ECE 484: Principles of Safe Autonomy (Fall 2025)
Lecture 15: Search and Planning

Professor: Huan Zhang

https://publish.illinois.edu/safe-autonomy/

https://huan-zhang.com

huanz@illinois.edu

Slides adapted from Prof. Sayan Mitra's slides for Spring 2025

Announcements

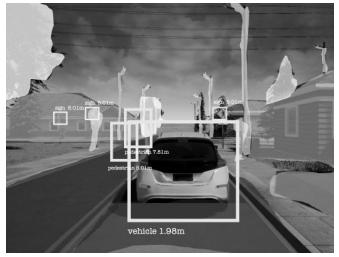
- GEM and F1-tenth groups: please sign up
- Many F1-tenth teams still need to finish their safety training
- Please ensure that you show up at the time slot signed up (if late by 15minutes, TAs will leave)
- Project checkpoint: week of 11/14 (tentative, check CampusWire)

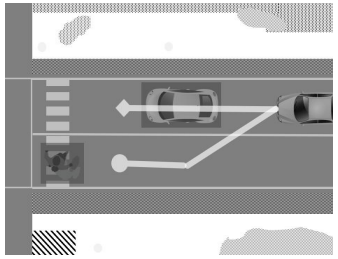
Final review of localization module

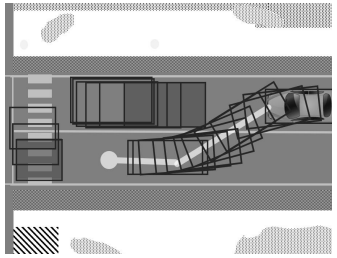
- Fundamentals: **Conditional Probability**, **Bayes Rule**, **Conditional Independence**
- Motion model and measurement model
- Bayes Filters: prediction + correction steps
  - Discrete Bayes Filter
  - Particle Filter
  - Kalman Filter
- Pros and cons of each type of filter
- SLAM will not appear in your exam, but you still need to know the key concept of **conditional independence** and the decomposition of the posterior

GEM platform

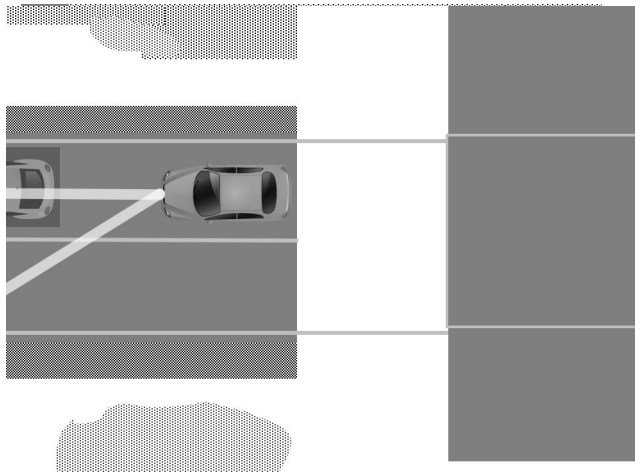
Autonomy pipeline







Sensing	Perception	Decisions and planning	Control
Physics-based models of camera, LIDAR, RADAR, GPS, etc.	Programs for object detection, lane tracking, scene understanding, etc.	Programs and multi-agent models of pedestrians, cars, etc.	Dynamical models of engine, powertrain, steering, tires, etc.



Decisions and planning

Programs and multi-agent models of pedestrians, cars, etc.

Search and planning problems appear in different levels of autonomy stack

Global path planner --- invoked at each new *checkpoint*

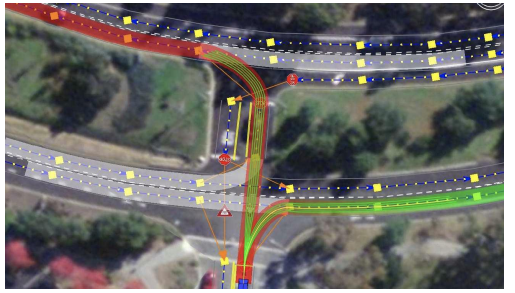
- finds paths from every point in the map to next checkpoint
- dynamic programming

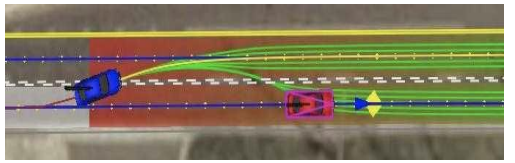
Road navigation

- For each path, the planner rolls out several discrete trajectories that are parallel to the smoothed center of the lane

Freeform navigation (parking lots)

- Generate arbitrary trajectories (irrespective of road structure) using modified A*



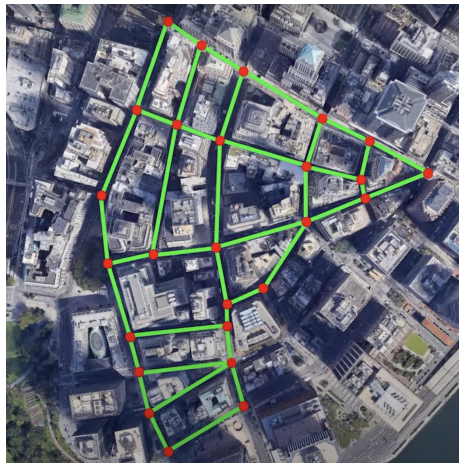


Junior: The Stanford Entry in the Urban Challenge, Thrun et al., 2008

Outline

- Deterministic search
  - Uninformed search
  - Informed search
  - Optimal search: A, A*
- Randomized search (next lecture)
  - Probabilistic Maps
  - Rapidly expanding random trees (RRT)

Planning as graph search



Making a Drone Smarter With Motion Planning Nicholas Rehm

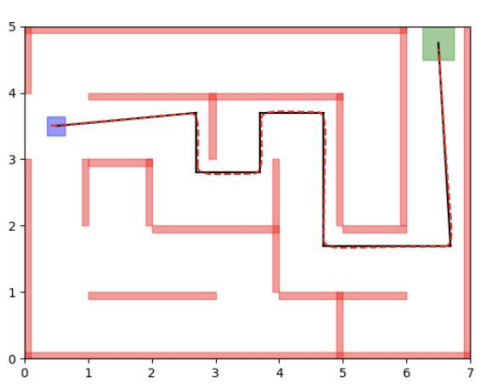
Search for collision free trajectories can be viewed abstractly as a graph search problem, where the nodes represent blocks of free space and the edges connect spatially adjacent nodes.

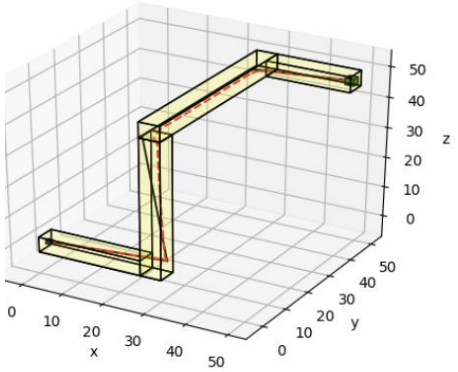
We can solve such problems using the graph search algorithms like (uninformed) Breadth-First Search and Depth-First Search

Roadmaps are not arbitrary graphs: Some paths are more preferable than others (e.g., shorter, faster, less costly in terms of fuel/tolls/fees, more stealthy, etc.)

Good guesses for distances can be made, even without knowing the full graph or the optimal paths

Can we *utilize this information* to find efficient paths, efficiently?





https://kmmille.github.io/FACTEST/

Problem statement: find shortest path

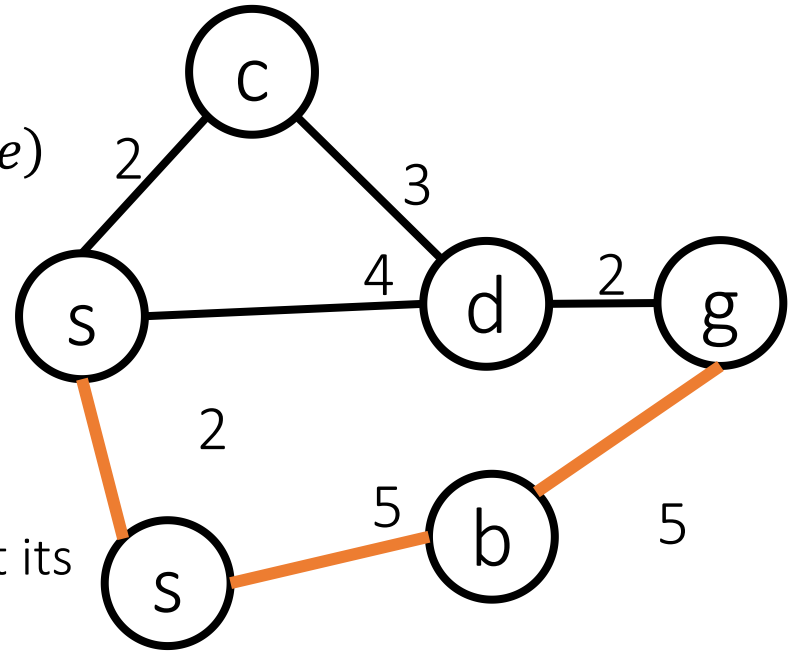
Input: $\langle V, E, w, start, goal \rangle$

- V: (finite) set of vertices
- $E \subseteq V \times V$: (finite) set of edges
- $w : E \to \mathbb{R}_{>0}$: associates to each edge e to a positive weight $w(e)$
- $start, goal \in V$: start and end vertices.

A path is a sequence of vertices $p = v_0 \ldots v_k$ such that $(v_i, v_{i+1}) \in E$

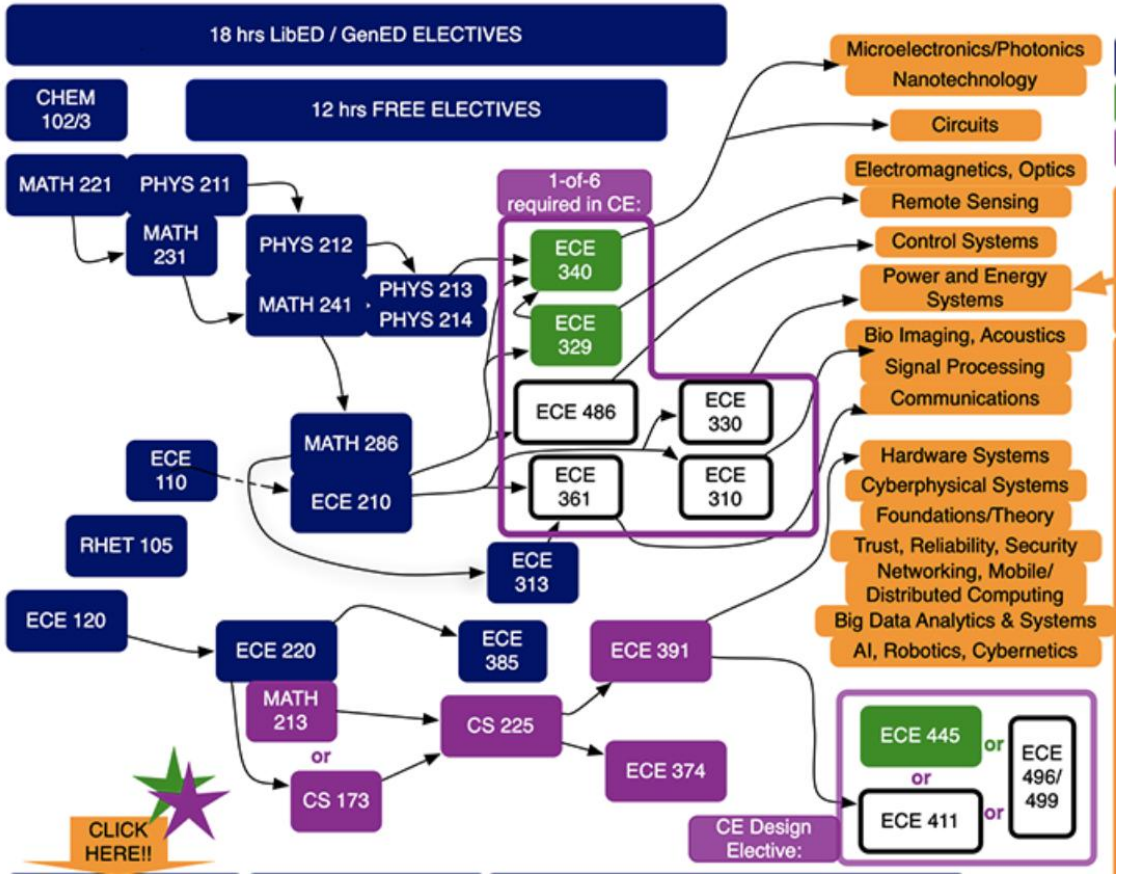
$w(p) = \sum_{i=0}^{k-1} w(v_i, v_{i+1})$ $\quad \mathrm{head}(p) = v_0 \quad \mathrm{end}(p) = v_k$

Output: a path p with $head(p) = start$ and $end(p) = gaol$, such that its $w(p)$ is minimal among all such paths



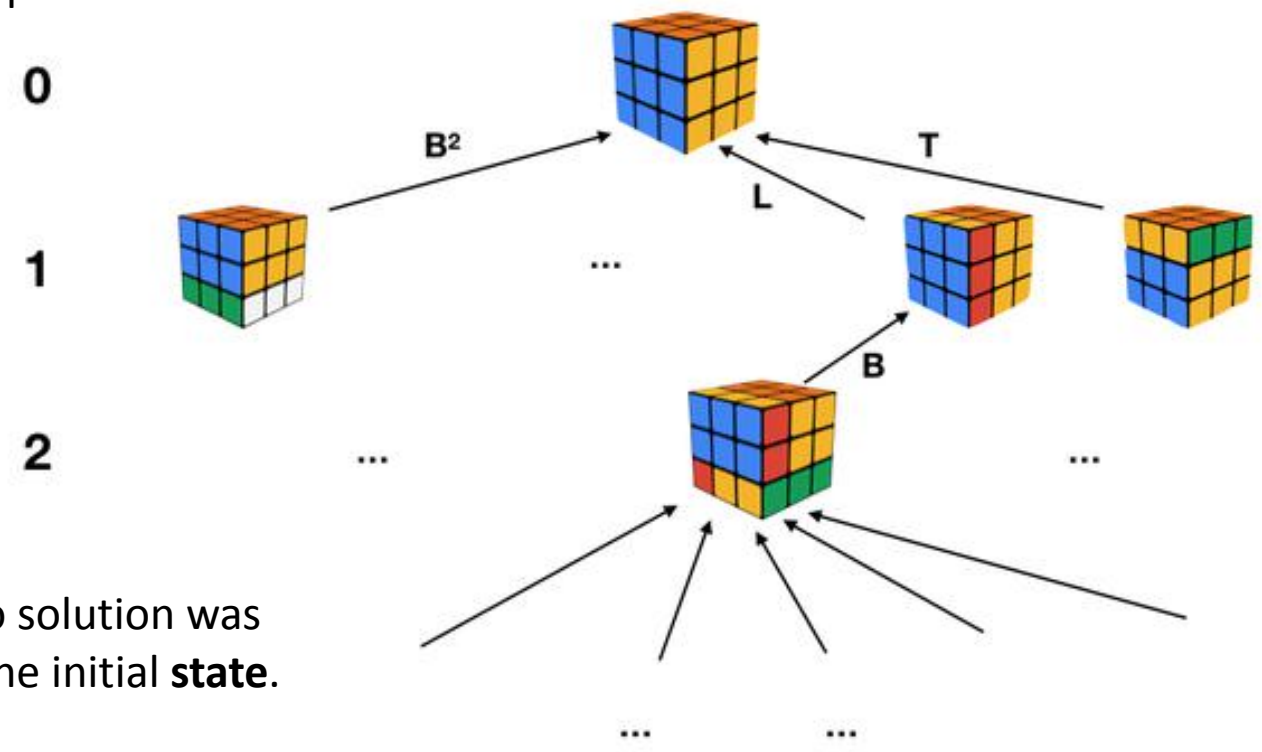
Algorithms: Dijkstra O((|V|+|E|)log|V|) , Bellman-Ford O(|V|x|E|),..

Example graph with imperfectly known information



The Graph Can be Large

Solving Rubik's cube as a graph problem



Number of states or vertices
43,252,003,274,489,856,000

Yet, maximum length of path to solution was shown to be 20, regardless of the initial **state**.

T. Rokicki, working with Google, proved "God's number" to be 20, in 2010.

Search Algorithm Performance Metrics

Soundness: when a solution is returned, is it guaranteed to be **correct** path?

Completeness: is the algorithm **guaranteed to find a solution** when one exists?

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?

Uniform cost search (Uninformed search)

$Q \leftarrow \langle start \rangle$ // maintains paths **sorted by cost**

// initialize queue with start

while $Q \neq \emptyset$:

    pick (and remove) the path P with lowest cost $w(P)$

    if $head(P) = goal$ then return P ; // Reached the goal

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

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

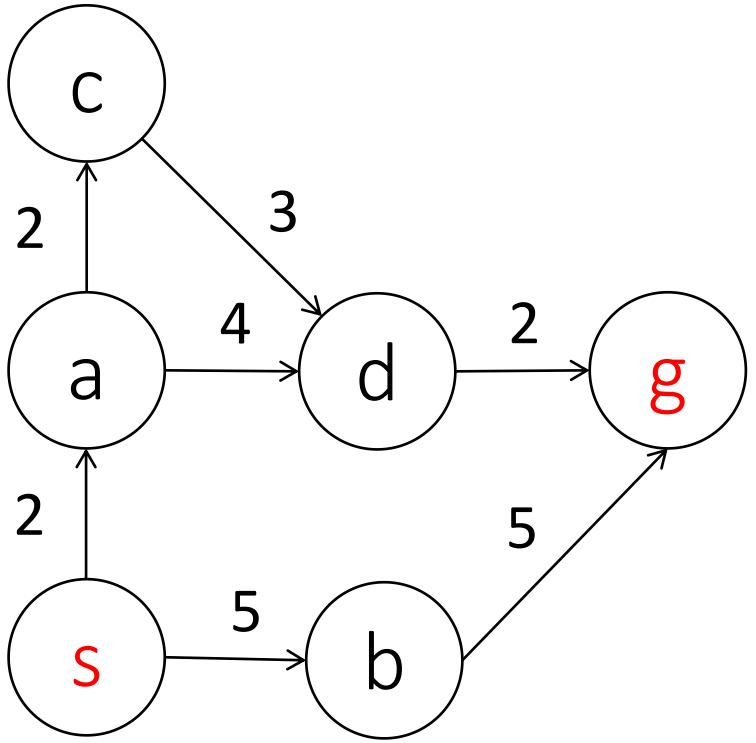
return $FAILURE$; // nothing left to consider

Note no ***visited*** list; Use no information obtained from the environment

Example of Uniform-Cost Search (s to g)

Q:

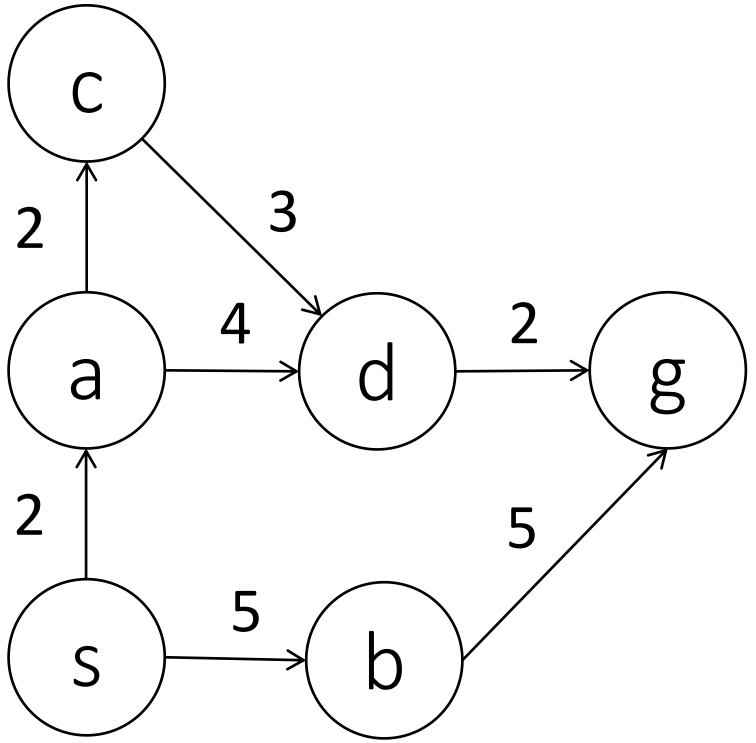
Path	Cost
$\langle s \rangle$	0



Example of Uniform-Cost Search

Q:

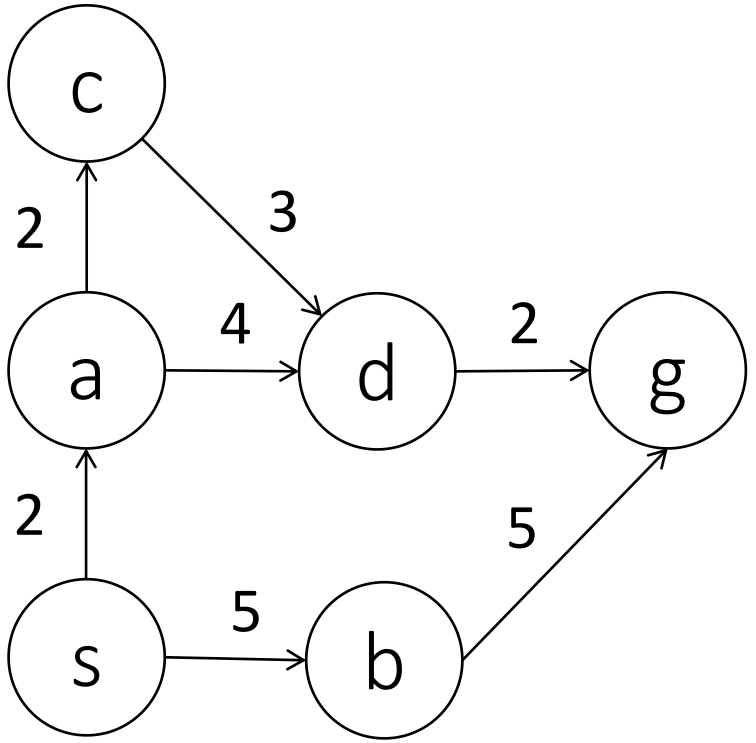
Path	Cost
$\langle a, s \rangle$	2
$\langle b, s \rangle$	5



Example of Uniform-Cost Search

Q:

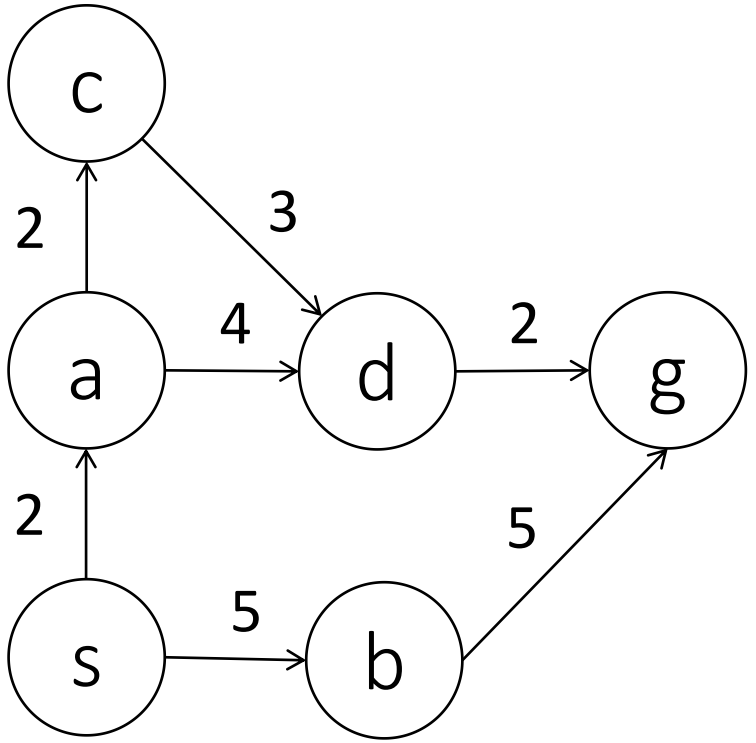
Path	Cost
$\langle c, a, s \rangle$	4
$\langle b, s \rangle$	5
$\langle d, a, s \rangle$	6



Example of Uniform-Cost Search

Q:

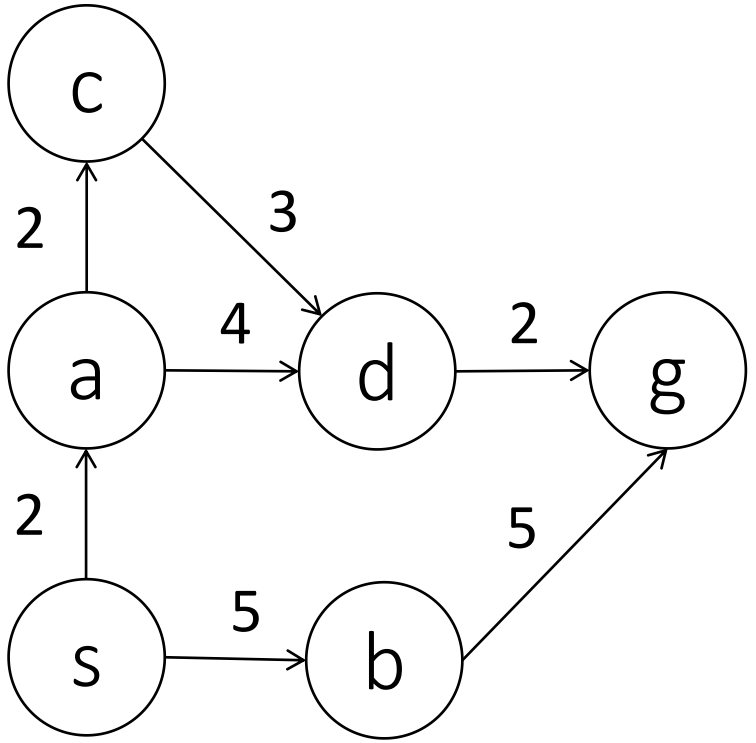
Path	Cost
$\langle b, s \rangle$	5
$\langle d, a, s \rangle$	6
$\langle d, c, a, s \rangle$	7



Example of Uniform-Cost Search

Q:

Path	Cost
$\langle d, a, s \rangle$	6
$\langle d, c, a, s \rangle$	7
$\langle g, b, s \rangle$	10

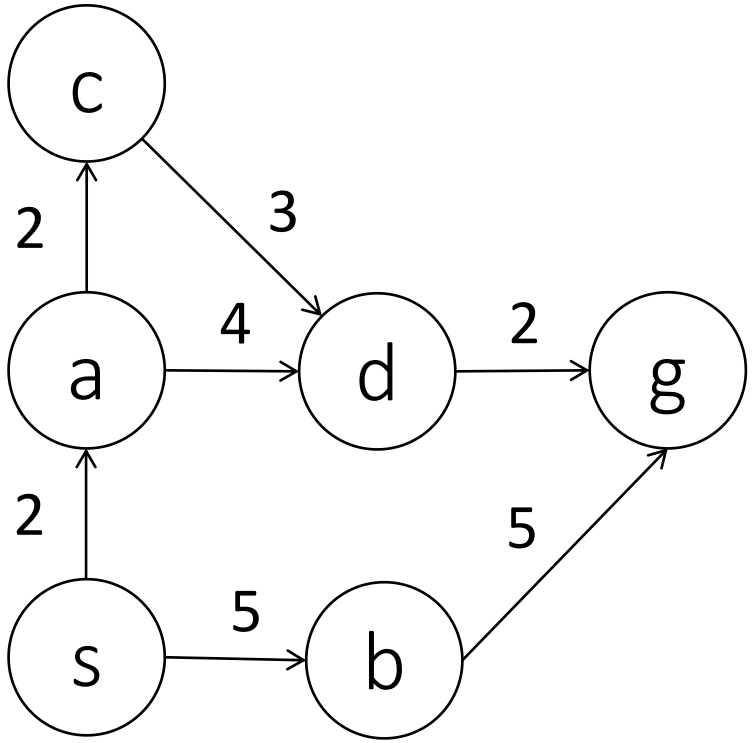


Notice a path to goal <g, b, s> has been found but the algorithm does not terminate, since the path is currently not at the head of Q

Example of Uniform-Cost Search

Q:

Path	Cost
$\langle d, c, a, s \rangle$	7
$\langle g, d, a, s \rangle$	8
$\langle g, b, s \rangle$	10

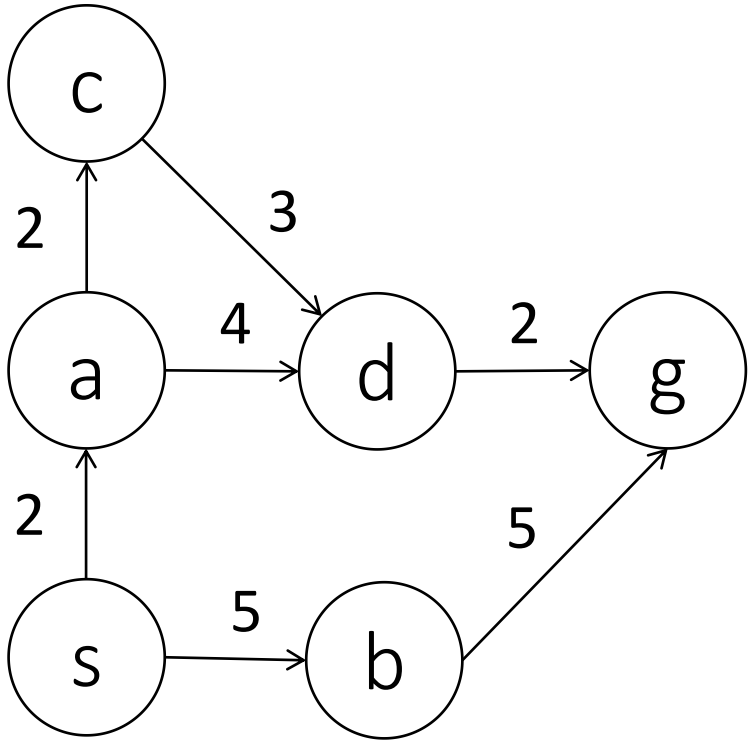


Another path to goal <g, d, a, s> has been found but the algorithm does not terminate, since the path is currently not at the head of Q

Example of Uniform-Cost Search

Q:

Path	Cost
$\langle \boldsymbol{g}, d, a, s \rangle$	8
$\langle g, d, c, a, s \rangle$	9
$\langle g, b, s \rangle$	10



Algorithm stops when smallest cost path in Q has head = g

Properties of Uniform Cost Search

UCS is an extension of BFS to the weighted-graph case (UCS = BFS if all edges have the same cost)

UCS is *sound, complete* and *optimal* (assuming costs bounded away from zero; zero or negative costs will cause infinite loops)

- Exercise: Prove the above claims
- Soundness: For any entry p in Q, is a path and head(p) = start

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 b is the branching factor, W^* is the optimal cost, and ϵ is such that all edge weights are no smaller than

Greedy or Best-First Search

UCS explores paths in all directions, with no bias towards the goal state

What if we 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

$h : V \rightarrow \mathbb{R}_{\geq 0}$. E.g., the Euclidean distance to the goal (as the crow flies)

$h(v)$ is the estimate of the distance from v to goal

A reasonable strategy is to always try to move in such a way to minimize the estimated distance to the goal: this is the basic idea of the greedy (best-first) search

Greedy/Best-first search

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

while $Q \neq \emptyset$:

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

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

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

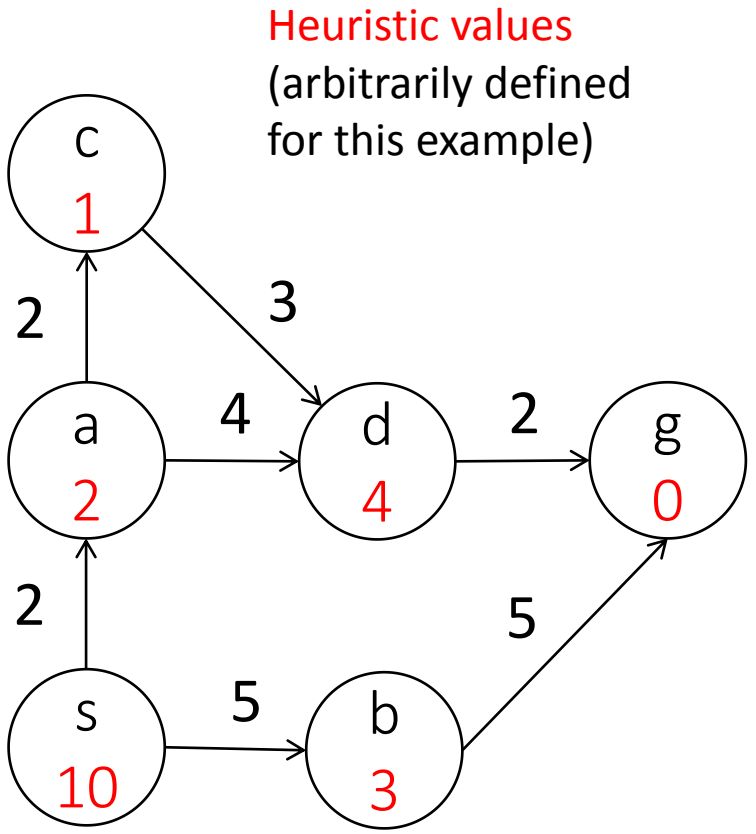
add $\langle v,\ P \rangle\ to\ Q$; *// Add expanded paths*

return $FAILURE$; *// nothing left to consider*

Example of Greedy search

Q:

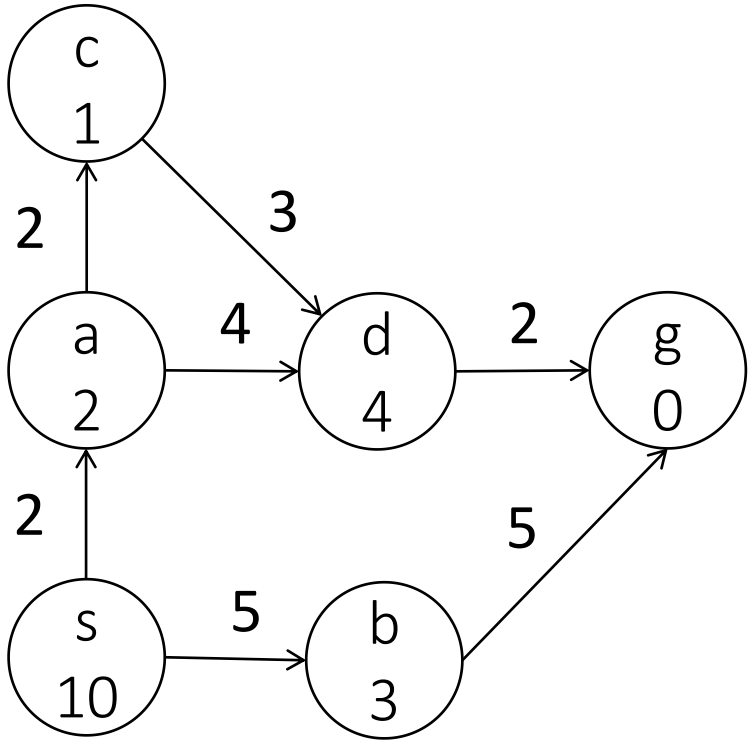
Path	Cost	h
$\langle s \rangle$	0	10



Example of Greedy search

Q:

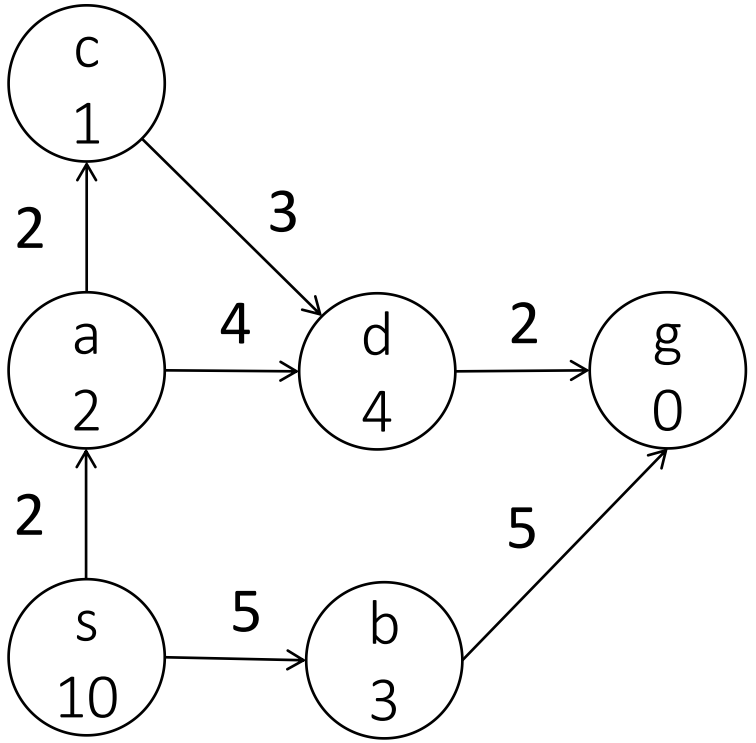
Path	Cost	h
$\langle a, s \rangle$	2	2
$\langle b, s \rangle$	5	3



Example of Greedy search

Q:

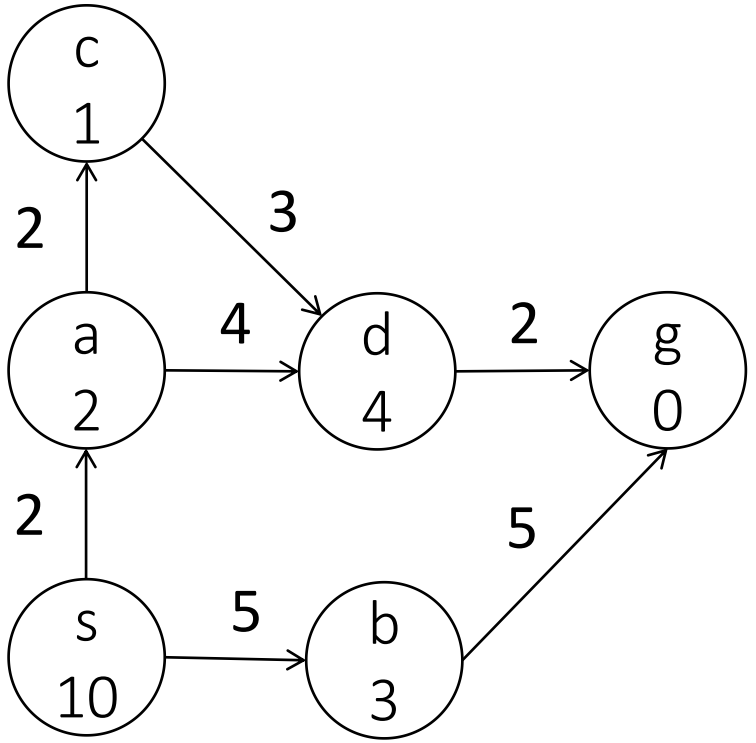
Path	Cost	h
$\langle c, a, s \rangle$	4	1
$\langle d, a, s \rangle$	6	4
$\langle b, s \rangle$	5	3



Example of Greedy search

Q:

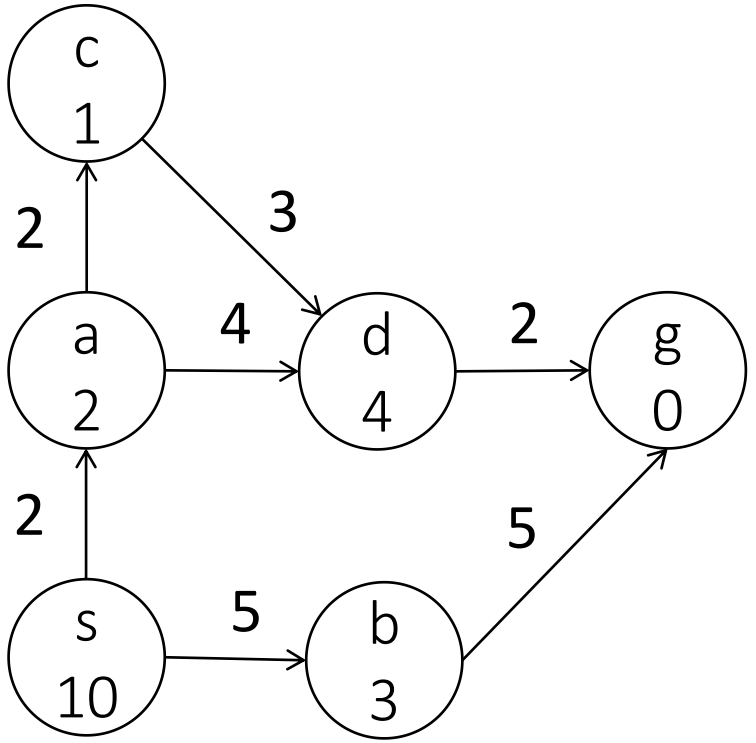
Path	Cost	h
$\langle d, c, a, s \rangle$	7	4
$\langle d, a, s \rangle$	6	4
$\langle b, s \rangle$	5	3



Example of Greedy search

Q:

Path	Cost	h
$\langle d, c, a, s \rangle$	7	4
$\langle d, a, s \rangle$	6	4
$\langle g, b, s \rangle$	10	0



Terminates with suboptimal path

Remarks on greedy/best-first 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 (why?) and not optimal, but is often fast and efficient, depending on the heuristic function h

- Exercise: Find a counter-example where path exists but bad heuristic function makes the algorithm loop forever

Worst-case time and space complexity?