td>
</tr>
<tr>
<td>Craiova</td>
<td>160</td>
</tr>
<tr>
<td>Dobreta</td>
<td>242</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
</tr>
<tr>
<td>Fagaras</td>
<td>178</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
</tr>
<tr>
<td>Hirsova</td>
<td>151</td>
</tr>
<tr>
<td>Iasi</td>
<td>226</td>
</tr>
<tr>
<td>Lugoj</td>
<td>244</td>
</tr>
<tr>
<td>Mehadia</td>
<td>241</td>
</tr>
<tr>
<td>Neamt</td>
<td>234</td>
</tr>
<tr>
<td>Oradea</td>
<td>380</td>
</tr>
<tr>
<td>Pitesti</td>
<td>98</td>
</tr>
<tr>
<td>Rimnicu Vilcea</td>
<td>193</td>
</tr>
<tr>
<td>Sibiu</td>
<td>253</td>
</tr>
<tr>
<td>Timisoara</td>
<td>329</td>
</tr>
<tr>
<td>Urziceni</td>
<td>80</td>
</tr>
<tr>
<td>Vaslui</td>
<td>199</td>
</tr>
<tr>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>
Example: Heuristic Function

Heuristic: the largest pancake that is still out of place
Greedy Best-First Search

- **Strategy**: expand a node that you think is closest to a goal state
  - **Heuristic**: estimate of distance to nearest goal for each state

- What can go wrong?
Example: Heuristic Function

Red Path (GBF) = 450
Green path (UCS) = 418

$h(x)$
Greedy Best-First 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

- **Not optimal**
  - heuristic is just an estimate to goal and GBF ignores the distance from root

- **Like DFS in completeness**
  - complete only if finite states with cycle checking
Combining UCS and Greedy

- Uniform-cost orders by path cost, or backward cost $g(n)$
- Greedy orders by goal proximity, or forward cost $h(n)$

- A* Search orders by the sum: $f(n) = g(n) + h(n)$
Combining UCS and Greedy

- **Uniform-cost** orders by path cost, or *backward cost* $g(n)$
- **Greedy** orders by goal proximity, or *forward cost* $h(n)$

**A* Search** orders by the sum: $f(n) = g(n) + h(n)$
When should A* terminate?

- Should we stop when we enter a goal in the frontier?

- No: only stop when we select a goal for expansion
Is A* optimal?

Overestimated $h$

- What went wrong?
- Actual cost of bad goal < estimated cost of good goal
- We need estimates to be less than actual costs!
Admissible Heuristics

• A heuristic $h$ is *admissible* if:

$$h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal

• Admissible heuristics are optimistic (underestimate the cost)

• Examples:

• Coming up with admissible heuristics is most of what’s involved in using A* in practice.
Optimality of A*: Blocking

Notation: …

- $g(n) =$ cost from root to node $n$
- $h(n) =$ estimated cost from $n$ to the nearest goal (heuristic)
- $f(n) = g(n) + h(n) =$ estimated total cost via $n$
- $G^*$: a lowest cost goal node
- $G$: another goal node
Optimality of A*: Blocking

Proof:

• What could go wrong?
• We’d have to have to pop a suboptimal goal \( G \) off the fringe before \( G^* \).

• This can’t happen if \( h \) admissible:
  – Imagine a suboptimal goal \( G \) is on the queue
  – Some node \( n \) which is a subpath of \( G^* \) must also be on the fringe (why?)
  – \( n \) will be popped before \( G \)

\[
\begin{align*}
f(n) &= g(n) + h(n) \\
g(n) + h(n) &\leq g(G^*) \quad h \text{ admissible} \\
g(G^*) &< g(G) \quad \text{by assumption} \\
g(G) &= f(G) \quad \text{for goals } h(G)=0 \\
f(n) &< f(G)
\end{align*}
\]
Properties of A*

- A* does not expand any node with $f(n) > C^*$ (Pruning).
  While UCS might expand nodes with $g(n) < C^*$ but $f(n) > C^*$.

- Optimally efficient, no other algorithm guarantees to expand nodes less than A*.
  (but not good choice for every search problem)

- Complete if costs exceeds positive epsilon and $b$ is finite

- Complexity $O(b^d)$ !!!!
UCS vs. A* Contours

- Uniform-cost expanded in all directions
  (Contours of UCS are cheapest $g$)

- A* expands mainly toward the goal, but does ensure optimality
  (Contours of A* are cheapest $f$.)
Creating Admissible Heuristics

• Most of the work in solving hard search problems optimally is in coming up with admissible heuristics

• Often, admissible heuristics are solutions to relaxed problems, where there are fewer restrictions on the actions (or new actions available)

• Inadmissible heuristics are often useful too (why?)
Example: 8 Puzzle

- What are the states?
- How many states?
- What are the actions?
- What states can I reach from the start state?
- What should the costs be?
8 Puzzle (I)

- Heuristic 1: Number of tiles misplaced

- Why is it admissible?

- $h(\text{start}) = 8$

- This is a relaxed-problem heuristic

<table>
<thead>
<tr>
<th>Average nodes expanded when optimal path has length...</th>
<th>...4 steps</th>
<th>...8 steps</th>
<th>...12 steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDS</td>
<td>112</td>
<td>6,300</td>
<td>3.6 x 10^6</td>
</tr>
<tr>
<td>TILES</td>
<td>13</td>
<td>39</td>
<td>227</td>
</tr>
</tbody>
</table>

Start State

Goal State
8 Puzzle (II)

- Heuristic 2: Sum of Manhattan distances of the tiles from their goal positions
- Why admissible?
- $h(\text{start}) = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$
- This is also a relaxed-problem heuristic

![Start State](image1.png) ![Goal State](image2.png)

<table>
<thead>
<tr>
<th>Average nodes expanded when optimal path has length...</th>
<th>4 steps</th>
<th>8 steps</th>
<th>12 steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>TILES</td>
<td>13</td>
<td>39</td>
<td>227</td>
</tr>
<tr>
<td>MANHATTAN</td>
<td>12</td>
<td>25</td>
<td>73</td>
</tr>
</tbody>
</table>
8 Puzzle (III)

• How about using the *actual cost* as a heuristic?
  – Would it be admissible?
  – Would we save on nodes expanded?
  – What’s wrong with it?

• With A*: a trade-off between quality of estimate and work per node!
Dominance

- **Dominance**: $h_a$ dominates $h_c$ if $\forall n : h_a(n) \geq h_c(n)$

- **Dominance $\rightarrow$ efficiency**
  A* using $h_a$ never expands more nodes than A* using $h_c$.

- **Heuristics form a semi-lattice:**
  - Max of admissible heuristics is admissible
    $h(n) = \max (h_a(n), h_b(n))$

- **Trivial heuristics**
  - Bottom of lattice is the zero heuristic
    (what does this give us?)
  - Top of lattice is the exact heuristic
Other A* Applications

• Pathing / routing problems
• Resource planning problems
• Robot motion planning
• Language analysis
• Machine translation
• Speech recognition
• …
Tree Search: Extra Work!

- Failure to detect repeated states can cause exponentially more work. Why?
Example

• In BFS, for example, we shouldn’t bother expanding the circled nodes (why?)
Graph Search

• **Idea:** never expand a state twice

• **How to implement:**
  – Tree search + set of expanded states ("closed-set")
  – Expand the search tree node-by-node, but…
  – Before expanding a node, check to make sure its state is new
    (neither in expanded set nor in frontier)
  – If not new, skip it

• **Important:** store the closed-set as a set, not a list

• Can graph search wreck completeness? Why/why not?

• How about optimality?
A* Graph Search Gone Wrong?

State space graph

Search tree

S (0+2)

A (1+4)

C (2+1)

G (5+0)

B (1+1)

C (3+1)

G (6+0)
Consistency of Heuristics

- Stronger than admissibility

- Definition:
  \[
  \text{cost}(A \text{ to } C) + h(C) \geq h(A) \\
  \text{cost}(A \text{ to } C) \geq h(A) - h(C) \\
  \text{real arc cost} \geq \text{cost implied by heuristic}
  \]

- Consequences:
  - The \( f \) value along a path never decreases
  - A* graph search is optimal
Optimality

• **Tree search:**
  – A* is optimal if heuristic is **admissible** (and non-negative)
  – UCS is a special case of A* (with $h = 0$)

• **Graph search:**
  – A* optimal if heuristic is **consistent**
  – UCS is optimal ($h = 0$ is consistent)

• **Consistency implies admissibility**

• In general, most natural admissible heuristics tend to be consistent, especially if from relaxed problems
Summary: A*

• A* uses both backward costs and (estimates of) forward costs

• A* is optimal with admissible / consistent heuristics

• Heuristic design is key: often use relaxed problems