Informed search algorithms

Chapter 3
(Based on Slides by Stuart Russell, Richard Korf, Subbarao Kambhampati, and UW-AI faculty)
“Intuition, like the rays of the sun, acts only in an inflexibly straight line; it can guess right only on condition of never diverting its gaze; the freaks of chance disturb it.”
AND WARREN HERE IS IN CHARGE OF OUR GUT FEELINGS
BRUTE-FORCE SOLUTION: \( O(n!) \)

DYNAMIC PROGRAMMING ALGORITHMS: \( O(n^22^n) \)

SELLING ON EBAY: \( O(1) \)

STILL WORKING ON YOUR ROUTE?

SHUT THE HELL UP.
Informed (Heuristic) Search

Idea: be **smart** about what paths to try.
Blind Search vs. Informed Search

• What’s the difference?

• How do we formally specify this?

A node is selected for expansion based on an evaluation function that estimates cost to goal.
function tree-search(root-node)
    fringe ← successors(root-node)
    while ( notempty(fringe) )
        {node ← remove-first(fringe)
         state ← state(node)
         if goal-test(state) return solution(node)
         fringe ← insert-all(successors(node),fringe) }
    return failure
end tree-search
function tree-search(root-node)
    fringe \leftarrow \text{successors}(root-node)
    explored \leftarrow \text{empty}
    while ( \text{notempty}(fringe) )
        \{ \text{node} \leftarrow \text{remove-first}(fringe) \\
            \text{state} \leftarrow \text{state}(\text{node}) \\
            \text{if} \ \text{goal-test}(\text{state}) \ \text{return} \ \text{solution}(\text{node}) \\
            \text{explored} \leftarrow \text{insert}(\text{node},\text{explored}) \\
            \text{fringe} \leftarrow \text{insert-all}(\text{successors}(\text{node}),\text{fringe}, \text{if} \ \text{node} \ \text{not in} \ \text{explored}) \\
        \}
    \text{return} \ \text{failure}
end tree-search
Best-First Search

- Use an evaluation function $f(n)$ for node $n$.
- Always choose the node from fringe that has the lowest $f$ value.
Best-first search

• A search strategy is defined by picking the order of node expansion
• Idea: use an evaluation function $f(n)$ for each node
  – estimate of "desirability"

→ Expand most desirable unexpanded node

• Implementation:
  Order the nodes in fringe in decreasing order of desirability

• Special cases:
  – greedy best-first search
  – A* search
Romania with step costs in km
Greedy best-first search

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

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

- Greedy best-first search expands the node that appears to be closest to goal
Properties of greedy best-first search

- **Complete?**
  - No – can get stuck in loops, e.g., Iasi → Neamt → Iasi → Neamt →

- **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 $f(n) = g(n) + h(n)$
  
  • $g(n) = \text{cost so far to reach } n$
  • $h(n) = \text{estimated cost from } n \text{ to goal}$
  • $f(n) = \text{estimated total cost of path through } n \text{ to goal}$
A* for Romanian Shortest Path
Arad

Sibiu

Arad 646=280+366
Fagaras 415=239+176
Oradea 671=291+380
Rimnicu Vilcea

Timisoara

447=118+329

Zerind

449=75+374

Craiova

526=366+160

Pitesti

417=317+100

Sibiu

553=300+253
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)

• **Theorem**: If $h(n)$ is admissible, A* using TREE-SEARCH is optimal
Consistent Heuristics

• $h(n)$ is consistent if
  – for every node $n$
  – for every successor $n'$ due to legal action $a$
  – $h(n) \leq c(n,a,n') + h(n')$

• Every consistent heuristic is also admissible.

• **Theorem**: If $h(n)$ is consistent, $A^*$ using **GRAPH-SEARCH** is optimal
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 (depending upon search algo and heuristic property)

http://www.youtube.com/watch?v=huJEGj82360
Example: Romania

Breadth-First goes level by level
Visualizing Breadth-First & Uniform Cost Search

Breadth-First goes level by level

This is also a proof of optimality…
It will not expand
Nodes with f > f*
(f* is f-value of the
Optimal goal which
is the same as g* since
h value is zero for goals)

Uniform
search

Visualizing A* Search

Lemma: A* expands nodes in order of increasing f value

Gradually adds “f-contours” of nodes (cf. breadth-first adds layers)
Contour i has all nodes with f = f_i, where f_i < f_{i+1}
Not always clear where the total minimum occurs
- Old wisdom was that the global min was closer to cheaper heuristics
- Current insights are that it may well be far from the cheaper heuristics for many problems
  - E.g. Pattern databases for 8-puzzle
  - Plan graph heuristics for planning

How informed should the heuristic be?

Reduced level of abstraction (i.e. more and more concrete)
Memory Problem?

• Iterative deepening A*
  – Similar to ID search

  – While (solution not found)
    • Do DFS but prune when cost (f) > current bound
    • Increase bound
Non-optimal variations

• Use more informative, but inadmissible heuristics

• Weighted A*
  – \( f(n) = g(n) + w \cdot h(n) \) where \( w > 1 \)
  – Typically \( w = 5 \).
  – Solution quality bounded by \( w \) for admissible \( h \).
Admissible heuristics

E.g., for the 8-puzzle:

- $h_1(n) = \text{number of misplaced tiles}$
- $h_2(n) = \text{total Manhattan distance}$ (i.e., no. of squares from desired location of each tile)

\[ h_1(S) = ? \]
\[ h_2(S) = ? \]
Admissible heuristics

E.g., for the 8-puzzle:

- $h_1(n) = \text{number of misplaced tiles}$
- $h_2(n) = \text{total Manhattan distance}$
  (i.e., no. of squares from desired location of each tile)

\[
\begin{array}{ccc}
7 & 2 & 4 \\
5 & \text{ } & 6 \\
8 & 3 & 1 \\
\end{array}
\quad
\begin{array}{ccc}
3 & 4 & 5 \\
6 & 7 & 8 \\
1 & 2 & \text{ } \\
\end{array}
\]

- $h_1(S) = ? \quad 8$
- $h_2(S) = ? \quad 3+1+2+2+2+3+3+2 = 18$
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 node 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
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.
## Sizes of Problem Spaces

<table>
<thead>
<tr>
<th>Problem</th>
<th>Nodes</th>
<th>Brute-Force Search Time (10 million nodes/second)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 Puzzle:</td>
<td>$10^5$</td>
<td>.01 seconds</td>
</tr>
<tr>
<td>$2^3$ Rubik’s Cube:</td>
<td>$10^6$</td>
<td>.2 seconds</td>
</tr>
<tr>
<td>15 Puzzle:</td>
<td>$10^{13}$</td>
<td>6 days</td>
</tr>
<tr>
<td>$3^3$ Rubik’s Cube:</td>
<td>$10^{19}$</td>
<td>68,000 years</td>
</tr>
<tr>
<td>24 Puzzle:</td>
<td>$10^{25}$</td>
<td>12 billion years</td>
</tr>
</tbody>
</table>
Performance of IDA* on 15 Puzzle

• Random 15 puzzle instances were first solved optimally using IDA* with Manhattan distance heuristic (Korf, 1985).
• Optimal solution lengths average 53 moves.
• 400 million nodes generated on average.
• Average solution time is about 50 seconds on current machines.
Limitation of Manhattan Distance

• To solve a 24-Puzzle instance, IDA* with Manhattan distance would take about 65,000 years on average.
• Assumes that each tile moves independently
• In fact, tiles interfere with each other.
• Accounting for these interactions is the key to more accurate heuristic functions.
Example: Linear Conflict

Manhattan distance is 2+2=4 moves
Example: Linear Conflict

Manhattan distance is 2+2=4 moves
Example: Linear Conflict

Manhattan distance is $2+2=4$ moves
Example: Linear Conflict

Manhattan distance is 2+2=4 moves
Example: Linear Conflict

Manhattan distance is 2+2=4 moves
Example: Linear Conflict

Manhattan distance is 2+2=4 moves
Example: Linear Conflict

Manhattan distance is $2+2=4$ moves, but linear conflict adds 2 additional moves.
Linear Conflict Heuristic

• Hansson, Mayer, and Yung, 1991
• Given two tiles in their goal row, but reversed in position, additional vertical moves can be added to Manhattan distance.
• Still not accurate enough to solve 24-Puzzle
• We can generalize this idea further.
More Complex Tile Interactions

M.d. is 19 moves, but 31 moves are needed.

M.d. is 20 moves, but 28 moves are needed

M.d. is 17 moves, but 27 moves are needed
Pattern Database Heuristics

• Culberson and Schaeffer, 1996
• A pattern database is a complete set of such positions, with associated number of moves.
• e.g. a 7-tile pattern database for the Fifteen Puzzle contains 519 million entries.
31 moves is a lower bound on the total number of moves needed to solve this particular state.
Combining Multiple Databases

31 moves needed to solve red tiles

22 moves need to solve blue tiles

Overall heuristic is maximum of 31 moves
Additive Pattern Databases

- Culberson and Schaeffer counted all moves needed to correctly position the pattern tiles.
- In contrast, we count only moves of the pattern tiles, ignoring non-pattern moves.
- If no tile belongs to more than one pattern, then we can add their heuristic values.
- Manhattan distance is a special case of this, where each pattern contains a single tile.
The 7-tile database contains 58 million entries. The 8-tile database contains 519 million entries.
Computing the Heuristic

20 moves needed to solve red tiles

25 moves needed to solve blue tiles

Overall heuristic is sum, or 20+25=45 moves
Performance on 15 Puzzle

• IDA* with a heuristic based on these additive pattern databases can optimally solve random 15 puzzle instances in less than 29 milliseconds on average.

• This is about 1700 times faster than with Manhattan distance on the same machine.