Recap: Search Problem

- States
  - configurations of the world
- Successor function:
  - function from states to lists of (state, action, cost) triples
- Start state
- Goal test

General Tree Search Paradigm

function tree-search(root-node)
  fringe ← successors(root-node)
  while ( notempty(fringe) )
    {node ← remove-first(fringe)
      state ← state(node)
      if goal-test(state) return solution(node)
      fringe ← insert-all(successors(node), fringe) }
  return failure
end tree-search

Extra Work?

- Failure to detect repeated states can cause exponentially more work (why?)

General Graph Search Paradigm

function tree-search(root-node)
  fringe ← successors(root-node)
  explored ← empty
  while ( notempty(fringe) )
    {node ← remove-first(fringe)
      state ← state(node)
      if goal-test(state) return solution(node)
      explored ← insert(node, explored)
      fringe ← insert-all(successors(node), fringe, if node not in explored) }
  return failure
end tree-search

Some Hints

- Graph search is almost always better than tree search (when not?)
  - Implement your closed list as a dict or set!
  - Nodes are conceptually paths, but better to represent with a state, cost, last action, and reference to the parent node
Informed (Heuristic) Search

Idea: be smart about what paths to try.

Blind Search vs. Informed Search

- What’s the difference?
- How do we formally specify this?

A node is selected for expansion based on an evaluation function that estimates cost to goal.

Best-First Search

- Use an evaluation function \( f(n) \) for node \( n \).
- Always choose the node from fringe that has the lowest \( f \) value.
  - Fringe = priority queue

Uniform Cost Search

- \( f(n) = \text{cost from root} \)
- The good: UCS is complete and optimal!
  - \( c \leq 3 \)
  - \( c \leq 2 \)
  - \( c \leq 1 \)
- The bad:
  - Explores options in every “direction”
  - No information about goal location

Greedy Search

- \( f(n) = \text{estimate of cost from } n \text{ to goal} \)
- A common case:
  - Takes you straight to the (wrong) goal
- Worst-case: like a badly-guided DFS

A* search

- \( f(n) = \text{estimated total cost of path thru } n \text{ to goal} \)
- \( f(n) = g(n) + h(n) \)
- \( g(n) = \text{cost so far to reach } n \)
- \( h(n) = \text{estimated cost from } n \text{ to goal} \)
  - (satisfying some important conditions)
Admissible heuristics

- A heuristic $h(n)$ is admissible if for every node $n$, $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from $n$.
- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic.
- Example: $h_{SLD}(n)$ (never overestimates the actual road distance).
- Theorem: If $h(n)$ is admissible, $A^*$ using TREE-SEARCH is optimal.

Consistent Heuristics

- $h(n)$ is consistent if:
  - for every node $n$
  - for every successor $n'$ due to legal action $a$
  - $h(n) \leq c(n,a,n') + h(n')$
- Every consistent heuristic is also admissible.
- Theorem: If $h(n)$ is consistent, $A^*$ using GRAPH-SEARCH is optimal.

When should $A^*$ terminate?

- Should we stop when we enqueue a goal?
- No: only stop when we dequeue a goal.

Which Algorithm?

Which Algorithm?
Which Algorithm?

- Uniform cost search (UCS):

Which Algorithm?

- A*, Manhattan Heuristic:

Which Algorithm?

- Best First / Greedy, Manhattan Heuristic:

Properties of A*

- Uniform-Cost
- A*

UCS vs A* Contours

- Uniform-cost expanded in all directions
- A* expands mainly toward the goal, but does hedge its bets to ensure optimality

Heuristics

It’s what makes search actually work
Admissable 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 transportation planning, relax requirement that car has to stay on road \( \rightarrow \) Euclidean dist
  - For blocks world, distance = # move operations; heuristic = number of misplaced blocks

What is relaxed problem?

- Cost of optimal soln to relaxed problem \( \leq \) cost of optimal soln for real problem

What's being relaxed?

Example: Pancake Problem

Action: Flip over the top \( n \) pancakes

Cost: Number of pancakes flipped

Example: Pancake Problem

State space graph with costs as weights
Example: Heuristic Function

Heuristic: the largest pancake that is still out of place

Traveling Salesman Problem

Heuristics for eight puzzle

- What can we relax?

Importance of Heuristics

h1 = number of tiles in wrong place

<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>39</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>

Importance of Heuristics

h1 = number of tiles in wrong place
h2 = \( \sum \) distances of tiles from correct loc

Combining Admissible Heuristics

- Can always take max
- Adding does not preserve admissibility in general

Decrease effective branching factor
Performance of IDA* on 15 Puzzle

- Random 15 puzzle instances were first solved optimally using IDA* with Manhattan distance heuristic (Korf, 1985).
- Optimal solution lengths average 53 moves.
- 400 million nodes generated on average.
- Average solution time is about 50 seconds on current machines.

Limitation of Manhattan Distance

- To solve a 24-Puzzle instance, IDA* with Manhattan distance would take about 65,000 years on average.
- Assumes that each tile moves independently.
- In fact, tiles interfere with each other.
- Accounting for these interactions is the key to more accurate heuristic functions.

Example: Linear Conflict

```plaintext
3 1
```

Manhattan distance is 2+2=4 moves

```plaintext
1 3
```

Example: Linear Conflict

```plaintext
3 1
```

Manhattan distance is 2+2=4 moves

```plaintext
1 3
```

Example: Linear Conflict

```plaintext
3 1
```

Manhattan distance is 2+2=4 moves

```plaintext
1 3
```
Example: Linear Conflict

Manhattan distance is 2+2=4 moves

Example: Linear Conflict

Manhattan distance is 2+2=4 moves

Example: Linear Conflict

Manhattan distance is 2+2=4 moves, but linear conflict adds 2 additional moves.

Linear Conflict Heuristic

- Hansson, Mayer, and Yung, 1991
- Given two tiles in their goal row, but reversed in position, additional vertical moves can be added to Manhattan distance.
- Still not accurate enough to solve 24-Puzzle
- We can generalize this idea further.

Pattern Database Heuristics

- Culberson and Schaeffer, 1996
- A pattern database is a complete set of such positions, with associated number of moves.
- e.g. a 7-tile pattern database for the Fifteen Puzzle contains 519 million entries.

Heuristics from Pattern Databases

31 moves is a lower bound on the total number of moves needed to solve this particular state.
Combining Multiple Databases

Additive Pattern Databases

- Culberson and Schaeffer counted all moves needed to correctly position the pattern tiles.
- In contrast, we count only moves of the pattern tiles, ignoring non-pattern moves.
- If no tile belongs to more than one pattern, then we can add their heuristic values.
- Manhattan distance is a special case of this, where each pattern contains a single tile.

Example Additive Databases

Computing the Heuristic

- Overall heuristic is maximum of 31 moves

Drawbacks of Standard Pattern DBs

- Since we can only take max
- Diminishing returns on additional DBs
- Would like to be able to add values

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