Week 13 - State Spaces and Search

EECS 16ML
Saagar Sanghavi, Tim Won, Kevin Zhu

Outline

1. State Space Representations
2. Types of Search Algorithms
3. Heuristic
4. Applications

Motivating Example: Robotic Manipulator

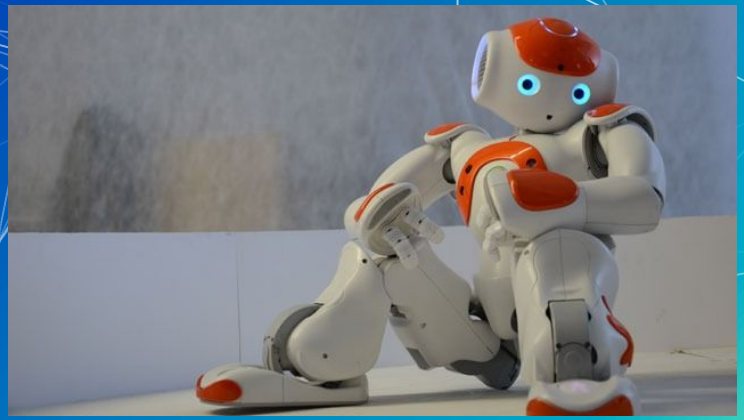
IN EECS 16B, we saw how we could discretize in time to model our system.

Now, we want to plan out a path. But there may be obstacles!

We can discretize in space, too!

Agent and Search Problems

- Agent - entity capable of rational decision given the knowledge about its environment
- World = Agent + Environment
- State Representation
- Search Problem



State Space

- **State Space** - The set of all possible states in the given world
- **Successor Function** - A function that takes in a state and an action and returns the cost of given action and the state after taking that action, which is called successor state
- **Start State** - The initial state where the agent begins in
- **Goal Test** - A function that takes in a state and returns whether the goal is achieved

Search Algorithms

How we create a plan once we have a state space representation!

General Structure of Search Algorithms

1. Make a fringe that contains potential nodes to explore
2. Choose the next node to explore using **strategy**, remove that node from fringe
3. Add removed node's children to the fringe
4. Repeat until reaching goal state

Properties of Search Algorithms

Some nice properties you might want from search algorithms:

1. Completeness - whether the algorithm is guaranteed to find a solution within reasonable time
2. Optimality - whether the algorithm's output is an optimal solution if it ever finds one
3. Time and space complexity - upper bound on the runtime and space usage of the algorithm

Uninformed Search

When given no information about where the solution is, we are forced to use uninformed searches, which are often much slower in the worst case than the informed searches. Here we will explore:

- Depth First Search
- Breadth First Search
- Uniform Cost Search

Depth First Search (DFS)

- Theseus—find a Minotaur in the Labyrinth!
- Wife: carry chalk and string!
  - Unravel the string behind you, so you can retrace your way back
  - Mark intersections in chalk, avoid exploring them again
- This is DFS!
  - String: Stack (or Recursive Stack)
  - Chalk: “seen” set

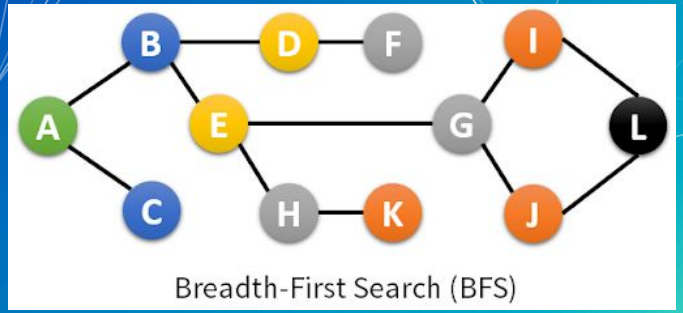
Depth First Search

Strategy - Select the deepest node in the fringe. There is usually a consistent tie-breaking scheme for multiple nodes with the same depth.

- Fringe - first-out (LIFO) stack.
- Completeness - No
- Optimality - No
- Time Complexity - $O(b^m)$
- Space Complexity - O(bm)

Breadth First Search (BFS)

- Strategy - Select the shallowest node in the fringe.
- Fringe - first-out (FIFO) queue.
- Completeness - Yes
- Optimality - Only when edge weights are uniform
- Time complexity - O(b^s) where s is depth of shallowest solution
- Space complexity - O(b^s)



Breadth-First Search (BFS)

Uniform Cost Search (UCS)

- Strategy - Select the lowest cost node in the fringe. Cost of the node is the sum of edge weights of the partial plan.
- Fringe - Priority queue, ranked by backward cost
- Completeness - Yes
- Optimality - Yes
- Time complexity - $O(b^{C^*/epsilon})$ where C* is cost of optimal path and epsilon is minimum cost in optimal path
- Space complexity - $O(b^{C^*/epsilon})$

Informed Searches

When given information about where the solution might be, performance can be improved by focusing on specific direction first.

Heuristic

- Heuristics refer to the techniques that allow us to make estimations very quickly while known algorithms can be very slow. Heuristics allow search algorithms to figure out quickly where to focus on.
- Heuristics need to meet some conditions for the search algorithms to work as intended, such as admissibility and consistency.

Greedy Search

- Strategy - Select the lowest heuristic node from the fringe.
- Fringe - Priority queue ranked by heuristic
- Completeness and Optimality - both can be varied depending on the chosen heuristic. In general greedy search is not complete or optimal

A* Search: Combining UCS and Greedy

- Strategy - Select the lowest estimated total cost node in the fringe, where estimated total cost is the sum of cost of partial plan and heuristic
- Fringe - Priority queue ranked by estimated total cost
- Completeness and Optimality - Given a heuristic that underestimates the total cost, A* search is both complete and optimal

Put it together: Path Planning for our Robotic Manipulator!

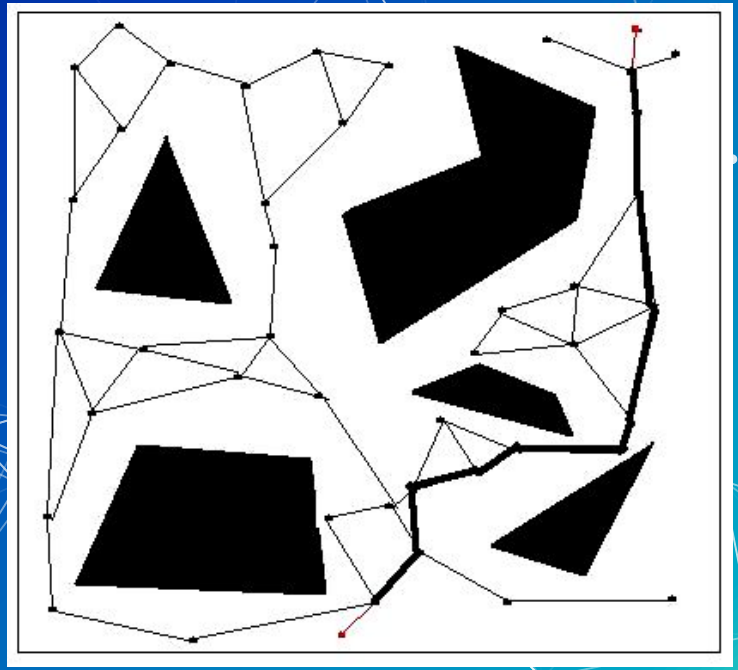
- Once we have a state representation, we can run our search algorithms on the search graph
- We may be able to use information about the configuration as a heuristic!

Path Planning In the Real World

- Discretizing the real, continuous world
  - Systematic intervals
  - Random sampling
- Probabilistic Road Map
- Rapidly Exploring Random Tree
  - RRT*

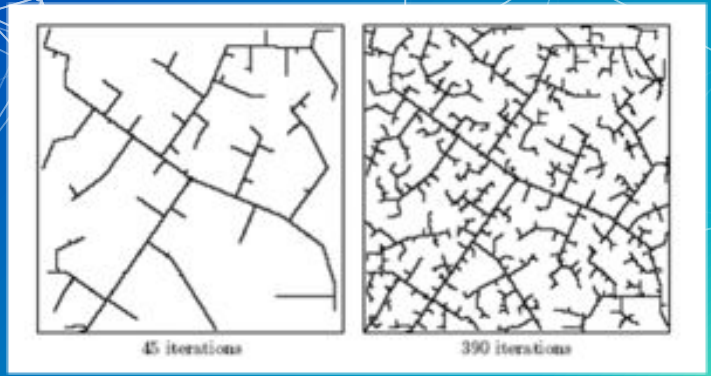
Probabilistic Road Map

- Choose points in the state space uniformly at random
- Remove the points that are in collision with obstacles.
- Connect each of sampled point to its k nearest neighbors
- Remove any edges that would collide with an obstacle.



Rapidly Exploring Random Tree (RRT)

- Bias towards unexplored regions when creating new edges and exploring them.
- Probabilistically complete, but the paths that are generated are often very suboptimal
  - Tends to zigzag alot



THANKS!

Any questions?