Heuristic Search and A*

CPSC 322 Lecture 7

September 19, 2007
Textbook §3.5

Lecture Overview

1. Recap
2. Heuristic Search
3. Best-First Search
4. A^* Search
5. Optimality of A^*

Breadth-first Search

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

- Complete even when the graph has cycles or is infinite
- Time complexity is $O(b^m)$
- Space complexity is $O(b^m)$

Search with Costs

- Sometimes there are costs associated with arcs.
  - The cost of a path is the sum of the costs of its arcs.
- 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: expand paths from the frontier in order of their costs.

Lecture Overview

1. Recap
2. Heuristic Search
3. Best-First Search
4. A^* Search
5. Optimality of A^*

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

Definition (search heuristic)

A search heuristic $h(n)$ 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.

Heuristic Search

Definition (search heuristic)

A search heuristic $h(n)$ 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.

Definition (admissible heuristic)

A search heuristic $h(n)$ is admissible if it is never an overestimate of the cost from n to a goal.

- there is never a 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

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

- 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
- 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!

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
- 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.

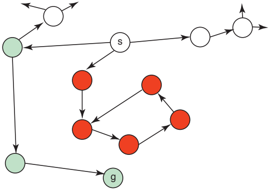
Lecture Overview

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

Complexity of Best-First Search

- Complete: no: a low heuristic value can mean that a cycle gets followed forever.



- Time complexity is $O(b^m)$
- Space complexity is $O(b^m)$
- Optimal: no (why not?)

Lecture Overview

A^* Search

- A^* search uses both path costs and heuristic values
  - $cost(p)$ is the cost of the path p.
  - $h(p)$ estimates the cost from the end of p to a goal.

A^* Search

- A^* search uses both path costs and heuristic values
  - $cost(p)$ is the cost of the path p.
  - $h(p)$ estimates the cost from the end of p to a goal.
- Let $f(p) = cost(p) + h(p)$.
  - $f(p)$ estimates the total path cost of going from a start node to a goal via p.

$$\underbrace{\underbrace{start \xrightarrow{\text{path } p} n}_{cost(p)} \underbrace{\xrightarrow{\text{estimate}} goal}_{h(p)}}_{f(p)}$$

A^* Search

- A^* search uses both path costs and heuristic values
  - $cost(p)$ is the cost of the path p.
  - $h(p)$ estimates the cost from the end of p to a goal.
- Let $f(p) = cost(p) + h(p)$.
  - $f(p)$ estimates the total path cost of going from a start node to a goal via p.

$$\underbrace{\underbrace{start \xrightarrow{\text{path } p} n}_{cost(p)} \underbrace{\xrightarrow{\text{estimate}} goal}_{h(p)}}_{f(p)}$$

- A^* treats the frontier as a priority queue ordered by $f(p)$.
  - It always selects the node on the frontier with the lowest estimated total distance.
  - It's a mix of lowest-cost-first and best-first search.

Analysis of A^*

Let's assume that arc costs are strictly positive.

- Completeness: yes.
- Time complexity: $O(b^m)$
  - the heuristic could be completely uninformative and the edge costs could all be the same, meaning that A^* does the same thing as BFS
- Space complexity: $O(b^m)$
  - like BFS, A^* maintains a frontier which grows with the size of the tree
- Optimality: yes.

Lecture Overview

1. Recap
2. Heuristic Search
3. Best-First Search
4. A^* Search
5. Optimality of A^*

Optimality[1] of A^*

If A^* returns a solution, that solution is guaranteed to be optimal, as long as

- the branching factor is finite
- arc costs are strictly positive
- $h(n)$ is an underestimate of the length of the shortest path from n to a goal node, and is non-negative

[1] Some literature, and the textbook, uses the word "admissiblity" here.

Why is A^* optimal?

Theorem

If A^* selects a path p, p is the shortest (i.e., lowest-cost) path.

- Assume for contradiction that some other path p' is actually the shortest path to a goal
- Consider the moment just before p is chosen from the frontier. Some part of path p' will also be on the frontier; let's call this partial path p''.
- Because p was expanded before p'', $f(p) \leq f(p'')$.
- Because p is a goal, $h(p) = 0$. Thus $cost(p) \leq cost(p'') + h(p'')$.
- Because h is admissible, $cost(p'') + h(p'') \leq cost(p')$ for any path p' to a goal that extends p''
- Thus $cost(p) \leq cost(p')$ for any other path p' to a goal. This contradicts our assumption that p' is the shortest path.