Informed Search

Reading: Chapter 4, paper
HW#1 due today
HW#2 out today, due Feb. 19th
Heuristics

- Suppose 8-puzzle off by one (demo)
- Is there a way to choose the best move next?
- Good news: Yes!
  - We can use domain knowledge or heuristic to choose the best move
Nature of heuristics

- Domain knowledge: some knowledge about the game, the problem to choose
- Heuristic: a guess about which is best, not exact
- Heuristic function, $h(n)$: estimate the distance from current node to goal
Heuristic for the 8-puzzle

- # tiles out of place \((h_1)\)

- Manhattan distance \((h_2)\)
  - Sum of the distance of each tile from its goal position
  - Tiles can only move up or down \(\rightarrow\) city blocks
Goal State

\begin{align*}
\text{Goal State:} & \\
\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 0 \\
\end{array} \\
\text{h}_1 &= 1 \\
\text{h}_2 &= 1 \\
\text{h}_1 &= 1+1+1+2+2 = 7 \\
\text{h}_2 &= 1+1+1+2+2 = 7
\end{align*}
Best first searches

- A class of search functions
- Choose the “best” node to expand next
- Use an evaluation function for each node
  - Estimate of desirability
- Implementation: sort fringe, open in order of desirability
- Today: greedy search, A* search
**Greedy search**

- Evaluation function = heuristic function
- Expand the node that *appears* to be closest to the goal
Greedy Search

- OPEN = start node; CLOSED = empty
- While OPEN is not empty do
  - Remove leftmost state from OPEN, call it X
  - If X = goal state, return success
  - Put X on CLOSED
  - SUCCESSORS = Successor function (X)
  - Remove any successors on OPEN or CLOSED
  - Compute heuristic function for each node
  - Put remaining successors on either end of OPEN
  - Sort nodes on OPEN by value of heuristic function
- End while
Demo
Analysis of Greedy Search

- Which uninformed search algorithm is it closest to?
- Optimal?
- Complete?
- Space?
- Time?
A* Search

- Try to expand node that is on least cost path to goal
- Evaluation function = $f(n)$
  - $f(n) = g(n) + h(n)$
  - $h(n)$ is heuristic function: cost from node to goal
  - $g(n)$ is cost from initial state to node
- $f(n)$ is the estimated cost of cheapest solution that passes through node $n$
- If $h(n)$ is an underestimate of true cost to goal
  - A* is complete
  - A* is optimal
  - A* is optimally efficient: no other algorithm using $h(n)$ is guaranteed to expand fewer states
Admissable heuristics

- A heuristic that never overestimates the cost to the goal

- $h_1$ and $h_2$ are admissable heuristics

- Consistency: the estimated cost of reaching the goal form n is no greater than the step cost of getting to $n'$ plus estimated cost to goal from $n'$
  - $h(n) \leq c(n,a,n') + h(n')$
Which heuristic is better?

- *Better* means that fewer nodes will be expanded in searches
- \( h_2 \) dominates \( h_1 \) if \( h_2 \geq h_1 \) for every node \( n \)
- Intuitively, if both are underestimates, then \( h_2 \) is more accurate
- Using a more informed heuristic is guaranteed to expand fewer nodes of the search space
- Which heuristic is better for the 8-puzzle?
Demo
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:

$d = 14$  \hspace{1em} \text{IDS} = 3,473,941$ nodes
\hspace{1em} $A^*(h_1) = 539$ nodes
\hspace{1em} $A^*(h_2) = 113$ nodes

$d = 24$  \hspace{1em} \text{IDS} \approx 54,000,000,000$ nodes
\hspace{1em} $A^*(h_1) = 39,135$ nodes
\hspace{1em} $A^*(h_2) = 1,641$ nodes
Relaxed Problems

- Admissable heuristics can be derived from the exact solution cost of a relaxed version of the problem.
- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1$ gives the shortest solution.
- If the rules are relaxed so that a tile can move to any adjacent square, then $h_2$ gives the shortest solution.
- **Key**: the optimal solution cost of a relaxed problem is no greater than the optimal solution cost of the real problem.
Goal State

h₁ = 1
h₂ = 1

h₁ = 5
h₂ = 1 + 1 + 1 + 2 + 2 = 7
Heuristics for other problems

- Shortest path from one city to another
  - Straight line distance

- Touring problem: visit every city exactly once

- Cryptograms
Romania with step costs in km

Straight-line distance to Bucharest

<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>Dobrogea</td>
<td>242</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
</tr>
<tr>
<td>Fagaras</td>
<td>178</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
</tr>
<tr>
<td>Hirsova</td>
<td>151</td>
</tr>
<tr>
<td>Iasi</td>
<td>226</td>
</tr>
<tr>
<td>Lugoj</td>
<td>244</td>
</tr>
<tr>
<td>Mehadia</td>
<td>241</td>
</tr>
<tr>
<td>Neamt</td>
<td>234</td>
</tr>
<tr>
<td>Oradea</td>
<td>380</td>
</tr>
<tr>
<td>Pitesti</td>
<td>98</td>
</tr>
<tr>
<td>Rimnicu Vilcea</td>
<td>193</td>
</tr>
<tr>
<td>Sibiu</td>
<td>253</td>
</tr>
<tr>
<td>Timisoara</td>
<td>329</td>
</tr>
<tr>
<td>Urziceni</td>
<td>80</td>
</tr>
<tr>
<td>Vaslui</td>
<td>199</td>
</tr>
<tr>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>
Greedy search example

Arad
366
Greedy search example
Greedy search example
Greedy search example
A* search example

Arad
366=0+366
A* search example

- **Sibiu**: 393 = 140 + 253
- **Timisoara**: 447 = 118 + 329
- **Zerind**: 449 = 75 + 374
A* search example

Arad

Sibiu

Timisoara

Zerind

Arad 446=280+366
Fagaras 415=239+176
Oradea 671=291+380
Rimnicu Vilcea 413=220+193

447=118+329
449=75+374
A* search example

- Arad
  - Fagaras: 415=239+176
  - Oradea: 671=291+380
  - Rimnicu Vilcea
    - Craiova: 526=366+160
    - Pitesti: 417=317+100
    - Sibiu: 553=300+253
- Sibiu
- Timisoara: 447=118+329
- Zerind: 449=75+374
A* search example

```
Arad
  ├── Sibiu
  │    ├── Arad 646=280+366
  │    │    ├── Fagaras 591=338+253
  │    │    │    └── Bucharest 450=450+0
  │    │    │        ├── Craiova 526=366+160
  │    │    │        │    └── Pitesti 417=317+100
  │    │    │        │        └── Sibiu 553=300+253
  │    │    └── Oradea 671=291+380
  │    └── Rimnicu Vilcea
  └── Timisoara 447=118+329
      └── Zerind 449=75+374
```
Online Search

- Agent operates by interleaving computation and action
  - No time for thinking
- The agent only knows
  - Actions (s)
  - The step-cost function $c(s,a,s')$
  - Goal-text (s)
- Cannot access the successors of a state without trying all actions
Assumptions

- Agent recognizes a state it has seen before
- Actions are deterministic
- Admissable heuristics
  - Competitive ratio: Compare cost that agent actually travels with cost of the actual shortest path
What properties of search are desirable?

- Will A* work?

- Expand nodes in a local order
  - Depth first
  - Variant of greedy search

- Difference from offline search:
  - Agent must physically backtrack
  - Record states to which agent can backtrack and has not yet explored
**Depth-first**

- OPEN = start node; CLOSED = empty
- While OPEN is not empty do
  - Remove leftmost state from OPEN, call it X
  - If X = goal state, return success
  - Put X on CLOSED
  - SUCCESSORS = Successor function (X)
    - Remove any successors on OPEN or CLOSED
    - Put remaining successors on *left* end of OPEN
- End while
Online DFS - setup

- Inputs: $s'$, a percept that identifies the current state

Static:
- **result**, a table indexed by action and state, initially empty
- **unexplored**: a table that lists, for each visited state, the actions not yet tried
- **unbacktracked**: a table that lists, for each visited state, the backtracks not yet tried
- **$s,a$**: the previous state and action, initially null
Online DFS – the algorithm

- If Goal-test(s’) then return stop
- If s’ is a new state then unexplored[s’] ← actions(s’)
- If s is not null then do
  - result[a,s]←s’
  - Add s to the front of unbacktracked[s’]
- If unexplored[s’] is empty
  - Then return stop
  - Else a← action b such that result[b,s’]=pop(unbacktracked[s’])
- Else a←pop(unexplored[s’])
- s← s’
- Return a
Homework