Informed (Heuristic) Search

Idea: be **smart** about what paths to try.
Blind Search vs. Informed Search

• What’s the difference?

• How do we formally specify this?

A node is selected for expansion based on an evaluation function that estimates cost to goal.
General Tree Search Paradigm

function tree-search(root-node)
    fringe ← successors(root-node)
    while ( notempty(fringe) )
        {node ← remove-first(fringe)
         state ← state(node)
         if goal-test(state) return solution(node)
         fringe ← insert-all(successors(node), fringe) }
    return failure
end tree-search

How do we order the successor list?
Best-First Search

- Use an evaluation function $f(n)$ for node $n$.
- Always choose the node from fringe that has the lowest $f$ value.
Heuristics

• What is a heuristic?

• What are some examples of heuristics we use?

• We’ll call the heuristic function $h(n)$. 
Greedy Best-First Search

- $f(n) = h(n)$

- What does that mean?

- What is it ignoring?
Romanian Route Finding

• **Problem**
  – Initial State: Arad
  – Goal State: Bucharest
  – $c(s,a,s')$ is the length of the road from $s$ to $s'$

• **Heuristic function**: $h(s) = $ the straight line distance from $s$ to Bucharest
What’s the real shortest path from Arad to Bucharest? What’s the distance on that path?
Greedy Search in Romania

(a) The initial state
(b) After expanding Arad
(c) After expanding Sibiu
(d) After expanding Fagaras

Distance = 450
Greedy Best-First Search

- Is greedy search optimal?
- Is it complete?
- What is its worst-case complexity for a tree search with branching factor $b$ and maximum depth $m$?
  - time
  - space
Greedy Best-First Search

- When would we use greedy best-first search or greedy approaches in general?
A* Search

• Hart, Nilsson & Rafael 1968
  – Best-first search with \( f(n) = g(n) + h(n) \)
    where \( g(n) = \text{sum of edge costs from start to } n \)
    and \( h(n) = \text{estimate of lowest cost path } n --> \text{goal} \)
  – If \( h(n) \) is \textit{admissible} then search will find optimal solution.

\[ \begin{align*}
\text{Never overestimates the true cost of any solution which can be reached from a node.}
\end{align*} \]

Space bound since the queue must be maintained.
Back to Romania

Straight-line distance to Bucharest

- Arad: 366
- Bucharest: 0
- Craiova: 160
- Dobroța: 242
- Eforie: 161
- Făgăraș: 178
- Giurgiu: 77
- Hirsova: 151
- Iași: 226
- Lugoj: 244
- Mehadia: 241
- Neamț: 234
- Oradea: 380
- Pitești: 98
- Râmnicu Vâlcea: 193
- Sibiu: 253
- Timișoara: 329
- Urziceni: 80
- Vaslui: 199
- Zerind: 374
A* for Romanian Shortest Path
8 Puzzle Example

• \( f(n) = g(n) + h(n) \)

• What is the usual \( g(n) \)?

• two well-known \( h(n) \)’s
  – \( h_1 \) = the number of misplaced tiles
  – \( h_2 \) = the sum of the distances of the tiles from their goal positions, using city block distance, which is the sum of the horizontal and vertical distances (Manhattan Distance)
8 Puzzle Using Number of Misplaced Tiles

1 2 3
8 4
7 6 5

goal

2 8 3
1 6 4
7 5

g=0
h=4
f=4
Optimality of A* with Admissibility
(h never overestimates the cost to the goal)

Suppose a suboptimal goal $G_2$ has been generated and is in the queue. Let $n$ be an unexpanded node on the shortest path to an optimal goal $G_1$.

\[
f(n) = g(n) + h(n) \\
\leq g(G_1) \quad \text{Why?} \\
< g(G_2) \quad \text{G2 is suboptimal} \\
= f(G_2) \quad f(G_2) = g(G_2)
\]

So $f(n) < f(G_2)$ and A* will never select $G_2$ for expansion.
Optimality of A* with Consistency (stronger condition)

- $h(n)$ is consistent if
  - for every node $n$
  - for every successor $n'$ due to legal action $a$
  - $h(n) \leq c(n,a,n') + h(n')$

- Every consistent heuristic is also admissible.
Algorithms for A*

- Since Nilsson defined A* search, many different authors have suggested algorithms.

- Using Tree-Search, the optimality argument holds, but you search too many states.

- Using Graph-Search, it can break down, because an optimal path to a repeated state can be discarded if it is not the first one found.

- One way to solve the problem is that whenever you come to a repeated node, discard the longer path to it.
The Rich/Knight Implementation

- a node consists of
  - state
  - g, h, f values
  - list of successors
  - pointer to parent

- OPEN is the list of nodes that have been generated and had h applied, but not expanded and can be implemented as a priority queue.

- CLOSED is the list of nodes that have already been expanded.
1) /* Initialization */

OPEN <- start node

Initialize the start node
g:
h:
f:

CLOSED <- empty list
2) repeat until goal (or time limit or space limit)

- if OPEN is empty, fail
- BESTNODE <- node on OPEN with lowest f
- if BESTNODE is a goal, exit and succeed
- remove BESTNODE from OPEN and add it to CLOSED
- generate successors of BESTNODE
for each successor $s$ do
  1. set its parent field
  2. compute $g(s)$
  3. if there is a node OLD on OPEN with the same state info as $s$
     { add OLD to successors(BESTNODE)
       if $g(s) < g(\text{OLD})$, update OLD and throw out $s$ }
4. if (s is not on OPEN and there is a node OLD on CLOSED with the same state info as s

   { add OLD to successors(BESTNODE)
     if g(s) < g(OLD), update OLD,
     remove it from CLOSED
     and put it on OPEN, throw out s

   }
5. If $s$ was not on OPEN or CLOSED
   
   { add $s$ to OPEN
   
   add $s$ to successors(BESTNODE)
   
   calculate $g(s)$, $h(s)$, $f(s)$ }

end of repeat loop
The Heuristic Function $h$

- If $h$ is a **perfect estimator** of the true cost then A* will always pick the correct successor with no search.

- If $h$ is **admissible**, A* with TREE-SEARCH is guaranteed to give the optimal solution.

- If $h$ is **consistent**, too, then GRAPH-SEARCH is optimal.

- If $h$ is not admissible, no guarantees, but it can work well if $h$ is not often greater than the true cost.
Complexity of A*

- Time complexity is exponential in the length of the solution path unless for “true” distance $h^*$
  
  $|h(n) - h^*(n)| < \Theta(\log h^*(n))$

  which we can’t guarantee.

- But, this is AI, computers are fast, and a good heuristic helps a lot.

- Space complexity is also exponential, because it keeps all generated nodes in memory.

Big Theta notation says 2 functions have about the same growth rate.
Why not always use A*?

• Pros

• Cons
Solving the Memory Problem

• Iterative Deepening A*
• Recursive Best-First Search
• Depth-First Branch-and-Bound
• Simplified Memory-Bounded A*
Iterative-Deepening A*

• Like iterative-deepening depth-first, but...

• Depth bound modified to be an \textbf{f-limit}
  – Start with \( f\text{-limit} = h(\text{start}) \)
  – Prune any node if \( f(\text{node}) > f\text{-limit} \)
  – Next f-limit=\text{min-cost of any node pruned}
Recursive Best-First Search

• Use a variable called \textit{f-limit} to keep track of the best alternative path available from any ancestor of the current node

• If $f($current node$) > f$-limit, back up to try that alternative path

• As the recursion unwinds, replace the f-value of each node along the path with the \textit{backed-up value}: the best f-value of its children
Simplified Memory-Bounded A*

- Works like A* until memory is full
- When memory is full, drop the leaf node with the highest f-value (the worst leaf), keeping track of that worst value in the parent
- Complete if any solution is reachable
- Optimal if any optimal solution is reachable
- Otherwise, returns the best reachable solution
Performance of Heuristics

• How do we evaluate a heuristic function?
• effective branching factor $b^*$
  – If A* using $h$ finds a solution at depth $d$ using $N$ nodes, then the effective branching factor is
    $$b^* \text{ where } N = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$$
• Example:
  
  $d=2$
  $b=3$

  depth 0
  depth 1
  depth 2
### Table of Effective Branching Factors

<table>
<thead>
<tr>
<th>b</th>
<th>d</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>63</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>364</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>88573</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>43</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>9331</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>72,559,411</td>
</tr>
</tbody>
</table>

How might we use this idea to evaluate a heuristic?
Generate Admissible Heuristics from Relaxed Problems

• A relaxed problem has fewer constraints.

• Search graph is a superset of the one for the original problem. (more legal actions)

• The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem. (Why?)
Example from Text

A tile can move from square A to square B if A is horizontally or vertically adjacent to B and B is blank.

we can generate three relaxed problems by removing one or both of the conditions:

(a) A tile can move from square A to square B if A is adjacent to B.
(b) A tile can move from square A to square B if B is blank.
(c) A tile can move from square A to square B.

Initial

\[
\begin{array}{ccc}
2 & 8 & 3 \\
1 & 6 & 4 \\
7 & 5 & \\
\end{array}
\]

(a)

\[
\begin{array}{ccc}
2 & 8 & 3 \\
1 & 4 & \\
7 & 5 & 6 \\
\end{array}
\]

(b)

\[
\begin{array}{ccc}
2 & 3 \\
1 & 6 & 4 \\
7 & 5 & 8 \\
\end{array}
\]

(c)

\[
\begin{array}{ccc}
2 & 8 & 3 \\
4 & 6 & 1 \\
7 & 5 & \\
\end{array}
\]
Generate Admissible Heuristics from Subproblems

- A subproblem may be much easier to solve.

- There can be pattern databases for particular problems that store the exact costs for solutions to all subproblem instances (if they are small enough).

- The cost of solving a subproblem is not greater than the cost of solving the full problem.
Still may not succeed

- In spite of the use of heuristics and various smart search algorithms, not all problems can be solved.

- Some search spaces are just too big for a classical search.

- So we have to look at other kinds of tools.