Title: Informed Search Methods
Required reading: AIMA, Chapter 3 (Sections 3.5, 3.6)
LWH: Chapters 6, 10, 13 and 14.
Outline

- Categorization of search techniques
- Ordered search (search with an evaluation function)
- Best-first search:
  (1) Greedy search  (2) A*
- Admissible heuristic functions:
  how to compare them?
  how to generate them?
  how to combine them?
Types of Search (I)

1- Uninformed vs. informed
2- Systematic/constructive vs. iterative improvement

**Uninformed:**
use only information available in problem definition,
no idea about distance to goal
→ can be incredibly ineffective in practice

**Heuristic:**
exploits some knowledge of the domain
also useful for solving optimization problems
Types of Search (II)

Systematic, exhaustive, constructive search:

a partial solution is incrementally extended into global solution

Partial solution =
sequence of transitions between states

Global solution =
Solution from the initial state to the goal state

Examples: \[
\begin{align*}
\text{Uninformed} \\
\text{Informed (heuristic): Greedy search, A*}
\end{align*}
\]

→ Returns the path; solution = path
Types of Search (III)

Iterative improvement:
A state is gradually modified and evaluated until reaching an (acceptable) optimum

→ We don’t care about the path, we care about ‘quality’ of state
→ Returns a state; a solution = good quality state
→ Necessarily an informed search

Examples (informed):
- Hill climbing
- Simulated Annealing (physics), Taboo search
- Genetic algorithms (biology)
Ordered search

• Strategies for systematic search are generated by choosing which node from the fringe to expand first

• The node to expand is chosen by an evaluation function, expressing ‘desirability’ → ordered search

• When nodes in queue are sorted according to their decreasing values by the evaluation function → best-first search

• Warning: ‘best’ is actually ‘seemingly-best’ given the evaluation function. Not always best (otherwise, we could march directly to the goal!)
Search using an evaluation function

- Example: uniform-cost search!

  What is the evaluation function?
  Evaluates cost from ............. to .................?

- How about the cost to the goal?

  \[ h(n) = \text{estimated cost of the cheapest path from the state at node } n \text{ to a goal state} \]

  \( h(n) \) would help focusing search
Cost to the goal

This information is **not** part of the problem description

<table>
<thead>
<tr>
<th>City</th>
<th>Cost to Goal</th>
<th>City</th>
<th>Cost to Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arad</td>
<td>366</td>
<td>Mehadia</td>
<td>241</td>
</tr>
<tr>
<td>Bucharest</td>
<td>0</td>
<td>Neamt</td>
<td>234</td>
</tr>
<tr>
<td>Craiova</td>
<td>160</td>
<td>Oradea</td>
<td>380</td>
</tr>
<tr>
<td>Dobrova</td>
<td>242</td>
<td>Pitesti</td>
<td>100</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
<td>Rimnicu Vilcea</td>
<td>193</td>
</tr>
<tr>
<td>Fagaras</td>
<td>176</td>
<td>Sibiu</td>
<td>253</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
<td>Timisoara</td>
<td>329</td>
</tr>
<tr>
<td>Hirsova</td>
<td>151</td>
<td>Urziceni</td>
<td>80</td>
</tr>
<tr>
<td>Iasi</td>
<td>226</td>
<td>Vaslui</td>
<td>199</td>
</tr>
<tr>
<td>Lugoj</td>
<td>244</td>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>
Best-first search

1. **Greedy search** chooses the node $n$ closest to the goal such as $h(n)$ is minimal

2. **A* search** chooses the least-cost solution

   $f(n) = \begin{cases} g(n): \text{cost from root to a given node } n \\ + \\ h(n): \text{cost from the node } n \text{ to the goal node} \end{cases}$

such as $f(n) = g(n) + h(n)$ is minimal
Greedy search

→ First expand the node whose state is ‘closest’ to the goal!

→ Minimize $h(n)$

```plaintext
function BEST-FIRST-SEARCH(problem, EVAL-FN) returns a solution sequence

inputs: problem, a problem
        Eval-Fn, an evaluation function

Queueing-Fn ← a function that orders nodes by Eval-Fn
return GENERAL-SEARCH(problem, Queueing-Fn)
```

→ Usually, cost of reaching a goal may be estimated, not determined exactly

→ If state at $n$ is goal, $h(n) =$ ?

→ How to choose $h(n)$? Problem specific! Heuristic!
Greedy search: Romania

$h_{\text{SLD}}(n) = \text{straight-line distance between } n \text{ and goal location}$
**Greedy search**: Trip from Arad to Bucharest

(a) The initial state

(b) After expanding Arad

(c) After expanding Sibiu

(d) After expanding Fagaras

... Greedy search! quick, but not optimal!
Greedy search: Problems

From Iasi to Fagaras?

False starts: Neamt is a dead-end
Looping
**Greedy search**: Properties

→ Like depth-first, tends to follow a single path to the goal

→ Like depth-first
→ Not complete
→ Not optimal

→ Time complexity: $O(b^m)$, $m$ maximum depth

→ Space complexity: $O(b^m)$ retains all nodes in memory

→ Good $h$ function (considerably) reduces space and time
   but $h$ functions are problem dependent :—(}
Hmm...

**Greedy search** minimizes estimated cost to goal $h(n)$
- cuts search cost considerably
- but not optimal, not complete

**Uniform-cost search** minimizes cost of the path so far $g(n)$
- is optimal and complete
- but can be wasteful of resources

**New-Best-First search** minimizes $f(n) = g(n) + h(n)$
- combines greedy and uniform-cost searches
  
  $f(n) =$ estimated cost of cheapest solution via $n$
- Provably: complete and optimal, if $h(n)$ is admissible
A* Search

- A* search
  Best-first search expanding the node in the fringe with minimal
  \[ f(n) = g(n) + h(n) \]

- A* search with admissible \( h(n) \)
  Provably complete, optimal, and optimally efficient using
  Tree-Search

- A* search with consistent \( h(n) \)
  Remains optimal even using Graph-Search

(See Tree-Search versus Graph-Search page 77)
Admissible heuristic

An admissible heuristic is a heuristic that never overestimates the cost to reach the goal

→ is optimistic
→ thinks the cost of solving is less than it actually is

Example:

\[
\begin{cases}
\text{travel: straight line distance} \\
\text{I need 3 years to finish college (at least!)} \\
\text{We are 3 years away from the first flight to Mars (at least!)}
\end{cases}
\]

If \( h \) is admissible,\

\[ f(n) \text{ never overestimates the actual cost of the best solution through } n. \]
A* Search From Arad to Bucharest

(a) The initial state

(b) After expanding Arad

(c) After expanding Sibiu

(d) After expanding Rimnicu Vilcea

(e) After expanding Fagaras

(f) After expanding Pitesti
A* Search is optimal

\[ G, G_2 \text{ goal states } \Rightarrow g(G) = f(G), \ f(G_2) = g(G_2) \quad h(G) = h(G_2) = 0 \]

\[ G \text{ optimal goal state } \Rightarrow C^* = f(G) \]

\[ G_2 \text{ suboptimal } \Rightarrow f(G_2) > C^* = f(G) \] (1)

Suppose \( n \) is not chosen for expansion

\[ h \text{ admissible } \Rightarrow C^* \geq f(n) \] (2)

Since \( n \) was not chosen for expansion \( \Rightarrow f(n) \geq f(G_2) \) (3)

(2) + (3) \( \Rightarrow C^* \geq f(G_2) \) (4)

(1) and (4) are contradictory \( \Rightarrow n \) should be chosen for expansion
Which nodes does A* expand?

**Goal-Test** is applied to **State(node)** when a node is chosen from the fringe for expansion, not when the node is generated.

Theorem 3 & 4 in Pearl 84, original results by Nilsson

- **Necessary condition:** Any node expanded by A* cannot have an \( f \) value exceeding \( C^* \): For all nodes expanded, \( f(n) \leq C^* \)

- **Sufficient condition:** Every node in the fringe for \( f(n) < C^* \) will eventually be expanded by A*

In summary

- A* expands all nodes with \( f(n) < C^* \)
- A* expands some nodes with \( f(n) = C^* \)
- A* expands no nodes with \( f(n) > C^* \)
Expanding contours

A* expands nodes from fringe in increasing $f$ value
We can conceptually draw contours in the search space

The first solution found is necessarily the optimal solution
Careful: a TEST-GOAL is applied at node expansion
**A* Search** is complete

Since A* search expands all nodes with \( f(n) < C^* \), it must eventually reach the goal state unless there are infinitely many nodes \( f(n) < C^* \)

- 1. \( \exists \) a node with infinite branching factor
- 2. \( \exists \) a path with infinite number of nodes along it

A* is complete if

- on locally finite graphs
- \( \exists \delta > 0 \) constant, the cost of each operator > \( \delta \)
**A* Search Complexity**

**Time:**
Exponential in (relative error in $h \times$ length of solution path) ... quite bad

**Space:** must keep all nodes in memory
Number of nodes within goal contour is exponential in length of solution.... unless the error in the heuristic function $|h(n) - h^*(n)|$ grows no faster than the log of the actual path cost: $|h(n) - h^*(n)| \leq O(\log h^*(n))$

In practice, the error is proportional... impractical..

major drawback of A*: runs out of space quickly

→ Memory Bounded Search IDA*(not addressed here)
A* Search is optimally efficient

... for any given evaluation function: no other algorithms that finds the optimal solution is guaranteed to expend fewer nodes than A*

Interpretation (proof not presented): Any algorithm that does not expand all nodes between root and the goal contour risks missing the optimal solution
Tree-Search vs. Graph-Search

After choosing a node from the fringe and before expanding it, Graph-Search checks whether State(node) was visited before to avoid loops.

→ Graph-search may lose optimal solution

Solutions

1. In Graph-Search, discard the more expensive path to a node

2. Ensure that the optimal path to any repeated state is the first one found
   → Consistency
Consistency

$h(n)$ is consistent
If $\forall \ n$ and $\forall \ n'$ successor of $n$ along a path, we have
$h(n) \leq k(n,n') + h(n'),\ k \ \text{cost of cheapest path from } n \ \text{to} \ n'$

Monotonicity

$h(n)$ is monotone
If $\forall \ n$ and $\forall \ n'$ successor of $n$ generated by action $a$, we have
$h(n) \leq c(n,a,n') + h(n'),\ n'$ is an immediate successor of $n$
Triangle inequality ($\langle n, n', \text{goal}\rangle$)

Important: $h$ is consistent $\iff$ $h$ is monotone

Beware: of confusing terminology ‘consistent’ and ‘monotone’
Values of $h$ not necessarily decreasing/nonincreasing
Properties of $h$: Important results

- $h$ consistent $\iff$ $h$ monotone \hfill (Pearl 84)
- $h$ consistent $\Rightarrow$ $h$ admissible \hfill (AIMA, Exercise 4.7)
  consistency is stricter than admissibility
- $h$ consistent $\Rightarrow$ $f$ is nondecreasing
  
  
  $$f(n') = g(n') + h(n') = g(n) + c(n, a, n') + h(n') \geq g(n) + h(n) = f(n)$$

- $h$ consistent $\Rightarrow$ $A^*$ using \textsc{Graph-Search} is optimally efficient
Pathmax equation

Monotonicity of $f$: values along a path are nondecreasing
When $f$ is not monotonic, use pathmax equation
\[
f(n') = \max(f(n), g(n') + h(n'))
\]
A* never decreases along any path out from root

Pathmax
- guarantees $f$ nondecreasing
- does not guarantee $h$ consistent
- does not guarantee A* + GRAPH-SEARCH is optimally efficient
Summarizing definitions for $A^*$

- $A^*$ is a best-first search that expands the node in the fringe with minimal $f(n) = g(n) + h(n)$

- An admissible function $h$ never overestimates the distance to the goal.

- $h$ admissible $\Rightarrow$ $A^*$ is complete, optimal, optimally efficient using Tree-Search

- $h$ consistent $\iff h$ monotone
  - $h$ consistent $\Rightarrow h$ admissible
  - $h$ consistent $\Rightarrow f$ nondecreasing

- $h$ consistent $\Rightarrow A^*$ remains optimal using Graph-Search
Admissible heuristic functions

Examples

• Route-finding problems: straight-line distance

• 8-puzzle:

\[
\begin{align*}
h_1(n) &= \text{number of misplaced tiles} \\
h_2(n) &= \text{total Manhattan distance}
\end{align*}
\]

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

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

\[
h_1(S) = ? \\
h_2(S) = ?
\]
Performance of admissible heuristic functions

Two criteria to compare admissible heuristic functions:

1. Effective branching factor: $b^*$

2. Dominance: number of nodes expanded
Effective branching factor $b^*$

- The heuristic expands $N$ nodes in total
- The solution depth is $d$

$\rightarrow b^*$ is the branching factor had the tree been uniform

\[ N = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d = \frac{(b^*)^{d+1} - 1}{b^* - 1} \]

- Example: $N=52$, $d=5 \rightarrow b^* = 1.92$
Dominance

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

Typical search costs: nodes expanded

<table>
<thead>
<tr>
<th>Sol. depth</th>
<th>IDS</th>
<th>( A^*(h_1) )</th>
<th>( A^*(h_2) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d = 12 )</td>
<td>3,644,035</td>
<td>227</td>
<td>73</td>
</tr>
<tr>
<td>( d = 24 )</td>
<td>too many</td>
<td>39,135</td>
<td>1,641</td>
</tr>
</tbody>
</table>

\( A^* \) expands all nodes \( f(n) < C^* \Rightarrow g(n) + h(n) < C^* \)
\( \Rightarrow h(n) < C^* - g(n) \)

If \( h_1 \leq h_2 \), \( A^* \) with \( h_1 \) will always expand at least as many (if not more) nodes than \( A^* \) with \( h_2 \)

\( \rightarrow \) It is always better to use a heuristic function with higher values, as long as it does not overestimate (remains admissible)
How to generate admissible heuristics?

→ Use *exact* solution cost of a relaxed (easier) problem

Steps:
- Consider problem $P$
- Take a problem $P'$ easier than $P$
- Find solution to $P'$
- Use solution of $P'$ as a heuristic for $P$
Relaxing the 8-puzzle problem

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
   The rules are relaxed so that a tile can move to any adjacent square: the shortest solution can be used as a heuristic
   \( (\equiv h_2(n)) \)

2. A tile can move from square A to square B if B is blank
   Gaschnig heuristic (Exercice 3.31, AIMA, page 119)

3. A tile can move from square A to square B
   The rules of the 8-puzzle are relaxed so that a tile can move anywhere: the shortest solution can be used as a heuristic
   \( (\equiv h_1(n)) \)
An admissible heuristic for the TSP

Let path be *any* structure that connects all cities

⇒ minimum spanning tree heuristic (polynomial)

(Exercice 3.30, AIMA, page 119)
Combining several admissible heuristic functions

We have a set of admissible heuristics \( h_1, h_2, h_3, \ldots, h_m \) but no heuristic that dominates all others, what to do?

\[ h(n) = \max(h_1(n), h_2(n), \ldots, h_m(n)) \]

\( h \) is admissible and dominates all others.

\( \rightarrow \) Problem:

Cost of computing the heuristic (vs. cost of expanding nodes)
Using subproblems to derive an admissible heuristic function

Goal: get 1, 2, 3, 4 into their correct positions, ignoring the ‘identity’ of the other tiles

Cost of optimal solution to subproblem used as a lower bound (and is substantially more accurate than Manhattan distance)

Pattern databases:

- Identify patterns (which represent several possible states)
- Store cost of exact solutions of patterns
- During search, retrieve cost of pattern and use as a (tight) estimate

Cost of building the database is amortized over ‘time’