Intuitions behind heuristic search

The separation property of GRAPH-SEARCH

Black Nodes - Explored
White Nodes - Frontier
Grey Nodes - Unexplored

Find a heuristic measure $h(n)$ which estimates how close a node $n$ in the frontier is to the nearest goal state and then order the frontier queue accordingly relative to closeness.

Introduce an evaluation function on nodes $f(n)$ which is a cost estimate. $f(n)$ will order the frontier by least cost.

$$f(n) = \ldots + h(n)$$

$h(n)$ will be part of $f(n)$

Recall Uniform-Cost Search

$$g(n) = \text{cost of path from root node to } n$$
$$f(n) = g(n)$$

Function $\text{UNIFORM-COST-SEARCH}(\text{problem})$ returns a solution, or failure

node — a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
frontier — a priority queue ordered by PATH-COST, with node as the only element explored — an empty set

loop do
if EMPTY?(frontier) then return failure
node = POP(frontier) /* chooses the lowest-cost node in frontier */
if problem.GOAL-TEST(node, STATE) then return SOLUTION(node)
add node, STATE to explored
don eact action in problem.ACTIONS(node, STATE) do
child — CHILD-NODE(node, problem, node, action)
if child.STATE is not in explored or frontier then
frontier — INSERT(child, frontier)
else if child.STATE is in frontier with higher PATH-COST then
replace that frontier node with child
loop do

Best-First Search

Function $\text{BEST-FIRST-SEARCH}(\text{problem})$ returns a solution, or failure

node — a node with STATE = problem.INITIAL-STATE, frontier — a priority queue ordered by $f(n)$, with node as the only element explored — an empty set

loop do
if EMPTY?(frontier) then return failure
node = POP(frontier) /* chooses the lowest-cost node in frontier */
if problem.GOAL-TEST(node, STATE) then return SOLUTION(node)
add node, STATE to explored
don eact action in problem.ACTIONS(node, STATE) do
child — CHILD-NODE(problem, node, action)
if child.STATE is not in explored or frontier then
frontier — INSERT(child, frontier)
else if child.STATE is in frontier with higher $f(n)$ then
replace that frontier node with child

$$f(n) = \ldots + h(n)$$

Most best-first search algorithms include $h(n)$ as part of $f(n)$

$h(n)$ is a heuristic function

Estimated cost of the cheapest path through state $n$ to a goal state
Greedy Best-First Search

Don't care about anything except how close a node is to a goal state

\[ f(n) = h(n) \]

Let's find a heuristic for the Romania Travel Problem:

Romania Travel Problem Heuristic

- Straight line distance from city \( n \) to goal city \( n' \)

Assume the cost to get somewhere is a function of the distance traveled

\[ h_{\text{SLD}}(n) \text{ for Bucharest} \]

\[ f(n) = h_{\text{SLD}}(n) \]
In this particular example, we rapidly reached the goal directly following a single path, but...

Is Greedy Best-First Search Optimal?

No, the actual costs:
Path Chosen: Arad-Sibiu-Fagaras-Bucharest = 450
Optimal Path: Arad-Sibiu-Rimnicu Vilcea-Pitești-Bucharest = 418

The search cost is minimal but not optimal!

What's missing?

Is Greedy Best-First Search Complete?

• GBF Graph search is complete in finite spaces but not in infinite spaces
• GBF Tree search is not even complete in finite spaces.

Consider going from Iasi to Fagaras?

Neamt is chosen 1st because \( h(\text{Neamt}) \) is closer than \( h(\text{Vaslui}) \), but Neamt is a deadend. Expanding Neamt still puts Iasi 1st on the frontier again since \( h(\text{Iasi}) \) is closer than \( h(\text{Vaslui}) \)...which puts Neamt 1st again!

Worst case time and space complexity for GBF tree search is \( O(b^m) \)

BUT

With heuristics performance is often much better with good choice of heuristic
Improving Greedy Best-First Search

Best-First Search finds a goal as fast as possible by using the \( h(n) \) function to estimate \( n \)'s closeness to the goal. Best-First Search chooses any goal node without concerning itself with the shallowness of the goal node or the cost of getting to \( n \) in the 1st place.

Rather than choosing a node based just on distance to the goal we could include a quality notion such as expected depth of the nearest goal.

- \( g(n) \) - the actual cost of getting to node \( n \)
- \( h(n) \) - the estimated cost of getting from \( n \) to a goal state

\[
f(n) = g(n) + h(n)
\]

\( f(n) \) is the estimated cost of the cheapest solution through \( n \)

A* Search

\[
f(n) = g(n) + h(n)
\]

A*-1

(a) The initial state

Heuristic:

\[
f(n) = g(n) + h(n)
\]

\( g(n) \) - Actual distance from root node to \( n \)

\( h(n) \) - \( h_{SLD}(n) \) straight line distance from \( n \) 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>Drobeta</td>
<td>242</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
</tr>
<tr>
<td>Fagaras</td>
<td>176</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
</tr>
<tr>
<td>Hlinsora</td>
<td>151</td>
</tr>
<tr>
<td>Iasi</td>
<td>226</td>
</tr>
<tr>
<td>Lugoj</td>
<td>244</td>
</tr>
<tr>
<td>Mahadia</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>100</td>
</tr>
<tr>
<td>Rimnicu Vilcea</td>
<td>193</td>
</tr>
<tr>
<td>Simia</td>
<td>253</td>
</tr>
<tr>
<td>Timisoara</td>
<td>329</td>
</tr>
<tr>
<td>Uruicenii</td>
<td>80</td>
</tr>
<tr>
<td>Vaslui</td>
<td>199</td>
</tr>
<tr>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>

A*-2

(b) After expanding Arad

TDDC17_Fo3 - 3 Sep 2014
A* Proof of Optimality for Tree Search

A* using TREE-SEARCH is optimal if h(n) is admissible

Proof:
Assume the cost of the optimal solution is C*.
Suppose a suboptimal goal node G2 appears on the fringe.
Since G2 is suboptimal and h(G2)=0 (G2 is a goal node),
f(G2) = g(G2) + h(G2) = g(G2) > C*
Now consider the fringe node n that is on an optimal solution path. If h(n) does not over-estimate the cost of completing the solution path then f(n) = g(n) + h(n) < or = C*
Then f(n) < or = C* < f(G2), so G2 will not be expanded and A* is optimal!

See example:
n = Pitești (417)
G2 = Bucharest (450)

A* Proof of Optimality for Graph Search

A* using GRAPH-SEARCH is optimal if h(n) consistent (monotonic)

h(n) is consistent if h(n) < or = c(n,a, succ(n)) + h(succ(n)) for all a, n, succ(n)

Step Cost

Step cost: c(n,a, succ(n))

successors(n):

n1 .... .... nk

Gn

Goal node closest to n

Triangle inequality argument:
Length of a side of a triangle is always less than the sum of the other two.
Otherwise it would violate the property that h(n) is a lower bound on the cost to reach

Optimality of graph search
Steps to show in the proof:
• If h(n) is consistent, then the values f(n) along any path are non-decreasing
• Whenever A* selects a node n for expansion, the optimal path to that node has been found

If this is the case, then the values along any path are non-decreasing and A* fans out in concentric bands of increasing f-cost

Some Properties of A*
• **Optimal** - for a given admissible heuristic (every consistent heuristic is an admissible heuristic)
• **Complete** - Eventually reach a contour equal to the path of the cost to the goal state.
• **Optimally Efficient** - No other algorithm, that extends search paths from a root is guaranteed to expand fewer nodes than A* for a given heuristic function.
• The exponential growth for most practical heuristics will eventually overtake the computer (run out of memory)
• The number of states within the goal contour is still exponential in the length of the solution.
• There are variations of A* that bound memory....
Admissible Heuristics

$h(n)$ is an admissible heuristic if it never over-estimates the cost to reach the goal from $n$.

Admissible Heuristics are optimistic because they always think the cost of solving a problem is less than it actually is.

The 8 Puzzle

How would we choose an admissible heuristic for this problem?

8 Puzzle Heuristics

$h_1(n)$: The number of pieces that are out of place.

Any tile that is out of place must be moved at least once. Definite underestimate of moves!

$h_2(n)$: The sum of the manhattan distances for each tile that is out of place.

The manhattan distance is an underestimate because there are tiles in the way.

Some Relaxations

Sample rule:

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

1. A tile can move from square A to square B if A is adjacent to B
2. A tile can move from square A to square B if B is blank
3. A tile can move from square A to square B

(1) gives us manhattan distance
Beyond Classical Search
Chapter 4

Local Search: 8 Queens Problem

Problem:
Place 8 queens on a chessboard such that No queen attacks any other.

Note:
- The path to the goal is irrelevant!
- Complete state formulation is a straightforward representation: 8 queens, one in each column

Candidate for use of local search!

Local Search Techniques

Definition: Two prisoners are questioned separately about a crime they committed. Each may give evidence against the other or may say nothing. If both say nothing, they get a minor reprimand and go free because of lack of evidence. If one gives evidence and the other says nothing, the first goes free and the second is severely punished. If both give evidence, both are severely punished. The overall (globally) best strategy is for both to say nothing. However not knowing (or trusting) what the other will do, each prisoner’s (locally) best strategy is to give evidence, which is the worst possible outcome.

Global Optimum: The best possible solution to a problem.
Local Optimum: A solution to a problem that is better than all other solutions that are slightly different, but worse than the global optimum

Greedy Algorithm: An algorithm that always takes the best immediate, or local, solution while finding an answer. Greedy algorithms find the overall, or globally, optimal solution for some optimization problems, but may find less-than-optimal solutions for some instances of other problems. (They may also get stuck!)

Hill-Climbing Algorithm
(steepest ascent version)

function HILL-CLIMBING(problem) returns a state that is a local maximum
    current ← MAKE-NODE(problem INITIAL-STATE)
    loop do
        neighbor ← a highest-valued successor of current
        if neighbor.VALUE ≤ current.VALUE then return current.STATE
        current ← neighbor
Greedy Progress: Hill Climbing

Aim: Find the Global Maximum

Hill Climbing: Modify the current state to try and improve it

Successor State Example

Current state: $h=17$

The value of $h$ is shown for each possible successor. The 12's are the best choices for the local move. (Use steepest descent) Choose randomly on ties.

Local minimum: $h=1$

Any move will increase $h$.

Hill Climbing: 8 Queens

Problem:
Place 8 queens on a chessboard such that
No queen attacks any other.

Successor Function
Return all possible states generated by moving a single queen to another square in the same column. (8*7=56)

Heuristic Cost Function
The number of pairs of queens that are attacking each other either directly or indirectly.
Global minimum - 0

Results

State Space: $8^8 = 17 \times 10^6$ states!
Branching factor of $8^7=56$

• Starting from a random 8 queen state:
  • Steepest hill ascent gets stuck 86% of the time.
  • It is quick: average of 3 steps when it fails, 4 steps when it succeeds.
  • $8^8 = 17$ million states!
Variants on Hill-Climbing

- Stochastic hill climbing
  - Chooses at random from among the uphill moves. Probability can vary with the steepness of the moves.
- Simulated Annealing
  - Combination of hill climbing and random walk.
- Local Beam search
  - Start with k randomly generated start states and generate their successors.
  - Choose the k best out of the union and start again.

Local Beam Search

- Start with k random states
- Determine successors of all k random states
- If any successors are goal states then finished
- Else select k best states from union of successors and repeat

Stochastic variant: choose k successors at random with probability of choosing the successor being an increasing function of its value.

Simulated Annealing

- Escape local maxima by allowing “bad” moves
- Idea: but gradually decrease their size and frequency
- Origin of concept: metallurgical annealing
- Bouncing ball analogy (gradient descent):
  - Shaking hard (= high temperature)
  - Shaking less (= lower the temperature)
- If Temp decreases slowly enough, best state is reached

The probability decreases exponentially with the “badness” of the move - the amount Delta E by which the evaluation is worsened.

The probability also decreases as the “temperature” T goes down: “bad” moves are more likely to be allowed at the start when the temperature is high, and more unlikely as T decreases.
Genetic Algorithms

Variant of Local Beam Search with the addition of sexual recombination

Genetic Algorithms

Using A* in Path Planning for a UAV (If time permits)

function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
inputs: population, a set of individuals
FITNESS-FN, a function that measures the fitness of an individual
repeat
new_population = empty set
for i = 1 to SIZE(population) do
  z = RANDOM-SELECTION(population, FITNESS-FN)
  y = RANDOM-SELECTION(population, FITNESS-FN)
  child = REPRODUCER(z, y)
  if (small random probability) then child = MUTATE(child)
  add child to new_population
population = new_population
until some individual is fit enough, or enough time has elapsed
return the best individual in population, according to FITNESS-FN

function REPRODUCER(x, y) returns an individual
inputs: x, y, parent individuals
n = LENGTH(x), c = random number from 1 to n
return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, c + 1, n))