**Informed Search**

Introduction to informed search
The A* search algorithm
Designing good admissible heuristics

(AIMA Chapter 3.5.1, 3.5.2, 3.6)

---

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

![Image](CIS 391 - Intro to AI)

**A Better Idea...**

- Node expansion based on an estimate which includes distance to the goal

- 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 (new idea!)

- Implementation: Sort frontier queue by this new \( f(n) \).
  - Special cases: greedy search, \( A^* \) search

---

**Simple, useful estimate heuristic: straight-line distances**

![Image](CIS 391 - Intro to AI)
Heuristic (estimate) functions

Heureka! --- Archimedes

[Dictionary] “A rule of thumb, simplification, or educated guess that reduces or limits the search for solutions in domains that are difficult and poorly understood.”

Heuristic knowledge is useful, but not necessarily correct. Heuristic algorithms use heuristic knowledge to solve a problem.

A heuristic function \( h(n) \) takes a state \( n \) and returns an estimate of the distance from \( n \) to the goal.

(A graphic: http://hyperbolegames.com/2014/10/20/eureka-moments/)

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

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

(A great site for practical AI & game Programming)

Informed search (A*) in a world with obstacles

Greedy best-first search in a world with obstacles
**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: \( f(n) = h(n) \)**
- Expands the node that is estimated 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 \( n \) to Bucharest

*Greedy best-first search example*

<table>
<thead>
<tr>
<th>Initial State = Arad</th>
<th>Goal State = Bucharest</th>
</tr>
</thead>
</table>

Frontier queue:
- Arad 366
- Bucharest 0
- Craiova 160
- Dubova 342
- Eforie 161
- Fagaras 176
- Giurgiu 77
- Hirssova 151
- Iasi 226
- Lugoj 244
- Mladia 241
- Neamt 234
- Oradura 380
- Pitesti 100
- Rimnicu Vilcea 193
- Sibiu 253
- Starnia 329
- Uzunce 46
- Vaslui 199
- Zerind 374

Goal reached!!

Frontier queue:
- Bucharest 0
- Rimnicu Vilcea 193
- Sibiu 253
- Timisoara 329
- Arad 366
- Oradura 380
- Pitesti 100
- Rimnicu Vilcea 193
- Sibiu 253
- Starnia 329
- Uzunce 46
- Vaslui 199
- Zerind 374
- Oradura 380
Properties of greedy best-first search

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

- **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 of average cost

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

- **Optimal?** No

---

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: $\forall n$, if $n$ is 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:** $h_{SFLP}(n)$ never overestimates the actual road distance

**Theorem:** If $h(n)$ is **admissible**, A* using Tree Search is optimal
We add the three nodes we found to the Frontier queue. We sort them according to the \( g(\cdot)+h(\cdot) \) 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 Vila.

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

Suppose some suboptimal goal $G_2$ (i.e. a goal on a suboptimal path) has been generated and is in the frontier along with an optimal goal $G$.

Must prove: $f(G_2) > f(G)$

(Why? Because if $f(G_2) > f(n)$, then $G_2$ will never get to the front of the priority queue.)

Proof:
1. $g(G_2) > g(G)$ since $G_2$ is suboptimal
2. $f(G_2) = g(G_2)$ since $f(G_2) = g(G_2) + h(G_2)$ & $h(G_2) = 0$, 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 $f(n) < f(G)$)

- 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,

\[
\begin{align*}
    f(n') &= g(n') + h(n') \\
    &= g(n) + c(n,a,n') + h(n') \\
    &\geq g(n) + h(n) = f(n)
\end{align*}
\]

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 $\leq 3$)
  - 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:
- $h_{\text{hop}}(n) =$ number of out of place tiles
- $h_{\text{md}}(n) =$ total Manhattan distance (i.e., # of moves from desired location of each tile)

\begin{align*}
\text{Initial State} & : \begin{pmatrix} 7 & 2 & 4 \\ 5 & 6 & 3 \\ 1 & 2 & 8 \end{pmatrix} \\
\text{Goal State} & : \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & \_ \end{pmatrix}
\end{align*}

- $h_{\text{hop}}(S) = ?$
- $h_{\text{md}}(S) = ?$

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 $h_{\text{hop}}(n)$ 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

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. ($h_{\text{md}}$)
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
  - \( A^*(h_{oop}) = 2,227 \) nodes
  - \( A^*(h_{md}) = 73 \) nodes
- \( d=24 \)
  - IDS = too many nodes
  - \( A^*(h_{oop}) = 39,135 \) 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)