Heuristic Search and A*
Lecture Overview

1. Recap

2. Heuristic Search
Recap

Heuristic Search

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 $O(bm)$
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 $O(b^m)$
At this point in the course you should be able to...

- Identify real world examples that make use of deterministic, goal-driven agents
- Differentiate between single/static and sequential problems
- Assess the size of the search space of a given search problem.
- Implement the generic solution to a search problem.
- Evaluate the complexity of a search problem in terms of number of nodes, paths, and frontier nodes.
- Define/read/write/trace/debug different uninformed search algorithms
- Define and determine basic properties of search algorithms: completeness, time and space complexity.
1 Recap

2 Heuristic Search
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
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
**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: $h(⟨n_0, \ldots, n_k⟩) = h(n_k)$
- $h(n)$ 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.

- 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.
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
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.