Search: Cost & Heuristics

Dan Weld

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]

Concept Learning

Labeled Training Examples

<p1,blond,32,mc,ok>
<p2,red,47,visa,ok>
<p3,blond,23,cash,ter>
<p4,…

Output: f: <pn…> ⇒ {ok, ter}

Graduation?

• Getting a BS in CSE as a search problem?
  (don’t think too hard)

• Space of States
• Operators
• Initial State
• Goal State

Depth First Search

• Maintain stack of nodes to visit
• Check path to root to prune duplicates

Evaluation

• Complete?  
  Not for infinite spaces

• Time Complexity?
  $O(b^m)$

• Space Complexity?
  $O(bm)$
Breadth First Search

- Maintain queue of nodes to visit
- Evaluation
  - Complete? Yes
  - Time Complexity? \( O(b^d) \)
  - Space Complexity? \( O(b^d) \)

Memory a Limitation?

- Suppose:
  - 2 GHz CPU
  - 4 GB main memory
  - 100 instructions / expansion
  - 5 bytes / node
  - 200,000 expansions / sec
    - Memory filled in 400 sec ... < 7 min

Iterative Deepening Search

- DFS with limit; incrementally grow limit
- Evaluation
  - Complete?
  - Time Complexity?
  - Space Complexity?
### Cost of Iterative Deepening

<table>
<thead>
<tr>
<th>b</th>
<th>ratio ID to DFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>1.5</td>
</tr>
<tr>
<td>10</td>
<td>1.2</td>
</tr>
<tr>
<td>25</td>
<td>1.08</td>
</tr>
<tr>
<td>100</td>
<td>1.02</td>
</tr>
</tbody>
</table>

### Speed

<table>
<thead>
<tr>
<th></th>
<th>BFS</th>
<th>Iter. Deep.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nodes</td>
<td>Time</td>
</tr>
<tr>
<td>8 Puzzle</td>
<td>$10^5$</td>
<td>.01 sec</td>
</tr>
<tr>
<td>2x2x2 Rubik’s</td>
<td>$10^6$</td>
<td>.2 sec</td>
</tr>
<tr>
<td>15 Puzzle</td>
<td>$10^{13}$</td>
<td>6 days</td>
</tr>
<tr>
<td>3x3x3 Rubik’s</td>
<td>$10^{19}$</td>
<td>68k yrs</td>
</tr>
<tr>
<td>24 Puzzle</td>
<td>$10^{25}$</td>
<td>12B yrs</td>
</tr>
</tbody>
</table>

Why the difference?
- Rubik has higher branch factor
- 15 puzzle has greater depth

### Costs on Actions

What does BFS do?
- ... finds the shortest path in terms of number of transitions.
- It does not find the least-cost path.

### Best-First Search

- Generalization of breadth-first search
- **Priority** queue of nodes to be explored
- Cost function $f(n)$ applied to each node

Add initial state to priority queue
While queue not empty
  - Node = head(queue)
  - If goal?(node) then return node
  - Add children of node to queue

### Priority Queue Refresher

- A priority queue is a data structure in which you can insert and retrieve (key, value) pairs with the following operations:
  - `pq.push(key, value)` inserts (key, value) into the queue.
  - `pq.pop()` returns the key with the lowest value, and removes it from the queue.

- You can decrease a key’s priority by pushing it again
- Unlike a regular queue, insertions aren’t constant time, usually $O(\log n)$
- We’ll need priority queues for cost-sensitive search methods

### Old Friends

- **Breadth First = Best First**
  - With $f(n) = \text{depth}(n)$
- **Dijkstra’s Algorithm (Uniform cost) = Best First**
  - With $f(n) = \text{the sum of edge costs from start to } n$
Uniform Cost Search

Best first, where
\[ f(n) = \text{“cost from start to } n\text{”} \]

aka “Dijkstra’s Algorithm”

Uniform Cost Search

Algorithm | Complete | Optimal | Time | Space
--- | --- | --- | --- | ---
DFS | Y if finite | N | O(b^d) | O(b^d)
BFS | Y | Y* | O(b^d) | O(b^d)
UCS | Y* | Y | O(b^C*/\epsilon) | O(b^C*/\epsilon)

\( C^* \) tiers

\( C^* = \text{Optimal cost} \)
\( \epsilon = \text{Minimum cost of an action} \)

Uniform Cost Issues

- Remember: explores increasing cost contours
- The good: UCS is complete and optimal!
- The bad:
  - Explores options in every “direction”
  - No information about goal location

Uniform Cost: Pac-Man

- Cost of 1 for each action
- Explores all of the states, but one

Search Heuristics

- Any *estimate* of how close a state is to a goal
- Designed for a particular search problem

- Examples: Manhattan distance, Euclidean distance
Heuristics

Greedy Search

Best first with \( f(n) = \) heuristic estimate of distance to goal

Greedy Search

Expand the node that seems closest…

What can go wrong?

Greedy Search

• A common case:
  - Best-first takes you straight to the (wrong) goal

• Worst-case: like a badly-guided DFS in the worst case
  - Can explore everything
  - Can get stuck in loops if no cycle checking

  - Like DFS in completeness (if finite # states w/ cycle checking)

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 \( n \to \) goal
  \( h(\text{goal}) = 0 \)

If \( h(n) \) is admissible and monotonic then A* is optimal

\( f \) values increase from node to descendants (triangle inequality)
A* Example

Optimality of A*
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 $G_1$.

\[ f(G_2) = g(G_2) \quad \text{since} \quad h(G_2) = 0 \]

\[ g(G_1) \quad \text{since} \quad G_2 \text{ is suboptimal} \]

\[ \geq f(n) \quad \text{since} \quad h \text{ is admissible} \]

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* Summary

- **Pros**
- **Cons**

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
  \( \Rightarrow 1+2+...+n = O(n^2) \) where \( n \)=nodes A* expands
  - if \( n \) is too big for main memory, \( n^2 \) is too long to wait!
- Generates duplicate nodes in cyclic graphs

Forgetfulness

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

SMA*

- Use all available memory
- Start like A*
- When memory is full...
  - Erase node with highest f-value
  - First, backup parent with this f-value
  - So... parent knows cost-bound on best child

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 |queue| = 1

- Problems?
  - Local maxima
  - Plateaus
  - Diagonal ridges

Randomizing Hill Climbing

- Randomly disobeying heuristic
- Random restarts

(heavy tailed distributions)

→ Local Search

To Do:

- Look at the course website:
  - http://www.cs.washington.edu/cse473/12sp
- Do the readings (Ch 3)
- Start PS1,