Last time: Problem-Solving

**Problem solving:**
- Goal formulation
- Problem formulation (states, operators)
- Search for solution

**Problem formulation:**
- Initial state
- ?
- ?
- ?

Last time: Problem-Solving

**Problem types:**
- single state:
  - accessible and deterministic environment
- multiple state: ?
- contingency: ?
- exploration: ?
Last time: Finding a solution

Solution: is ???

Basic idea: offline, systematic exploration of simulated state-space by generating successors of explored states (expanding)

**Function** General-Search(*problem*, *strategy*) returns a *solution*, or failure

- initialize the search tree using the initial state problem

**loop do**
- **if** there are no candidates for expansion **then return** failure
- choose a leaf node for expansion according to strategy
- **if** the node contains a goal state **then return** the corresponding solution
- **else** expand the node and add resulting nodes to the search tree

**end**

Strategy: The search strategy is determined by ???
Last time: search strategies

**Uninformed:** Use only information available in the problem formulation
- Breadth-first
- Uniform-cost
- Depth-first
- Depth-limited
- Iterative deepening

**Informed:** Use heuristics to guide the search
- Best first
- A*  

Evaluation of search strategies

Search algorithms’ four criteria:
- **Completeness:**
- **Time complexity:**
- **Space complexity:**
- **Optimality:**

Complexity are measured in terms of:
- $b$ – max branching factor of the search tree
- $d$ – depth of the least-cost solution
- $m$ – max depth of the search tree (may be infinity)
Last time: uninformed search strategies

**Uninformed search:**

Use only information available in the problem formulation
- Breadth-first
- Uniform-cost
- Depth-first
- Depth-limited
- Iterative deepening

Comparing uninformed search strategies

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Breadth</th>
<th>Uniform</th>
<th>DepthFirst</th>
<th>DepthLim</th>
<th>Iterative</th>
<th>Bidirectional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>$b^d$</td>
<td>$b^d$</td>
<td>$b^m$</td>
<td>$b^l$</td>
<td>$b^d$</td>
<td>$b^{(d/2)}$</td>
</tr>
<tr>
<td>Space</td>
<td>$b^d$</td>
<td>$b^d$</td>
<td>$bm$</td>
<td>$bl$</td>
<td>$bd$</td>
<td>$b^{(d/2)}$</td>
</tr>
<tr>
<td>Optimal?</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Complete?</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes if $l \geq d$</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

$b$ – max branching factor of the search tree
$d$ – depth of the least-cost solution
$m$ – max depth of the state-space (may be infinity)
$l$ – depth cutoff
This time: informed search

**Informed search:**

Use heuristics to guide the search
- Best first
- A*
- Heuristics
- Hill-climbing
- Simulated annealing

**Best-first search**

**Idea:** use an evaluation function for each node; estimate of "desirability"

**Implementation:**

QueueingFn = insert successors in decreasing order of desirability

**Special cases:**
- greedy search
- A* search
Romania with step costs in km

Greedy search

- **Estimation function:**
  \[ h(n) = \text{estimate of cost from } n \text{ to goal (heuristic)} \]

- For example:
Greedy search example
Properties of Greedy Search

- Complete?
- Time?
- Space?
- Optimal?
Properties of Greedy Search

- Complete?
- Time?
- Space?
- Optimal?

A* search

- Idea: combine the advantages of uniform cost and greedy approach
A* search

- A* search uses an admissible heuristic,

- Theorem: A* search is optimal
Optimality of A* (standard proof)

$G_2$: suboptimal goal has been generated and is in the queue.

$n$: an unexpanded node on a shortest path to an optimal goal $G_j$. 
**Optimality of A* (more useful proof)**

**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 $f_i < f_{i+1}$

---

**f-contours**

How do the contours look like when $h(n) = 0$?
Properties of A*

- Complete?
- Time?
- Space?
- Optimal?

Proof of lemma: pathmax

For some admissible heuristics, $f$ may decrease along a path.
E.g., suppose $n'$ is a successor of $n$

```
  n  g=5  h=4  f=9
     1
  n' g'=6  h'=2  f'=8
```

But this throws away information!
$f(n) = 9 \Rightarrow$ true cost of a path through $n$ is $\geq 9$
Hence true cost of a path through $n'$ is $\geq 9$ also

Pathmax modification to A*:
Instead of $f(n') = g(n') + h(n')$, use $f(n') = \max(g(n') + h(n'), f(n))$

With pathmax, $f$ is always nondecreasing along any path.
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)

\[
\begin{array}{cccc}
5 & 4 & \text{Start State} \\
6 & 1 & 8 \\
7 & 3 & 2 \\
\end{array}
\]

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

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

Relaxed Problem

- How to determine an admissible heuristics?
Relaxed Problem

- Example:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
</tr>
</tbody>
</table>

Next time

- Iterative improvement
- Hill climbing
- Simulated annealing