Learning Objectives

At the end of the class you should be able to:

- explain how cycle checking and multiple-path pruning can improve efficiency of search algorithms
- explain the complexity of cycle checking and multiple-path pruning for different search algorithms
- justify why the monotone restriction is useful for A^* search
- ~~predict whether forward, backward, bidirectional or island-driven search is better for a particular problem~~
- ~~demonstrate how dynamic programming works for a particular problem~~

Summary of Search Strategies

Strategy	Frontier Selection	Complete	Halts	Space
Depth-first	Last node added			
Breadth-first	First node added			
Heuristic depth-first	Local min $h(p)$			
Best-first	Global min $h(p)$			
Lowest-cost-first	Minimal $cost(p)$			
A^*	Minimal $f(p)$			

Complete — if there a path to a goal, it can find one, even on infinite graphs.

Halts — on finite graph (perhaps with cycles).

Space — as a function of the length of current path

Summary of Search Strategies

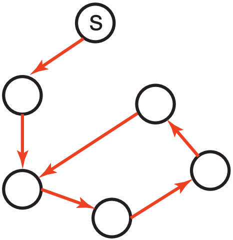
Strategy	Frontier Selection	Complete	Halts	Space
Depth-first	Last node added	No	No	Linear
Breadth-first	First node added	Yes	No	Exp
Heuristic depth-first	Local min $h(p)$	No	No	Linear
Best-first	Global min $h(p)$	No	No	Exp
Lowest-cost-first	Minimal $cost(p)$	Yes	No	Exp
A^*	Minimal $f(p)$	Yes	No	Exp

Complete — if there a path to a goal, it can find one, even on infinite graphs.

Halts — on finite graph (perhaps with cycles).

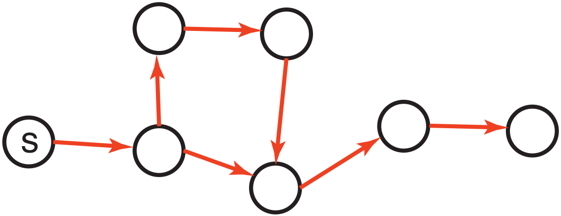
Space — as a function of the length of current path

Cycle Checking



- A searcher can prune a path that ends in a node already on the path, without removing an optimal solution.
- In depth-first methods, checking for cycles can be done in ___________ time in path length.
- For other methods, checking for cycles can be done in ___________ time in path length.
- Does cycle checking mean the algorithms halt on finite graphs?

Multiple-Path Pruning



- Multiple path pruning: prune a path to node n that the searcher has already found a path to.
- What needs to be stored?
- How does multiple-path pruning compare to cycle checking?
- Do search algorithms with multiple-path pruning always halt on finite graphs?
- What is the space & time overhead of multiple-path pruning?
- Can multiple-path pruning prevent an optimal solution being found?

Multiple-Path Pruning & Optimal Solutions

Problem: what if a subsequent path to n is shorter than the first path to n?

Multiple-Path Pruning & Optimal Solutions

Problem: what if a subsequent path to *n* is shorter than the first path to *n*?

- remove all paths from the frontier that use the longer path.
- change the initial segment of the paths on the frontier to use the shorter path.
- ensure this doesn't happen. Make sure that the shortest path to a node is found first.

Multiple-Path Pruning & A^*

- Suppose path p to n was selected, but there is a shorter path to n. Suppose this shorter path is via path p' on the frontier.
- Suppose path p' ends at node n'.
- p was selected before p', so:

Multiple-Path Pruning & A^*

- Suppose path p to n was selected, but there is a shorter path to n. Suppose this shorter path is via path p' on the frontier.
- Suppose path p' ends at node n'.
- p was selected before p', so:
  $cost(p) + h(n) \leq cost(p') + h(n')$.
- Suppose $cost(n', n)$ is the actual cost of a path from n' to n. The path to n via p' is shorter that p so:

Multiple-Path Pruning & A^*

- Suppose path p to n was selected, but there is a shorter path to n. Suppose this shorter path is via path p' on the frontier.
- Suppose path p' ends at node n'.
- p was selected before p', so:
  $cost(p) + h(n) \leq cost(p') + h(n')$.
- Suppose $cost(n', n)$ is the actual cost of a path from n' to n. The path to n via p' is shorter that p so:
  $cost(p') + cost(n', n) < cost(p)$.

$$cost(n', n) <$$

Multiple-Path Pruning & A^*

- Suppose path p to n was selected, but there is a shorter path to n. Suppose this shorter path is via path p' on the frontier.
- Suppose path p' ends at node n'.
- p was selected before p', so: $cost(p) + h(n) \leq cost(p') + h(n')$.
- Suppose $cost(n', n)$ is the actual cost of a path from n' to n. The path to n via p' is shorter that p so: $cost(p') + cost(n', n) < cost(p)$.

$$cost(n', n) < cost(p) - cost(p') \leq$$

Multiple-Path Pruning & A^*

- Suppose path p to n was selected, but there is a shorter path to n. Suppose this shorter path is via path p' on the frontier.
- Suppose path p' ends at node n'.
- p was selected before p', so: $cost(p) + h(n) \leq cost(p') + h(n')$.
- Suppose $cost(n', n)$ is the actual cost of a path from n' to n. The path to n via p' is shorter that p so: $cost(p') + cost(n', n) < cost(p)$.

$$cost(n', n) < cost(p) - cost(p') \leq h(n') - h(n).$$

We can ensure this doesn't occur if $|h(n') - h(n)| \leq cost(n', n)$.

Monotone Restriction

- Heuristic function h satisfies the monotone restriction if $|h(m) - h(n)| \leq cost(m, n)$ for every arc $\langle m, n \rangle$.
- If h satisfies the monotone restriction, A^* with multiple path pruning always finds the shortest path to a goal.
- This is a strengthening of the admissibility criterion.

Direction of Search

- The definition of searching is symmetric: find path from start nodes to goal node or from goal node to start nodes.
- Forward branching factor: number of arcs out of a node.
- Backward branching factor: number of arcs into a node.
- Search complexity is b^n. Should use forward search if forward branching factor is less than backward branching factor, and vice versa.
- Note: when graph is dynamically constructed, the backwards graph may not be available.

Bidirectional Search

- Idea: search backward from the goal and forward from the start simultaneously.
- This wins as $2b^{k/2} \ll b^k$. This can result in an exponential saving in time and space.
- The main problem is making sure the frontiers meet.
- This is often used with one breadth-first method that builds a set of locations that can lead to the goal. In the other direction another method can be used to find a path to these interesting locations.

Island Driven Search

- Idea: find a set of islands between s and g.

$$s \longrightarrow i_1 \longrightarrow i_2 \longrightarrow \ldots \longrightarrow i_{m-1} \longrightarrow g$$

  There are m smaller problems rather than 1 big problem.
- This can win as $mb^{k/m} \ll b^k$.
- The problem is to identify the islands that the path must pass through. It is difficult to guarantee optimality.
- The subproblems can be solved using islands $\Longrightarrow$ hierarchy of abstractions.

Dynamic Programming

~~Idea: for statically stored graphs, build a table of $dist(n)$ the actual distance of the shortest path from node n to a goal.~~

~~This can be built backwards from the goal:~~

$$\cancel{dist(n) = \begin{cases} 0 & \text{if } is\_goal(n), \\ \min_{\langle n,m\rangle \in A}(|\langle n, m\rangle| + dist(m)) & \text{otherwise.} \end{cases}}$$

~~This can be used locally to determine what to do.~~

~~There are two main problems:~~

- ~~It requires enough space to store the graph.~~
- ~~The *dist* function needs to be recomputed for each goal.~~