CSE 473: Artificial Intelligence
Autumn 2016

Search: Heuristics and Pattern DBs

Travis Mandel
(subbing for Dan Weld)

With slides from
Dan Weld, Dan Klein, Stuart Russell, Andrew Moore, Luke Zettlemoyer

Announcements

P0: You’re good unless you saw an email from us

Now in More 220!

Project 1: “Search” - due Friday 10/14
Should have started by now!

Dan will be back Friday!
Search thru a Problem Space / State Space

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

• Output:
  ▪ Path: start ⇒ a state satisfying goal test
  ▪ [May require shortest path]
  ▪ [Sometimes just need state passing test]

Tree vs Graph search

- In BFS, for example, we shouldn’t bother expanding the circled nodes (why?)
Graph Search

- Very simple fix: never expand a state type twice

function Graph-Search( problem, fringe) returns a solution, or failure

closed — an empty set
fringe ← Insert(Make-Node(Initial-State[problem]), fringe)
loop do
  if fringe is empty then return failure
  node ← Remove-Front(fringe)
  if Goal-Test(problem, State[node]) then return node
  if State[node] is not in closed then
    add State[node] to closed
    fringe ← InsertAll(Expand(node, problem), fringe)
  end
end

Some Hints

- Graph search is almost always better than tree search

- Implement your closed list as a dict or set!

- Space huge concern!
Search with Heuristics

A* Search

Hart, Nilsson & Rafael 1968

Best first search with $f(n) = g(n) + h(n)$
- $g(n)$ = sum of costs from start to $n$
- $h(n)$ = estimate of lowest cost path $n \rightarrow$ goal
  $h(\text{goal}) = 0$
A* Search

Hart, Nilsson & Rafael 1968

Best first search with $f(n) = g(n) + h(n)$
- $g(n) = \text{sum of costs from start to } n$
- $h(n) = \text{estimate of lowest cost path } n \rightarrow \text{goal}$
  $h(\text{goal}) = 0$

Can view as cross-breed:
- $g(n) \sim \text{uniform cost search}$
- $h(n) \sim \text{greedy search}$

Best of both worlds…

Admissible Heuristics

---

Slide credit: Travis Mandel
Monotonic/Consistent Heuristics

Monotonic

Not Monotonic (but admissible)

State (x)

Value

True (optimal) cost remaining

h(x) Heuristic-estimated cost remaining

Monotonic/Consistent Heuristics

Monotonic

Not Monotonic (but admissible)

State (x)

Value

True (optimal) cost remaining

h(x) Heuristic-estimated cost remaining

f(x) Heuristic + cost so far

Slide credit: Travis Mandel
Optimality of A* (tree search)

Suppose some suboptimal goal $G_2$ has been generated and is in the queue. Let $n$ be an unexpanded node on a shortest path to an optimal goal $G_1$.

\[
\begin{align*}
    f(G_2) &= g(G_2) \quad \text{since } h(G_2) = 0 \\
    &> g(G_1) \quad \text{since } G_2 \text{ is suboptimal} \\
    &\geq f(n) \quad \text{since } h \text{ is admissible}
\end{align*}
\]

Since $f(G_2) > f(n)$, A* will never select $G_2$ for expansion.

Optimality Continued

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

Arad
366=0+366

A* Example

Arad
Sibiu
393=140+253
Timisoara
447=118+329
Zerind
449=75+374
A* Example

A* Example
A* Example

10/5/2016
A* Summary

- **Pros**
  
  Produces optimal cost solution!

  Does so quite quickly (focused)

- **Cons**
  
  Maintains priority queue…

  Which can get exponentially big 😞
Iterative-Deepening A*

- Like iterative-deepening depth-first, but...
- Depth bound modified to be an f-limit
  - Start with f-limit = h(start)
  - Prune any node if f(node) > f-limit
  - Next f-limit = min-cost of any node pruned

IDA* Analysis

- Complete & Optimal (ala A*)
- Space usage $\propto$ depth of solution
- Each iteration is DFS - no priority queue!
- # nodes expanded relative to A*
  - Depends on # unique values of heuristic function
  - In 8 puzzle: few values $\Rightarrow$ close to # A* expands
  - In traveling salesman: each f value is unique
    $\Rightarrow$ $1+2+\ldots+n = O(n^2)$ where $n$=nodes A* expands
    if $n$ is too big for main memory, $n^2$ is too long to wait!
Forgetfulness

- A* used exponential memory
- How much does IDA* use?
  - During a run?
  - In between runs?
    - SMA*

Heuristics

It's what makes search actually work
Dominance

If \( h_2(n) \geq h_1(n) \) for all \( n \) (both admissible) then \( h_2 \) dominates \( h_1 \)

\( h_2 \) is better - guaranteed never to expand more nodes.

Admissible Heuristics

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

Where do heuristics come from?

© Daniel S. Weld
Relaxed Problems

- Derive admissible heuristic from exact cost of a solution to a relaxed version of problem
  - For blocks world, distance = # move operations
  - heuristic = number of misplaced blocks
  - What is relaxed problem?

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

© Daniel S. Weld

What’s being relaxed?

Heuristic = Euclidean distance
Example: Pancake Problem

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

Cost: Number of pancakes flipped

Example: Pancake Problem

---

**BOUNDS FOR SORTING BY PREFIX REVERSAL**

William H. GATES  
*Microsoft, Albuquerque, New Mexico*

Christos H. PAPADIMITRIOU*†  
*Department of Electrical Engineering, University of California, Berkeley, CA 94720, U.S.A.*

Received 18 January 1978  
Revised 28 August 1978

For a permutation \( \sigma \) of the integers from 1 to \( n \), let \( f(\sigma) \) be the smallest number of prefix reversals that will transform \( \sigma \) to the identity permutation, and let \( f(n) \) be the largest such \( f(\sigma) \) for all \( \sigma \) in (the symmetric group) \( S_n \). We show that \( f(n) \leq (5n + 5)/3 \), and that \( f(n) \geq 17n/16 \) for \( n \) a multiple of 16. If, furthermore, each integer is required to participate in an even number of reversed prefixes, the corresponding function \( g(n) \) is shown to obey \( 3n/2 - 1 \leq g(n) \leq 2n + 3 \).
Example: Pancake Problem

State space graph with costs as weights

Pancake Heuristic?

Heuristic: the largest pancake that is still out of place
Traveling Salesman Problem

Objective: shortest path visiting every city

What can be Relaxed?

Groundedness.
If can fly to previously seen city $\rightarrow$ minimum spanning tree

Heuristics for eight puzzle

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

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

- What can we relax?

$h_1 =$ number of tiles in wrong place

$h_2 = \sum$ distances of tiles from correct loc
## Importance of Heuristics

**h1 = number of tiles in wrong place**

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

Decrease effective branching factor

**h2 = \( \sum \) distances of tiles from correct loc**

<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></td>
<td>3056</td>
<td>363</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td>39135</td>
<td>1641</td>
</tr>
</tbody>
</table>
Need More Power!

Performance of Manhattan Distance Heuristic

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

Need even better heuristics!

Subgoal Interactions

- Manhattan distance assumes
  - Each tile can be moved independently of others
- Underestimates because
  - Doesn’t consider interactions between tiles
Pattern Databases [Culberson & Schaeffer 1996]

- 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?
Combining Multiple Databases

- Can choose another set of tiles
  - Precompute multiple tables
- How combine table values?

- 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
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.

Beam Search

- Idea
  - Best first but only keep N best items on priority queue

- Evaluation
  - Complete?

  - Time Complexity?

  - Space Complexity?
Hill Climbing

**Idea**
- Always choose best child; no backtracking
- Beam search with $|\text{queue}| = 1$

**Problems?**
- Local maxima
- Plateaus
- Diagonal ridges