Informed Search II

Outline for today’s lecture

Informed Search
• Optimal informed search: A* (AIMA 3.5.2)
• Creating good heuristic functions
• Hill Climbing

Review: Greedy best-first search:
• \(f(n)\): estimated total cost of path through \(n\) to goal
• \(g(n)\): the cost to get to \(n\), UCS: \(f(n) = g(n)\)
• \(h(n)\): heuristic function that estimates the remaining distance from \(n\) to goal
• Greedy best-first search idea: \(f(n) = h(n)\)
  • Expand the node that is estimated by \(h(n)\) to be closest to goal
  • Here, \(h(n) = h_{SLD}(n)\) = straight-line distance from \(n\) to Bucharest

Review: Properties of greedy best-first search
• Complete? No – can get stuck in loops,
  • e.g., Iasi \(\rightarrow\) Neamt \(\rightarrow\) Iasi \(\rightarrow\) Neamt \(\rightarrow\) ...
• Time? \(O(b^m)\) – worst case (like Depth First Search)
  • But a good heuristic can give dramatic improvement of average cost
• Space? \(O(b^m)\) – priority queue, so worst case: keeps all (unexpanded) nodes in memory
  • Optimal? No

Review: Optimal informed search: A*
• Best-known informed search method
• Key Idea: avoid expanding paths that are already expensive, but expand most promising first.
  • Simple idea: \(f(n) = g(n) + h(n)\)
• Implementation: Same as before
  • Frontier queue as priority queue by increasing \(f(n)\)

Review: Admissible heuristics
• A heuristic \(h(n)\) is admissible if it never overestimates the cost to reach the goal; i.e. it is optimistic
  • Formally: \(\forall n, h(n) \leq h^*(n)\) where \(h^*(n)\) is the true cost from \(n\)
  1. \(h(n) \geq g(n)\) so \(h(G) = 0\) for any goal \(G\).
• Example: \(h_{SLD}(n)\) never overestimates the actual road distance

Theorem: If \(h(n)\) is admissible, A* using Tree Search is optimal
A* search example

Frontier queue:
Arad 366

We add the three nodes we found to the Frontier queue. We sort them according to the $g() + h()$ calculation.

Frontier queue:
Sibiu 393
Timisoara 447
Zerind 449

When we expand Sibiu, we run into Arad again. Note that we've already expanded this node once; but we still add it to the Frontier queue again.

When we expand Fagaras, we find Bucharest, but we're not done. The algorithm doesn't end until we "expand" the goal node -- it has to be at the top of the Frontier queue.

When we expand Fagaras, we find Bucharest, but we're not done. The algorithm doesn't end until we "expand" the goal node -- it has to be at the top of the Frontier queue.

We expand Rimincu Visea.

Note that we just found a better value for Bucharest! Now we expand this better value for Bucharest since it's at the top of the queue.

We're done and we know the value found is optimal!
Optimality of $A^*$ (intuitive)

- Lemma: $A^*$ expands nodes on frontier in order of increasing $f$-value
- Gradually adds "$f$-contours" of nodes
- Contour $i$ has all nodes with $f = f_i$, where $f_i < f_{i+1}$
- (After all, $A^*$ is just a variant of uniform-cost search...)

$A^*$ search, evaluation

- Completeness: YES
  - Since bands of increasing $f$ are added
  - As long as $b$ is finite
    - (guaranteeing that there aren't infinitely many nodes $n$ with $f(n) < f(G)$)
- Time complexity: Same as UCS worst case
  - Number of nodes expanded is still exponential in the length of the solution.
- Space complexity: Same as UCS worst case
  - It keeps all generated nodes in memory so exponential
    - Hence space is the major problem not time
- Optimality: YES
  - Cannot expand $f_{i+1}$ until $f_i$ is finished.
  - $A^*$ expands all nodes with $f(n) < f(G)$
  - $A^*$ expands one node with $f(n) = f(G)$
  - $A^*$ expands no nodes with $f(n) > f(G)$

Consistency

- A heuristic is consistent if
  \[ h(n) \leq c(n, a, n') + h(n') \]
- Consistency enforces that $h(n)$ is optimistic

Theorem: if $h(n)$ is consistent, $A^*$ using Graph-Search is optimal

See book for details

Outline for today's lecture

**Informed Search**

- Optimal informed search: $A^*$
- Creating good heuristic functions (AIMA 3.6)
- Hill Climbing

Heuristic functions

- For the 8-puzzle
  - Avg. solution cost is about 22 steps
    - (branching factor $\leq 3$)
  - Exhaustive search to depth 22: $3.1 \times 10^{12}$ states
  - A good heuristic function can reduce the search process
Example Admissible heuristics

For the 8-puzzle:

- $h_{\text{loop}}(n) =$ number of out of place tiles
- $h_{\text{md}}(n) =$ total Manhattan distance (i.e., # of moves from desired location of each tile)

- $h_{\text{loop}}(S) = 8$
- $h_{\text{md}}(S) = 3+1+2+2+2+3+3+2 = 18$

Relaxed problems

- A problem with fewer restrictions on the actions than the original 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 $h_{\text{loop}}(n)$ gives the shortest solution.
- If the rules are relaxed so that a tile can move to any adjacent square, then $h_{\text{md}}(n)$ gives the shortest solution.

Defining Heuristics: $h(n)$

- Cost of an exact solution to a relaxed problem (fewer restrictions on operator)

  - Constraints on Full Problem:
    A tile can move from square A to square B if A is adjacent to B and B is blank.
    - A tile can move from square A to square B if A is adjacent to B. ($h_{\text{loop}}$)
    - A tile can move from square A to square B if B is blank. ($h_{\text{md}}$)
    - A tile can move from square A to square B. ($h_{\text{teleport}}$)

Dominance: A metric on better heuristics

- If $h_2(n) \geq h_1(n)$ for all n (both admissible)
  - then $h_2$ dominates $h_1$
  - So $h_2$ is optimistic, but more accurate than $h_1$
    - $h_2$ is therefore better for search
    - Notice: $h_{\text{md}}$ dominates $h_{\text{loop}}$

Typical search costs (average number of nodes expanded):

  - $d=12$ Iterative Deepening Search = 3,644,035 nodes
  - $A^*(h_{\text{loop}}) = 227$ nodes
  - $A^*(h_{\text{md}}) = 73$ nodes
  - $d=24$ IDS = too many nodes
  - $A^*(h_{\text{loop}}) = 38,135$ nodes
  - $A^*(h_{\text{md}}) = 1,641$ nodes

The best and worst admissible heuristics

- $h^*(n)$ - the (unachievable) Oracle heuristic
  - $h^*(n)$ = the true distance from the root to n
- $h_{\text{we're here already}}(n) = h_{\text{teleport}}(n)=0$

Admissible: both yes!!!
- $h^*(n)$ dominates all other heuristics
- $h_{\text{teleport}}(n)$ is dominated by all heuristics

Iterative Deepening $A^*$ and beyond

Beyond our scope:

- Iterative Deepening $A^*$
- Recursive best first search (incorporates $A^*$ idea, despite name)
- Memory Bounded $A^*$
- Simplified Memory Bounded $A^*$ - R&N say the best algorithm to use in practice, but not described here at all.
  - (If interested, follow reference to Russell article on Wikipedia article for SMA*)

(see 3.5.3 if you're interested in these topics)
Outline for today's lecture

**Informed Search**
- Optimal informed search: A*
- Creating good heuristic functions
- When informed search doesn’t work: hill climbing (AIMA 4.1.1)
  - A few slides adapted from CS 471, UBMC and Eric Eaton (in turn, adapted from slides by Charles R. Dyer, University of Wisconsin-Madison)

Local search and optimization
- Local search:
  - Use single current state and move to neighboring states.
  - Idea: start with an initial guess at a solution and incrementally improve it until it is one
- Advantages:
  - Use very little memory
  - Find often reasonable solutions in large or infinite state spaces.
- Useful for pure optimization problems.
  - Find or approximate best state according to some objective function
  - Optimal if the space to be searched is convex

Hill climbing on a surface of states
- h(s): Estimate of distance from a peak (smaller is better)
- OR: Height Defined by Evaluation Function (greater is better)

Hill-climbing search
1. While (3 uphill points):
   - Move in the direction of increasing evaluation function f
2. Let \( s_{\text{next}} = \arg \max f(s) \) if a successor state to the current state
   - If \( f(s) < f(n) \) then move to \( s \)
   - Otherwise halt at \( n \)
- Properties:
  - Terminates when a peak is reached.
  - Does not look ahead of the immediate neighbors of the current state.
  - Chooses randomly among the set of best successors, if there is more than one.
  - Doesn't backtrack, since it doesn't remember where it's been
- a.k.a. greedy local search

"Like climbing Everest in thick fog with amnesia"

Hill climbing example I (minimizing h)

<table>
<thead>
<tr>
<th></th>
<th>h(_{\text{loop}}) = 5</th>
<th></th>
<th>h(_{\text{loop}}) = 0</th>
<th></th>
<th>h(_{\text{loop}}) = 1</th>
<th></th>
<th>h(_{\text{loop}}) = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>3 1 2</td>
<td></td>
<td>3 1 2</td>
<td></td>
<td>3 1 2</td>
<td></td>
<td>3 1 2</td>
</tr>
<tr>
<td></td>
<td>4 5 8</td>
<td></td>
<td>4 5 8</td>
<td></td>
<td>4 5 8</td>
<td></td>
<td>4 5 8</td>
</tr>
<tr>
<td></td>
<td>6 7</td>
<td></td>
<td>6 7</td>
<td></td>
<td>6 7</td>
<td></td>
<td>6 7</td>
</tr>
<tr>
<td>6</td>
<td>h = 4.5</td>
<td>h(_{\text{loop}}) = 4.5</td>
<td>h = 4</td>
<td>h(_{\text{loop}}) = 4</td>
<td>h = 4</td>
<td>h(_{\text{loop}}) = 3</td>
<td></td>
</tr>
<tr>
<td>3 1 2</td>
<td>4 5 8</td>
<td>3 1 2</td>
<td>4 5 8</td>
<td>3 1 2</td>
<td>4 5 8</td>
<td>3 1 2</td>
<td>4 5 8</td>
</tr>
<tr>
<td>4 5 8</td>
<td>6 7</td>
<td>4 5 8</td>
<td>6 7</td>
<td>4 5 8</td>
<td>6 7</td>
<td>4 5 8</td>
<td>6 7</td>
</tr>
<tr>
<td>5</td>
<td>h(_{\text{loop}}) = 3</td>
<td></td>
<td>3 1 2</td>
<td></td>
<td>6 7 8</td>
<td></td>
<td>6 7 8</td>
</tr>
<tr>
<td>3 1 2</td>
<td>4 5</td>
<td>3 1 2</td>
<td>4 5</td>
<td>3 1 2</td>
<td>4 5</td>
<td>3 1 2</td>
<td>4 5</td>
</tr>
<tr>
<td>4 5 8</td>
<td>6 7 8</td>
<td>4 5 8</td>
<td>6 7 8</td>
<td>4 5 8</td>
<td>6 7 8</td>
<td>4 5 8</td>
<td>6 7 8</td>
</tr>
<tr>
<td>goal</td>
<td>1 2</td>
<td>1 2</td>
<td>3 1 2</td>
<td>3 1 2</td>
<td>1 2</td>
<td>1 2</td>
<td>3 1 2</td>
</tr>
<tr>
<td></td>
<td>3 4 5</td>
<td>3 4 5</td>
<td>4 5 8</td>
<td>4 5 8</td>
<td>3 4 5</td>
<td>3 4 5</td>
<td>4 5 8</td>
</tr>
<tr>
<td></td>
<td>6 7 8</td>
<td>6 7 8</td>
<td>6 7 8</td>
<td>6 7 8</td>
<td>6 7 8</td>
<td>6 7 8</td>
<td>6 7 8</td>
</tr>
</tbody>
</table>

Hill-climbing Example: n-queens
- n-queens problem: Put n queens on an \( n \times n \) board with no two queens on the same row, column, or diagonal
- Good heuristic: \( h = \) number of pairs of queens that are attacking each other

\( h = 5 \) (for illustration)
Hill-climbing example: 8-queens

A state with $h=17$ and the $h$-value for each possible successor

A local minimum of $h$ in the 8-queens state space ($h=1$).

$h =$ number of pairs of queens that are attacking each other

Search Space features

Drawbacks of hill climbing

- Local Maxima: peaks that aren't the highest point in the space
- Plateaus: the space has a broad flat region that gives the search algorithm no direction (random walk)
- Ridges: dropoffs to the sides; steps to the North, East, South and West may go down, but a step to the NW may go up.

Toy Example of a local "maximum"

The Shape of an Easy Problem (Convex)

Gradient ascent/descent

- Gradient descent procedure for finding the $\arg\min_x f(x)$
  - choose initial $x_0$ randomly
  - repeat
    - $x_{n+1} \leftarrow x_n - \eta \cdot f'(x_n)$
  - until the sequence $x_0, x_1, \ldots, x_n$ converges
- Step size $\eta$ (eta) is small (perhaps 0.1 or 0.05)
A reminder of Newton’s method from Calculus:

\[ x_{i+1} = x_i - \frac{\eta f'(x_i)}{f''(x_i)} \]

• Newton’s method uses 2nd order information (the second derivative, or, curvature) to take a more direct route to the minimum.

• The second-order information is more expensive to compute, but converges quicker.

Contour lines of a function
Gradient descent (green)
Newton’s method (red)

(this and previous slide from Eric Eaton)

Gradient methods vs. Newton’s method

The Shape of a Harder Problem

The Shape of a Yet Harder Problem

One Remedy to Drawbacks of Hill Climbing: Random Restart

In the end: Some problem spaces are great for hill climbing and others are terrible.

Fantana muzicala Arad - Musical fountain Arad, Romania