ECE 484: Principles of Safe Autonomy (Fall 2025)
Lecture 16: Search and Planning: A* and heuristics

Professor: Huan Zhang

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

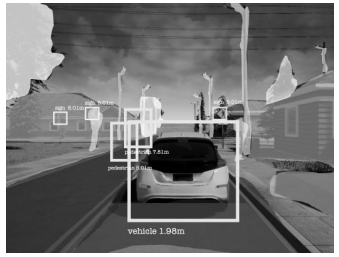
https://huan-zhang.com

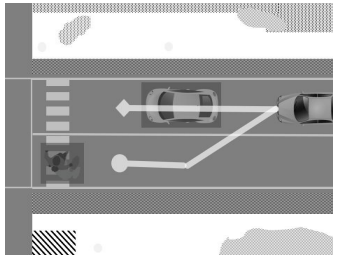
huanz@illinois.edu

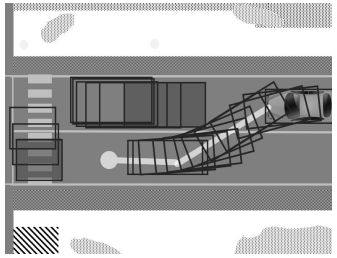
Slides adapted from Prof. Sayan Mitra's slides for Spring 2025

GEM platform

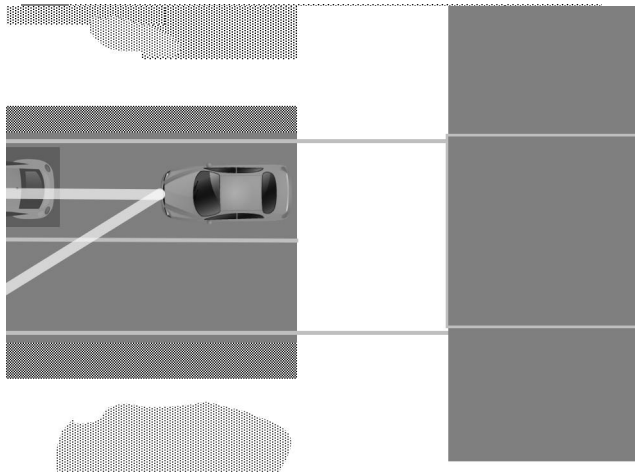
Autonomy pipeline







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.

Outline

- Deterministic search
  - Uniform Cost Search (uninformed search)
  - Informed search
    - Greedy/Best-first search
    - **A, A* search**
      - Admissible heuristics
      - design of heuristics

Problem statement: find shortest path

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

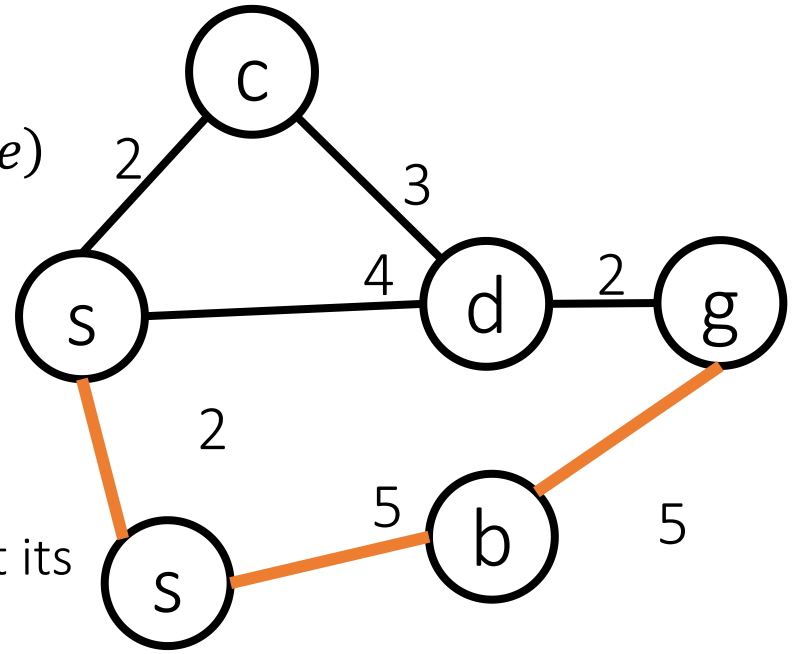
- V: (finite) set of vertices
- $E \subseteq V \times V$: (finite) set of edges
- $w : E \to \mathbb{R}_{>0}$: associates to each edge e to a positive weight $w(e)$
- $start, goal \in V$: start and end vertices.

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

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

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

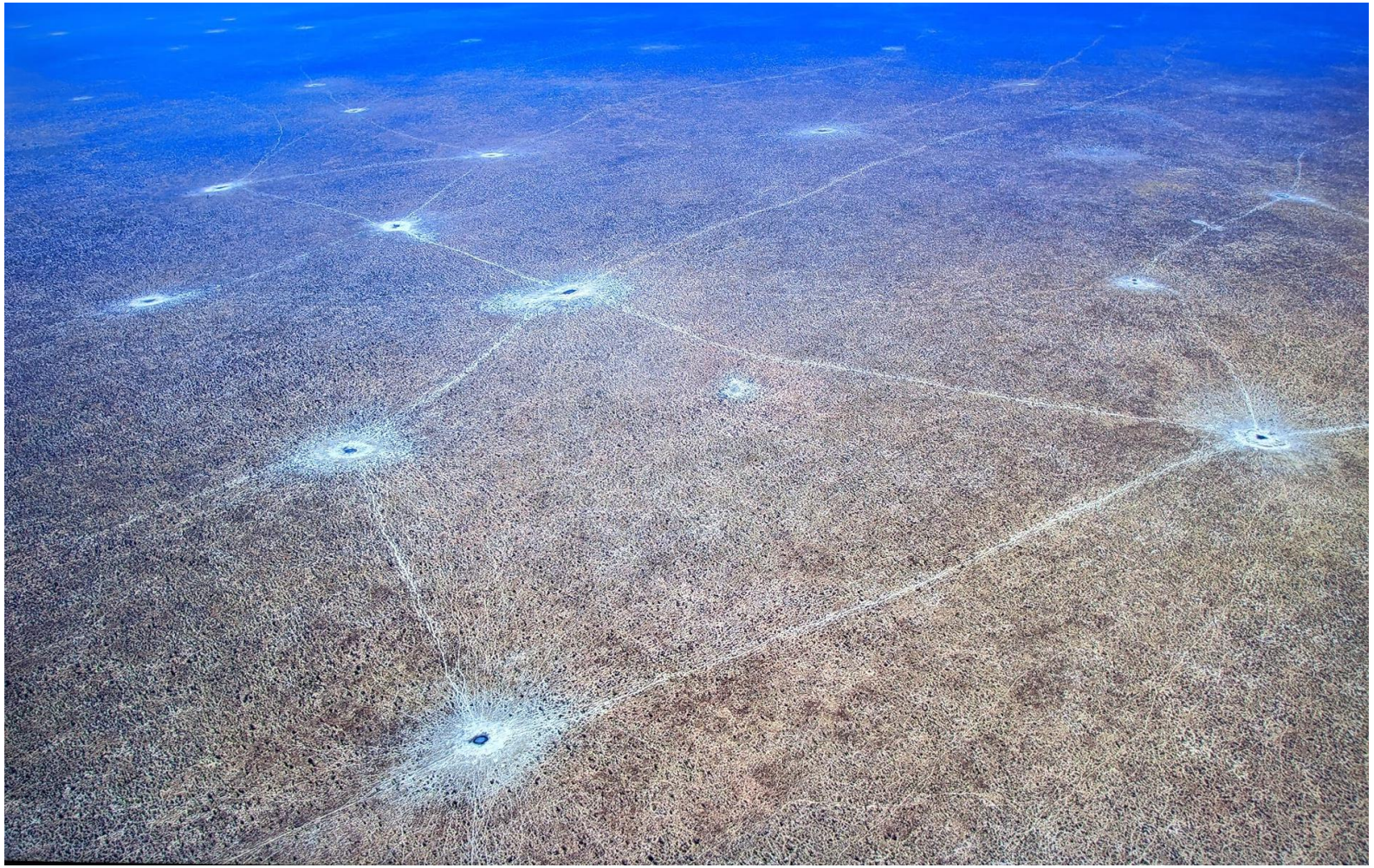
Algorithms: Dijkstra O((|V|+|E|)log|V|) , Bellman-Ford O(|V|x|E|),..



Review: Properties of search algorithm

- Soundness
- Completeness
- Optimality
- Speed (time complexity)
- Memory usage

Slow search can be life or death



Elephants migrate in thousands from Okavango delta, Botswana, drawn by the need to find water

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

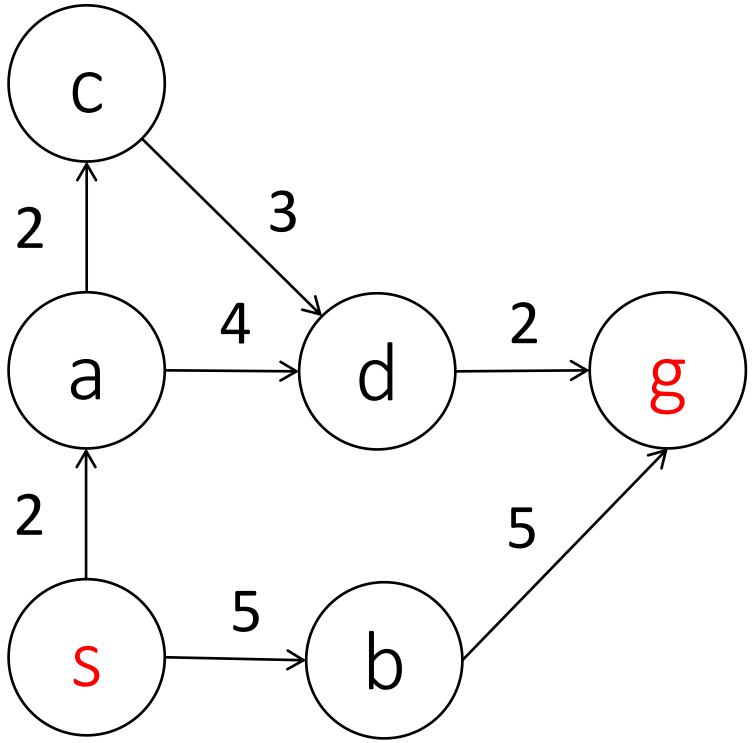
return $FAILURE$; // nothing left to consider

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

Example of Uniform-Cost Search (s to g)

Q:

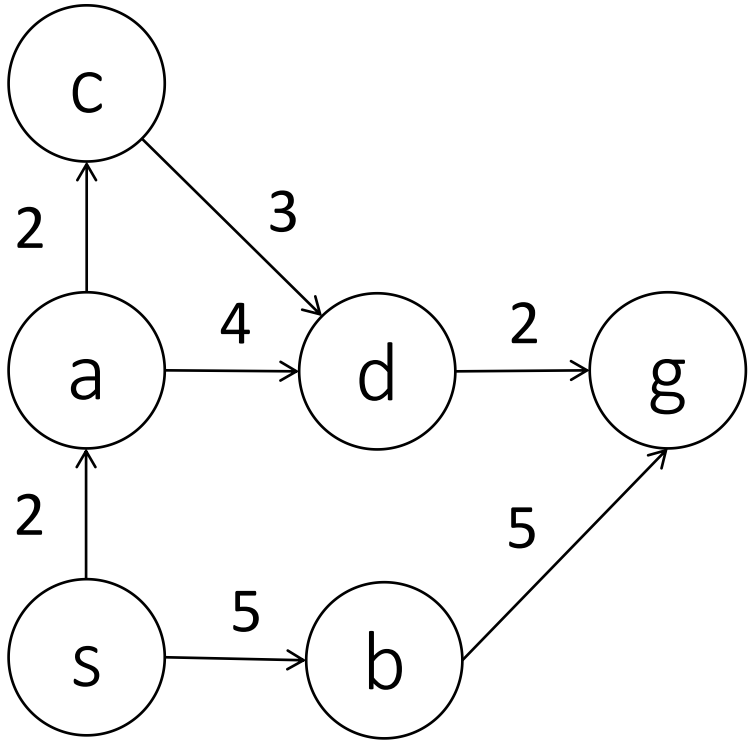
Path	Cost
$\langle s \rangle$	0



Example of Uniform-Cost Search

Q:

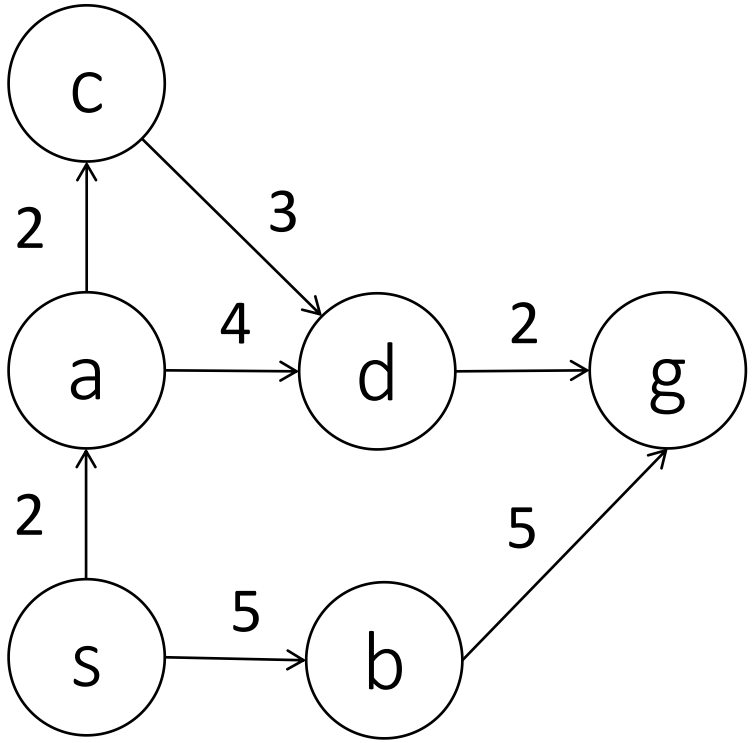
Path	Cost
$\langle a, s \rangle$	2
$\langle b, s \rangle$	5



Example of Uniform-Cost Search

Q:

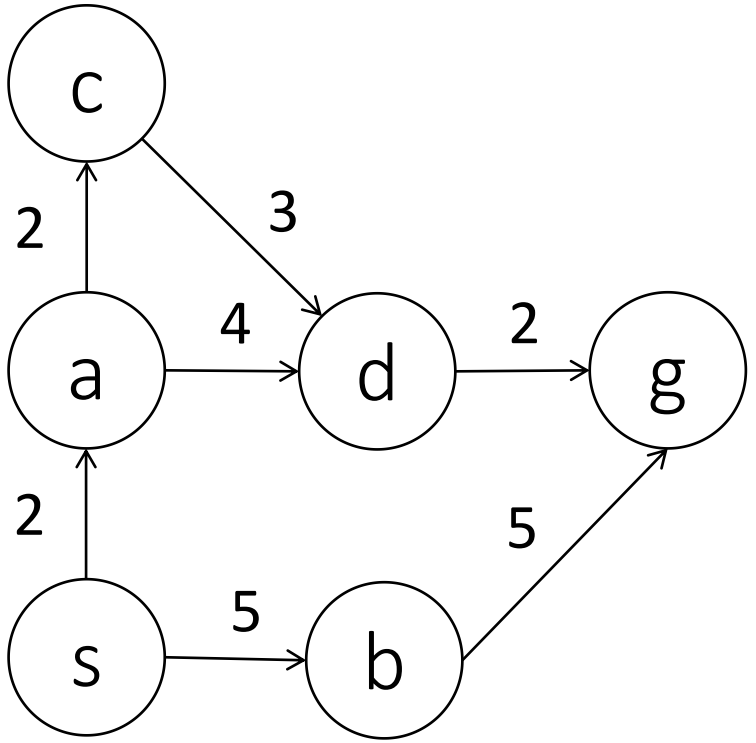
Path	Cost
$\langle c, a, s \rangle$	4
$\langle b, s \rangle$	5
$\langle d, a, s \rangle$	6



Example of Uniform-Cost Search

Q:

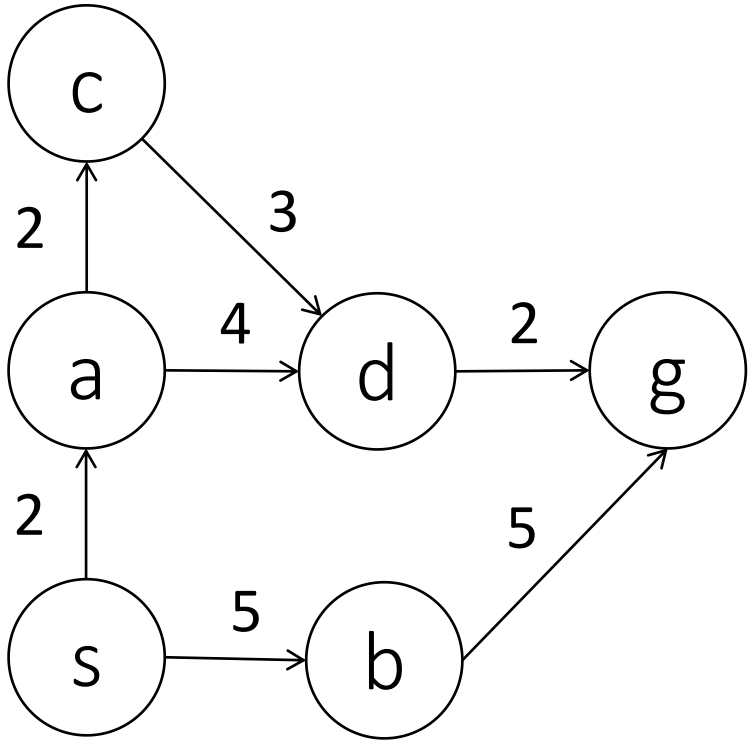
Path	Cost
$\langle b, s \rangle$	5
$\langle d, a, s \rangle$	6
$\langle d, c, a, s \rangle$	7



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

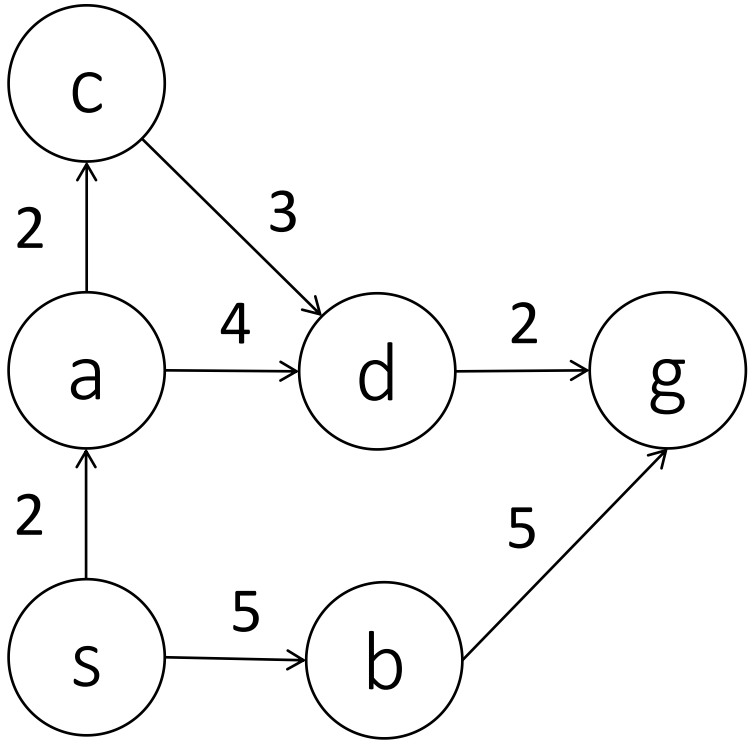


Notice a path to goal <g, b, s> has been found but the algorithm does not terminate, since the path is currently not at the head of Q

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

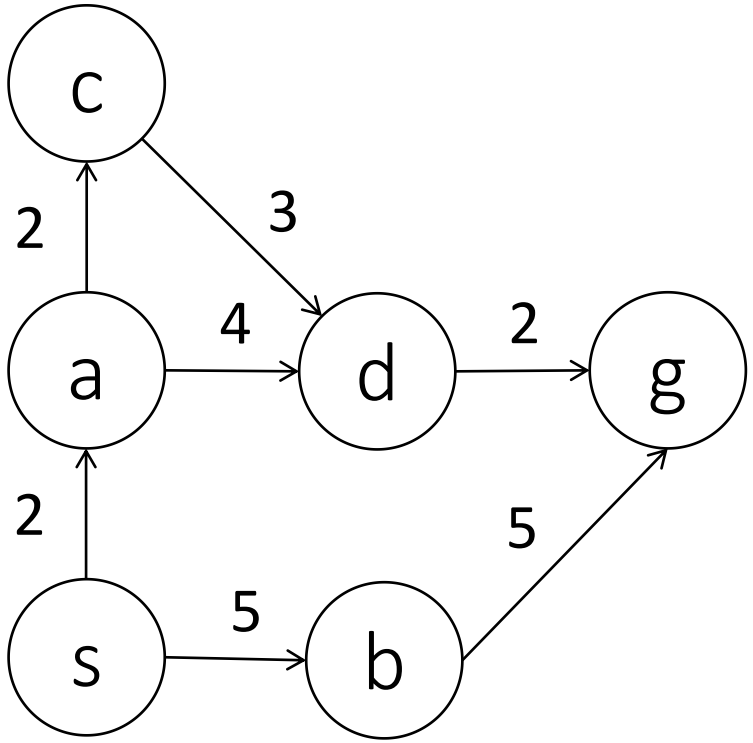


Another path to goal <g, d, a, s> has been found but the algorithm does not terminate, since the path is currently not at the head of Q

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*

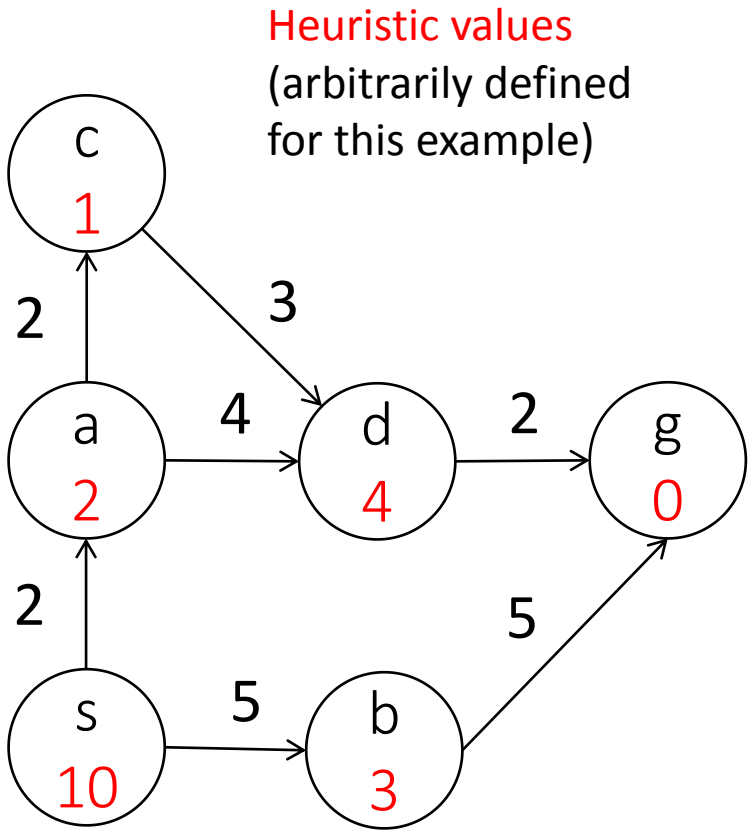
add $\langle v,\ P \rangle\ to\ Q$; *// Add expanded paths*

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

Example of Greedy search

Q:

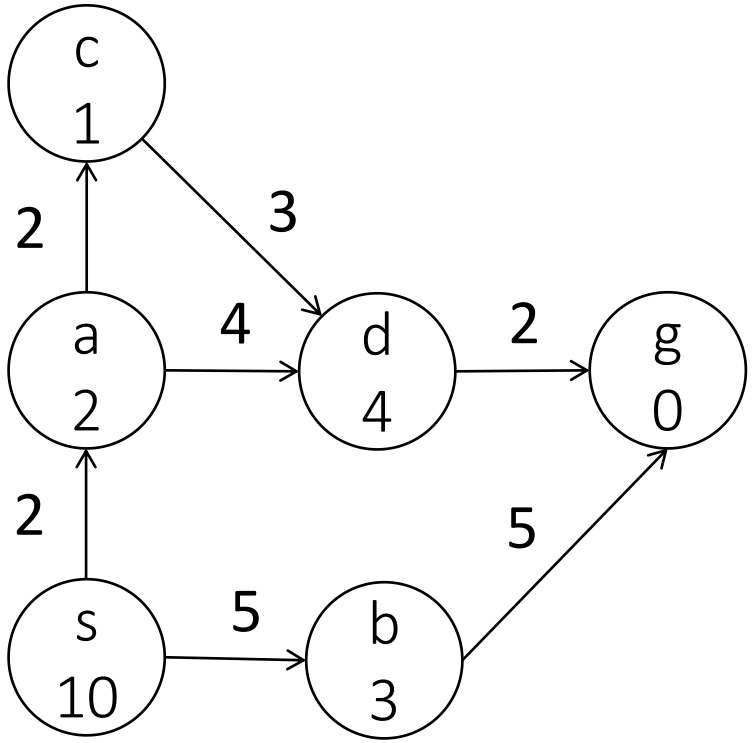
Path	Cost	h
$\langle s \rangle$	0	10



Example of Greedy search

Q:

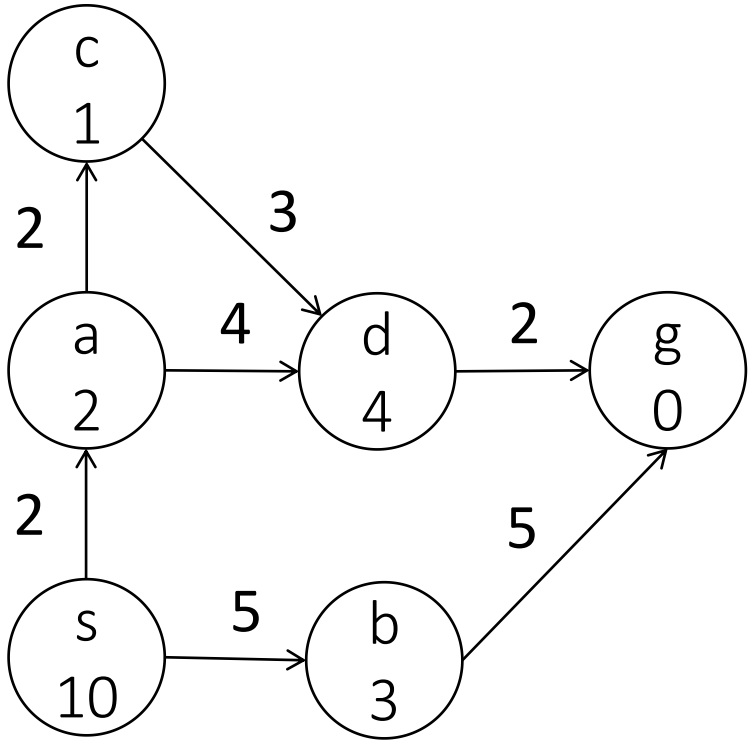
Path	Cost	h
$\langle a, s \rangle$	2	2
$\langle b, s \rangle$	5	3



Example of Greedy search

Q:

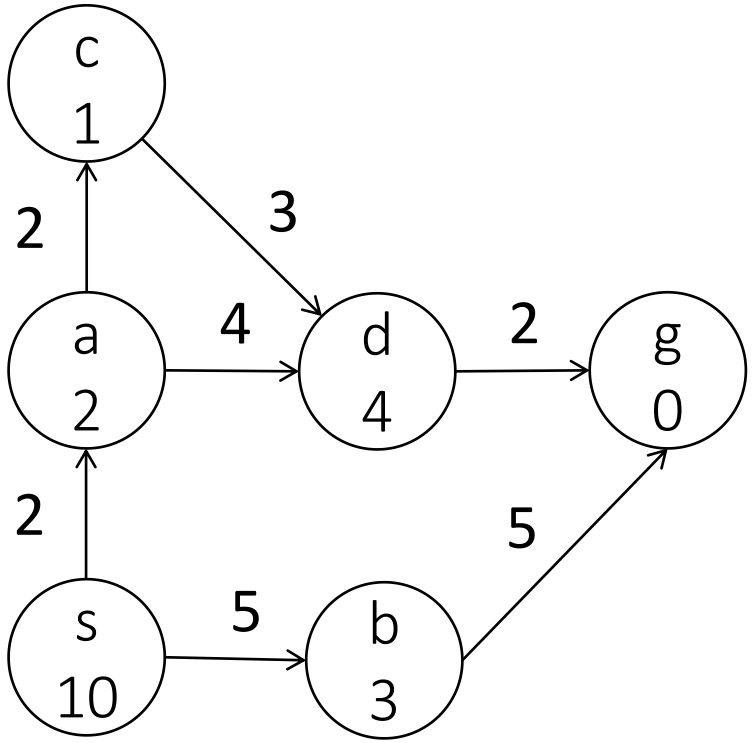
Path	Cost	h
$\langle c, a, s \rangle$	4	1
$\langle d, a, s \rangle$	6	4
$\langle b, s \rangle$	5	3



Example of Greedy search

Q:

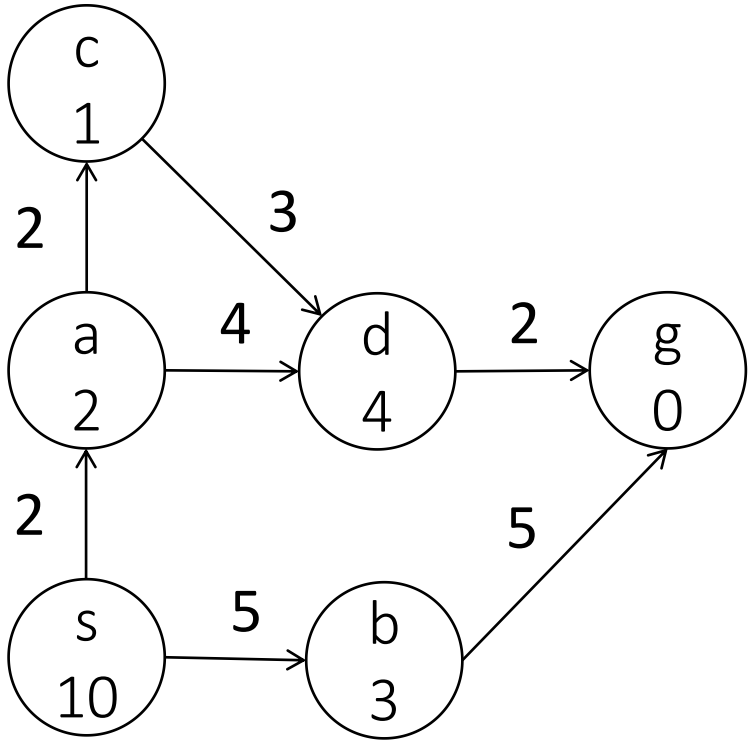
Path	Cost	h
$\langle d, c, a, s \rangle$	7	4
$\langle d, a, s \rangle$	6	4
$\langle b, s \rangle$	5	3



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?

A search: informed search

The problems

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. The non-optimality comes from neglecting "the past."

The idea

Keep track both of the cost of the partial path to get to a vertex, say g(v), and of the heuristic function estimating the cost to reach the goal from a vertex, h(v)

In other words, choose as a "ranking" function the sum of the two costs:

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

g(v) cost-to-come (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)

A and A* search

The challenges

Uninformed search without cost-to-go heuristic h (future prediction) is optimal, but may wander around a lot before finding the goal

Greedy can be biased to explore efficiently towards the goal but may not be optimal as it neglect the cost to arrive at a node (past)

The idea of A Search is to combine past and future

Use both of the cost of the partial path to get to a vertex, g(v), and of the heuristic function estimating the cost to reach the goal from a vertex, h(v) to guide the exploration

sum of the two costs: $f(v) = g(v) + h(v)$

g(v) cost-to-come (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)

A search

A search is similar to UCS, with a bias induced by the heuristic h

If h = 0 then A search = UCS.

The A search is complete, but is *not optimal*

What is wrong? (Recall that if h = 0 then A = UCS, and hence optimal...)

A Search

Choose an admissible heuristic, i.e., such $that\ h(v) \leq h^*(v)$

$h^*(v)$ is the "optimal" heuristic---perfect cost to go

To be admissible $h(v)$ should be at most $h^*(v)$

A search with an admissible heuristic is called A* --- guaranteed to find optimal path

A* search

open set and closed set

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

while $Q \neq \emptyset$:

pick $(and\ remove)\ \ path\ P\ with\ lowest\ estimated\ cost\ f(P) = g(P) + h\big(head(P)\big)\ from\ Q$

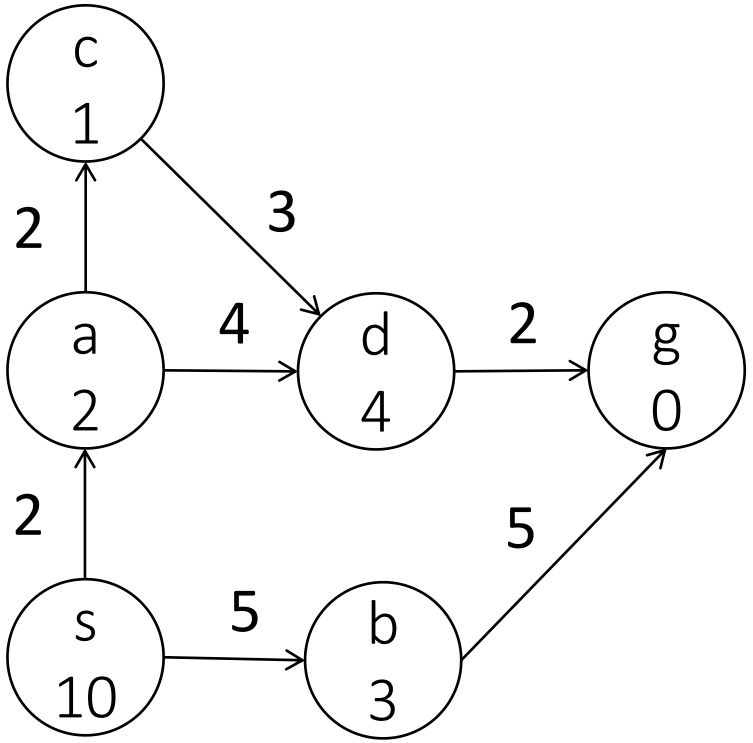
if $head(P) = \ goal$ then return P // Reached the goal

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

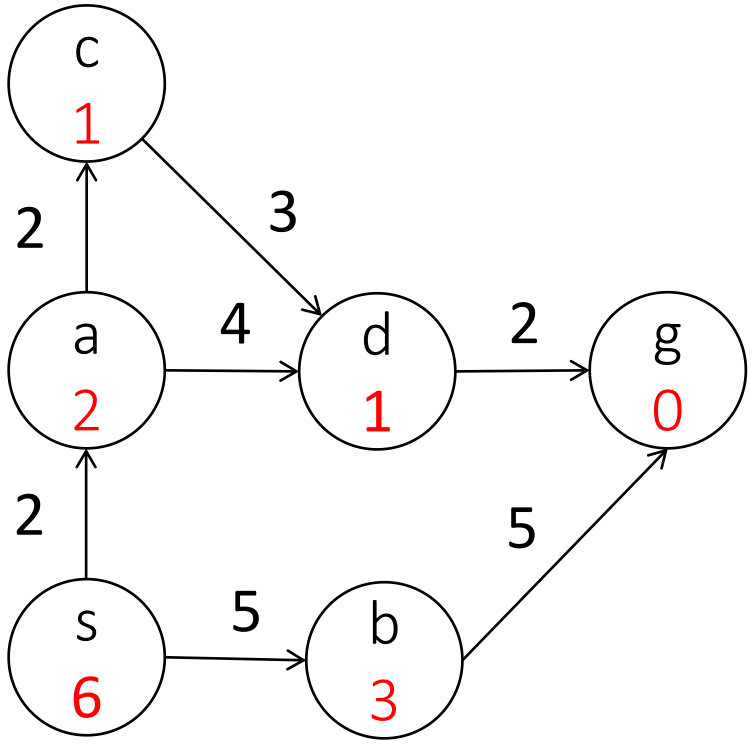
add $\langle v,\ P \rangle\ to\ Q$; // Add expanded paths

return $FAILURE$; // nothing left to consider

Example of A search



These are the heuristic values we used for greedy search. Are they admissible?

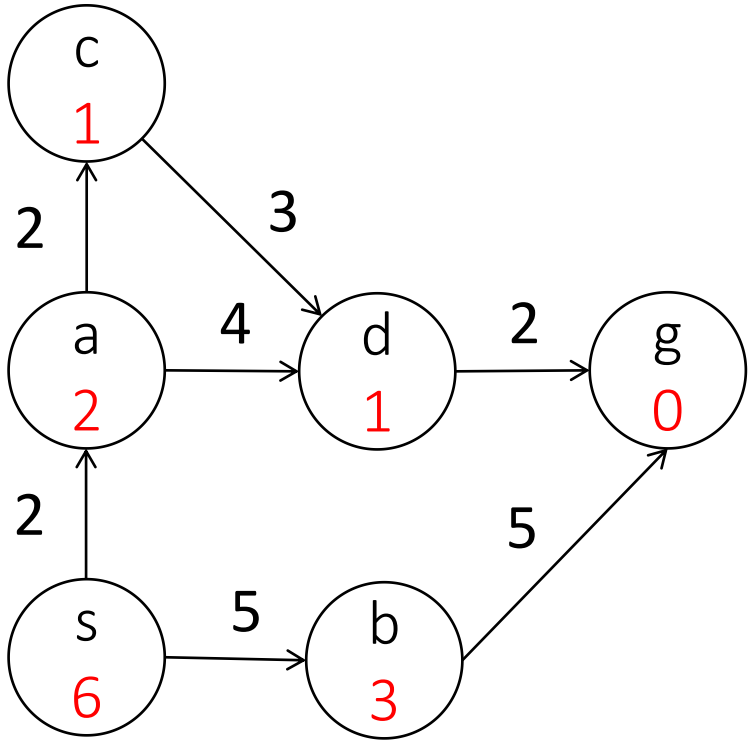


Updated heuristic values

Example of A search

Q:

Path	g	h	f
$\langle s \rangle$	0	6	0+6

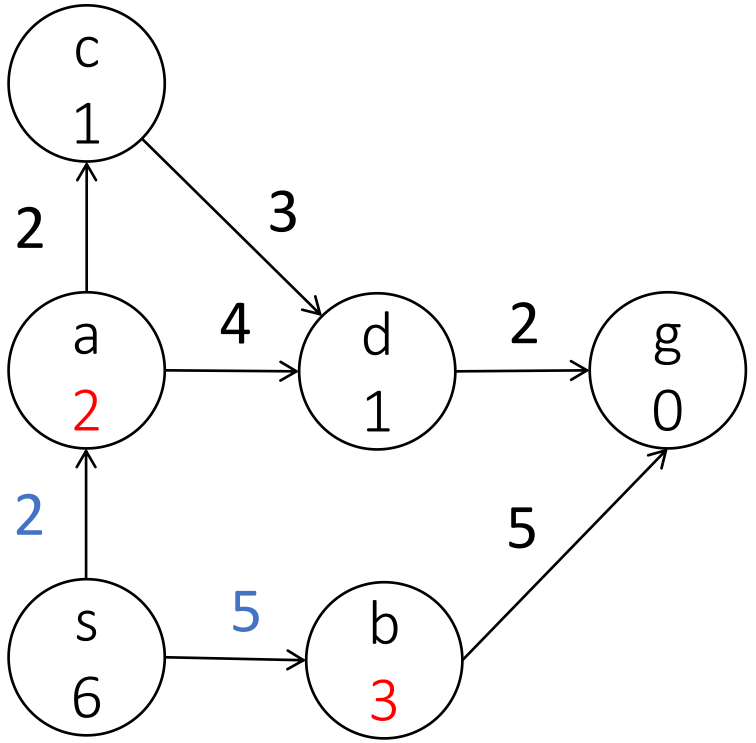


Admissible
heristic values

Example of A search

Q:

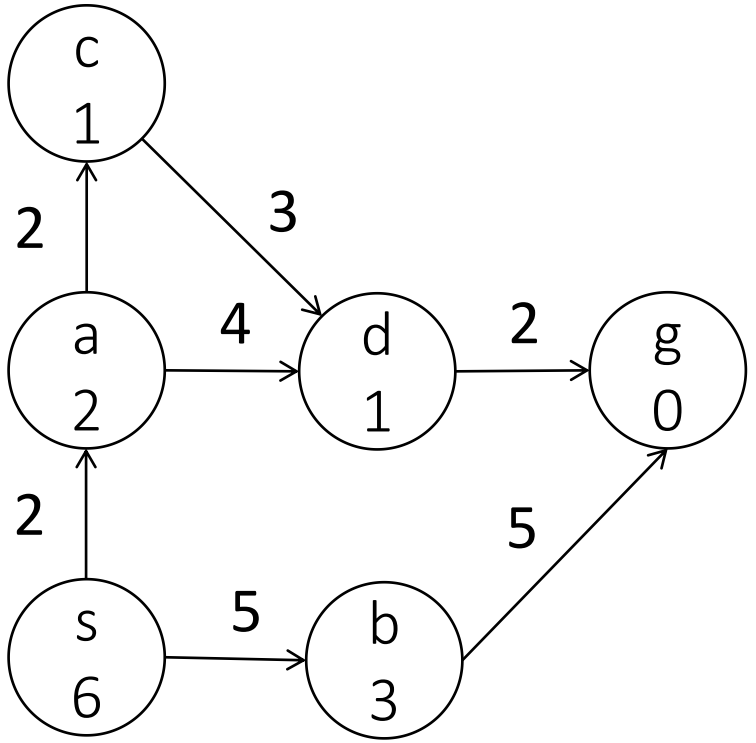
Path	g	h	f
$\langle a, s \rangle$	2	2	2+2=4
$\langle b, s \rangle$	5	3	5+3=8



Example of A search

Q:

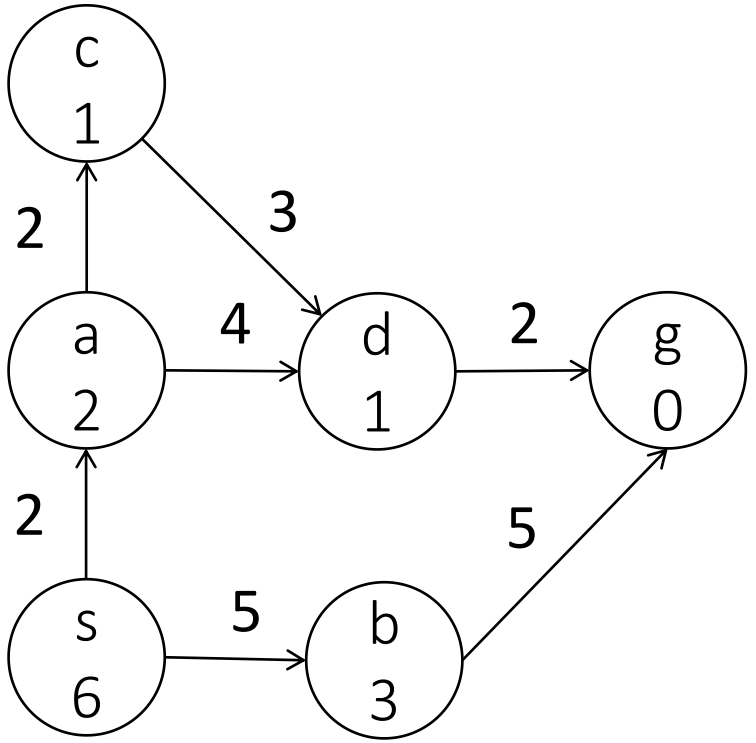
Path	g	h	f
$\langle c, a, s \rangle$	4	1	5
$\langle d, a, s \rangle$	6	1	7
$\langle b, s \rangle$	5	3	8



Example of A search

Q:

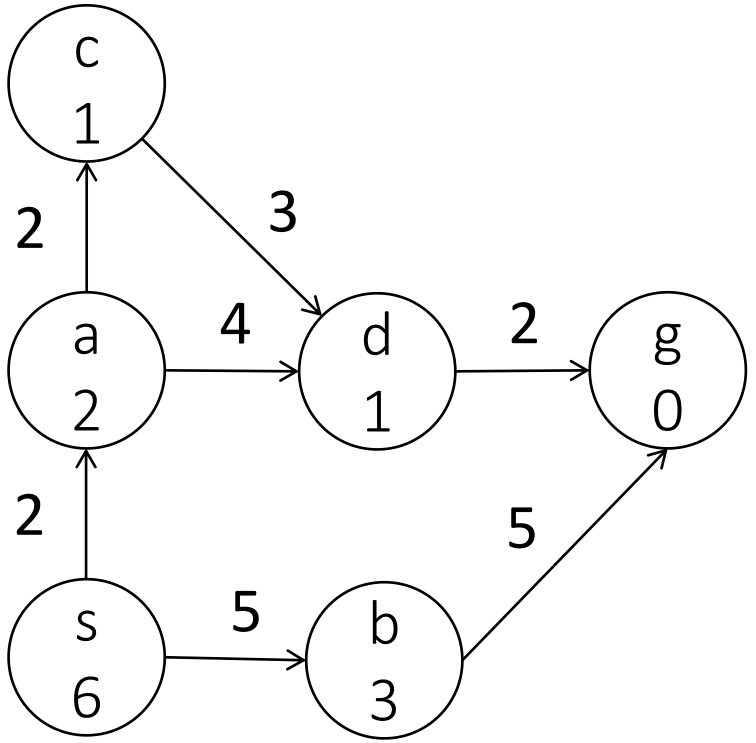
Path	g	h	f
$\langle d, c, a, s \rangle$	7	1	8
$\langle d, a, s \rangle$	6	1	7
$\langle b, s \rangle$	5	3	8



Example of A search

Q:

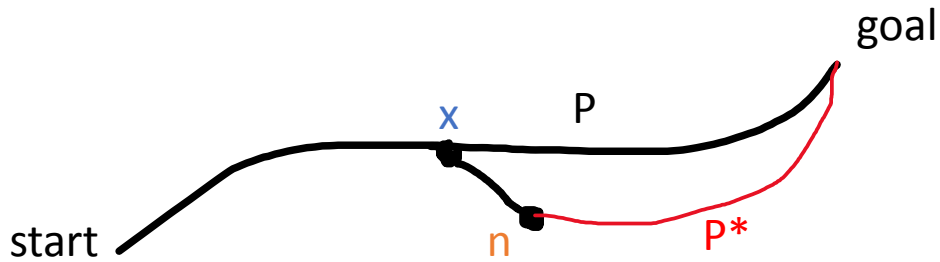
Path	g	h	f
$\langle g, d, a, s \rangle$	8	0	8
$\langle d, c, a, s \rangle$	7	1	8
$\langle b, s \rangle$	5	3	8



Uniform cost search vs A*

https://en.wikipedia.org/wiki/A*_search_algorithm

Proof of optimality of A*



Theorem. A* Search can find the optimal path.

Let w* be the **cost of the optimal path** P*.

Suppose for the sake of contradiction, that A* returns suboptimal P with w(P) > w*

Find the first node *n* on the optimal path P* that is **not** on P

When *n*'s parent x was expanded the path $\langle n, x, \ldots \rangle$, f(n) was added in Q but then *n* was not expanded later in the algorithm

Thus, we must have f(n) > w(P), otherwise n would have been expanded (1)

But, f(n) = g(n) + h(n)

= g*(n) + h(n) [since n is on the optimal path]

<= g*(n) + h*(n) [since h is admissible]

= w* [since w* is the cost of the optimal path which goes through n]

=> f(n) < w(P) (2) [since w* is the optimal cost < W(P)]

(1) and (2) contradict.

Admissible heuristics

How to find admissible heuristics? i.e., a heuristic that never overestimates the cost-to-go.

Some examples

- $h(v) = 0$: this always works! However, it is not very useful, A* = UCS
- $h(v) = distance(v, g)$ when vertices are physical locations
- $h(v) = ||v - g||_p$, when vertices are points in a normed vector space

More generally

- Choose h as the optimal cost-to-go function for a relaxed problem, that is easy to compute
- Relaxed problem: ignore some of the constraints in the original problem
  - Ignore or under-approximate some of the obstacles
  - Allow less restrictive dynamics

Admissible heuristics for the 8-puzzle

Initial state:

7	5	4
	3	2
8	1	6

Goal state:

1	2	3
4	5	6
7	8	

First, think about how to solve this game using search:

- What is the vertices on the graph?
- How to define the edges?
- What are the start and goal vertices?
- How the distance is defined?
- What heuristic to use?

7	5	4
	3	2
8	1	6

Admissible heuristics for the 8-puzzle

Initial state:

7	5	4
	3	2
8	1	6

Goal state:

1	2	3
4	5	6
7	8	

Which of the following are admissible heuristics?

- h = 0 — YES, always good (uniform cost search), but can be slow
- h = 1 — not valid in goal state
- h = number of tiles in the wrong position — YES, “teleport” each tile to the goal in one move
- h = sum of (Manhattan) distance between tiles and their goal position — YES, move each tile to the goal ignoring other tiles.

Which heuristc is better? A partial order of heuristic functions

Some heuristics are better than others

- h = 0 is an admissible heuristic, but is not very useful
- h = h* is also an admissible heuristic, and it is the "best" possible one (it give us the optimal path directly, no searches/backtracking)

Partial order

- We say that h_1 dominates h_2 if $h_1(v) \geq h_2(v)$ for all vertices v
- h^* dominates all admissible heuristics, and 0 is dominated by all admissible heuristics

Choosing the right heuristic

- In general, we want a heuristic that is as close to h* as possible
- However, such a heuristic may be too complicated to compute
- There is a tradeoff between complexity of computing h and the complexity of the search

Consistent heuristics

- An additional useful property for A* heuristics is called consistency
  - A heuristic $h : X \rightarrow \mathbb{R}_{\geq 0}$ is said consistent if
    $h(u) \leq w(e = (u, v)) + h(v), \forall (u, v) \in E$
  - In other words, a consistent heuristics satisfies a triangle inequality
- If h is a consistent heuristics, then $f = g + h$ is non-decreasing along paths:
  $f(v) = g(v) + h(v) = g(u) + w(e = (u, v)) + h(v) \geq g(u) + h(u) = f(u)$
- Hence, the values of f on the sequence of nodes expanded by A* is non-decreasing: the **when we expand path P with the lowest f(P), P is the optimal path to head(P)**
- With consistent heuristics, A* is not only optimal (finds the best path), but also has optimal efficiency (explores fewest node)

A* search

open set and closed set

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

while $Q \neq \emptyset$:

pick $(and\ remove)\ \ path\ P\ with\ lowest\ estimated\ cost\ f(P) = g(P) + h\big(head(P)\big)\ from\ Q$

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

Review

- Deterministic search
  - Uniform Cost Search (uninformed search)
  - Informed search
    - Greedy/Best-first search
    - **A, A* search**
      - Admissible heuristics
      - design of heuristics
- How to transform a planning problem in to a search problem?
- Know how to apply each search algorithm
- Soundness, completeness, optimality
- Design of heuristics

A* to Continuous State Spaces

Applying A* to vehicles

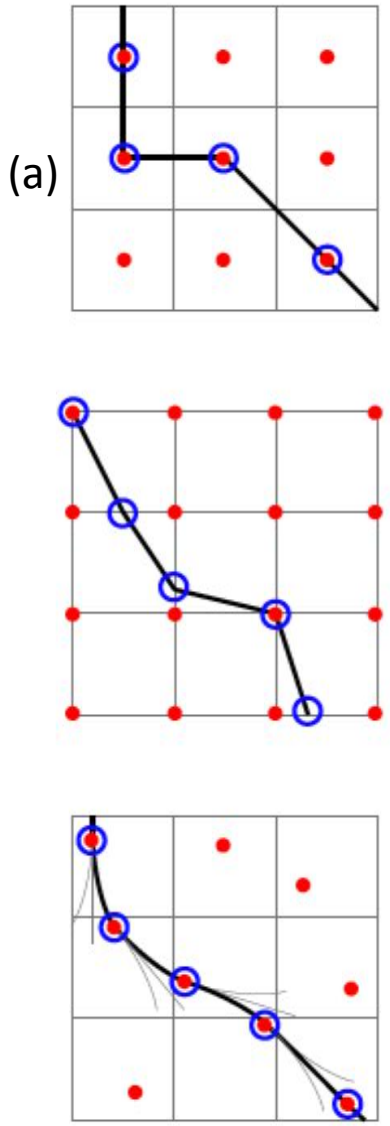
Vehicle models have continuous state spaces and kinematic constraints

We could represent vehicle state in a *uniform* discrete grid, for example, a 4D grid: $x, y, \theta\ (heading), dir$ (fwd,rev)

A path (a) over this discrete grid can be a starting plan

But, the discrete path (a) may not be executable by the vehicle dynamics

Hybrid A:* An extension of the traditional A* algorithm that operates on a continuous state space an incorporates kinematic (e.g., non-holonomic) **constraints**



Montemerlo, Michael, et al. "Junior: The stanford entry in the urban challenge." *Journal of field Robotics* 25.9 (2008): 569-597.

Hybrid A* Search

O = {(x_0, y_0, θ_0, g=0, h=0)} // open list
C = {} // closed list
While O ≠ {}:
  n = (x, y, θ) node with min cost f(n) = g(n) + h(n) in O
  If n = goal, **return** the path.
  C = C ∪ {n}
  For each motion primitive {m}:
    s = (x', y', θ') = motion(x, y, θ, m)
      g(s) = g(n) + cost(n, s) // Calculate cost
      h(s) = heuristic(s, goal)
      f(s) = g(s) + h(s)
      If s ∉ C or g(s) is lower than previously recorded cost:
        Add/update s in O with g(s), h(s)
Return failure if O is empty

Input: Source node x_0, y_0, θ_0, goal node
Motion primitives

Junior: The Stanford Entry in the Urban Challenge

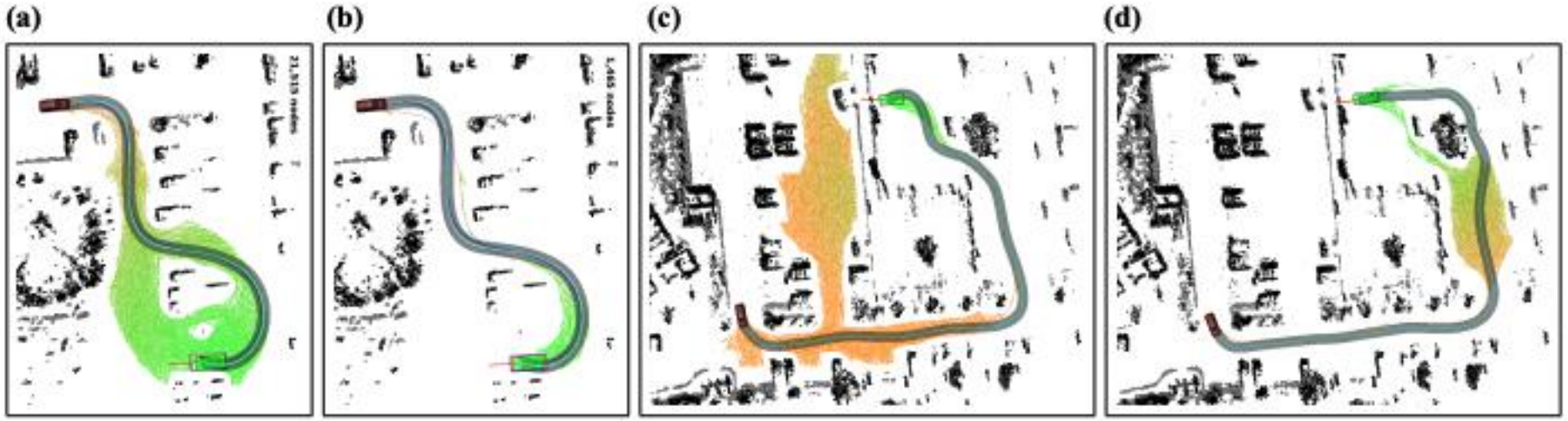


Figure 16: Hybrid-state A* heuristics. (a) Euclidean distance in 2-D expands 21,515 nodes. (b) The non-holonomic-without-obstacles heuristic is a significant improvement, as it expands 1,465 nodes, but as shown in (c), it can lead to wasteful exploration of dead-ends in more complex settings (68,730 nodes). (d) This is rectified by using the latter in conjunction with the holonomic-with-obstacles heuristic (10,588 nodes).

http://robots.stanford.edu/papers/junior08.pdf

Properties of Hybrid A*

- **Soundness**
  - sound when the heuristic function is admissible and consistent.
  - However, soundness may be compromised if motion primitives do not accurately represent feasible trajectories.
- **Completeness:**
  - Hybrid A* is not strictly complete due to the continuous state space.
  - Completeness can be approximated with dense discretization of the state space.
- **Optimality:**
  - Hybrid A* is not guaranteed to find the optimal path since it approximates the continuous space with motion primitives.
  - The quality of the solution depends on the choice of motion primitives and heuristics.
- **Feasibility:**
  - The generated path is kinematically feasible, respecting vehicle constraints (e.g., turning radius)

Summary

A* algorithm combines cost-to-come g(v) and a heuristic function h(v) for cost-to-go to find shortest path

- informed search

Heuristic function must be *admissible* $h(v) \leq h^*(v)$

- Never over-estimate the actual cost to go
- Dominant and consistent heuristics are preferrable

Hybrid A* extends A* to continuous state spaces and incorporates kinematic constraints