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 $g$ is the sum of the costs of its arcs.
- Lowest-cost-first search finds the shortest path to a goal node.
- Frontier is implemented as a priority queue ordered by $g$.
  - At each stage, it selects the shortest path on the frontier.
- When arc costs are equal $\Rightarrow$ breadth-first search.
Overview

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

• Previous methods do not take into account goal until at goal node
• Often there is extra knowledge that can be used to guide the search: heuristics
• Use $h(n)$ as estimate of distance from node $n$ to a goal node
• $h(n)$ is underestimate if it is less than or equal to the actual cost of the shortest path from node $n$ to a goal
• $h(n)$ uses only readily obtainable information about a node
Best-first Search

• Idea: always select node on the frontier with smallest $h$-value
• Treat the frontier as a priority queue ordered by $h$
• Uses space exponential in path length
Applying Best-First Search to Top-Down Proofs

- How could we use this in searching through resolutions?

\[
\begin{align*}
    a & \leftarrow b \land c. & a & \leftarrow g. & \text{yes} \leftarrow b \land c \land d & \text{yes} \leftarrow g \land d \\
    a & \leftarrow h. & b & \leftarrow j. & \text{yes} \leftarrow j \land c \land d & \text{yes} \leftarrow f \land d \\
    b & \leftarrow k. & d & \leftarrow m. & \text{yes} \leftarrow k \land c \land d & \text{yes} \leftarrow p \land d \\
    d & \leftarrow p. & f & \leftarrow m. & \text{yes} \leftarrow m \land c \land d & \text{yes} \leftarrow d \\
    f & \leftarrow p. & g & \leftarrow m. & \text{yes} \leftarrow m \land c \land d & \text{yes} \leftarrow m \\
    g & \leftarrow f. & k & \leftarrow m. & \text{yes} \leftarrow p \\
    h & \leftarrow m. & p. & & \text{yes} \leftarrow \\
\end{align*}
\]

\(?a \land d\)
• Not guaranteed to find a solution, even if one exists

• It doesn’t always find the shortest path
Example with Top-Down Theorem Proving

• Not guaranteed to find a solution, even if one exists
  \[ g \leftarrow a \]
  \[ g \leftarrow d \land e \]
  \[ a \leftarrow b \]
  \[ b \leftarrow a \]
  \[ d \]
  \[ e \]
  \[ ? \ g \]
Overview

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

- A* search takes path to a node and heuristic value into account
  - \( g(n) \) be the cost of the path found to node \( n \)
    + From lowest-cost first search
  - \( h(n) \) be the estimate of the cost from \( n \) to goal
    + From best-first search
  - Let \( f(n) = g(n) + h(n) \).
    + \( f(n) \) is estimate of path from start to goal via \( n \)

\[
\begin{align*}
\text{start} & \quad \rightarrow \quad \text{actual} \quad n \quad \rightarrow \quad \text{estimate} \quad \text{goal} \\
& \quad \underline{g(n)} \quad \underline{h(n)} \\
& \quad \underline{f(n)}
\end{align*}
\]

- A* orders the frontier by \( f(n) \)
  + Stops when min node in frontier is goal node
A* Finds Optimal Solution

• If there is a solution, A* always finds an optimal solution
  - the first path to goal that it finds is optimal

• If ...
  - the branching factor is finite (not necessarily a finite number of nodes)
  - arc costs are bounded above zero
    (there is some $\epsilon > 0$ such that all of the arc costs are greater than $\epsilon$)
  - $h(n)$ is an underestimate of the cost of the best path from $n$ to a goal node and $\geq 0$
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 $\geq 0$ 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 $\leq f^*$ (since $f$-values never over estimate)
  - Hence, it would have been higher in the frontier than $p$,
    and so $p$ would not have been chosen
Proof that it will find a path

• Let an optimal path have weight $f_1$

• Only a finite number of subpaths $m$ have $g$-score $\leq f_1$
  - Because each arc has weight at least $\epsilon$ and finite branching
  - Note: subpath might not end at a goal node and $g$-score measures the full cost of the subpath

• So, finite number of subpaths $n \leq m$ have $f$-score at most $f_1$
  - Because $f$-score of subpath is greater than its $g$-score

• A subpath of the optimal path is always in frontier and its $f$-score always at most $f_1$

• After at most $n$ steps, optimal path must be on top of frontier (if we haven’t stopped earlier)
## Summary of Search Strategies

<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>No</td>
<td>Linear</td>
</tr>
<tr>
<td>Breadth-first</td>
<td>First node added</td>
<td>Yes</td>
<td>Exp</td>
</tr>
<tr>
<td>Best-first</td>
<td>Global min (h(n))</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>
Overview

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

⇒ Iterative Deepening
Iterative Deepening

• So far all search strategies that are guaranteed to halt use exponential space

• Idea: let’s recompute elements of the frontier rather than saving them

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

• You need a depth-bounded depth-first searcher

• If proof cannot be found at depth $B$, look for proof at depth $B + 1$
Depth-bounded depth-first search

- $\text{dbsearch}(N, D, P)$ is true if $P$ is path of length $D$ from $N$ to goal

\[
\text{dbsearch}(\text{Node}, 0, [\text{Node}]) :- \\
\text{is\_goal(}\text{Node}).
\]

\[
\text{dbsearch}(\text{Node}, D, \text{NewP}) :- \\
D > 0, \\
\text{neighbors(}\text{Node, Neighbors}), \\
\text{member(}\text{NewNode, Neighbors}), \\
D1 \text{ is } D - 1, \\
\text{dbsearch(}\text{NewNode, D1, P}), \\
\text{NewP} = [\text{Node}|P].
\]

? dbsearch(start, 5, Path).

- A bit different from previous versions
  - Gathers up all neighbors, and then non-deterministically chooses one
  - Builds the path on the way out

© P. Heeman, 2020