Informed Search I

Review: Uniform Cost Search
Introduction to informed search
The A* search algorithm
Designing good admissible heuristics

(AIMA Chapter 3.5.1, 3.5.2, 3.6)

Motivation: Romanian Map Problem

- All our search methods so far assume step-cost = 1
- This isn’t always true

$g(N)$: the path cost function

- If all moves equal in cost:
  - Cost = # of nodes in path - 1
  - $g(N) = \text{depth}(N)$
  - Equivalent to what we’ve been assuming so far

- Assigning a (potentially) unique cost to each step
  - $N_0, N_1, N_2, \ldots$ = nodes visited on path $p$
  - $C(i,j)$: Cost of going from $N_i$ to $N_j$
  - $g(N_3) = C(0,1) + C(1,2) + C(2,3)$

Uniform-cost search (UCS)

- Extension of BF-search:
  - Expand node with lowest path cost
- Implementation: frontier = queue ordered by $g(n)$
- Differs from BF-search:
  - Tests if a node is a goal state when it is selected for expansion, not when it is added to the frontier.
  - Updates a node on the frontier if a better path to the same state is found.
  - So always enqueues nodes before checking whether they are goals.
  - WHY???
- (Dijkstra’s algorithm just UCS without goal)
Outline – Informed Search

PART I
- Informed = use problem-specific knowledge
- Best-first search and its variants
- A* - Optimal Search using Knowledge
- Proof of Optimality of A*
- Heuristic functions?
- How to invent them

PART II
- Local search and optimization
  - Hill climbing, local beam search, genetic algorithms...
  - Local search in continuous spaces
  - Online search agents

Is Uniform Cost Search the best we can do?
Consider finding a route from Bucharest to Arad...

A Better Idea...
- Node expansion based on some estimate of distance to the goal, extending current path
- General approach of informed search:
  - Best-first search: node selected for expansion based on an evaluation function \( f(n) \)
    - \( f(n) \) includes estimate of distance to goal
- Implementation:
  - Sort frontier queue monotonically by \( f(n) \).
  - Special cases: greedy search, A* search

A heuristic function
- Let evaluation function \( f(n) = h(n) \) (heuristic)
  - \( h(n) = \text{estimated cost of the cheapest path from node } n \text{ to goal node} \)
  - If \( n \text{ is goal then } h(n) = 0 \)
- Here: \( h_{\text{SLD}}(n) \) = straight-line distance from \( n \) to Bucharest

Heureka! ---Archimedes
Breadth First for Games, Robots, ...

- Pink: Starting Point
- Blue: Goal
- Teal: Scanned squares
  - Darker: Closer to starting point...

Graphics from http://theory.stanford.edu/~amitp/GameProgramming
(A great site for practical AI & game Programming

An optimal informed search algorithm (A*)

- We add a heuristic estimate of distance to the goal
- Yellow: examined nodes with high estimated distance
- Blue: examined nodes with low estimated distance

Breadth first in a world with obstacles

Informed search (A*) in a world with obstacles

Greedy best-first search

- Informally: Expands the node that is estimated to be closest to goal
- Expands nodes based on \( f(n) = h(n) \)
  - (Again, \( h(n) \): Some heuristic estimate of cost from \( n \) to goal)
- Ignores cost so far to get to that node (\( g(n) \))
- Here, \( h(n) = h_{SLD}(n) \) = straight-line distance from \( n \) to Bucharest

Greedy best-first search example

- Initial State = Arad
- Goal State = Bucharest

<table>
<thead>
<tr>
<th>Initial State</th>
<th>Goal State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arad</td>
<td>Bucharest</td>
</tr>
<tr>
<td>366</td>
<td>203</td>
</tr>
<tr>
<td>Cracovia</td>
<td>Oradea</td>
</tr>
<tr>
<td>365</td>
<td>351</td>
</tr>
<tr>
<td>Dokiet</td>
<td>Pitesti</td>
</tr>
<tr>
<td>342</td>
<td>100</td>
</tr>
<tr>
<td>Esztergom</td>
<td>Rimnicu Vilcea</td>
</tr>
<tr>
<td>372</td>
<td>212</td>
</tr>
<tr>
<td>Fagaras</td>
<td>Sighisoara</td>
</tr>
<tr>
<td>277</td>
<td>312</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>Uroseni</td>
</tr>
<tr>
<td>77</td>
<td>90</td>
</tr>
<tr>
<td>Huedin</td>
<td>Vaslui</td>
</tr>
<tr>
<td>151</td>
<td>199</td>
</tr>
<tr>
<td>Iasi</td>
<td>Zerind</td>
</tr>
<tr>
<td>236</td>
<td>254</td>
</tr>
</tbody>
</table>
Greedy best-first search example

Frontier queue:
- Sibiu 253
- Timisoara 329
- Zerind 374

<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>Dobra</td>
<td>242</td>
</tr>
<tr>
<td>Eforie</td>
<td>176</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
</tr>
<tr>
<td>Hersona</td>
<td>153</td>
</tr>
<tr>
<td>Iasi</td>
<td>226</td>
</tr>
<tr>
<td>Lugoj</td>
<td>244</td>
</tr>
</tbody>
</table>

Arad 366 Mehadia 246
Bucharest 0 Neamt 236
Craiova 160 Oradea 386
Dobra 242 Ploesti 108
Eforie 176 Rumania Vitea 193
Giurgiu 77 Timisoara 324
Hersona 153 Urziceni 80
Iasi 226 Vaslui 199
Lugoj 244 Zerind 374

Goal reached!!

Greedy best-first search example

Frontier queue:
- Fagaras 176
- Rimnicu Vilcea 193
- Timisoara 329
- Arad 366
- Zerind 374
- Oradea 380

<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>Dobra</td>
<td>242</td>
</tr>
<tr>
<td>Eforie</td>
<td>176</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
</tr>
<tr>
<td>Hersona</td>
<td>153</td>
</tr>
<tr>
<td>Iasi</td>
<td>226</td>
</tr>
<tr>
<td>Lugoj</td>
<td>244</td>
</tr>
</tbody>
</table>

Arad 366 Neckaia 246
Bucharest 0 Neamt 236
Craiova 160 Oradea 386
Dobra 242 Ploesti 108
Eforie 176 Rumania Vitea 193
Giurgiu 77 Timisoara 324
Hersona 153 Uziceni 80
Iasi 226 Vaslui 199
Lugoj 244 Zerind 374

Properties of greedy best-first search

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

- **Time?** $O(b^m)$ – worst case (like Depth First Search)
  - But a good heuristic can give dramatic improvement

- **Space?** $O(b^m)$ – priority queue, so worst case: keeps all nodes in memory

- **Optimal?** No

Properties of greedy best-first search

- **Optimal?**
  - No!
  - Found: Arad → Sibiu → Fagaras → Bucharest (450km)
  - Shortest: Arad → Sibiu → Rimnicu Vilcea → Pitesti → Bucharest (418km)
**A* search**

- Best-known form of best-first search.
- 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: \( \forall n, \exists h(n) \) where \( h(n) \) is the true cost from \( n \)
  - \( h(n) \geq 0 \) 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:
- Arad 366

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 Vlea.
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 in order of increasing \( f \) value
- Gradually adds “f-contours” of nodes
- Contour has all nodes with \( f = f_i \), where \( f_i < f_{i+1} \)
- (After all, A* is just a variant of uniform-cost search....)

**Optimality of A* using Tree-Search I (proof)**

1. \( f(G_2) > f(n) \)
2. \( h(n) = 0 \) since \( h \) is admissible, and therefore optimistic
3. \( g(n) + h(n) \leq f(n) \) from definitions of \( f \) and \( h \) (think about it!)
4. \( f(G_2) > f(G) \) repeated
5. \( f(n) = h(n) \)
6. \( g(n) + h(n) \geq g(n) + h(n) \) from 5
7. \( h(n) \leq f(n) \)
8. \( f(G_2) \geq f(n) \) from 1,2,3
9. \( f(G_2) > f(n) \)

So A* will never select \( G_2 \) for expansion
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: (exponential with path length)
- Space complexity:
  - It keeps all generated nodes in memory
  - 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 $f(n) > f(G)$
  - Also optimally efficient (not including ties)

---

Proof of Lemma: Consistency

- A heuristic is consistent if
  \[ h(n) \leq c(n,a,n') + h(n') \]
- Lemma: If $h$ is consistent,
  \[ f(n') = g(n') + h(n') \]
  \[ \geq g(n) + c(n,a,n') + h(n') \]
  \[ \geq g(n) + h(n) \]
  \[ f(n') \geq f(n) \]
  i.e. $f(n)$ is nondecreasing along any path.

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

Creating Good Heuristic Functions

AIMA 3.6
Heuristic functions

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

Admissible heuristics

E.g., for the 8-puzzle:
- $h_{oop}(n) =$ number of out of place tiles
- $h_{md}(n) =$ total Manhattan distance (i.e., # of moves from desired location of each tile)

$$\begin{array}{|c|c|}
\hline
7 & 2 & 4 \\
5 & 6 & 3 \\
8 & 3 & 1 \\
\hline
\end{array}$$

$$\begin{array}{|c|c|}
\hline
1 & 2 \\
3 & 4 \\
6 & 7 & 8 \\
\hline
\end{array}$$

- $h_{oop}(S) =$ ?
- $h_{md}(S) =$ ?

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 $h_{oop}(n)$ gives the shortest solution
- If the rules are relaxed so that a tile can move to any adjacent square, then $h_{md}(n)$ 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.

Constraints on relaxed problems:
- A tile can move from square A to square B if A is adjacent to B.
- A tile can move from square A to square B if B is blank.
- A tile can move from square A to square B (A to blank)

Dominance

- 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: $h_{md}$ dominates $h_{oop}$

Typical search costs (average number of nodes expanded):
- $d=12$ Iterative Deepening Search = 3,644,035 nodes
  - A*(h_{noop}) = 227 nodes
  - A*(h_{md}) = 73 nodes
- $d=24$ IDS = too many nodes
  - A*(h_{noop}) = 39,135 nodes
  - A*(h_{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*)

(see 3.5.3 if you're interested in these topics)