Artificial Intelligence

Heuristic Search
(Ch. 3.5-6)
Best-first search

- Idea: use an evaluation function \( f(n) \) for each node
  - estimate of “promisingness”

\( \rightarrow \) Expand the most promising unexpanded node

- Implementation:
  Order the nodes in fringe in decreasing order ofpromisingness

- Special cases:
  - greedy best-first search
  - A* search
Greedy best-first search

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

- e.g., $h_{SLD}(n) =$ straight-line distance from $n$ to Bucharest

- Greedy best-first search expands the node that appears to be closest to $goal$
Romania with step costs in km

Straight-line distance to Bucharest

<table>
<thead>
<tr>
<th>City</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>
Greedy best-first search example
Greedy best-first search example
Greedy best-first search example
Greedy best-first search example
Properties of greedy best-first search

• Complete? \( \times \)
• Optimal? \( \times \)
• Time? \( \text{ expo} \)
• Space? \( \text{ expo} \)
A* search

• Idea: avoid expanding paths that are already expensive

• Evaluation function $f(n) = g(n) + h(n)$
  – $g(n) =$ cost so far to reach $n$
  – $h(n) =$ estimated cost from $n$ to goal
  – $f(n) =$ estimated total cost of path through $n$ to goal
A* search example

Arad
366 = 0 + 366
A* search example
A* search example
A* search example
A* search example
A* search 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**

• Example: $h_{SLD}(n)$ (never overestimates the actual road distance)
Optimality of A*

- **Theorem:** If $h(n)$ is admissible, A* guarantees to be optimal

\[ f(n) \leq f(n*) \]

Path $n*\rightarrow n$ is optimal.

\[ f(G) \geq f(n*) \]
**Optimality of A**

- **Lemma**: A* expands nodes in order of increasing $f$ value
- Gradually adds "$f$-contours" of nodes
- Contour $i$ has all nodes with $f=f_i$, where $f_i < f_{i+1}$
create the open list of nodes, initially containing only starting node
create the closed list of nodes, initially empty
while (we have not reached our goal) {
    consider the best node in the open list (the node with the lowest f value)
    if (this node is the goal) {
        then we're done;
    } else {
        move the current node to the closed list and consider all of its neighbors;
        for (each neighbor) {
            if (this neighbor is in the closed list and our current g value is lower) {
                move the node from closed list to open list;
                update the neighbor with the new, lower, g value;
                change the neighbor's parent to our current node;
            } else if (this neighbor is in the open list and our current g value is lower) {
                update the neighbor with the new, lower, g value;
                change the neighbor's parent to our current node;
            } else this neighbor is not in either the open or closed list {
                add the neighbor to the open list and set its g value;
            }
        }
    }
}
Properties of A*

- Complete? yes
- Optimal? yes
- Time? expo
- Space? expo
Consistent heuristics

- A heuristic is **consistent (monotonic)** if for every node $n$, every successor $n'$ of $n$ generated by any action $a$,

  $$h(n) \leq c(n, a, n') + h(n')$$

- If $h$ is consistent, $f(n) \leq f(n')$. 

Optimality of A*

• **Theorem:** If $h(n)$ is consistent, A* guarantees to find an optimal path without reexpanding nodes.
Example

Graph:
- Nodes: S, A, B, C, D, E, F, G
- Edges:
  - S → D: 5
  - A → B: 1
  - B → E: 4
  - C → F: 3
  - S → G: 11.0
  - A → G: 10.4
  - B → G: 6.7
  - C → G: 4.0
  - D → G: 8.9
  - E → G: 6.9
  - F → G: 3.0
A* Algorithm in action

Diagram of a graph with nodes labeled S, A, B, C, D, E, F, G, and edges connecting them with weights.

Weights: 2, 5, 2, 4, 5, 4, 3, 11.0, 8.9, 6.7, 4.0, 6.9, 3.0.
Depth-first Branch and Bound Search

- Although A* guarantees to generate no more nodes than any other optimal algorithm, its space requirement can be prohibitive for large search problems.
- Depth-first search algorithm can be enhanced to utilize heuristic:
  - So called Depth-first Branch and Bound Search
  - Uses heuristic functions to bound solution quality, and only expand nodes that can still lead to solutions better than the incumbent.
Depth-first Branch and Bound Search

Initialize:
Let open list \( Q = \{ S \} \)
Let closed list \( C = \{ \} \)
\( L \leftarrow \infty \) // score of best solution so far
While \( Q \) is not empty
    pull \( Q_1 \), the first element in \( Q \)
    if \( Q_1 \) is a goal compute the cost of the solution and update
        \( L \leftarrow \) minimum between new cost and old cost
    else
        child_nodes = expand(\( Q_1 \)),
        For each child node \( n \) do:
            Eliminate child_nodes which represent simple loops by checking against the closed list,
            evaluate \( f(n) \).
            If \( f(n) \) is greater than \( L \), discard \( n \).
        end-for
        Put remaining child_nodes on top of queue in the order of their evaluation function, \( f \).
    end
Continue
Properties of Branch-and-Bound

- **Complete?** yes
- **Optimal?** yes
- **Time?** linear
Iterative Deepening A* (IDA*)

- Extend iteratively deepening search by
  - Using depth-first branch and bound
  - Updating the search limit according heuristic evaluations

- Properties:
  Guarantee to find an optimal solution
  time: exponential, like A*
  space: linear, like B&B.
Iterative Deepening A* (IDA*)

- **Pseudocode:**
  
  Initialize: $f \leftarrow$ the evaluation function of the start node
  until goal node is found
  
  Loop:
  
  Do Branch-and-bound with upper-bound $L$ equal current evaluation function
  
  Increment evaluation function to next contour level
  
  (Update evaluation function to the minimum $f$-value which exceeded $f$ among states which were generated)
  
  end
  
  continue

- 2nd: $L_2 = \min \{ f(0), f(10), f(13), f(40) \}$
Relationships among search algorithms

- Depth first (LIFO ordering)
- $\hat{f} =$ depth (Breadth first)
- $\hat{h} = 0$ (Uniform cost)
- $\hat{h} \leq h$
- A* ($\hat{f} = \hat{g} + \hat{h}$) (Best-first search)
- (Generic graph-search algorithms)
Admissible heuristics

E.g., for the 8-puzzle:

- \( h_1(n) = \)
- \( h_2(n) = \)

\[ h_1(S) = 8 \]
\[ h_2(S) = 3 + 1 + 2 + 2 + 3 + 2 + 2 + 2 \]
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

- Typical search costs (average number of nodes expanded):

  - $d=12$
    - 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
Combination of Multiple Heuristics

- Given any admissible heuristics $h_a, h_b$,
  \[ h(n) = \max(h_a(n); h_b(n)) \]
  is also admissible and dominates $h_a, h_b$
Relaxed problems

• A problem with fewer restrictions on the actions is called a relaxed problem.
• The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem.
• If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution.
• If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution.