Recall: Admissible Heuristics

- $f(x) = g(x) + h(x)$
- $g$: cost so far
- $h$: underestimate of remaining costs

Where do heuristics come from?
Relaxed Problems

- Derive admissible heuristic from **exact** cost of a solution to a **relaxed** version of problem

*For route planning, what is a relaxed problem?*

Relax requirement that car has to stay on road
Straight Line Distance becomes optimal cost

- Cost of optimal soln to relaxed problem ≤ cost of optimal soln for real problem

---

Heuristics for eight puzzle

- What can we relax?
Heuristics for eight puzzle

Original: Tile can move from location A to B if A is horizontally or vertically next to B and B is blank

Relaxed 1: Tile can move from any A to any B
Cost = \( h_1 = \) number of misplaced tiles

Relaxed 2: Tile can move from A to B if A is horizontally or vertically next to B
Cost = \( h_2 = \) total Manhattan distance

Importance of Heuristics

<table>
<thead>
<tr>
<th>d</th>
<th>IDS</th>
<th>( A^*(h1) )</th>
<th>( A^*(h2) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>10</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>112</td>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>680</td>
<td>20</td>
<td>18</td>
</tr>
<tr>
<td>8</td>
<td>6384</td>
<td>39</td>
<td>25</td>
</tr>
<tr>
<td>10</td>
<td>47127</td>
<td>93</td>
<td>39</td>
</tr>
<tr>
<td>12</td>
<td>364404</td>
<td>227</td>
<td>73</td>
</tr>
<tr>
<td>14</td>
<td>3473941</td>
<td>539</td>
<td>113</td>
</tr>
<tr>
<td>18</td>
<td>3056</td>
<td>363</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>39135</td>
<td>1641</td>
<td></td>
</tr>
</tbody>
</table>

Recall from last time: \( h_2 \) dominates \( h_1 \)
Need for Better Heuristics

Performance of $h_2$ (Manhattan Distance Heuristic)

- 8 Puzzle: < 1 second
- 15 Puzzle: 1 minute
- 24 Puzzle: 65000 years

Can we do better?

Creating New Heuristics

- Given admissible heuristics $h_1$, $h_2$, ..., $h_m$, none of them dominating any other, how to choose the best?

- Answer: No need to choose only one! Use:
  
  $$h(n) = \max \{h_1(n), h_2(n), \ldots, h_n(n)\}$$

- $h$ is admissible (why?)
- $h$ dominates all $h_i$ (by construction)
- Can we do better with:
  
  $$h'(n) = h_1(n) + h_2(n) + \ldots + h_n(n)$$
**Pattern Databases** [Culberson & Schaeffer 1996]

- **Idea:** Use solution cost of a subproblem as heuristic. For 8-puzzle: pick any subset of tiles
  - E.g., 3, 7, 11, 12
- **Precompute a table**
  - Compute optimal cost of solving just these tiles
  - This is a lower bound on actual cost with all tiles
  - For all possible configurations of these tiles
  - Could be several million
  - Use breadth first search back from goal state
    - State = position of just these tiles (& blank)
  - Admissible heuristic $h_{DB}$ for complete state = cost of corresponding sub-problem state in database

**Combining Multiple Databases**

- **Can choose another set of tiles**
  - Precompute multiple tables
- **How to combine table values?**
  - Use the *max* trick!

- **E.g. Optimal solutions to Rubik’s cube**
  - First found w/ IDA* using pattern DB heuristics
  - Multiple DBs were used (diff subsets of cubies)
  - Most problems solved optimally in 1 day
  - Compare with **574,000 years** for IDDFS
Drawbacks of Standard Pattern DBs

- Since we can only take \( \text{max} \)
  Diminishing returns on additional DBs

- Would like to be able to add values
  - But not exceed the actual solution cost (admissible)
  - How?

Disjoint Pattern DBs

- Partition tiles into disjoint sets
  For each set, precompute table
  Don’t count moves of tiles not in set
  - This makes sure costs are disjoint
  - Can be added without overestimating!
  - E.g. 8 tile DB has 519 million entries
  - And 7 tile DB has 58 million

- During search
  Look up costs for each set in DB
  Add values to get heuristic function value

Manhattan distance is a special case of this idea where each set is a single tile
Performance

- **15 Puzzle**: 2000x speedup vs Manhattan dist
  IDA* with the two DBs solves 15 Puzzles optimally in 30 milliseconds

- **24 Puzzle**: 12 millionx speedup vs Manhattan
  IDA* can solve random instances in 2 days.
  Requires 4 DBs as shown
  - Each DB has 128 million entries
  Without PDBs: 65000 years

Adapted from Richard Korf presentation

Enuff’bout heuristics - let’s investigate local search!
Local search algorithms

- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution.
- Find configuration satisfying constraints, e.g., $n$-queens.
- In such cases, we can use local search algorithms.
- Keep a single "current" state, try to improve it.

Example: $n$-queens

- Put $n$ queens on an $n \times n$ board with no two queens on the same row, column, or diagonal.
Hill-climbing search

• "Like climbing Everest in thick fog with amnesia"

function Hill-Climbing(problem) returns a state that is a local maximum
inputs: problem, a problem
local variables: current, a node
neighbor, a node

current <- Make-Node(Initial-State[problem])
loop do
    neighbor <- a highest-valued successor of current
    if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
    current <- neighbor

Hill-climbing search

• Problem: depending on initial state, can get stuck in local maxima
Example: 8-queens problem

1. $h$ = number of pairs of queens that are attacking each other, either directly or indirectly
2. $h = 17$ for the above state

Heuristic?

Example: 8-queens problem

1. A local minimum with $h = 1$. Need $h = 0$
2. How to find global minimum/maximum?
Simulated Annealing

• Idea: escape local maxima by allowing some "bad" moves but gradually decrease their frequency

function Simulated-Annealing(problem, schedule) returns a solution state
inputs: problem, a problem
        schedule, a mapping from time to "temperature"
local variables: current, a node
                next, a node
        T, a "temperature" controlling prob. of downward steps

current ← Make-Node(Initial-State[problem])
for t ← 1 to ∞ do
    T ← schedule[t]
    if T = 0 then return current
    next ← a randomly selected successor of current
    ΔE ← Value[next] − Value[current]
    if ΔE > 0 then current ← next
    else current ← next only with probability e^ΔE/T

Properties of simulated annealing

• One can prove: If T decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1

• Widely used in VLSI layout, airline scheduling, etc
Local Beam Search

- Keep track of $k$ states rather than just one
- Start with $k$ randomly generated states
- At each iteration, all the successors of all $k$ states are generated
- If any one is a goal state, stop; else select the $k$ best successors from the complete list and repeat.

Next Time

- Gaming search and searching for Games
- Homework #1 due

Have a great weekend!