Announcements

- Solution to Assignment 1 is posted
- Assignment 2 is available

Review: Searching

- A frontier is a set of paths
- Generic search algorithm: Repeatedly:
  - select a path from the frontier
  - stop of it is a path to a goal
  - otherwise expand it in all ways, and add the resulting paths to the frontier
- Frontier is a stack $\longrightarrow$ depth-firt search
- Frontier is a queue $\longrightarrow$ breadth-firt search
- Frontier is a priority queue ordered by path cost $\longrightarrow$ least-cost-first search

AIspace examples

- Vancouver neighbourhood graph
- Misleading heuristic demo

How do good heuristics help?

Suppose c is the cost of an optimal solution. What happens to a path p from start, where

- $cost(p) + h(p) < c$

How do good heuristics help?

Suppose c is the cost of an optimal solution. What happens to a path p from start, where

- $cost(p) + h(p) < c$
  It will be expanded
- $cost(p) + h(p) > c$

How do good heuristics help?

Suppose c is the cost of an optimal solution. What happens to a path p from start, where

- $cost(p) + h(p) < c$
  It will be expanded
- $cost(p) + h(p) > c$
  It will not be expanded
- $cost(p) + h(p) = c$

How do good heuristics help?

Suppose c is the cost of an optimal solution. What happens to a path p from start, where

- $cost(p) + h(p) < c$
  It will be expanded
- $cost(p) + h(p) > c$
  It will not be expanded
- $cost(p) + h(p) = c$
  It might or might not be expanded.

How do good heuristics help?

Suppose c is the cost of an optimal solution. What happens to a path p from start, where

- $cost(p) + h(p) < c$
  It will be expanded
- $cost(p) + h(p) > c$
  It will not be expanded
- $cost(p) + h(p) = c$
  It might or might not be expanded.

How can a better heuristic function help?

Summary of Search Strategies

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

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
Lowest-cost-first	Minimal $cost(p)$	Yes	No	Exp
Best-first	Minimal $h(p)$	No	No	
A^*	Minimal $f(p)$	Yes		

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
Lowest-cost-first	Minimal $cost(p)$	Yes	No	Exp
Best-first	Minimal $h(p)$	No	No	
A^*	Minimal $f(p)$	Yes	No	

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
Lowest-cost-first	Minimal $cost(p)$	Yes	No	Exp
Best-first	Minimal $h(p)$	No	No	Exp
A^*	Minimal $f(p)$	Yes	No	

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

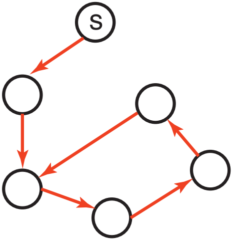
Strategy	Frontier Selection	Complete	Halts	Space
Depth-first	Last node added	No	No	Linear
Breadth-first	First node added	Yes	No	Exp
Lowest-cost-first	Minimal $cost(p)$	Yes	No	Exp
Best-first	Minimal $h(p)$	No	No	Exp
A^*	Minimal $f(p)$	Yes	No	Exp

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

Halts — on finite graph (perhaps with cycles).

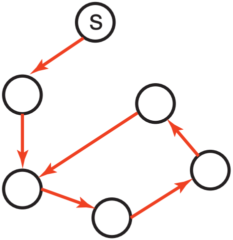
Space — as a function of the length of current path

Cycle Pruning



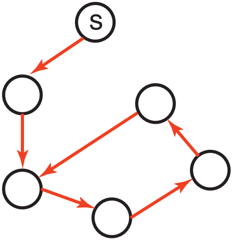
- In depth-first search, checking for cycles can be done in ________ time in path length.

Cycle Pruning



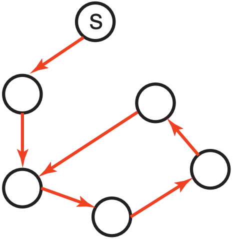
- In depth-first search, checking for cycles can be done in <u>constant</u> time in path length.

Cycle Pruning



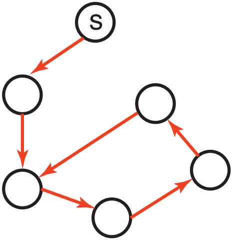
- In depth-first search, checking for cycles can be done in <u>constant</u> time in path length.
- For other methods, checking for cycles can be done in _____ time in path length.

Cycle Pruning



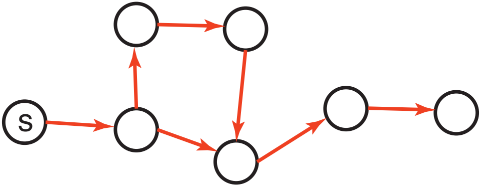
- In depth-first search, checking for cycles can be done in <u>constant</u> time in path length.
- For other methods, checking for cycles can be done in <u>linear</u> time in path length.

Cycle Pruning



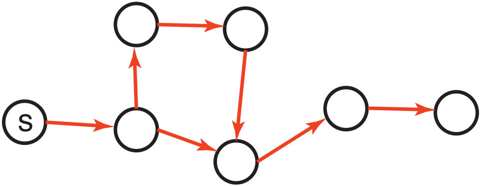
- In depth-first search, checking for cycles can be done in <u>constant</u> time in path length.
- For other methods, checking for cycles can be done in <u>linear</u> time in path length.
- With cycle pruning, which algorithms halt on finite graphs?

Multiple-Path Pruning



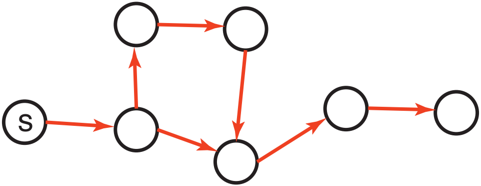
- Multiple path pruning: prune a path to node n that the searcher has already found a path to.

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?

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?
- Lowest-cost-first search with multiple-path pruning is Dijkstra's algorithm.

Graph searching with multiple-path pruning

Input: a graph,
a set of start nodes,
Boolean procedure $goal(n)$ that tests if n is a goal node.
$frontier := \{\langle s \rangle : s \text{ is a start node}\}$
$expanded := \{\}$
while *frontier* is not empty:
select and **remove** path $\langle n_0, \ldots, n_k \rangle$ from *frontier*
if $n_k \notin expanded$:
add n_k to *expanded*
if $goal(n_k)$:
return $\langle n_0, \ldots, n_k \rangle$
$Frontier := Frontier \cup \{\langle n_0, \ldots, n_k, n \rangle : \langle n_k, n \rangle \in A\}$

Multiple-Path Pruning

- How does multiple-path pruning compare to cycle pruning?

Multiple-Path Pruning

- How does multiple-path pruning compare to cycle pruning?
- Which search algorithms with multiple-path pruning always halt on finite graphs?

Multiple-Path Pruning

- How does multiple-path pruning compare to cycle pruning?
- Which search algorithms with multiple-path pruning always halt on finite graphs?
- What is the time overhead of multiple-path pruning?

Multiple-Path Pruning

- How does multiple-path pruning compare to cycle pruning?
- Which search algorithms with multiple-path pruning always halt on finite graphs?
- What is the time overhead of multiple-path pruning?
- What is the space overhead of multiple-path pruning?

Multiple-Path Pruning

- How does multiple-path pruning compare to cycle pruning?
- Which search algorithms with multiple-path pruning always halt on finite graphs?
- What is the time overhead of multiple-path pruning?
- What is the space overhead of multiple-path pruning?
- Is it better for depth-first or breadth-first searches?

Multiple-Path Pruning

- How does multiple-path pruning compare to cycle pruning?
- Which search algorithms with multiple-path pruning always halt on finite graphs?
- What is the time overhead of multiple-path pruning?
- What is the space overhead of multiple-path pruning?
- Is it better for depth-first or breadth-first searches?
- Can multiple-path pruning prevent an optimal solution being found?

Summary of Search Strategies

Strategy	Frontier	Complete	Halts	Space
Depth-first w/o CP	Last added			
Depth-first w CP	Last added			
Depth-first w MPP	Last added			

Summary of Search Strategies

Strategy	Frontier	Complete	Halts	Space
Depth-first w/o CP	Last added	No	No	Linear
Depth-first w CP	Last added	No	Yes	Linear
Depth-first w MPP	Last added	No	Yes	Exp

Summary of Search Strategies

Strategy	Frontier	Complete	Halts	Space
Depth-first w/o CP	Last added	No	No	Linear
Depth-first w CP	Last added	No	Yes	Linear
Depth-first w MPP	Last added	No	Yes	Exp
Breadth-first w/o MPP	First added	Yes	No	Exp
Breadth-first w MPP	First added	Yes	Yes	Exp

Summary of Search Strategies

Strategy	Frontier	Complete	Halts	Space
Depth-first w/o CP	Last added	No	No	Linear
Depth-first w CP	Last added	No	Yes	Linear
Depth-first w MPP	Last added	No	Yes	Exp
Breadth-first w/o MPP	First added	Yes	No	Exp
Breadth-first w MPP	First added	Yes	Yes	Exp
Best-first w/o MPP	Min $h(p)$	No		
Best-first w MPP	Min $h(p)$	No		
A^* w/o MPP	Min $f(p)$	Yes		
A^* w MPP	Min $f(p)$	Yes		

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

Assume graph satisfies the assumptions of A^* proof + montonicity

Summary of Search Strategies

Strategy	Frontier	Complete	Halts	Space
Depth-first w/o CP	Last added	No	No	Linear
Depth-first w CP	Last added	No	Yes	Linear
Depth-first w MPP	Last added	No	Yes	Exp
Breadth-first w/o MPP	First added	Yes	No	Exp
Breadth-first w MPP	First added	Yes	Yes	Exp
Best-first w/o MPP	Min $h(p)$	No	No	Exp
Best-first w MPP	Min $h(p)$	No	Yes	Exp
A^* w/o MPP	Min $f(p)$	Yes	No	Exp
A^* w MPP	Min $f(p)$	Yes	Yes	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

Assume graph satisfies the assumptions of A^* proof + montonicity

Clicker Question

Which of the following is false:

- A All of the search methods based on the generic search algorithm (without cycle pruning and MPP) can go into an infinite loop on finite graphs
- B Heuristics in spatial domains have to be straight-line distances
- C Arc costs must be non-negative to make sure least-cost-search finds least-cost solutions first
- D Complete search algorithms find a solution if one exists even in infinite graphs.
- E A^* uses the cost of the path to a node as well as heuristic information about the node.

Clicker Question

"A^* is admissible" means:

A The first solution returned is a least-cost solution

B It always halts on finite graphs

C It can get stuck in cycles

D Multiple-path pruning is not used

E Multiple-path pruning is used

Clicker Question

Which of the following assumptions was not made in the result that A^* is admissible:

A Arc costs are bounded above 0

B Branching factor is finite

C $h(n)$ is an underestimate of the cost of the shortest path from n to a goal

D The costs around a cycle must sum to zero

Clicker Question

The monotone restriction:

A restricts the possible graphs that can be searched

B restricts the possible heuristics that can be used

C restricts which goals can be searched for

D implies that spatial domains must use the straight-line (Euclidean or Manhattan) distance

E means poor singers won't get recording contracts

Clicker Question

Which of the following is true:

A Multiple-path pruning increases the space used by breadth-first search from linear to exponential

B Multiple-path pruning increases the space used by A^* search from linear to exponential

C Multiple-path pruning increases the space used by depth-first search from linear to exponential

D Multiple-path pruning doesn't increase the space used of any of these methods.

Clicker Question

With a heuristic that does not satisfy the monotone restriction, how might A^* search with multiple-path pruning not be admissible?

- A it might not expand a path on frontier with lowest f-value
- B it might not return a lowest-cost path
- C it is always admissible, even without the monotone restriction
- D it only considers the heuristic value and not both path cost and heuristic cost
- E it might use space exponential in the path length instead of linear

Clicker Question

With of the following is false:

A With multiple-path pruning, we don't need cycle pruning

B With multiple path pruning all search algorithms halt on finite graphs

C All algorithms have exponential space with multiple-path pruning

D Cycle pruning without multiple-path pruning makes A^* no longer admissible

Direction of Search

- The definition of searching is symmetric: find path from start nodes to goal node or from goal node to start nodes (with reversed arcs).

Direction of Search

- The definition of searching is symmetric: find path from start nodes to goal node or from goal node to start nodes (with reversed arcs).
- Forward branching factor: number of arcs out of a node.
- Backward branching factor: number of arcs into a node.

Direction of Search

- The definition of searching is symmetric: find path from start nodes to goal node or from goal node to start nodes (with reversed arcs).
- 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.

Direction of Search

- The definition of searching is symmetric: find path from start nodes to goal node or from goal node to start nodes (with reversed arcs).
- 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. One might be more difficult to compute than the other.

Bidirectional Search

- Idea: search backward from the goal and forward from the start simultaneously.

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.

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.

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
  - a breadth-first method (e.g., least-cost-first search) that builds a set of states that can lead to the goal quickly.
  - in the other direction, another method (typically depth-first) can be used to find a path to these interesting states.

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
  - a breadth-first method (e.g., least-cost-first search) that builds a set of states that can lead to the goal quickly.
  - in the other direction, another method (typically depth-first) can be used to find a path to these interesting states.
  - How much is stored in the breadth-first method, can be tuned depending on the space available.

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 $mb^{k/m} \ll b^k$.

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.
- Requires more knowledge than just the graph and a heuristic function.

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.
- Requires more knowledge than just the graph and a heuristic function.
- 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:

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

using least-cost-first search in the reverse graph.

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:

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

using least-cost-first search in the reverse graph.

- This can be used locally to determine what to do from any state.

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:

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

using least-cost-first search in the reverse graph.

- This can be used locally to determine what to do from any state.
- Why not use A^*?

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:

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

using least-cost-first search in the reverse graph.

- This can be used locally to determine what to do from any state.
- Why not use A^*?
- There are two main problems:

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:

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

using least-cost-first search in the reverse graph.

- This can be used locally to determine what to do from any state.
- Why not use A^*?
- There are two main problems:
  - It requires enough space to store the graph.
  - The *dist* function needs to be recomputed for each goal.