Heuristics

Introduction to Intelligent Systems
Heuristics

- Heuristic: a rule or other piece of information that is used to make methods such as search more efficient or effective
- In search, often use a heuristic evaluation function, f(n):
  - f(n) tells you the approximate distance of a node, n, from a goal node
- f(n) may not be 100% accurate, but it should give better results than pure guesswork
Heuristics

- A heuristic should reduce the number of nodes that need to be examined
- The more informed a heuristic is, the better it will perform
- Heuristics are used for solving constraint satisfaction problems
  - Generate a possible solution, and then make small changes to bring it closer to satisfying constraints
Best-First Search

• Pick the *most likely* node (based on some heuristic value) from the partially expanded tree at each stage

• Tends to find a shorter path than depth-first or breadth-first search, but does not guarantee to find the best path
Best-First Search

• Use an evaluation function $f(n)$ for each node
  – $f(n)$ is an estimate of "desirability"
  – Expand the most desirable unexpanded node

• Implementation:
  – Order the nodes in fringe (candidate nodes) in order of decreasing desirability

• Special cases:
  – Greedy best-first search
  – $A^*$ search
Greedy Best-First Search

• Greedy Best-First Search always expands the node that appears to be closest to the goal
• Not optimal, and not guaranteed to find a solution at all
• Can easily be fooled into taking poor paths
Greedy Best-First Search

• Evaluation function \( f(n) = h(n) \) (heuristic)

• The heuristic is the estimate of cost from node \( n \) to the goal
  
  – e.g., \( h_{SLD}(n) \) = straight-line distance (“as the crow flies”) from start node to destination node

• Greedy best-first search expands the node that \textit{appears} to be closest to goal
Romania with Step Costs in km

Start: Arad
Destination: Bucharest

Straight-line distance to Bucharest:
- Arad: 366
- Bucharest: 0
- Craiova: 160
- Dobrota: 242
- Eforie: 161
- Fagaras: 176
- Giurgiu: 77
- Hirssova: 151
- Iasi: 226
- Lugoj: 244
- Mehadia: 241
- Neamt: 234
- Oradea: 380
- Pitești: 100
- Rimnicu Vilcea: 193
- Sibiu: 253
- Timisoara: 329
- Urziceni: 80
- Vaslui: 199
- Zerind: 374
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?
  - No – can get stuck in loops, e.g., Iasi → Neamt → Iasi → Neamt → etc.

- Time?
  - $O(b^m)$, but a good heuristic can give dramatic improvement

- Space?
  - $O(b^m)$ -- keeps all nodes in memory

- Optimal?
  - No
A* Search

• Idea: Avoid expanding paths that are already expensive

• Evaluation function \( f(n) = g(n) + h(n) \)

• \( f(n) \) = estimated total cost of path through \( n \) to goal

• \( g(n) \) = cost so far to reach \( n \)

• \( h(n) \) = estimated cost from \( n \) to goal
A* Search Example

• Choose $h(n)$ to be the straight-line distance to the goal (Bucharest), as with Greedy Best-First
• Choose $g(n)$ to be distance from the start node to the current node
A* Search Example
A* Search Example

366 = 0 + 366
A* Search Example
A* Search Example

![A* Search Diagram]

- **Ariad**
  - **Sibiu**
    - **Ariad**
      - Distance: 646 = 280 + 366
    - **Fagras**
      - Distance: 415 = 239 + 176
    - **Oradea**
      - Distance: 671 = 291 + 380
    - **Rimniu Vicoa**
      - Distance: 413 = 220 + 193
- **Timisoara**
  - Distance: 447 = 118 + 329
- **Zerind**
  - Distance: 449 = 75 + 374
A* Search Example
A* Search Example

Ariad

Sibiu

Fagaras

Oradea

Rimnicu Vida

Timisoara

447 = 118 + 329

Zerind

449 = 75 + 374

Ariad

646 = 280 + 366

Sibiu

591 = 338 + 253

Bucharest

450 = 450 + 0

Craiova

526 = 366 + 160

Pitesti

417 = 317 + 100

Sibiu

553 = 300 + 253
A* Search Example
A* Search

- A* algorithms are optimal:
  - They are guaranteed to find the shortest path to a goal node, provided $h$ is never an overestimate
    - i.e., $h$ is an admissible heuristic
- A* methods are also optimally efficient – they expand the fewest possible paths to find the right one
- If $h$ is not admissible, the method is called A, rather than A*
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: $h(n) = “straight-line distance”$ never overestimates the actual road distance.
Admissible Heuristics

• A heuristic $h$ is \textit{consistent} 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, then for every node $n$ and every successor $n'$ of $n$ generated by any action $a'$, the estimated cost of reaching the goal from $n$ is no greater than the step cost of getting to $n'$ plus the estimated cost of reaching the goal from $n'$ (also know as the \textit{triangle inequality})

• i.e., $f(n)$ is non-decreasing along any path ($f(n)$ is \textit{monotonic})
Admissible Heuristics

- All consistent heuristics are admissible, but not all admissible heuristics are consistent

http://en.wikipedia.org/wiki/Consistent_heuristic
Admissible Heuristics

- $A^*$ expands nodes in order of increasing $f$ value
- It gradually adds "$f$-contours" of nodes, where contour $i$ has all nodes with $f_i < f_{i+1}$
Other Admissible Heuristics

The start state, first move, and goal state for our example 8-puzzle:
Other Admissible Heuristics

For the 8-puzzle:

- \( h_1(n) \) = number of misplaced tiles
- \( h_2(n) \) = total Manhattan distance (i.e., the number of squares from desired location of each tile)
- \( h_3(n) \) = 2 * number of direct tile reversals

\[ \begin{align*}
\text{Start State} & : & \begin{array}{ccc}
7 & 2 & 4 \\
5 & 6 & \\
8 & 3 & 1 \\
\end{array} \\
\text{Goal State} & : & \begin{array}{ccc}
1 & 2 \\
3 & 4 & 5 \\
6 & 7 & 8 \\
\end{array}
\end{align*} \]

- \( h_1(S) = 8 \)
- \( h_2(S) = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18 \)
- \( h_3(S) = 0 \)
Other Admissible Heuristics

<table>
<thead>
<tr>
<th>Tiles out of place</th>
<th>Sum of distances out of place</th>
<th>$2 \times$ the number of direct tile reversals</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>

Goal: A square grid with numbers from 1 to 8, with one blank space.
Other Admissible Heuristics

Values of $f(n)$ for each state,

where:

- $f(n) = g(n) + h(n)$,
- $g(n) =$ actual distance from $n$ to the start state, and
- $h(n) =$ number of tiles out of place.
Other Admissible Heuristics
Properties of A*

- Complete?
  - Yes (unless there are infinitely many nodes with $f \leq f(G)$)
- Time?
  - Exponential
- Space?
  - Keeps all nodes in memory
- Optimal?
  - Yes
Dominance

• If $h_2(n) \geq h_1(n)$ for all $n$ (and both are admissible) then $h_2$ dominates $h_1$ ($h_2$ is better)

• Typical search costs (average number of nodes expanded) for iterative deepening search (IDS), $A^*(h_1)$ and $A^*(h_2)$ for the 8-puzzle:
  
  • $depth = 12$: 
    
    | Method      | Nodes Expanded |
    |-------------|----------------|
    | IDS         | 3,644,035      |
    | $A^*(h_1)$  | 227            |
    | $A^*(h_2)$  | 73             |
  
  • $depth = 24$: 
    
    | Method      | Nodes Expanded |
    |-------------|----------------|
    | IDS         | too many       |
    | $A^*(h_1)$  | 39,135         |
    | $A^*(h_2)$  | 1,641          |
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 exact solution
• If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the exact solution
Relaxed Problems

- If a problem is written down in a formal language, it is possible to construct heuristics automatically. Consider the following rule:
  - “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 heuristics by removing one or both of the conditions from the above rule:
  a) “A tile can move from square A to square B”
  b) “A tile can move from square A to square B if A is adjacent to B”
  c) “A tile can move from square A to square B if B is blank”
Constraint Satisfaction Problems

• A constraint satisfaction problem is a combinatorial optimization problem with a set of constraints
• Can be solved using search
• With many variables, it is essential to use heuristics
The Eight Queens Problem

• Example of a constraint satisfaction problem:
  – Place eight queens on a chess board so that no two queens are on the same row, column or diagonal

• Can be solved by search, but the search tree is large

• *Heuristic repair* (constraint satisfaction) is very efficient at solving this problem
The Eight Queens Problem

- Initial state – one queen is conflicting with another (locations c7 and h2)
- Constrain the problem by only moving a queen in the same column (column h)
- The numbers in the right-most squares indicate how many conflicts will arise if we move the queen at h2 to that row
- Move the queen at h2 to the square with the fewest conflicts (h6)
The Eight Queens Problem

• Second state – after moving the queen at h2 to h6
• Now the queen at f6 is conflicting with the queen at h6, so we’ll move f6 to the square with the fewest conflicts (f2)
The Eight Queens Problem

• Final state – a solution!
Local Search Algorithms

• In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution

• Like heuristic repair, local search methods start from a random state, and make small changes until a goal state is achieved

• Most local search methods are susceptible to local maxima, like *hill-climbing*
Hill-Climbing

- Hill-climbing is an informed, irrevocable (choices cannot be “un-done”) search method
- Easiest to understand when considered as a method for finding the highest point in a three dimensional search space:
  - Check the height one unit away from your current location in each direction
  - As soon as you find a position whose height is higher than your current position, move to that location, and restart the algorithm
Hill-Climbing

- Problem: depending on the initial state, you can get stuck in local maxima
Hill-Climbing: 8-Queens Problem

- $h =$ number of pairs of queens that are attacking each other, either directly or indirectly.
- $h = 17$ for the state shown here
- The number in each square indicates how many conflicts will result by moving the queen in that column to the indicated row
Hill-Climbing: 8-Queens Problem

A local minimum with $h = 1$
Improved Heuristics For The Eight Queens Problem

• One way to solve this problem is to have a tree that is 8-ply deep (a ply is a level), with a branching factor of 64 for the first level (because there are 64 squares to place the first queen on an empty board), 63 for the next level, and so on, down to 57 for the eighth level
Improved Heuristics For The Eight Queens Problem

• This can be improved further by noting that each row and column must contain exactly one queen
  – If we assume the first queen is placed in row 1, the second in row 2, etc. the first level will have a branching factor of 8, the next 7, the next 6, etc.
  – In addition, as each queen is placed on the board, it “uses up” a diagonal, meaning that the branching factor is only 5 or 6 after the first choice has been made
Simulated Annealing Search

• A method borrowed from metallurgy, based on the way in which metal is heated and then cooled very slowly in order to make it extremely strong

• Goal is to obtain a minimum value for some function of a large number of variables.
  – This value is known as the energy of the system
Simulated Annealing Search

- Simulated annealing escapes local minima by allowing some "bad" moves but gradually decreases the frequency with which they are allowed.

```plaintext
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
inputs: problem, a problem
         schedule, a mapping from time to "temperature"
local variables: current, a node
                next, a node
                T, a "temperature" controlling prob. of downward steps

current ← MAKE-NODE(INITIAL-STATE[problem])
for t ← 1 to ∞ do
    $T ← schedule[t]$
    if $T = 0$ then return current
    next ← a randomly selected successor of current
    $ΔE ← VALUE[next] − VALUE[current]$
    if $ΔE > 0$ then current ← next
    else current ← next only with probability $e^{ΔE/T}$
```
Simulated Annealing Search

- A random start state is selected
- A small random change is made to this state
  - If this change lowers the system energy, it is accepted
  - If it increases the energy, it may be accepted, depending on a probability called the Boltzmann acceptance criteria:
    - \( e^{-\frac{dE}{T}} \)
Simulated Annealing Search

• Because the energy of the system is sometimes allowed to increase, simulated annealing is able to escape from local minima
• Simulated annealing is a widely used local search method for solving problems with a very large numbers of variables
  – For example: scheduling problems, traveling salesman, placing VLSI (chip) components
Heuristics for Games of Strategy

• *Minimax* is a method used to evaluate game trees, where the goal is to maximize a utility function

• This will result in *perfect play* for a 2-player, deterministic, zero-sum game

• A static evaluator is applied to leaf nodes, and values are passed back up the tree to determine the best score a player can obtain against a rational opponent
Partial Game Tree for Tic-Tac-Toe

+1 is a win for max (X)
-1 is a win for min (O)
0 is a tie
Minimax

• 2 players: “max”, who wants to maximize his own score and “min” who wants to minimize max’s score
• Example: 2-ply game, with “max” taking the first turn (assume heuristic values at leaves are known by both players):

Max’s scores →
Minimax

- Minimax was originally about a two-player zero-sum game
- It essentially says that there are optimal strategies for such games
- Minimax strategies are essentially pessimistic (Murphy’s Law) strategies
  - What would you do if you knew that your opponent was going to make his best response against you?
  - What is the best strategy for you to play?
Example: Game of “Nim”

• A number of tokens are placed on a table between the two players
• At each move a player must divide a pile of tokens into two non-empty piles of different sizes
  – For example, 6 tokens may be divided into piles of 5 and 1, or 4 and 2, but not 3 and 3
• The first player who can no longer make a move loses the game
Game Tree for “Nim” with 7 Tokens
Game Tree for “Nim” with 7 Tokens

• In this game, assume Min moves first
• Give each leaf node a value of 1 or 0, depending on if it is a win for Max (1) or for Min (0)
• Propagate the values up the tree through successive parent nodes according to the rule:
  – If the parent state is a Max node, give it the maximum value among its children
  – If the parent state is a Min node, give it the minimum value among its children
Game Tree for “Nim” with 7 Tokens
Game Tree for “Nim” with 7 Tokens

- Because all of Min’s possible first moves lead to nodes with a derived value of 1, the second player, Max, can always force the game to a win, regardless of Min’s first move.
- Min can choose any of the first move alternatives and will still be guaranteed to lose.
- The resulting “win paths” for Max are in bold.
- Min can only win if Max played foolishly.
Properties of Minimax

- Complete?
  - Yes (if tree is finite)

- Optimal?
  - Yes (against an optimal opponent)

- Time complexity?
  - $O(b^m)$
    - For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games → exact solution completely infeasible!

- Space complexity?
  - $O(bm)$
Bounded Look-Ahead

• For trees with high depth or very high branching factor, minimax cannot be applied to the entire tree

• In such cases, *bounded look-ahead* is applied:
  – When search reaches a specified depth, the search is cut off, and the static evaluator applied

• Must be applied carefully:
  – In some positions a static evaluator will not take into account significant changes that are about to happen
α-β Pruning

- A method that can often cut off a large part of the game tree
- Based on the idea that if a move is clearly bad, there is no need to follow the consequences of it
α-β Pruning

• In this tree, having examined the leaf nodes with values 7 and 1, there is no need to examine the final leaf node (we can prune the tree at this node)
• To see why, notice that min is going to pass up to max the maximum of (3,1) which is 3
• But notice also that min is going to always choose the minimum value of its children, so once min sees a 1 in the right sub-tree, it knows that it will never pass anything from that tree up to max, because 1 < 3.
**α-β Pruning**

Max is waiting for a value that is $\geq 3$. 

Diagram:
- MAX
- MIN
  - 3
  - 12
  - 8

Root node: $\geq 3$
$\alpha$-$\beta$ Pruning

Max is waiting for a value that is $\geq 3$.

This sub-tree can be pruned because $2 < 3$.
$\alpha$-$\beta$ Pruning

Max is waiting for a value that is $\geq 3$.

This sub-tree cannot be pruned because $14 > 3$. 
$\alpha$-$\beta$ Pruning

Min is looking for a value <=14 but >= 3
Min is looking for a value $\leq 14$ but $\geq 3$. Since $2 < 5$ Min will choose that, and Max will have to settle for 3.
Properties of $\alpha$-$\beta$

- Pruning does not affect the final result
- Good move ordering improves effectiveness of pruning
- With "perfect ordering," time complexity = $O(b^{m/2})$
  - Reduces number of states that have to be checked by $\sqrt{2}$
- A simple example of the value of reasoning about which computations are relevant (a form of meta-reasoning)
Why is it Called $\alpha$-$\beta$?

- $\alpha$ is the value of the best (i.e., highest-value) choice found so far at any choice point along the path for $\text{max}$

- If $v$ is worse than $\alpha$, $\text{max}$ will avoid it
  - Prune that branch

- Define $\beta$ similarly for $\text{min}$