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]

Graduation?

- Getting a BS in CSE as a search problem? (don’t think too hard)
  - Space of States
  - Operators
  - Initial State
  - Goal State

Search Methods

- Depth first search (DFS)
- Breadth first search (BFS)
- Iterative deepening depth-first search (IDS)

Heuristic search

- Best first search
- Uniform cost search (UCS)
- Greedy search
- $A^*$
- Iterative Deepening $A^*$ (IDA*)
- Beam search
- Hill climbing
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?
    - $O(bm)$

Memory a Limitation?
- Suppose:
  - 4 GHz CPU
  - 16 GB main memory
  - 100 instructions / expansion
  - 10 bytes / node
  - 400,000 expansions / sec
    - Memory filled in 400 sec … < 7 min

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

Iterative Deepening Search
- DFS with limit; incrementally grow limit
- Evaluation
  - Complete?
  - Time Complexity?
  - Space Complexity?

© Daniel S. Weld

© Daniel S. Weld
Iterative Deepening Search
- DFS with limit; incrementally grow limit
- Complete?
  - Yes
- Time Complexity?
  - $O(b^d)$
- Space Complexity?
  - $O(b^d)$

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>Puzzle</th>
<th>BFS</th>
<th>Iter. Deep.</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 Puzzle</td>
<td>$10^5$</td>
<td>$10^5$</td>
</tr>
<tr>
<td>2x2x2 Rubik’s</td>
<td>$10^6$</td>
<td>$10^6$</td>
</tr>
<tr>
<td>15 Puzzle</td>
<td>$10^{13}$</td>
<td>1Mx</td>
</tr>
<tr>
<td>3x3x3 Rubik’s</td>
<td>$10^{19}$</td>
<td>8x</td>
</tr>
<tr>
<td>24 Puzzle</td>
<td>$10^{25}$</td>
<td>$10^{37}$</td>
</tr>
</tbody>
</table>

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

Costs on Actions

Objective: Path with smallest overall cost

Best-First Search
- Generalization of breadth-first search
- Fringe = Priority queue of nodes to be explored
- Cost function $f(n)$ applied to each node

BFS

What will BFS return?
- … finds the shortest path in terms of number of transitions.
- It does not find the least-cost path.
### 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 \(f(n) = \text{the sum of edge costs from start to } n\)

- **Dijkstra’s Algorithm (Uniform cost)** =
  - Best First
  - with \(f(n) = \text{the sum of edge costs from start to } n\)

- **Old Friends**

---

### Uniform Cost Search

- **Best-First Search**
  - Generalization of breadth-first search
  - Fringe = *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
  ```

### Uniform Cost Search

- **Best-First Search**
  - Generalization of breadth-first search
  - Fringe = *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
  ```

### Uniform Cost Search

#### Expansion order:

- **S, p, d, b, e, a, f, e, G**

#### Cost contours (not all shown)

- **C* / \(\varepsilon\)** tiers
  - \(C^*\) = Optimal cost
  - \(\varepsilon\) = Minimum cost of an action

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS</td>
<td>Y if finite</td>
<td>N</td>
<td>(O(b^d))</td>
<td>(O(b^d))</td>
</tr>
<tr>
<td>BFS</td>
<td>Y</td>
<td>Y*</td>
<td>(O(b^d))</td>
<td>(O(b^d))</td>
</tr>
<tr>
<td>UCS</td>
<td>Y*</td>
<td>Y</td>
<td>(O(b^{d*}))</td>
<td>(O(b^{d*}))</td>
</tr>
</tbody>
</table>
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

What is a Heuristic?

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

What is a Heuristic?

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

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 a (suboptimal) 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 $n \rightarrow$ goal
  - $h(\text{goal}) = 0$

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

Underestimates cost of reaching goal from node

$f$ values increase from node to descendants (triangle inequality)

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
  - $h(\text{goal}) = 0$

Can view as cross-breed:
- $g(n)$ ~ uniform cost search
- $h(n)$ ~ greedy search

Best of both worlds…

Is Manhattan distance admissible?
- Underestimate?

Is Manhattan distance monotonic?
- $f$ values increase from node to children
- (triangle inequality)

Euclidean Distance
- Admissible?
- Monotonic?
European 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 $G_1$.

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

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**
  - Produces optimal cost solution!
  - Does so quite quickly (focused)

- **Cons**
  - Maintains priority queue…
  - Which can get exponentially big 😞