Overview

- Lowest-Cost-First
- Best-First Search
- A* Search
- Iterative Deepening

Lowest-Cost-First Search

- Sometimes there are costs associated with arcs. The cost of a path is the sum of the costs of its arcs.
- When all costs are equal => depth-first search.
- At each stage, it selects the shortest path on the frontier.
- Frontier is implemented as a priority queue ordered by g.
- Lowest-cost-first search finds the shortest path to a goal node.
- Lowest-cost-first search is a best-first search where the frontier is implemented as a priority queue ordered by g.
- In each stage, it selects the shortest path on the frontier.
Applying Best-First Search to Top-Down Proofs

How could we use this in searching through resolutions?

Best-first Search

Idea: always select node on the frontier with smallest h-value

Uses space exponential in path length
Treat the frontier as a priority queue ordered by h

Heuristic Search

• Previous methods do not take into account goal until goal
• When there is extra knowledge we can be used to guide the search
• Heuristics: heuristics use only readily obtainable information about a node

Previous methods do not take into account goal until goal
Overview

• Lowest-Cost-First
• Best-First Search
⇒ A* Search
• Iterative Deepening

Example with Top-Down Theorem Proving

Not guaranteed to find a solution, even if one exists

g ← a
a ← b
d
¬ e

Seems like a good idea but...

It doesn’t always find the shortest path

No guarantee to find a solution, even if one exists
Proof that if it finds a path, the path is optimal

• Let an optimal path have weight $f^*$.
• Cells in the frontier are ordered by $g(n) + h(n)$.
  - Where $g(n)$ is strictly increasing as you go down the path.
  - And $h(n)$ is a lower-estimate of the remaining distance.
• Assume A* stops at a goal node with non-optimal path $p$.
• So, $p$ was on top of the frontier.
• Since $p$ is not optimal, $g(p) > f^*$.
• Since $p$ ends at the goal, $g(p) = f(p)$, and so $f(p) > f^*$.
• But, part of the optimal path will be in the frontier, and it will have an $f$-value $\leq f^*$.
  - Hence, it would have been higher in the frontier than $p$.
  - And if it had been in the frontier, $f < (d)f = (d)g + (d)h$.
  - Hence, if it finds an $f$-value of some path, that path is not optimal.
  - But, part of the optimal path will be in the frontier.
  - Hence, it would have been higher in the frontier than that path.

A* Finds Optimal Solution

• If there is a solution, A* always finds an optimal solution.
• A* orders the frontier by $f(n)$.
  - A* stops when the node in the frontier is a goal node.
  - And so $d$ would not have been chosen.
  - Hence, it would have been higher in the frontier than $d$.
  - And if it had been in the frontier, $f < (d)f = (d)g + (d)h$.
  - Hence, if it finds an $f$-value of some path, that path is not optimal.
  - But, part of the optimal path will be in the frontier.
  - Hence, it would have been higher in the frontier than that path.
  - But, part of the optimal path will be in the frontier.
  - Hence, it would have been higher in the frontier than that path.

A* Search

• A* orders the frontier by $f(n)$.
  - A* starts from the start and heuristic value into account.
  - At each step it adds the cost from the start to the node.
  - Then, for each new frontier node, $f(n)$ is increased by the heuristic.
  - At each level the path to the goal is calculated.
  - At each level the path to the goal is calculated.
  - At each level the path to the goal is calculated.
  - At each level the path to the goal is calculated.
  - At each level the path to the goal is calculated.
  - At each level the path to the goal is calculated.
  - At each level the path to the goal is calculated.
  - At each level the path to the goal is calculated.
  - At each level the path to the goal is calculated.
Overview

- Lowest-Cost-First
- Best-First Search
- A* Search
- Iterative Deepening

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Frontier Selection</th>
<th>Halts?</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth-first</td>
<td>Last node added</td>
<td>Yes</td>
<td>Exp</td>
</tr>
<tr>
<td>Breadth-first</td>
<td>First node added</td>
<td>No</td>
<td>Exp</td>
</tr>
<tr>
<td>Best-first</td>
<td>Last node added</td>
<td>No</td>
<td>Exp</td>
</tr>
<tr>
<td>Lowest-cost-first</td>
<td>Global min g(n)</td>
<td>Yes</td>
<td>Exp</td>
</tr>
<tr>
<td>A*</td>
<td>Global min f(n)</td>
<td>Yes</td>
<td>Exp</td>
</tr>
</tbody>
</table>

Summary of Search Strategies

- We haven't stopped anywhere.
- At most \( n \) steps, an optimal path must be on top of frontier if
- \( f \)-score always is minimal.
- A subpath of the optimal path is always in frontier and its \( f \)-score of subpath is greater than its \( g \)-score.
- Hence, the number of subpaths with \( f \)-score is at most \( n \).
- Each edge has weight at least \( \epsilon \) and finite branching.
- Only a finite number of subpaths have \( f \)-score \( \leq \epsilon \).
- Let an optimal path have \( f \)-score \( \epsilon \).
- Proof that it will find a path.
Depth-bounded depth-first search

- A bit different from previous versions

- Gather all neighbors and then non-deterministically choose one

\[
\text{dbsearch}(\text{Node}, D, \text{NewP}) \\
\text{if } D > 0 \land \text{neighbors}(\text{Node}, \text{Neighbors}) \land \text{member}(\text{NewNode}, \text{Neighbors}) \land \text{non-deterministic} \\
\quad \text{D1 = D - 1} \\
\quad \text{dbsearch}(\text{NewNode}, D1, P) \\
\quad \text{NewP = [Node | P].}
\]

- Build the path on the way out

Depth-bounded depth-first search

- Theorem

If proof cannot be found at depth B, look for proof at depth B + 1

- Look for proofs of depth 0, then 1, then 2, then 3, etc.

- The idea: As we compute elements of the frontier rather than saving them

- So far all search strategies that are guaranteed to halt we