Informed Search

Outline for today’s lecture

Informed Search
- 1st attempt: Greedy Best-first search (AIMA 3.5.1)
- Optimal informed search: A*
- Creating good heuristic functions
- Beyond A*

(When Informed Search doesn’t work…)

Review: Best-first search

Basic idea:
- select node for expansion with minimal evaluation function \( f(n) \)
  - where \( f(n) \) is some function that includes estimate heuristic \( h(n) \) of the remaining distance to goal
- Implement using priority queue
- Exactly UCS with \( f(n) \) replacing \( g(n) \)

Greedy best-first search:
- Simplest informed search: \( f(n) = h(n) \)
- Idea: Expand the node that is estimated by \( h(n) \) to be closest to goal
- Completely ignores \( g(n) \): the cost to get to \( n \)
- Here, \( h(n) = h_{SLD}(n) = \) straight-line distance from ` \( \) to Bucharest

Greedy best-first search example

- Initial State = Arad
- Goal State = Bucharest

Greedy best-first search example

Frontier queue:
Arad 366
Sibiu 253
Timisoara 329
Zerind 374

Frontier queue:
Arad 366
Bucharest 0
Crakow 152
Dubrovnik 243
Elnor 151
Fagras 156
Ghent 77
Hiroseva 151
Iasi 226
Lagaj 244
Mehabia 241
Naneti 234
Oradea 380
Pleurol 109
Riminos Venice 197
Sibiu 253
Timozuran 329
Ucrzini 84
Vadui 199
Zerzil 374
Greedy best-first search example

Frontier queue:
- Fagaras 176
- Rimnicu Vilcea 193
- Timisoara 329
- Arad 365
- Zerind 374
- Oradea 380

Greedy best-first search example

Frontier queue:
- Bucharest 0
- Rimnicu Vilcea 193
- Sibiu 253
- Timisoara 329
- Arad 365
- Zerind 374
- Oradea 380

Properties of greedy best-first search

- **Optimal?**
  - No!
    - Found: Arad → Sibiu → Fagaras → Bucharest (450km)
    - Shorter: Arad → Sibiu → Rimnicu Vilcea → Ploiesti → Bucharest (418km)

Outline for today’s lecture

**Informed Search**

- 1st attempt: Greedy Best-first search
- **Optimal informed search: A**ór (AIMA 3.5.2)
- Creating good heuristic functions
- Beyond A

(When Informed Search doesn’t work...)
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: \( f(n) = g(n) + h(n) \)
  - \( g(n) \) the cost (so far) to reach the node
  - \( h(n) \) estimated cost to get from the node to the goal
  - \( f(n) \) estimated total cost of path through \( n \) to goal
- Implementation: Frontier queue as priority queue by increasing \( f(n) \) (as expected…)

Admissible heuristics

- A heuristic \( h(n) \) is admissible if it never overestimates the cost to reach the goal; i.e. it is optimistic
  - Formally: \( \forall n \), \( n \) a node:
    1. \( h(n) \leq h^*(n) \) where \( h^*(n) \) is the true cost from \( n \)
    2. \( h(n) \geq 0 \) 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:

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

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.

We expand Rimnicu Visea.
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.

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 / has all nodes with \( f(n) \) where \( f(n) \leq f(n') \)
- (After all, A* is just a variant of uniform-cost search...)

Proof:
1. \( g(G_2) > g(G) \) since \( G_2 \) is suboptimal
2. \( f(G_2) = g(G_2) \) since \( f(G) = g(n) + h(n) \) and \( h(n) = 0 \), since \( G_2 \) is a goal
3. \( f(G) = g(G) \) similarly
4. \( f(G_2) > f(G) \) from 1,2,3

Also must show that \( G \) is added to the frontier before \( G_2 \) is expanded – see AIMA for argument in the case of Graph 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 \( f(n) < f(G) \))
A* search, evaluation

- Completeness: YES
- Time complexity: (exponential with path length)
- Space complexity:
  - It keeps all generated nodes in memory
  - Hence space is the major problem not time

Proof of Lemma: Consistency

- A heuristic is consistent if
  \[ h(n) \leq c(n, a, n') + h(n') \]
- Lemma: If \( h \) is consistent,
  \[ f(n') = g(n') + h(n') \]
  \[ = g(n) + c(n, a, n') + h(n') \]
  \[ \geq g(n) + h(n) = f(n) \]

i.e. \( f(n) \) is nondecreasing along any path.

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

Outline for today’s lecture

Informed Search

- 1st attempt: Greedy Best-first search
- Optimal informed search: A*
- Creating good heuristic functions (AIMA 3.6)
- Beyond A*

(When Informed Search doesn’t work…)

Heuristic functions

Creating Good Heuristic Functions

AIMA 3.6

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

E.g., for the 8-puzzle:

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

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 \( h_{\text{oop}}(n) \) 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

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: \( h_2 \) dominates \( h_{\text{oop}} \)
- Typical search costs (average number of nodes expanded):
  - \( d=12 \):
    - Iterative Deepening Search = 3,644,035 nodes
    - \( A^*(h_{\text{oop}}) = 227 \) nodes
    - \( A^*(h_{\text{md}}) = 73 \) nodes
  - \( d=24 \):
    - IDS = too many nodes
    - \( A^*(h_{\text{oop}}) = 39,135 \) nodes
    - \( A^*(h_{\text{md}}) = 1,641 \) nodes

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*)

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.
  - Constraints on relaxed problems:
    - A tile can move from square A to square B if A is adjacent to B. (\( h_{\text{oop}} \))
    - 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. (\( h_{\text{md}} \))

(see 3.5.3 if you're interested in these topics)
When A* doesn’t work
AIMA 4.1

A few slides adapted from CS 471, UBMC and Eric Eaton (in turn, adapted from slides by Charles R. Dyer, University of Wisconsin-Madison).

Outline
• Local Search: Hill Climbing
• Escaping Local Maxima: Simulated Annealing
• Genetic Algorithms

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

CIS 521 - Intro to AI - Spring 2016

Hill Climbing

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
I. While (g hill points):
   • Move in the direction of increasing evaluation function f
II. Let $s_{next} = \arg \max_s f(s)$, s a successor state to the current state n
   • If $f(n) < f(s)$ 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>Start</th>
<th>$h_{\text{sup}}$ = 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 1 2</td>
<td>1 2</td>
</tr>
<tr>
<td>4 5 8</td>
<td>3 4 5</td>
</tr>
<tr>
<td>6 7</td>
<td>6 7 8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Goal</th>
<th>$h_{\text{sup}}$ = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3</td>
<td>1 2</td>
</tr>
<tr>
<td>4 5 6</td>
<td>3 4 5</td>
</tr>
<tr>
<td>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: $h = \text{number of pairs of queens that are attacking each other}$

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

Hill-climbing example: 8-queens

<table>
<thead>
<tr>
<th>Start</th>
<th>$h_{\text{sup}}$ = 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 1 2</td>
<td>3 1 2</td>
</tr>
<tr>
<td>4 5 8</td>
<td>4 5 8</td>
</tr>
<tr>
<td>6 7</td>
<td>6 7 8</td>
</tr>
</tbody>
</table>

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

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 $\text{arg min } f(x)$
  - choose initial $x_0$ randomly
  - repeat
  - $x_{i+1} \leftarrow x_i - \eta f'(x_i)$
  - until the sequence $x_0, x_1, \ldots, x_i, x_{i+1}$ converges
- Step size $\eta$ (eta) is small (perhaps 0.1 or 0.05)

Gradient methods vs. Newton's method

- A reminder of Newton's method from Calculus:
  - $x_{i+1} \leftarrow x_i - \frac{f'(x_i)}{f''(x_i)}$
- 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.

The Shape of a 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.