Heuristic Search

CPSC 322 Lecture 6

January 19, 2007
Textbook §2.5

Lecture Overview

Graph Search Algorithm

Input: a graph,
a set of start nodes,
Boolean procedure $goal(n)$ that tests if n is a goal node.
$frontier := \{\langle s \rangle : s$ is a start node$\}$;
while $frontier$ is not empty:
    select and **remove** path $\langle n_0, \ldots, n_k \rangle$ from $frontier$;
    if $goal(n_k)$
        return $\langle n_0, \ldots, n_k \rangle$;
    for every neighbor n of n_k
        add $\langle n_0, \ldots, n_k, n \rangle$ to $frontier$;
end while

- After the algorithm returns, it can be asked for more answers and the procedure continues.
- Which value is selected from the frontier defines the search strategy.
- The *neighbor* relationship defines the graph.
- The *goal* function defines what is a solution.

Depth-first Search

- Depth-first search treats the frontier as a stack
  - It always selects one of the last elements added to the frontier.

- Complete when the graph has no cycles and is finite
- Time complexity is $O(b^m)$
- Space complexity is $O(bm)$

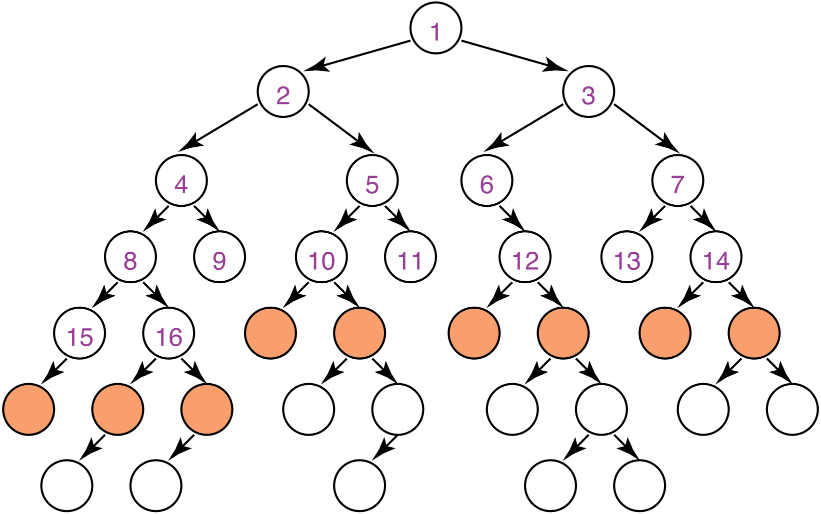
Lecture Overview

Breadth-first Search

- Breadth-first search treats the frontier as a queue
  - it always selects one of the earliest elements added to the frontier.

- Example:
  - the frontier is $[p_1, p_2, \ldots, p_r]$
  - neighbours of p_1 are $\{n_1, \ldots, n_k\}$
- What happens?
  - p_1 is selected, and tested for being a goal.
  - Neighbours of p_1 follow p_r at the end of the frontier.
  - Thus, the frontier is now $[p_2, \ldots, p_r, (p_1, n_1), \ldots, (p_1, n_k)]$.
  - p_2 is selected next.

Illustrative Graph — Breadth-first Search



Analysis of Breadth-First Search

- Is BFS complete?

Analysis of Breadth-First Search

- Is BFS complete?
  - Yes (but it wouldn't be if the branching factor for any node was infinite)
  - In fact, BFS is guaranteed to find the path that involves the fewest arcs (why?)

Analysis of Breadth-First Search

- Is BFS complete?
  - Yes (but it wouldn't be if the branching factor for any node was infinite)
  - In fact, BFS is guaranteed to find the path that involves the fewest arcs (why?)
- What is the time complexity, if the maximum path length is m and the maximum branching factor is b?

Analysis of Breadth-First Search

- Is BFS complete?
  - Yes (but it wouldn't be if the branching factor for any node was infinite)
  - In fact, BFS is guaranteed to find the path that involves the fewest arcs (why?)
- What is the time complexity, if the maximum path length is m and the maximum branching factor is b?
  - The time complexity is $O(b^m)$: must examine every node in the tree.
  - The order in which we examine nodes (BFS or DFS) makes no difference to the worst case: search is unconstrained by the goal.

Analysis of Breadth-First Search

- Is BFS complete?
  - Yes (but it wouldn't be if the branching factor for any node was infinite)
  - In fact, BFS is guaranteed to find the path that involves the fewest arcs (why?)
- What is the time complexity, if the maximum path length is m and the maximum branching factor is b?
  - The time complexity is $O(b^m)$: must examine every node in the tree.
  - The order in which we examine nodes (BFS or DFS) makes no difference to the worst case: search is unconstrained by the goal.
- What is the space complexity?

Analysis of Breadth-First Search

- Is BFS complete?
  - Yes (but it wouldn't be if the branching factor for any node was infinite)
  - In fact, BFS is guaranteed to find the path that involves the fewest arcs (why?)
- What is the time complexity, if the maximum path length is m and the maximum branching factor is b?
  - The time complexity is $O(b^m)$: must examine every node in the tree.
  - The order in which we examine nodes (BFS or DFS) makes no difference to the worst case: search is unconstrained by the goal.
- What is the space complexity?
  - Space complexity is $O(b^m)$: we must store the whole frontier in memory

Using Breadth-First Search

- When is BFS appropriate?

Using Breadth-First Search

- When is BFS appropriate?
  - space is not a problem
  - it's necessary to find the solution with the fewest arcs
  - although all solutions may not be shallow, at least some are
  - there may be infinite paths

Using Breadth-First Search

- When is BFS appropriate?
  - space is not a problem
  - it's necessary to find the solution with the fewest arcs
  - although all solutions may not be shallow, at least some are
  - there may be infinite paths

- When is BFS inappropriate?

Using Breadth-First Search

- When is BFS appropriate?
  - space is not a problem
  - it's necessary to find the solution with the fewest arcs
  - although all solutions may not be shallow, at least some are
  - there may be infinite paths

- When is BFS inappropriate?
  - space is limited
  - all solutions tend to be located deep in the tree
  - the branching factor is very large

Lecture Overview

Search with Costs

- Sometimes there are costs associated with arcs.
  - The cost of a path is the sum of the costs of its arcs.

$$cost(\langle n_0, \ldots, n_k \rangle) = \sum_{i=1}^{k} |\langle n_{i-1}, n_i \rangle|$$

- In this setting we often don't just want to find just any solution
  - Instead, we usually want to find the solution that minimizes cost
- We call a search algorithm which always finds such a solution optimal

Lowest-Cost-First Search

- At each stage, lowest-cost-first search selects a path on the frontier with lowest cost.
  - The frontier is a priority queue ordered by path cost.
  - We say "a path" because there may be ties
- When all arc costs are equal, LCFS is equivalent to BFS.
- Example:
  - the frontier is $[\langle p_1, 10\rangle, \langle p_2, 5\rangle, \langle p_3, 7\rangle]$
  - p_2 is the lowest-cost node in the frontier
  - neighbours of p_2 are $\{\langle p_9, 12\rangle, \langle p_{10}, 15\rangle\}$
- What happens?
  - p_2 is selected, and tested for being a goal.
  - Neighbours of p_2 are inserted into the frontier (it doesn't matter where they go)
  - Thus, the frontier is now $[\langle p_1, 10\rangle, \langle p_9, 12\rangle, \langle p_{10}, 15\rangle, \langle p_3, 7\rangle]$.
  - p_3 is selected next.
  - Of course, we'd really implement this as a priority queue.

Analysis of Lowest-Cost-First Search

- Is LCFS complete?

Analysis of Lowest-Cost-First Search

- Is LCFS complete?
  - not in general: a cycle with zero or negative arc costs could be followed forever.

Analysis of Lowest-Cost-First Search

- Is LCFS complete?
  - not in general: a cycle with zero or negative arc costs could be followed forever.
  - yes, as long as arc costs are strictly positive
- What is the time complexity, if the maximum path length is m and the maximum branching factor is b?

Analysis of Lowest-Cost-First Search

- Is LCFS complete?
  - not in general: a cycle with zero or negative arc costs could be followed forever.
  - yes, as long as arc costs are strictly positive
- What is the time complexity, if the maximum path length is m and the maximum branching factor is b?
  - The time complexity is $O(b^m)$: must examine every node in the tree.
  - Knowing costs doesn't help here.

Analysis of Lowest-Cost-First Search

- Is LCFS complete?
  - not in general: a cycle with zero or negative arc costs could be followed forever.
  - yes, as long as arc costs are strictly positive
- What is the time complexity, if the maximum path length is m and the maximum branching factor is b?
  - The time complexity is $O(b^m)$: must examine every node in the tree.
  - Knowing costs doesn't help here.
- What is the space complexity?

Analysis of Lowest-Cost-First Search

- Is LCFS complete?
  - not in general: a cycle with zero or negative arc costs could be followed forever.
  - yes, as long as arc costs are strictly positive
- What is the time complexity, if the maximum path length is m and the maximum branching factor is b?
  - The time complexity is $O(b^m)$: must examine every node in the tree.
  - Knowing costs doesn't help here.
- What is the space complexity?
  - Space complexity is $O(b^m)$: we must store the whole frontier in memory.

Analysis of Lowest-Cost-First Search

- Is LCFS complete?
  - not in general: a cycle with zero or negative arc costs could be followed forever.
  - yes, as long as arc costs are strictly positive
- What is the time complexity, if the maximum path length is m and the maximum branching factor is b?
  - The time complexity is $O(b^m)$: must examine every node in the tree.
  - Knowing costs doesn't help here.
- What is the space complexity?
  - Space complexity is $O(b^m)$: we must store the whole frontier in memory.
- Is LCFS optimal?
  - Not in general. Why not?

Analysis of Lowest-Cost-First Search

- Is LCFS complete?
  - not in general: a cycle with zero or negative arc costs could be followed forever.
  - yes, as long as arc costs are strictly positive
- What is the time complexity, if the maximum path length is m and the maximum branching factor is b?
  - The time complexity is $O(b^m)$: must examine every node in the tree.
  - Knowing costs doesn't help here.
- What is the space complexity?
  - Space complexity is $O(b^m)$: we must store the whole frontier in memory.
- Is LCFS optimal?
  - Not in general. Why not?
  - Arc costs could be negative: a path that initially looks high-cost could end up getting a "refund".
  - However, LCFS *is* optimal if arc costs are guaranteed to be non-negative.

Lecture Overview

Past knowledge and search

- Some people believe that they are good at solving hard problems without search
  - However, consider e.g., public key encryption codes (or combination locks): the search problem is clear, but people can't solve it
  - When people do perform well on hard problems, it is usually because they have useful knowledge about the structure of the problem domain
- Computers can also improve their performance when given this sort of knowledge
  - in search, they can estimate the distance from a given node to the goal through a search heuristic
  - in this way, they can take the goal into account when selecting path

Heuristic Search

- $h(n)$ is an estimate of the cost of the shortest path from node n to a goal node.
  - h can be extended to paths: $h(\langle n_0, \ldots, n_k \rangle) = h(n_k)$
- $h(n)$ uses only readily obtainable information (that is easy to compute) about a node.
- Admissible heuristic: $h(n)$ is an underestimate if there is no path from n to a goal that has path length less than $h(n)$.
  - another way of saying this: $h(n)$ is a lower bound on the cost of getting from n to the nearest goal.

Example Heuristic Functions

- If the nodes are points on a Euclidean plane and the cost is the distance, we can use the straight-line distance from n to the closest goal as the value of $h(n)$.
  - this makes sense if there are obstacles, or for other reasons not all adjacent nodes share an arc

Example Heuristic Functions

- If the nodes are points on a Euclidean plane and the cost is the distance, we can use the straight-line distance from n to the closest goal as the value of $h(n)$.
  - this makes sense if there are obstacles, or for other reasons not all adjacent nodes share an arc
- Likewise, if nodes are cells in a grid and the cost is the number of steps, we can use "Manhattan distance"
  - this is also known as the L_1 distance; Euclidean distance is L_2 distance

Example Heuristic Functions

- If the nodes are points on a Euclidean plane and the cost is the distance, we can use the straight-line distance from n to the closest goal as the value of $h(n)$.
  - this makes sense if there are obstacles, or for other reasons not all adjacent nodes share an arc
- Likewise, if nodes are cells in a grid and the cost is the number of steps, we can use "Manhattan distance"
  - this is also known as the L_1 distance; Euclidean distance is L_2 distance
- In the 8-puzzle, we can use the number of moves between each tile's current position and its position in the solution

How to Construct a Heuristic

- Overall, a cost-minimizing search problem is a constrained optimization problem
  - e.g., find a path from A to B which minimizes distance traveled, subject to the constraint that the robot can't move through walls
- A relaxed version of the problem is a version of the problem where one or more constraints have been dropped
  - e.g., find a path from A to B which minimizes distance traveled, *allowing* the agent to move through walls
  - A relaxed version of a minimization problem will always return a value which is weakly smaller than the original value: thus, it's an admissible heuristic

How to Construct a Heuristic

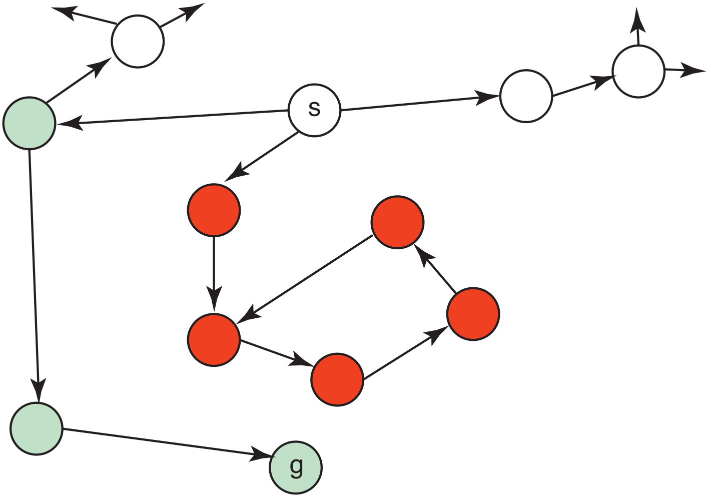
- It's usually possible to identify constraints which, when dropped, make the problem extremely easy to solve
  - this is important because heuristics are not useful if they're as hard to solve as the original problem!
- Another trick for constructing heuristics: if $h_1(n)$ is an admissible heuristic, and $h_2(n)$ is also an admissible heuristic, then $\max(h_1(n), h_2(n))$ is also admissible.

Lecture Overview

Best-First Search

- Idea: select the path whose end is closest to a goal according to the heuristic function.
- Best-First search selects a path on the frontier with minimal h-value.
- It treats the frontier as a priority queue ordered by h.
- This is a greedy approach: it always takes the path which appears locally best

Illustrative Graph — Best-First Search



Complexity of Best-First Search

- Complete: no: a heuristic of zero for an arc that returns to the same state can be followed forever.
- Time complexity is $O(b^m)$
- Space complexity is $O(b^m)$
- Optimal: no (why not?)