Search thru a Problem Space / State Space

• Input:
  - Set of states
  - Operators [and costs]
  - Start state
  - Goal state [test]

• Output:
  - Path: start $\Rightarrow$ a state satisfying goal test
  - [May require shortest path]
  - [Sometimes just need state passing test]
Heuristics

It’s what makes search actually work
Traveling Salesman

- **Input**: undirected graph
- **Output**: connected path traversing each vertex *exactly* once
- **As a search problem**
  - States?
    - Graphs w/ partial paths
  - Operators?
    - Adding a edge to the path
Traveling Salesman

- **Input:** undirected graph
- **Output:** connected path traversing each vertex *exactly* once

- **As a search problem**
  - **States?** Graphs w/ partial paths
  - **Operators?** Adding a edge to the path
  - **Heuristic estimate of cost to complete a path?**
    - What to relax?
    - What is a path?
      - Subgraph…
      - Degree 2
    - Min spanning tree
      - $O(n^2)$
Heuristics for eight puzzle

What can we relax?

- h1 = number of tiles in wrong place
  - start
- h2 = \( \sum \) distances of tiles from correct loc
  - goal
Relaxed Problem

- Can describe move as a Strips operator
- Predicates:
  - On(x,y) tile x is on cell y
  - Clear(y) no tiles are on cell y
  - Adj(y, z) cell y is adjacent to cell z
- States are conjunctions, eg initial state:
  - On(6,1-1), On(3, 2-1), …, Clear(1-2), Adj(1-1, 1-2), Adj(…
- Move(x,y,z)
  - Preconditions: on(x,y), clear(z), adj(y,z)
  - Add-list: on(x,z), clear(y)
  - Delete-list: on(x,y), clear(z)
Relaxed Problem

- Can describe move as a Strips operator

- Predicates:
  - On(x,y) tile x is on cell y
  - Clear(y) no tiles are on cell y
  - Adj(y, z) cell y is adjacent to cell z

- States are conjunctions, eg initial state:
  - On(6,1-1), On(3, 2-1), …, Clear(1-2), Adj(1-1, 1-2), Adj(…

- Move(x,y,z)
  - Preconditions: on(x,y), clear(z), adj(y,z)
  - Add-list: on(x,z), clear(y)
  - Delete-list: on(x,y), clear(z)
**Importance of Heuristics**

$h_1 = \text{number of tiles in wrong place}$

$h_2 = \sum \text{distances of tiles from correct loc}$

<table>
<thead>
<tr>
<th>D</th>
<th>IDS</th>
<th>$A^*(h_1)$</th>
<th>$A^*(h_2)$</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></td>
<td>3056</td>
<td>363</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td>39135</td>
<td>1641</td>
</tr>
</tbody>
</table>

Decrease effective branching factor
Need More Power!

Performance of Manhattan Distance Heuristic

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

Need even better heuristics!

© Daniel S. Weld
Subgoal Interactions

- Manhattan distance assumes
  - Each tile can be moved independently of others
- Underestimates because
  - Doesn’t consider interactions between tiles
Pattern Databases

- Pick any subset of tiles
  - E.g., 3, 7, 11, 12, 13, 14, 15
  - (or as drawn)
- Precompute a table
  - Optimal cost of solving just these tiles
  - For all possible configurations
    - 57 Million in this case
  - Use A* or IDA*
    - State = position of just these tiles (& blank)
Using a Pattern Database

- As each state is generated
  - Use position of chosen tiles as index into DB
  - Use lookup value as heuristic, \( h(n) \)

- Admissible?
- Monotonic?
Combining Multiple Databases

- Can choose another set of tiles
  - Precompute multiple tables
- How combine table values?
  - Min, Max, Sum, RandomlyChoose
- E.g. Optimal solutions to Rubik’s cube
  - First found w/ IDA* using pattern DB heuristics
  - Multiple DBs were used (dif cubie subsets)
  - Most problems solved optimally in 1 day
  - Compare with 574,000 years for IDDFS
Drawbacks of Standard Pattern DBs

- Since we can only take $max$
  - Diminishing returns on additional DBs

- Would like to be able to add values

© Daniel S. Weld
Adapted from Richard Korf presentation
Disjoint Pattern DBs

- Partition tiles into **disjoint** sets
  - For each set, precompute table
    - E.g. 8 tile DB has 519 million entries
    - And 7 tile DB has 58 million

- During search
  - Look up heuristic values for each set
  - *Can add values without overestimating!*

- 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 shown previously solves 15 Puzzles optimally in 30 milliseconds

- **24 Puzzle:** 12 million x speedup vs Manhattan
  - IDA* can solve random instances in 2 days.
  - Requires 4 DBs as shown
    - Each DB has 128 million entries
  - Without PDBs: 65,000 years
Alternative Approach…

- Optimality is nice to have, but…

- Sometimes space is too vast! Find suboptimal solution using local search.