Lecture 13: Planning

Professor Katie Driggs-Campbell

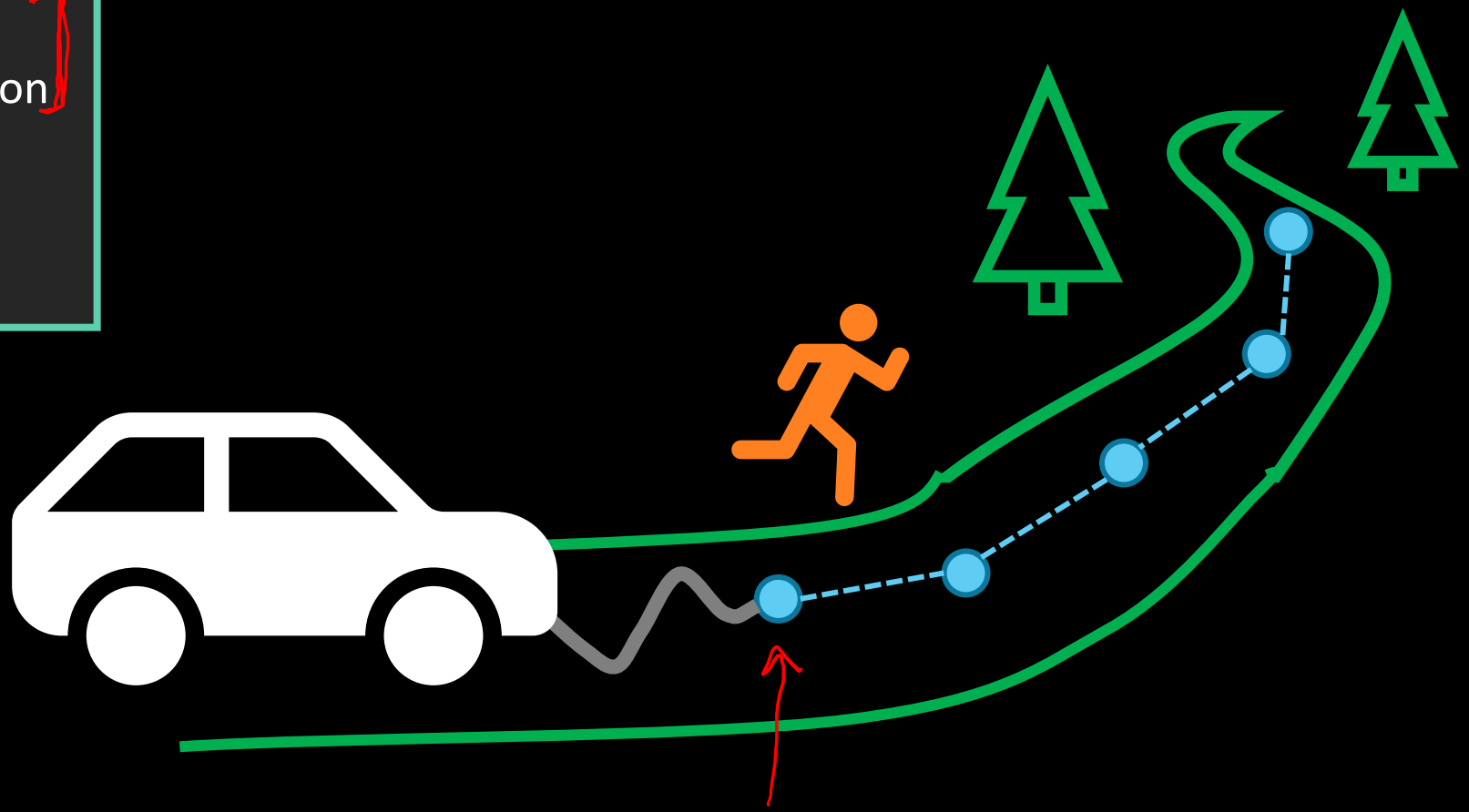
March 16, 2021

ECE484: Principles of Safe Autonomy

Administrivia

- Bayes Thm / Filter examples posted
- Milestone Report due Friday

- Vehicle Modeling
- Localization
- Detection & Recognition
- Control
- 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
  - A search

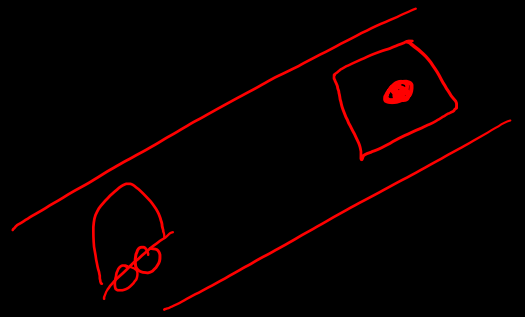
Today's Plan

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

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
- **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

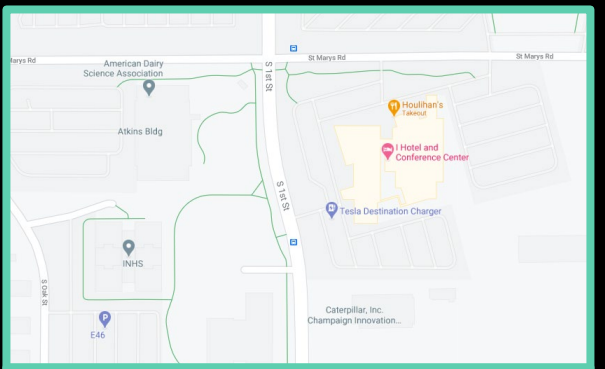
- **Multiple-query versus single-query planning**
- **“Anytime” planning**
  - Continues to look for better solutions after first solution is found
- **Computational complexity**
  - Characterization of the amount of time a planner takes to run or the amount of memory it requires
- **Completeness**
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

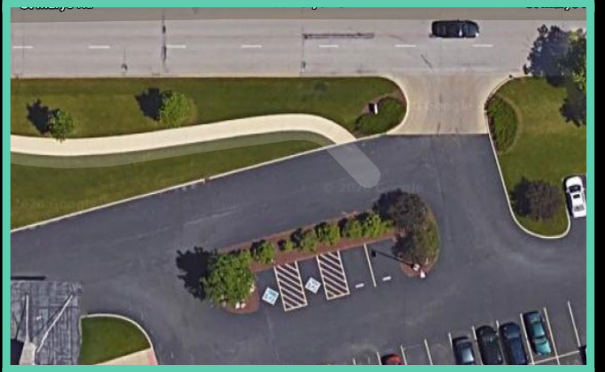
Search Performance Metrics

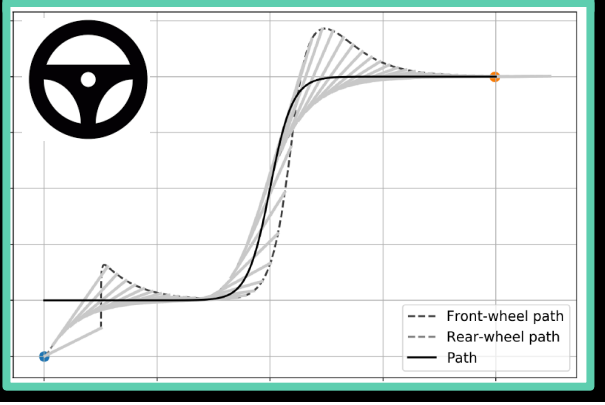
- **Soundness:** when a solution is returned, is it guaranteed to be a correct path?
- **Completeness:** is the algorithm guaranteed to find a solution when there is one?
- **Optimality:** How close is the found solution to the best solution?
- **Space complexity:** How much memory is needed?
- **Time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

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







Break-out Room Discussion

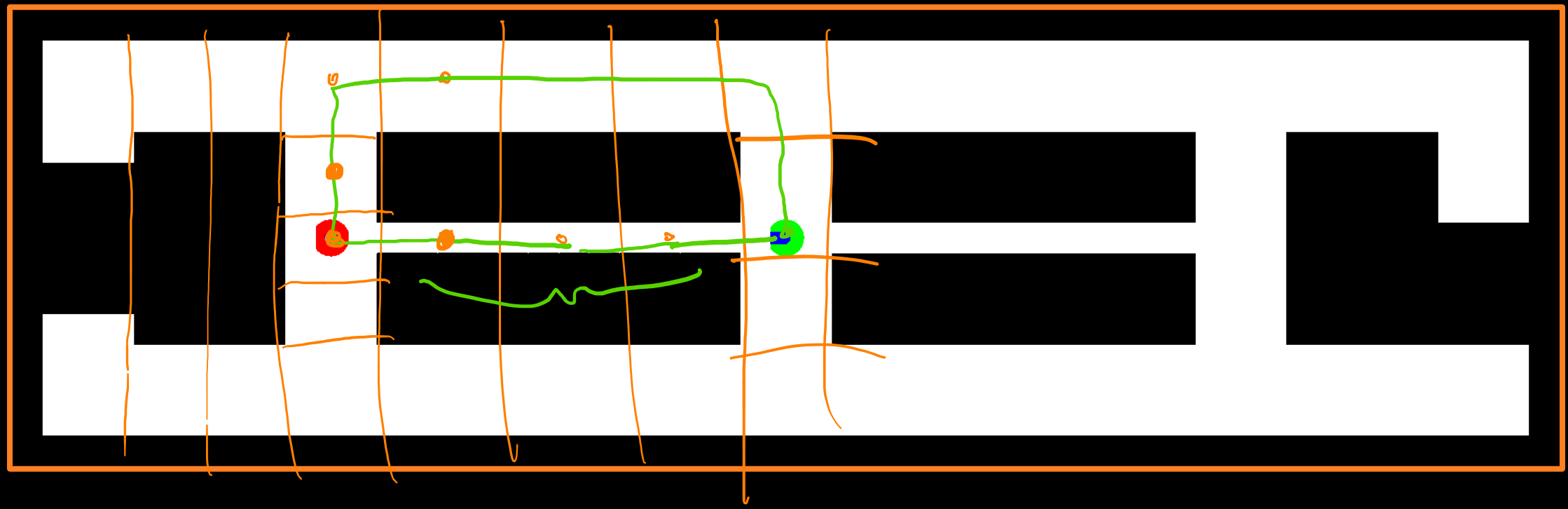
- What are some use cases, considerations, and requirements for different planning modules?
  - Ex: navigation, trajectory or motion planning, behavior planning

Today's Plan

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

Planning as a Search Problem

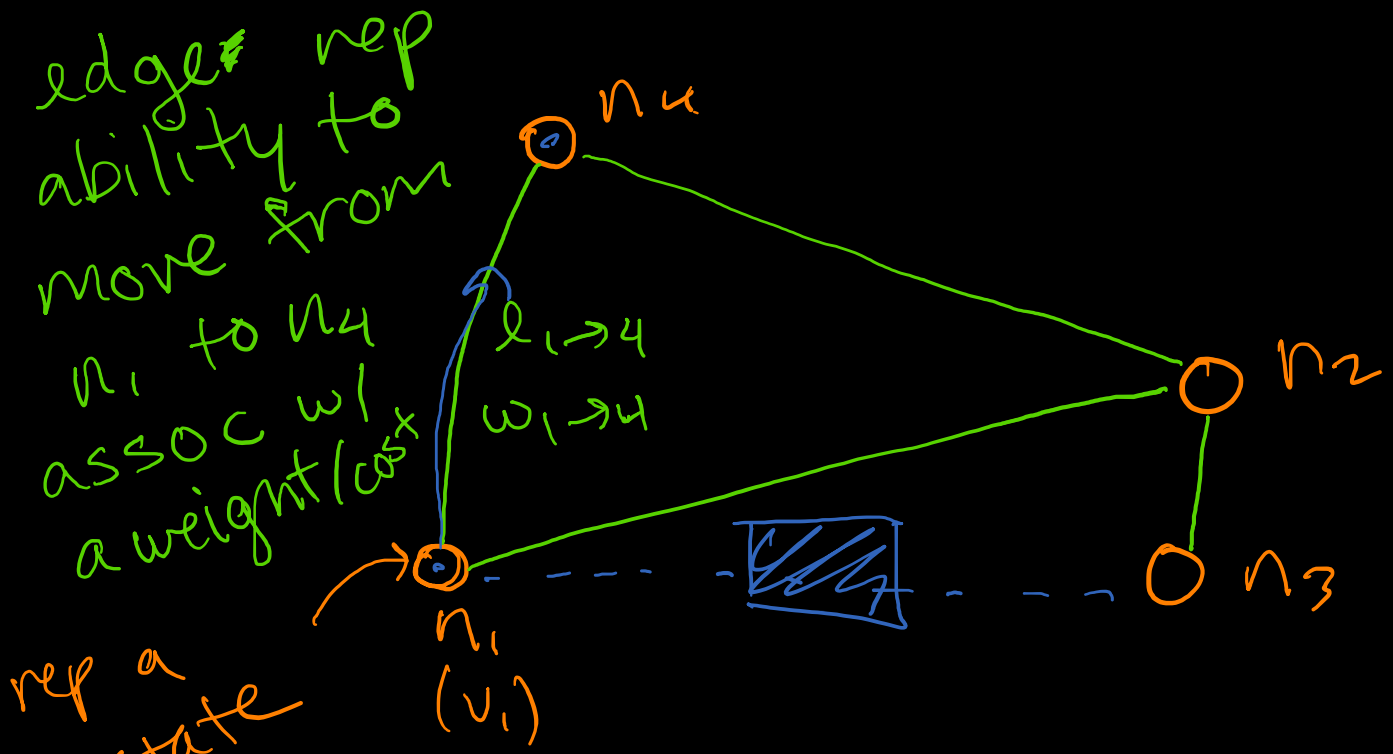
This is a 2D discretization, but we can generalize to higher dimensions (e.g., position, heading, mode)



Graphs and Trees

vertices V

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



rep as a matrix

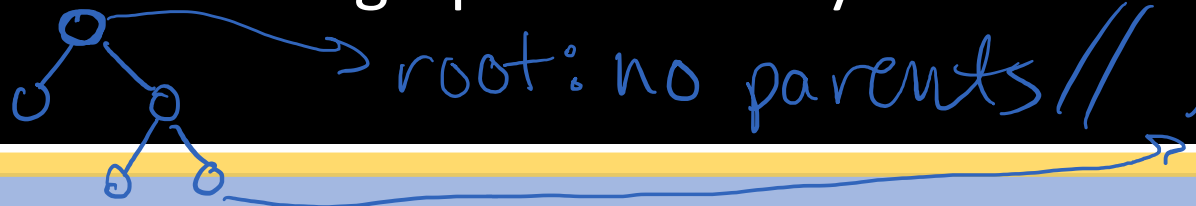
$A \in \mathbb{R}^{n \times n}$

$a_{ij} \leftarrow w_{i\to j}$

0 if no link

some graphs are directed

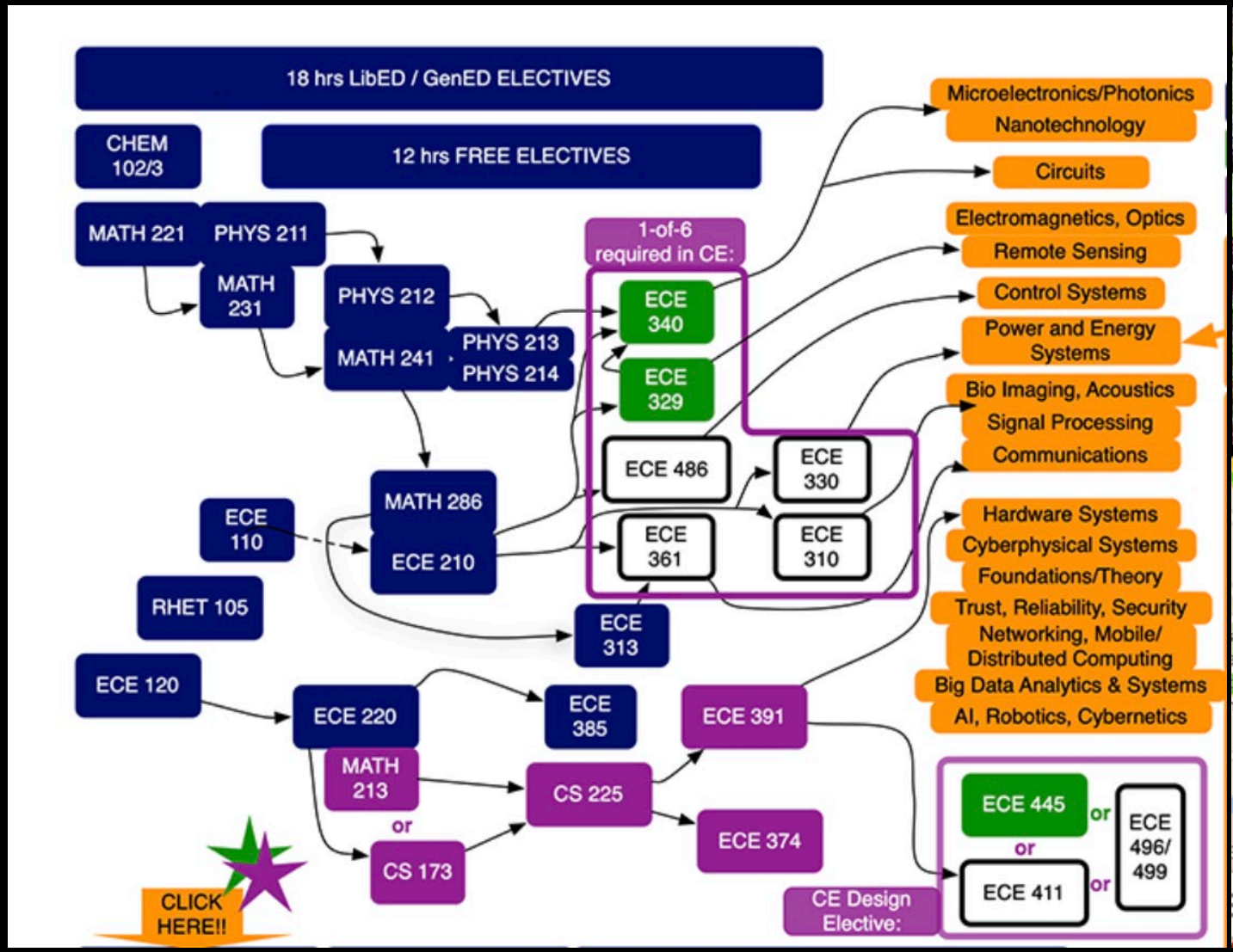
A **tree** is a directed graph with no cycles and each node has at least one parent

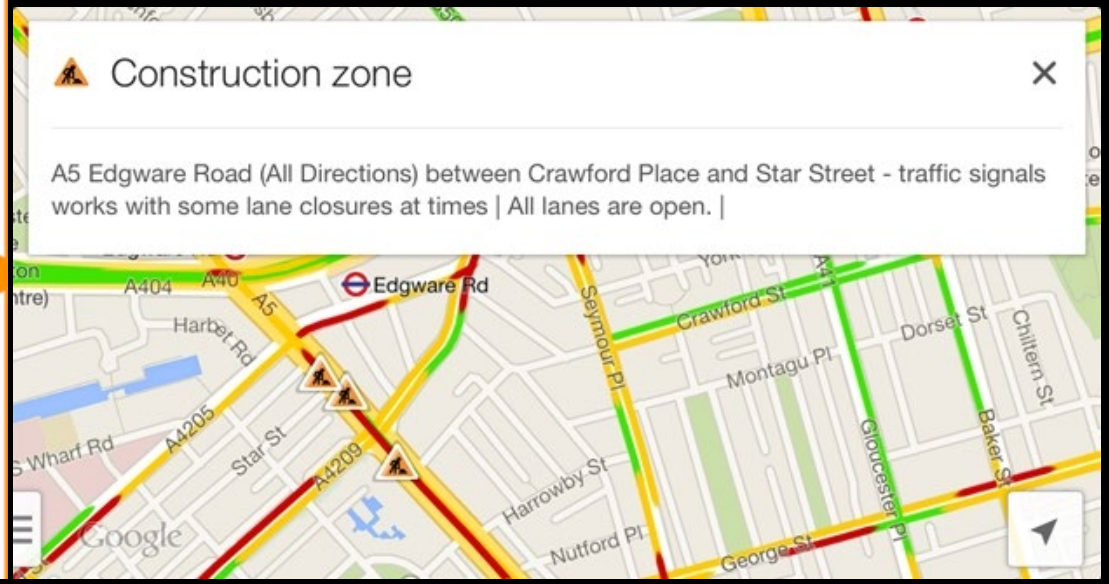


Problem Statement: find shortest path

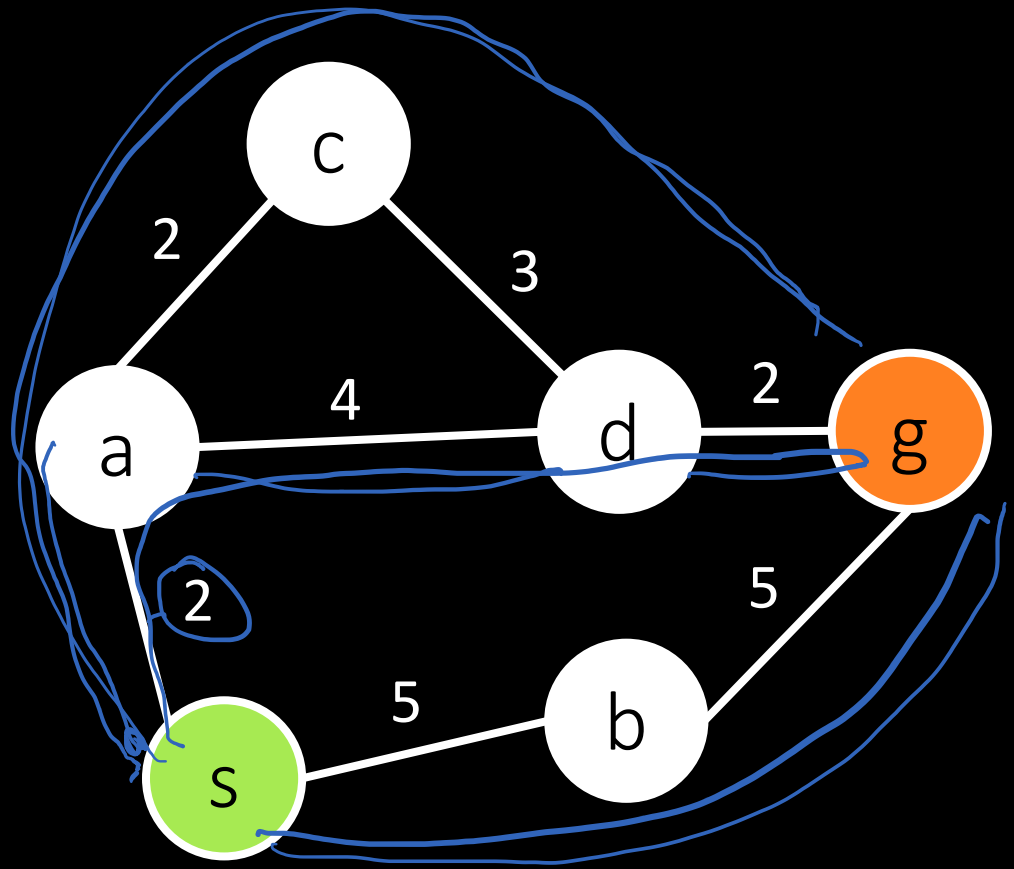
- Input: $\langle V, E, w, x_{start}, x_{goal} \rangle$
  - V: (finite) set of vertices
  - $E \subseteq V \times V$: (finite) set of edges
  - $w: E \to \mathbb{R}_{>0}$: a function that associates to each edge e to a strictly positive weight $w(e)$ (e.g., cost, distance, time, fuel)
  - $x_{start},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


18 hrs LibED / GenED ELECTIVES
CHEM 102/3
12 hrs FREE ELECTIVES
MATH 221
PHYS 211
MATH 231
PHYS 212
MATH 241
PHYS 213
PHYS 214
1-of-6 required in CE:
ECE 340
ECE 329
ECE 486
ECE 330
ECE 361
ECE 310
ECE 313
MATH 286
ECE 210
ECE 110
RHET 105
ECE 120
ECE 220
ECE 385
MATH 213
or
CS 173
CS 225
ECE 374
ECE 391
CE Design Elective:
ECE 445
or
ECE 411
or
ECE 496/ 499
CLICK HERE!!
Microelectronics/Photonics
Nanotechnology
Circuits
Electromagnetics, Optics
Remote Sensing
Control Systems
Power and Energy Systems
Bio Imaging, Acoustics
Signal Processing
Communications
Hardware Systems
Cyberphysical Systems
Foundations/Theory
Trust, Reliability, Security
Networking, Mobile/ Distributed Computing
Big Data Analytics & Systems
AI, Robotics, Cybernetics


Construction zone
A5 Edgware Road (All Directions) between Crawford Place and Star Street - traffic signals works with some lane closures at times | All lanes are open. |
Edgware Rd
A404
A40
A5
Harbet Rd
Crawford St
Seymour Pl
Montagu Pl
Dorset St
Chiltern St
Baker St
Gloucester Pl
A4205
Star St
A4209
Harrowby St
Nutford Pl
George St
Google

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
  - A search

Uniform cost search (Uninformed search)

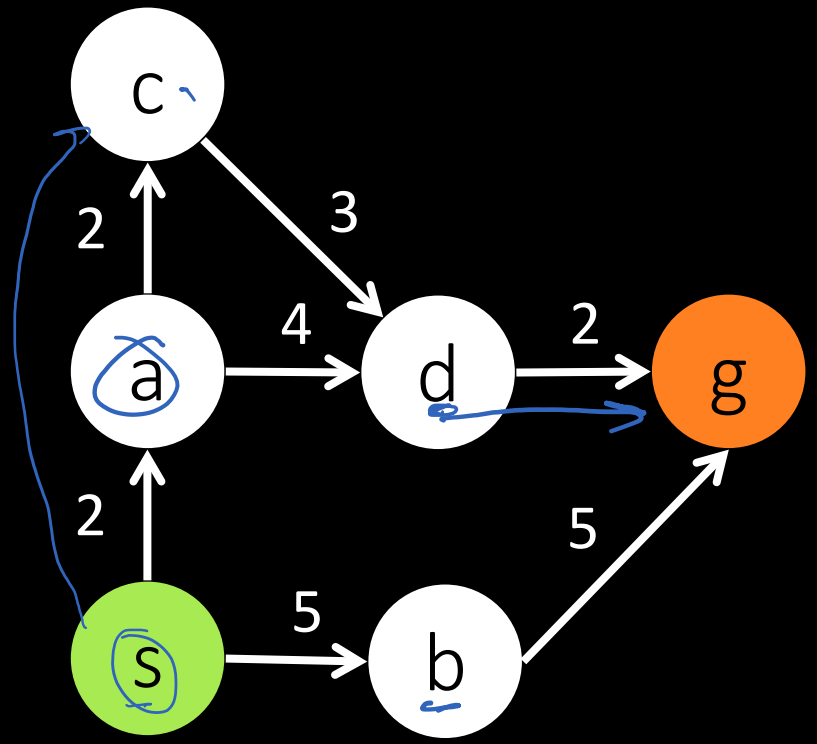
$\boldsymbol{Q} \leftarrow \langle \boldsymbol{start} \rangle$	*// maintains paths*
	// initialize queue with start
while $\boldsymbol{Q} \neq \emptyset$:	
pick (and remove) the path $\boldsymbol{P}$ with the lowest cost ($\boldsymbol{g} = \boldsymbol{w}(\boldsymbol{P})$) from Q	
if $\boldsymbol{head}(\boldsymbol{P}) = \boldsymbol{x}_{\boldsymbol{goal}}$ 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

Q:

Path	Cost
$\langle s \rangle$	0
$\langle a, s \rangle$	2
$\langle b, s \rangle$	5
$\langle c, a, s \rangle$	4
$\langle d, a, s \rangle$	6
$\langle g, d, a, s \rangle$	8

$\langle g, b, s \rangle$	10
$\langle d, c, a, s \rangle$	7
$\langle g, d, c, a, s \rangle$	9



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!
- 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*

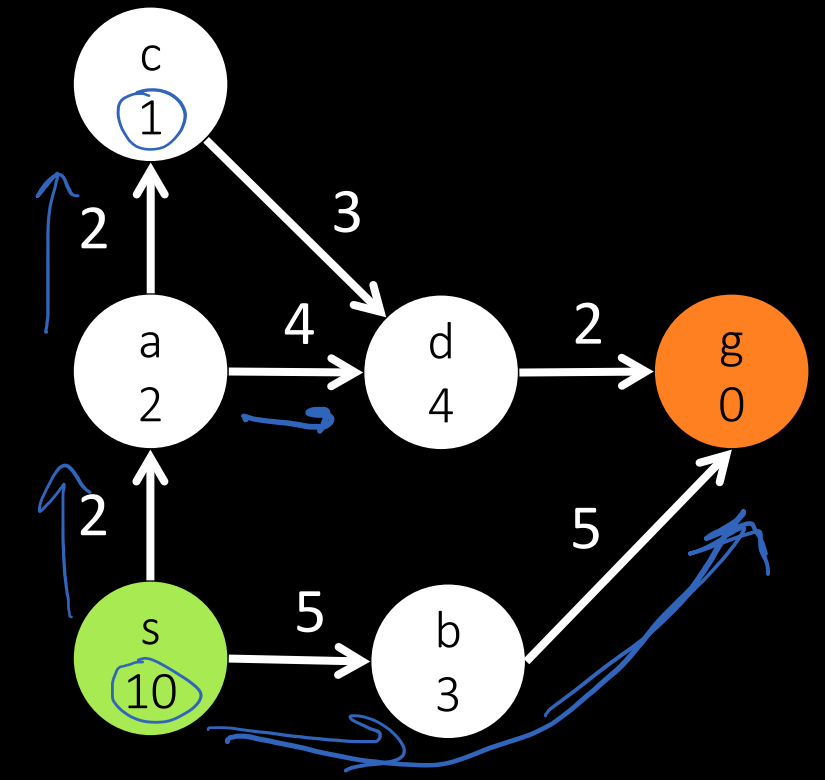
add $\langle v, P \rangle$ to Q *// Add expanded paths*

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

Example of Greedy Search

Q:

Path	Cost	h
$\langle s \rangle$	0	10
$\langle a, s \rangle$	2	2
$\langle b, s \rangle$	5	3
$\langle c, a, s \rangle$	4	1
$\langle d, a, s \rangle$	6	4
$\langle g, b, s \rangle$	10	0



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

Informed Search: A* Search

- UCS is optimal, but may wander around a lot before finding the goal
- Greedy is not optimal, but can be efficient, as it is heavily biased towards moving towards the goal
- A new idea:
  - Keep track of both the cost of the partial path to get to a vertex $g(v)$ and the heuristic function estimating the cost to reach the goal from a vertex $h(v)$
  - Choose a "ranking" function to be the sum of the two costs:

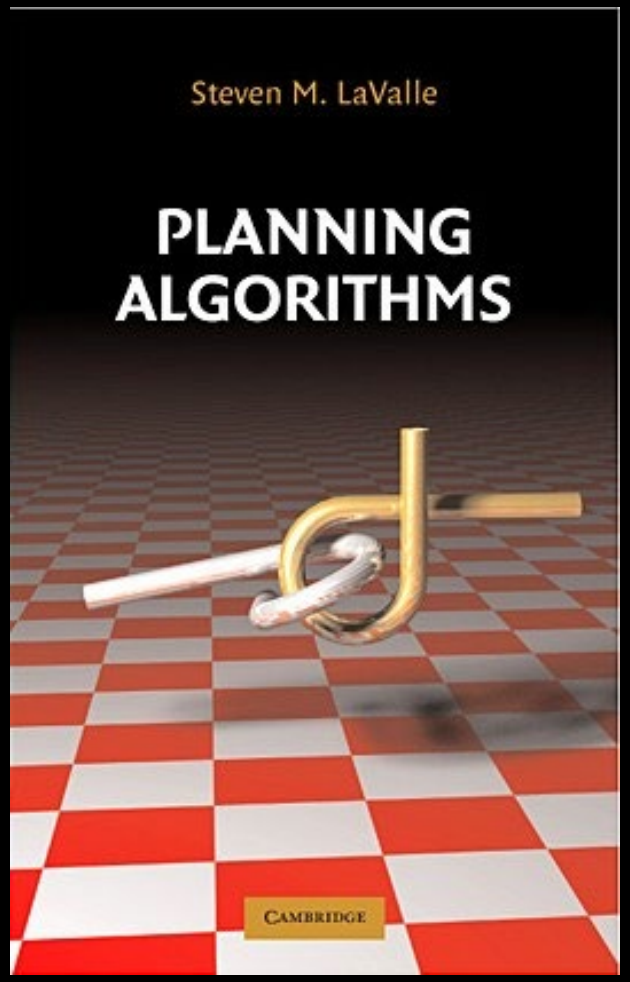
$$f(v) = g(v) + h(v)$$

  - $g(v)$: cost-to-arrive (from the start to v)
  - $h(v)$: cost-to-go estimate (from v to the goal)
  - $f(v)$: estimated cost of the path (from the start to v and then to the goal)

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 A Search (A* and Hybrid A*)

Extra Slides



CSL Prof. LaValle central to Oculus' $2 billion success

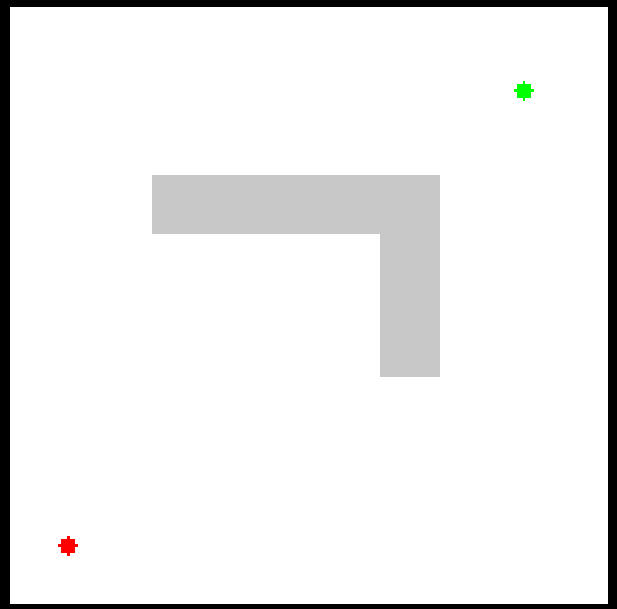
Apr 17, 2014

Rick Kubetz, Engineering Communications Office

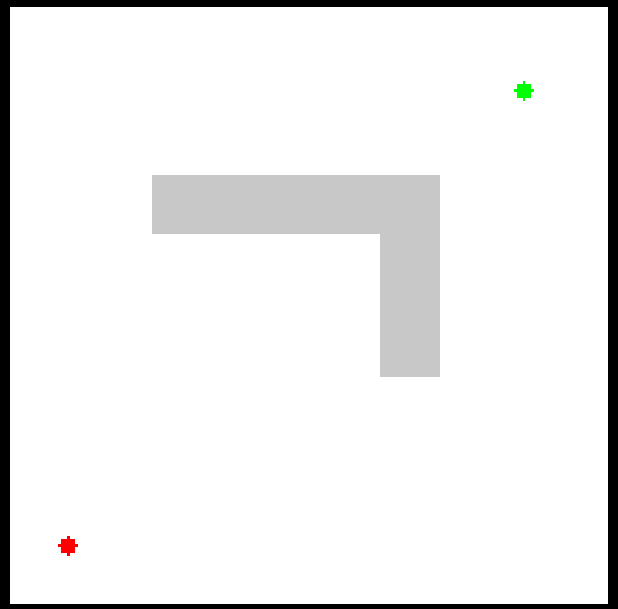
On March 25, both the business and technology news pages excitedly announced Facebook's $2 billion acquisition of Oculus VR, the maker of a virtual reality gaming headset called Oculus Rift.

Graph Search Methods

A* search algorithm.

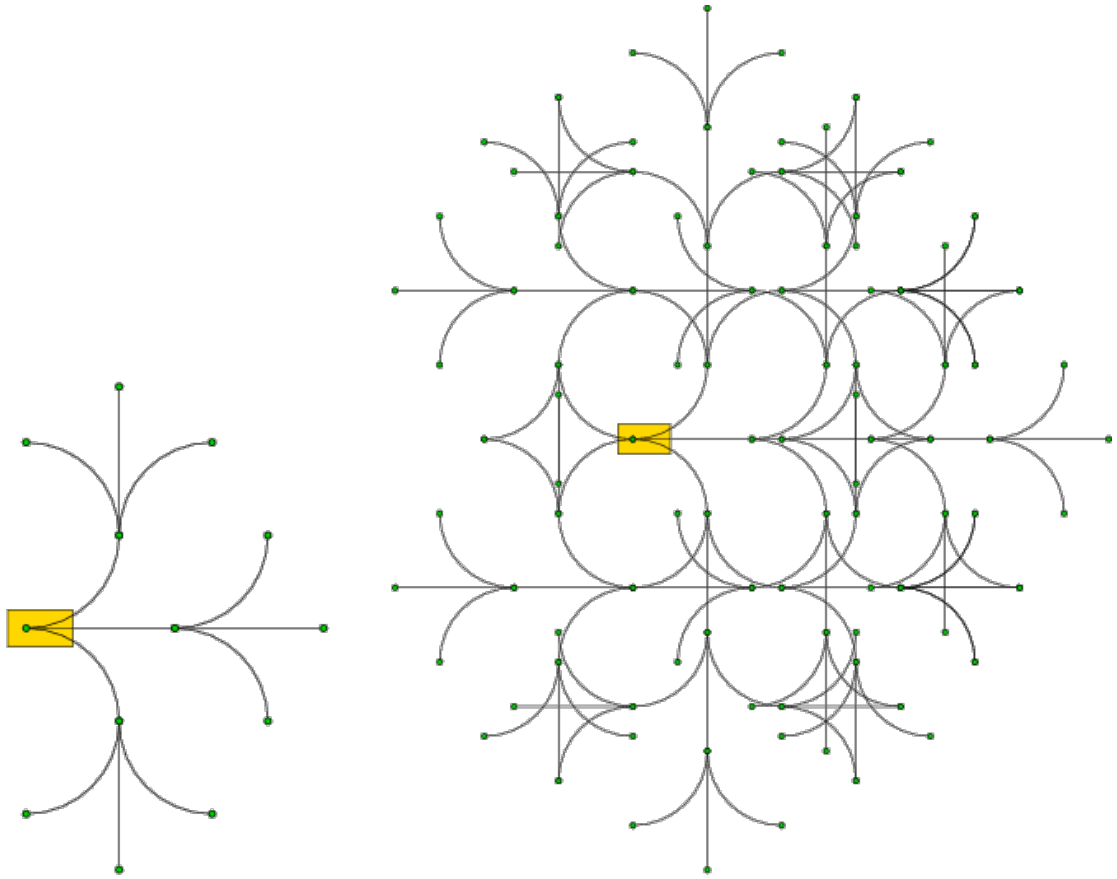


Dijkstra's algorithm.



Credit: Subh83 on Wikipedia

Reachability Tree for Dubin's Car



Credit: Steven LaValle, Planning Algorithms