Lecture 14: Planning I

Professor Katie Driggs-Campbell

March 19, 2024

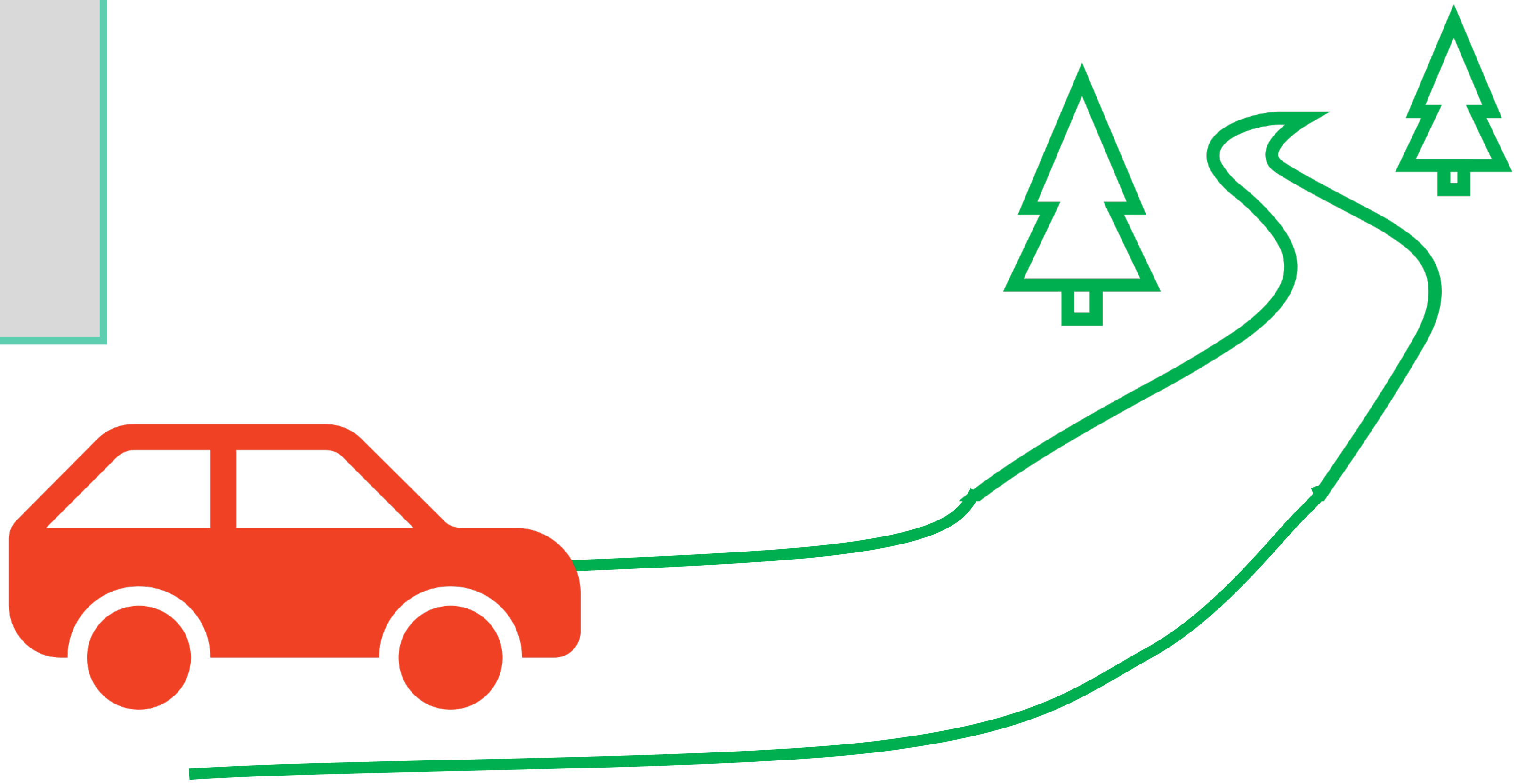
ECE484: Principles of Safe Autonomy

Administrivia

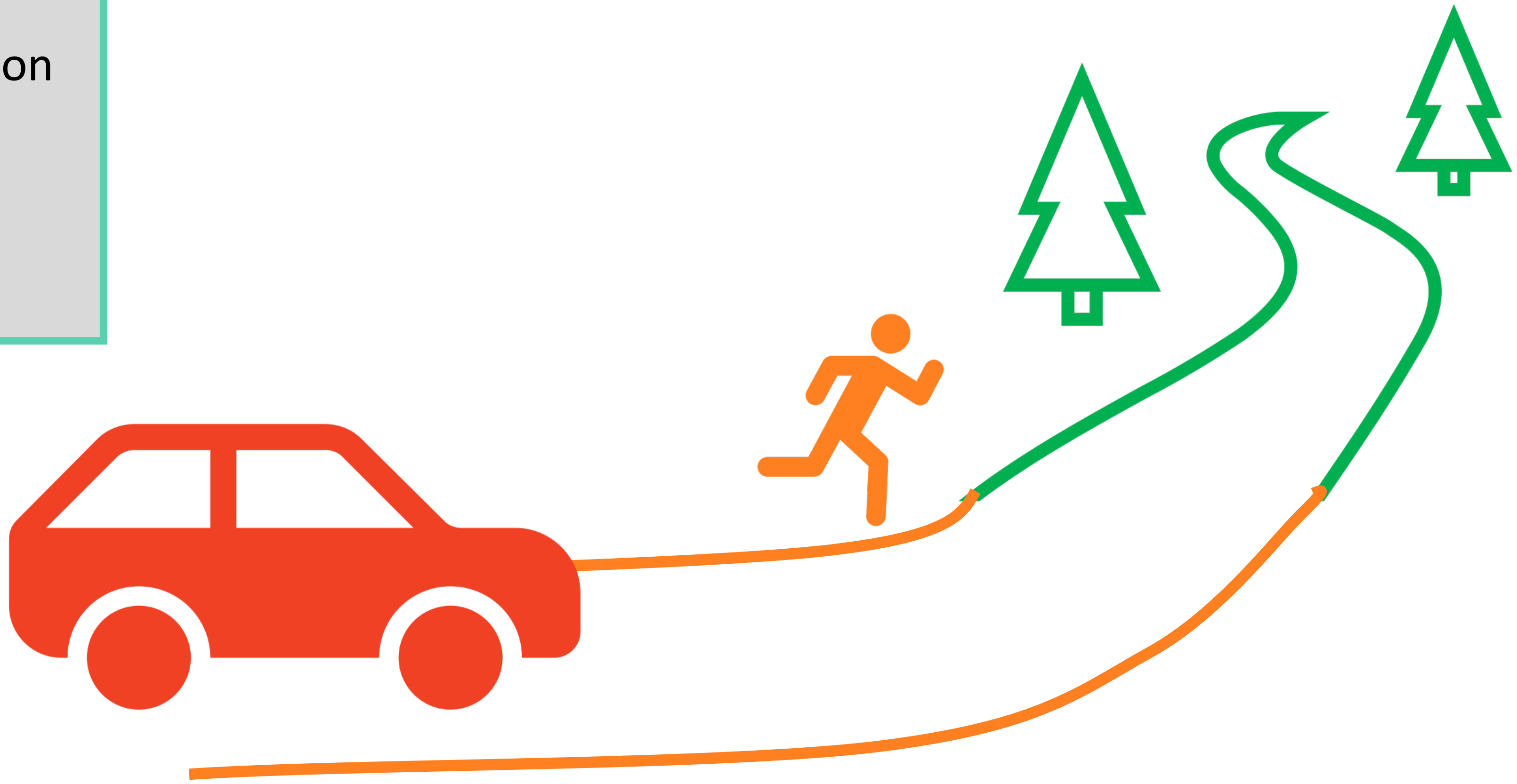
- Upcoming due dates:
  - HW3 and MP3 due Friday 3/22
  - Final Presentations in class on 4/23 and 4/25
  - Final Video due 5/3
- Guest Lectures next week (3/26 and 3/28)
  - Attendance will be taken as that week's pop quiz: Attending both will give 100% for that week's pop quiz, attending one will give 50%
  - Tuesday will start at 10am – I'll have office hours at 9:30am
- Safety discussion and Bonus MP walkthrough on 4/2
- Project support starting this week – information on Canvas
- Exam on 4/18 at 7pm
  - Email me about conflict exams
  - Will use testing center for DRES accommodations

- Vehicle Modeling

- Vehicle Modeling
- Localization



- Vehicle Modeling
- Localization
- Detection & Recognition



- Vehicle Modeling
- Localization
- Detection & Recognition
- Control
- Recall Simple Safety

- Vehicle Modeling
- Localization
- Detection & Recognition
- Control
- Recall Simple Safety
- Next up: Planning!

Today's Plan

- Overview of Motion Planning
- Planning as a graph search problem
- Finding the shortest path
  - Uninformed (uniform) search
  - Greedy search

Today’s Plan

- Overview of Motion Planning
- Planning as a graph search problem
- Finding the shortest path
  - Uninformed (uniform) search
  - Greedy search

Overview of Motion Planning

- **Motion planning** is the problem of finding a robot motion from start state to a goal state that avoids obstacles in the environment
- Recall the **configuration space or C-space:** every point in the C-space $\mathcal{C} \subset \mathbb{R}^n$ corresponds to a unique configuration q of the robot
  - E.g., configuration of a simple car is $q = (x, y, v, \theta)$
- The **free C-space** $\mathcal{C}_{\text{free}}$ consists of the configurations where the robot neither collides with obstacles nor violates constraints

Motion Planning

Given an initial state $x(0) = x_{start}$ and a desired final state x_{goal}, find a time T and a set of controls $u: [0, T] \rightarrow \mathcal{U}$ such that the motion satisfies $x(T) = x_{goal}$ and $q\big(x(t)\big) \in \mathcal{C}_{\text{free}}$ for all $t \in [0, T]$

Assumptions:

1. A feedback controller can ensure that the planned motion is followed closely
2. An accurate model of the robot and environment will evaluate $\mathcal{C}_{\text{free}}$ during motion planning

Quick Discussion

What are some use cases / tasks, considerations, and requirements for planning?

Typical planning and control modules

- Global navigation and planner
  - Find paths from source to destination with static obstacles
  - Algorithms: Graph search, Dijkstra, Sampling-based planning
  - Time scale: Minutes
  - Look ahead: Destination
  - Output: reference center line, semantic commands
- Local planner
  - Dynamically feasible trajectory generation
  - Dynamic planning w.r.t. obstacles
  - Time scales: 10 Hz
  - Look ahead: Seconds
  - Output: Waypoints, high-level actions, directions / velocities
- Controller
  - Waypoint follower using steering, throttle
  - Algorithms: PID control, MPC, Lyapunov-based controller
  - Lateral/longitudinal control
  - Time scale: 100 Hz
  - Look ahead: current state
  - Output: low-level control actions

Types of Motion Planning Problems

- **Path planning versus motion planning**
- **Control inputs: $m = n$ versus $m < n$**
  - Holonomic versus nonholonomic
- **Online versus offline**
  - How reactive does your planner need to be?
- **Optimal versus satisficing**
  - Minimum cost or just reach goal?
- **Exact versus approximate**
  - What is sufficiently close to goal?
- **With or without obstacles**
  - How challenging is the problem?

Motion Planning Methods

- **Complete methods:** exact representations of the geometry of the problem and space
- **Grid methods:** discretize $\mathcal{C}_{\text{free}}$ and search the grid from q_{start} to goal
- **Sampling Methods:** randomly sample from the C-space, evaluate if the sample is in $\mathcal{X}_{\text{free}}$, and add new sample to previous samples

Motion Planning Methods

- **Complete methods:** exact representations of the geometry of the problem and space
- **Grid methods:** discretize $\mathcal{C}_{\text{free}}$ and search the grid from q_{start} to goal
- **Sampling Methods:** randomly sample from the C-space, evaluate if the sample is in $\mathcal{X}_{\text{free}}$, and add new sample to previous samples
- **Virtual potential fields:** create forces on the robot that pull it toward goal and away from obstacles
- **Nonlinear optimization:** minimize some cost subject to constraints on the controls, obstacles, and goal
- **Smoothing:** given some guess or motion planning output, improve the smoothness while avoiding collisions

Properties of Motion Planners

- **Multiple-query versus single-query planning**
- **“Anytime” planning**
  - Continues to look for better solutions after first solution is found
- **Computational complexity**
  - Characterization of the amount of time a planner takes to run or the amount of memory it requires

Properties of Motion Planners

- **Multiple-query versus single-query planning**
- **“Anytime” planning**
  - Continues to look for better solutions after first solution is found
- **Computational complexity**
  - Characterization of the amount of time a planner takes to run or the amount of memory it requires
- **Completeness**
  - A planner is **complete** if it is guaranteed to find a solution in finite time if one exists, and report failure if no feasible plan exists
  - A planner is **resolution complete** if it is guaranteed to find a solution, if one exists, at the resolution of a discretized representation
  - A planner is **probabilistically complete** if the probability of finding a solution, if one exists, tends to 1 as planning time goes to infinity

Search Performance Metrics

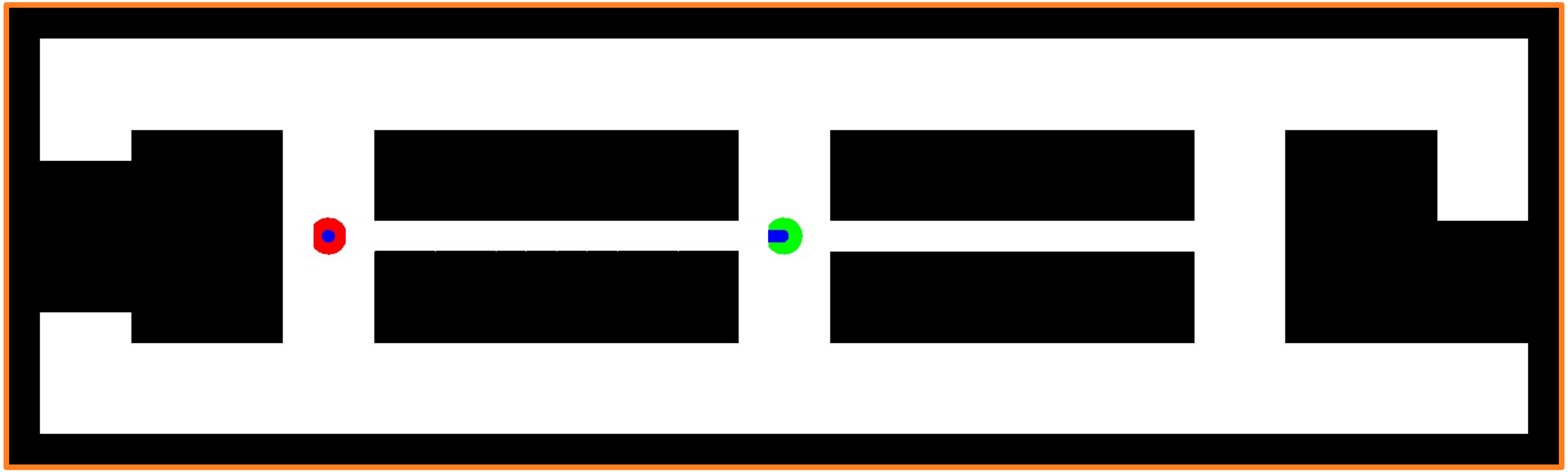
- **Soundness:** when a solution is returned, is it guaranteed to be a correct path?
- **Completeness:** is the algorithm guaranteed to find a solution when there is one?
- **Optimality:** How close is the found solution to the best solution?
- **Space complexity:** How much memory is needed?
- **Time complexity:** What is the running time? Can it be used for online planning?

Today's Plan

- Overview of Motion Planning
- Planning as a graph search problem
- Finding the shortest path
  - Uninformed (uniform) search
  - Greedy search

Planning as a Search Problem

This is a 2D discretization, but we can generalize to higher dimensions (e.g., position, heading, mode)



Graphs and Trees

A **graph** is a collection of **nodes** $\mathcal{N}$ and **edges** $\mathcal{E}$, where edge e connects two nodes

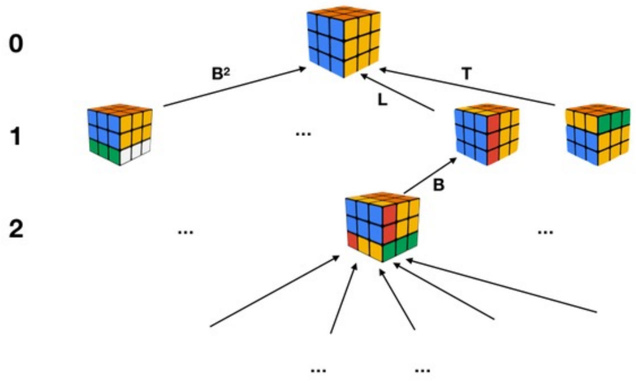
Problem Statement: find shortest path

- Input: $\langle V, E, w, x_{start}, x_{goal} \rangle$
  - V: (finite) set of vertices
  - $E \subseteq V \times V$: (finite) set of edges
  - $w: E \rightarrow \mathbb{R}_{>0}$: a function that associates to each edge e to a strictly positive weight $w(e)$ (e.g., cost, distance, time, fuel)
  - $x_{start}, x_{goal} \in V$: start and end vertices (i.e., initial and desired configuration)

Problem Statement: find shortest path

- Input: $\langle V, E, w, x_{start}, x_{goal} \rangle$
  - V: (finite) set of vertices
  - $E \subseteq V \times V$: (finite) set of edges
  - $w: E \rightarrow \mathbb{R}_{>0}$: a function that associates to each edge e to a strictly positive weight $w(e)$ (e.g., cost, distance, time, fuel)
  - $x_{start}, x_{goal} \in V$: start and end vertices (i.e., initial and desired configuration)
- Output: $\langle P \rangle$
  - P is a path starting at x_{start} and ending in x_{goal}, such that its weight $w(P)$ is minimal among all such paths
  - The weight of a path is the sum of the weights of its edges
  - *The graph may be unknown, partially known, or known*

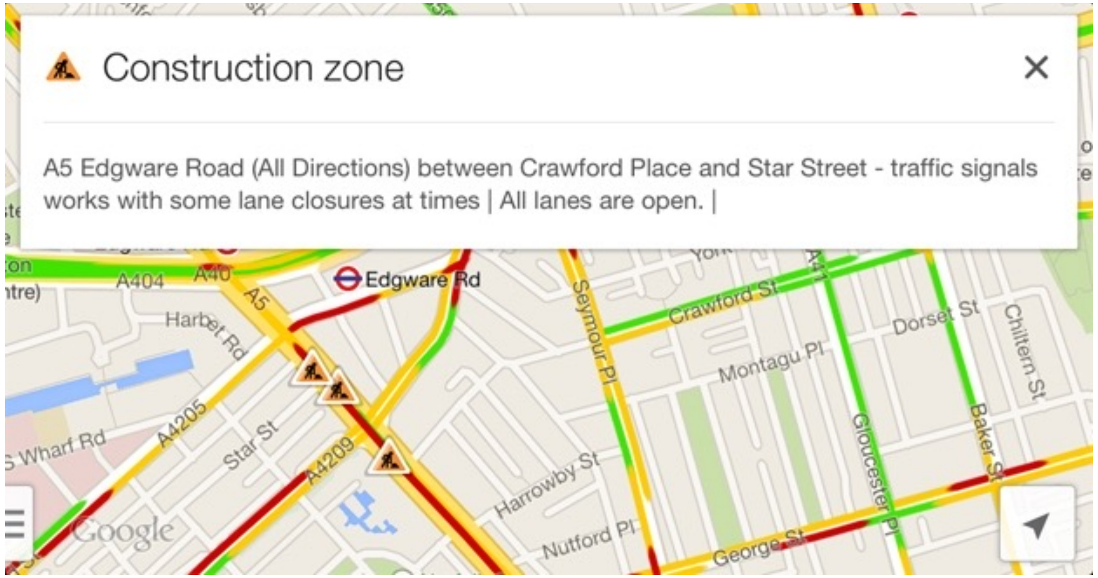
Examples

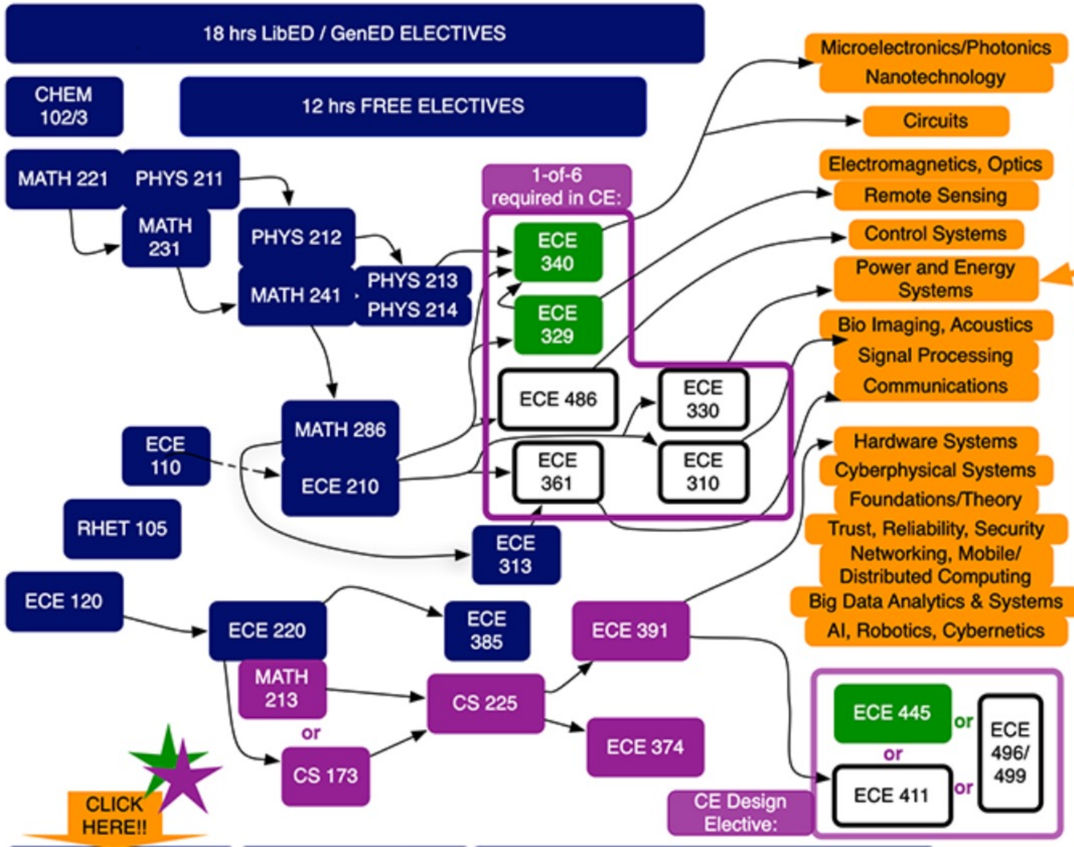


The number of states or vertices can be large!

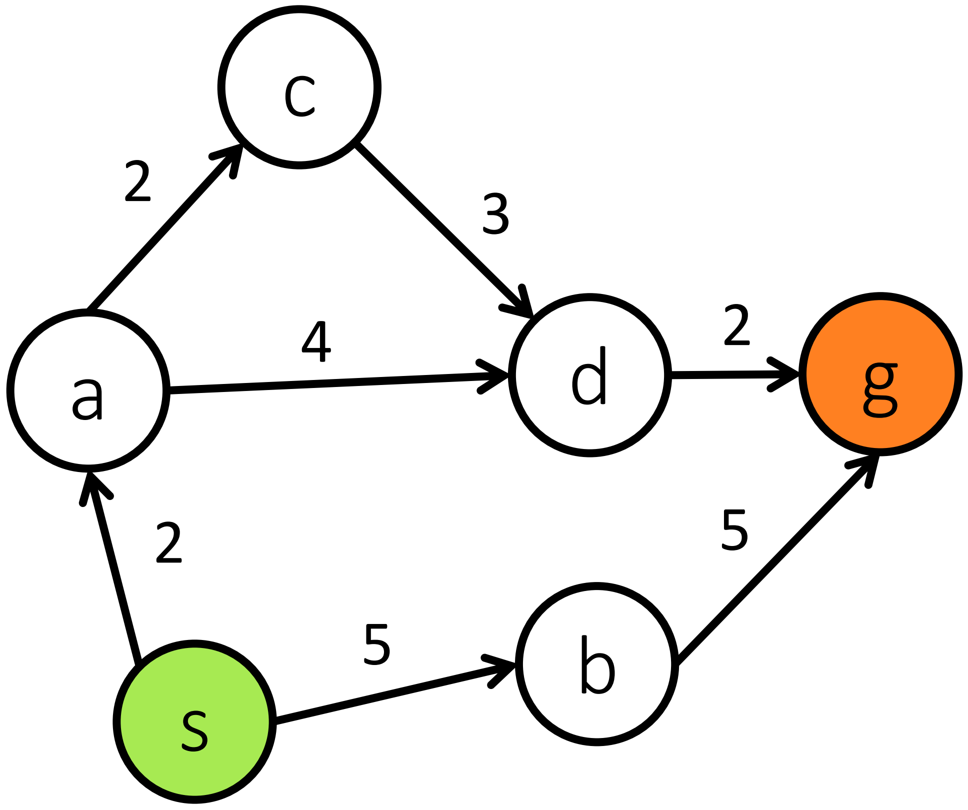
Rubik's cube num states:
43,252,003,274,489,856,000

Many paths and weights are not often known upfront!





Example: Find the minimal path from s to g



Today's Plan

- Overview of Motion Planning
- Planning as a graph search problem
- Finding the shortest path
  - Uninformed (uniform) search
  - Greedy search

Uniform cost search (Uninformed search)

$\boldsymbol{Q} \leftarrow \langle \boldsymbol{start} \rangle$ *// maintains paths*

// initialize queue with start

while $\boldsymbol{Q} \neq \emptyset$:

  pick (and remove) the path $\boldsymbol{P}$ with the lowest cost ($\boldsymbol{g} = \boldsymbol{w}(\boldsymbol{P})$) from Q

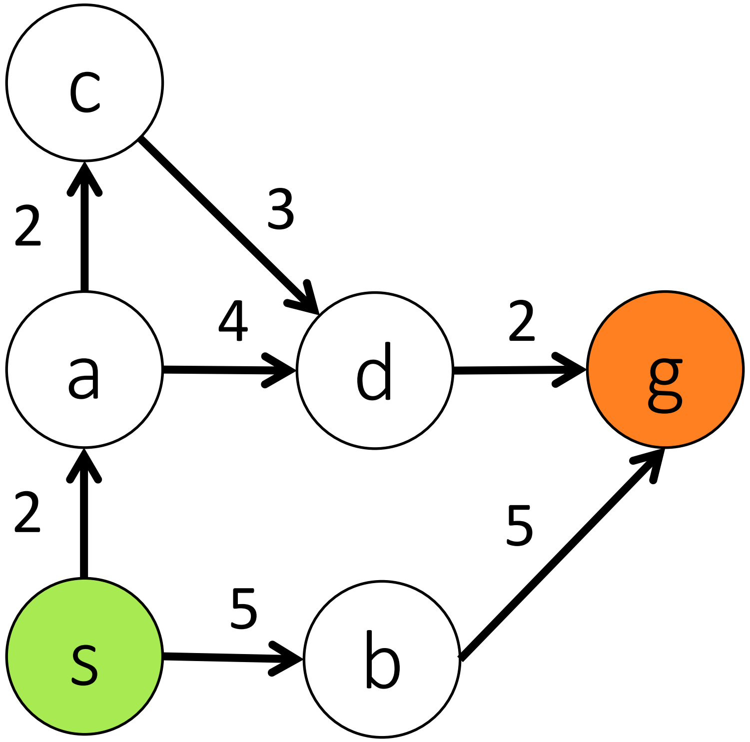
  if $\boldsymbol{head}(\boldsymbol{P}) = \boldsymbol{x}_{\boldsymbol{goal}}$ then return $\boldsymbol{P}$ *// Reached the goal*

  for each vertex $\boldsymbol{v}$ such that $(\boldsymbol{head}(\boldsymbol{P}), \boldsymbol{v}) \in \boldsymbol{E}$, do *// for all neighbors*

    add $\langle \boldsymbol{v}, \boldsymbol{P} \rangle$ **to** $\boldsymbol{Q}$ *// Add expanded paths*

Return FAILURE ***// nothing left to consider***

Example of Uniform-Cost Search



Remarks on Uniform Cost Search (UCS)

- UCS is an extension of Breadth First Search (BFS) to the weighted-graph case
  - i.e., UCS is equivalent BFS if all edges have the same cost
- UCS is *complete* and *optimal* assuming costs bounded away from zero
  - UCS is guided by path cost rather than path depth, so it may get in trouble if some edge costs are very small
- Worst-case time and space complexity $O(b^{W^*/\epsilon})$, where W^* is the optimal cost, and ϵ is such that all edge weights are no smaller than

Greedy (Best-First) Search

- UCS explores paths in all directions through all neighbor nodes
- Can we bias the search to try to get “closer” to the goal?
  - We need a measure of distance to the goal
    - → It would be ideal to use the length of the shortest path
    - → but this is exactly what we are trying to compute!

Greedy (Best-First) Search

- UCS explores paths in all directions through all neighbor nodes
- Can we bias the search to try to get "closer" to the goal?
  - We need a measure of distance to the goal
    - → It would be ideal to use the length of the shortest path
    - → but this is exactly what we are trying to compute!
- We can *estimate* the distance to the goal through a heuristic function:

$$h: V \rightarrow \mathbb{R}_{\geq 0}$$

  - $h(v)$ is the estimate of the distance from v to goal
  - Ex: the Euclidean distance to the goal (as the crow flies)
- A reasonable strategy is to always try to move in such a way to minimize the estimated distance to the goal

Greedy Search

$Q \leftarrow \langle start \rangle$ *// initialize queue with start*

while $Q \neq \emptyset$:

  pick (and remove) the path P with the lowest *heuristic* cost *($h(head(P))$)* from Q

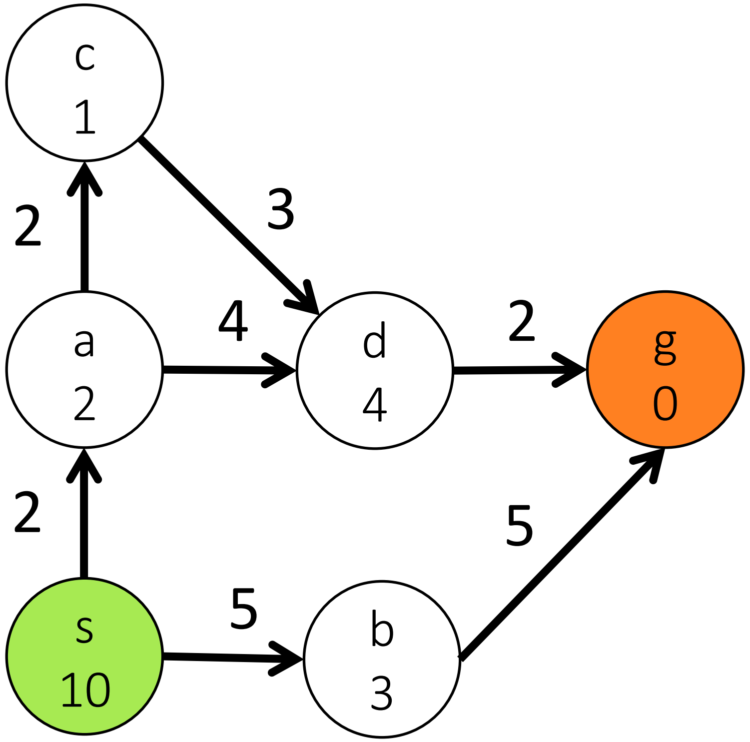
  if $head(P) = x_{goal}$ then return P *// Reached the goal*

  for each vertex v such that $(head(P), v) \in E$, do *// for all neighbors*

    add $\langle v, P \rangle$ to Q *// Add expanded paths*

Return FAILURE ***// nothing left to consider***

Example of Greedy Search



Remarks on Greedy Search

- Greedy (Best-First) search is similar to Depth-First Search
  - keeps exploring until it has to back up due to a dead end
- **Not complete** and not optimal, but is often fast and efficient, depending on the heuristic function h

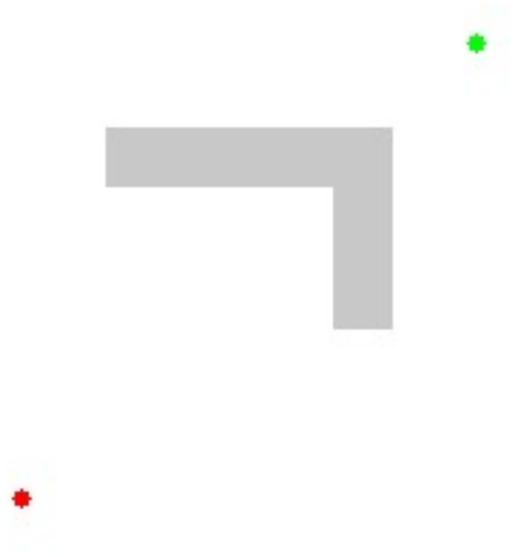
Summary

- Introduced basic concepts important for path and motion planning
  - Discussed the differences between the two planning strategies and considerations for various algorithms
- Reviewed graph definitions and naïve search methods
  - Uninformed and Greedy searches are okay, but not perfect
- *Next time:* Learn about the final search method that is better informed: A Search (A* and Hybrid A*)

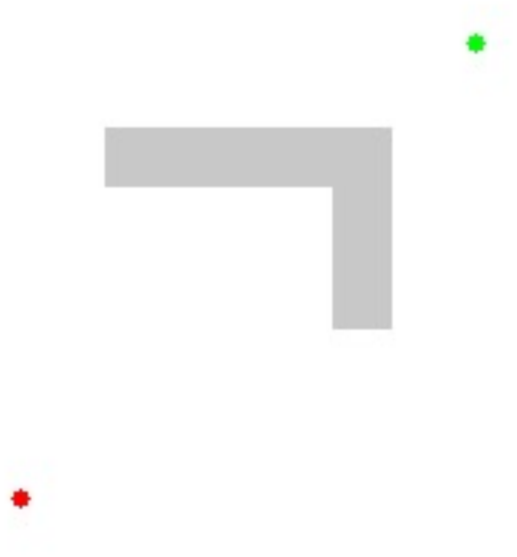
Extra Slides

Graph Search Methods

A* search algorithm.



Dijkstra's algorithm.



Credit: Subh83 on Wikipedia