Search and Planning

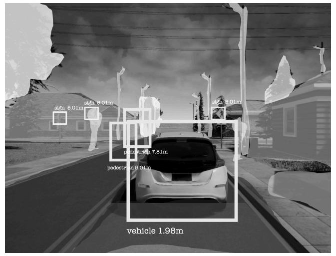
Sayan Mitra

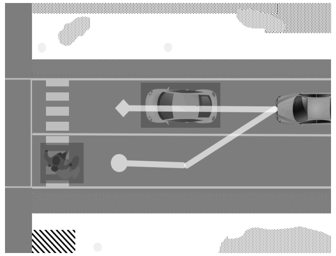
Based on some lectures by Emilio Frazzoli

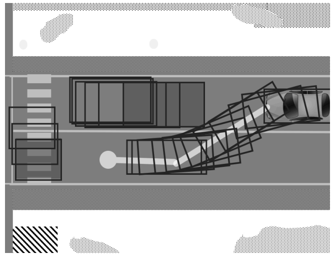
March 24

GEM platform

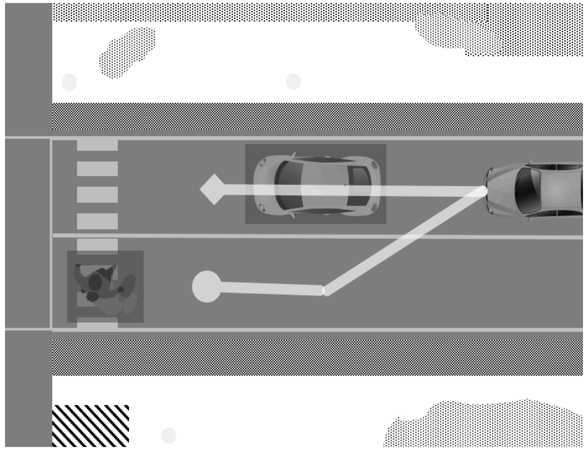
Autonomy pipeline







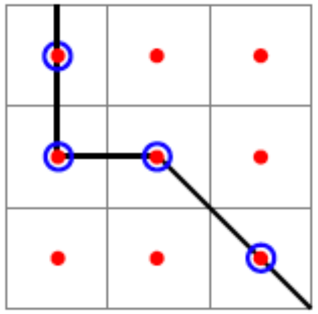
Sensing	**Perception**	**Decisions and planning**	**Control**
Physics-based models of camera, LIDAR, RADAR, GPS, etc.	Programs for object detection, lane tracking, scene understanding, etc.	Programs and multi-agent models of pedestrians, cars, etc.	Dynamical models of engine, powertrain, steering, tires,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

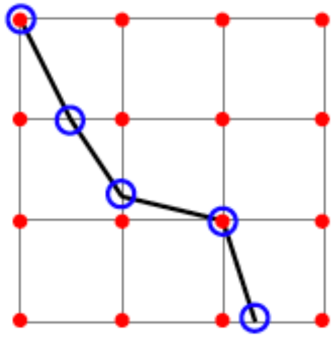


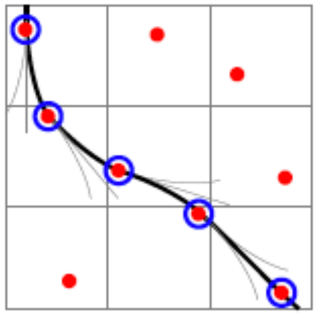
Decisions and planning

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

A search-based strategy for planning

(a)






- Represent vehicle state in a *uniform* discrete grid
  - 4D grid: $x, y, \theta\ (heading), dir$ (fwd,rev)
- A path (a) over this discrete grid is a start for a plan
- But, the discrete path (a) may not be executable by the vehicle dynamics
- *Hybrid A** solves this problem by shifting the points that represent the discrete cells
  - More on this in the next lecture

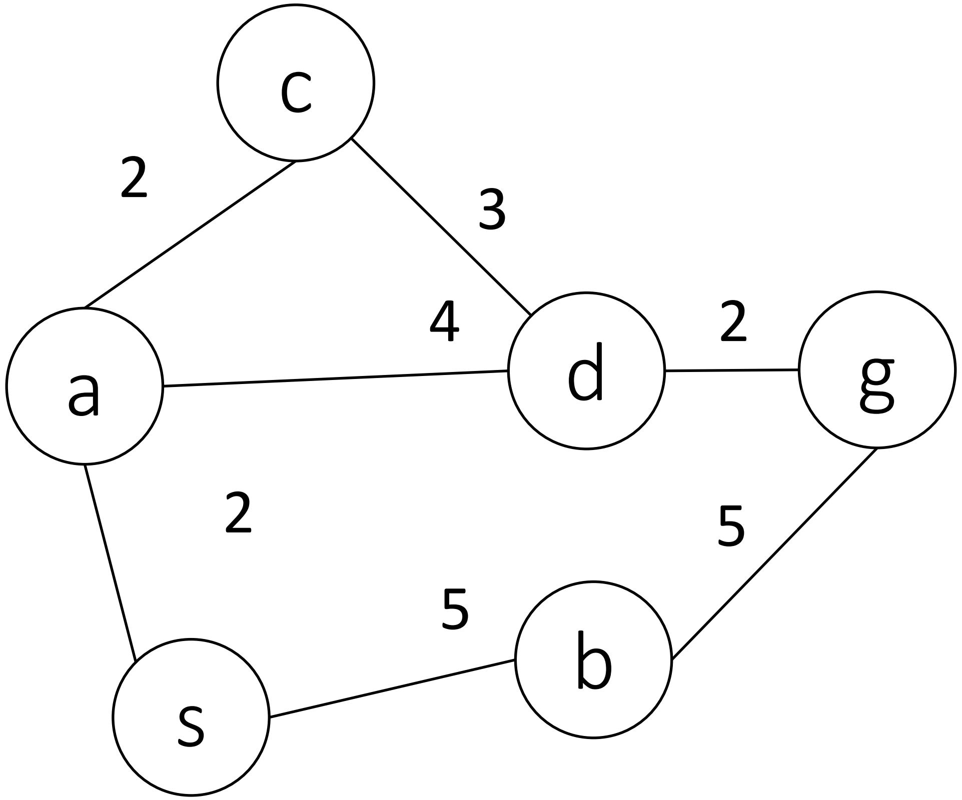
Shortest path problems

- Input: $\langle V, E, w, start, goal \rangle$
  - 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 $w(e)$ (cost, length, time, fuel, prob. of detection)
  - $start, goal \in V$: respectively, start and end vertices.
- Output: $\langle P \rangle$
  - P is a path (seq of vertices)
  - The weight of a path is the sum of the weights of its edges
    - Ultimately, we'd want a path starting in start and ending in goal, such that its weight $w(P)$ is minimal among all such paths
  - The graph may be unknown, partially known, or known

Example: Find the minimal path from s to g:

a simple path P:

w(P):



Search Performance Metrics

- Soundness: when a solution is returned, is it guaranteed to be correct
- Completeness: – the algorithm guaranteed to find a solution when one exists
- Optimality: How close is the found solution to the best solution
- Space complexity: memory needed
- Time complexity: running time; can it be used for online planning?

Uniform cost search (Uninformed search)

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

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

while $Q \neq \emptyset$:

    from Q pick $(and\ remove)\ the\ path\ P\ with\ lowest\ cost,\ \ say\ \ g = w(P)$

    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*

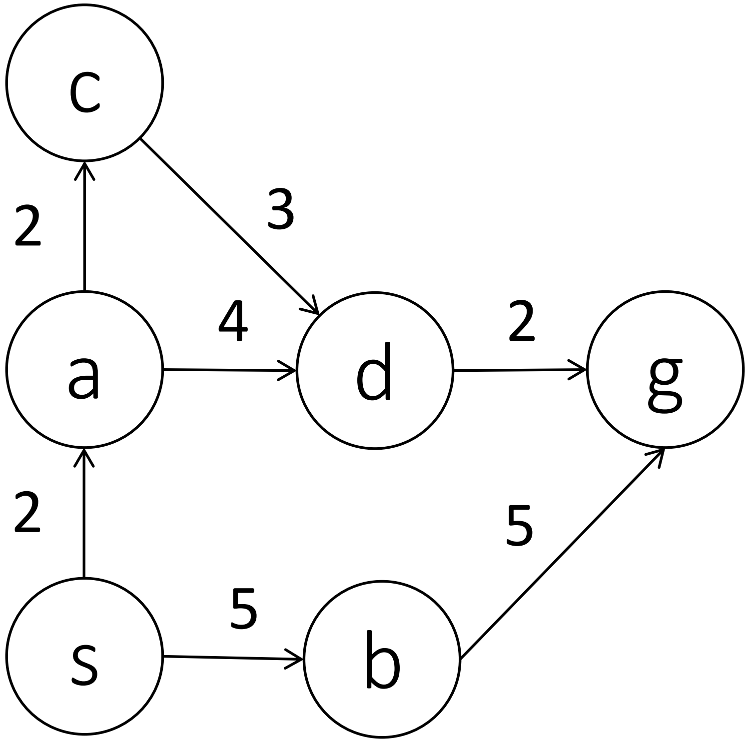
return $FAILURE$; *// nothing left to consider*

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

Example of Uniform-Cost Search

Q:

Path	Cost



Remarks on Uniform Cost Search

- UCS is an extension of BFS to the weighted-graph case (UCS = BFS if all edges have the same cost)
- Note. Algorithm stops when lowest cost path has ’goal’ as the head
- UCS is *complete* and *optimal* (assuming edge weights bounded away from zero)
  - Exercise: prove these
- 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
  - b is the max number of branches out of each node

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," $h : V \to \mathbb{R}_{\geq 0}$. E.g., 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: this is the basic idea of the greedy (best-first) search

Greedy/Best-first search

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

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

while $Q \neq \emptyset$:

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

  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*

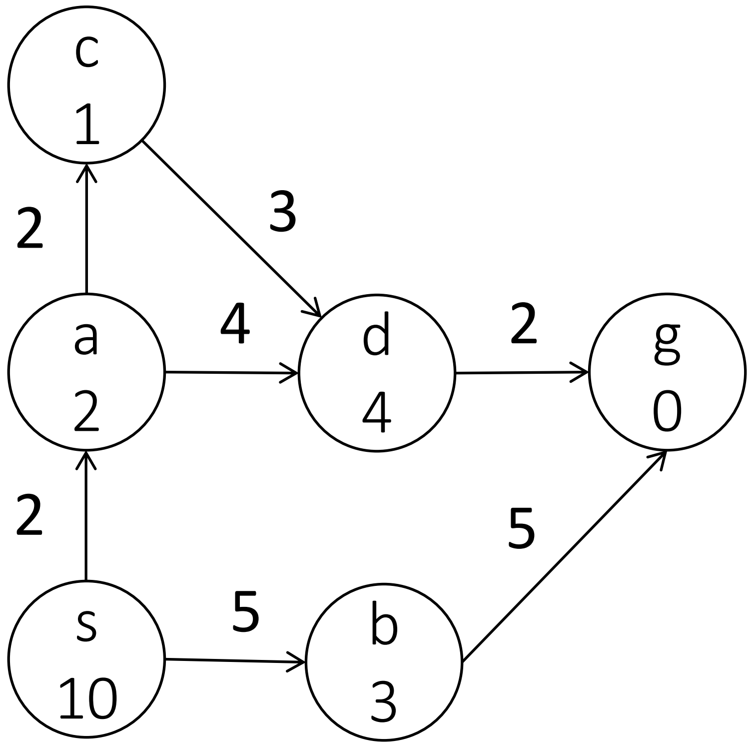
return $FAILURE$; // *nothing left to consider*

Note no visited list

Example of Greedy search

Q:

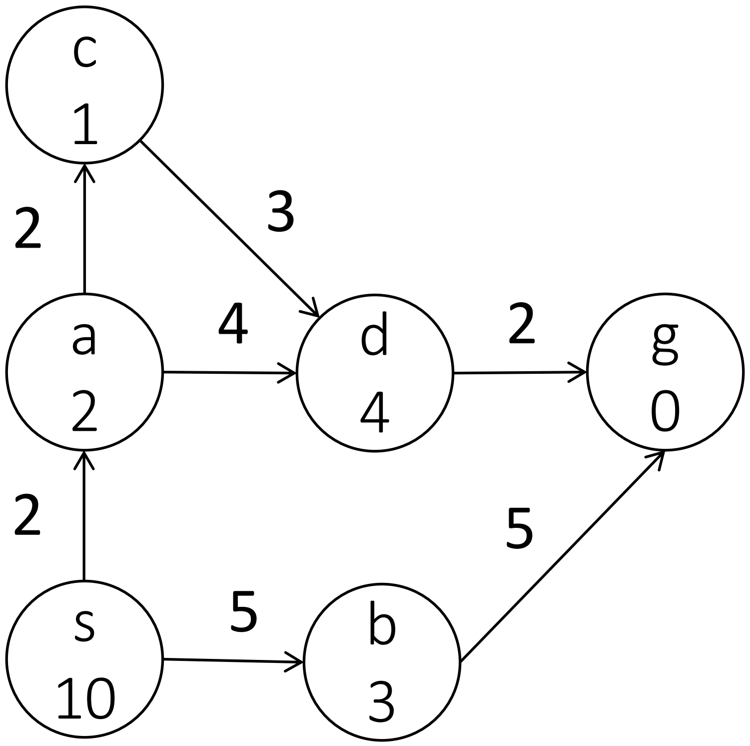
Path	Cost	h
$\langle s \rangle$	0	10



Example of Greedy search

Q:

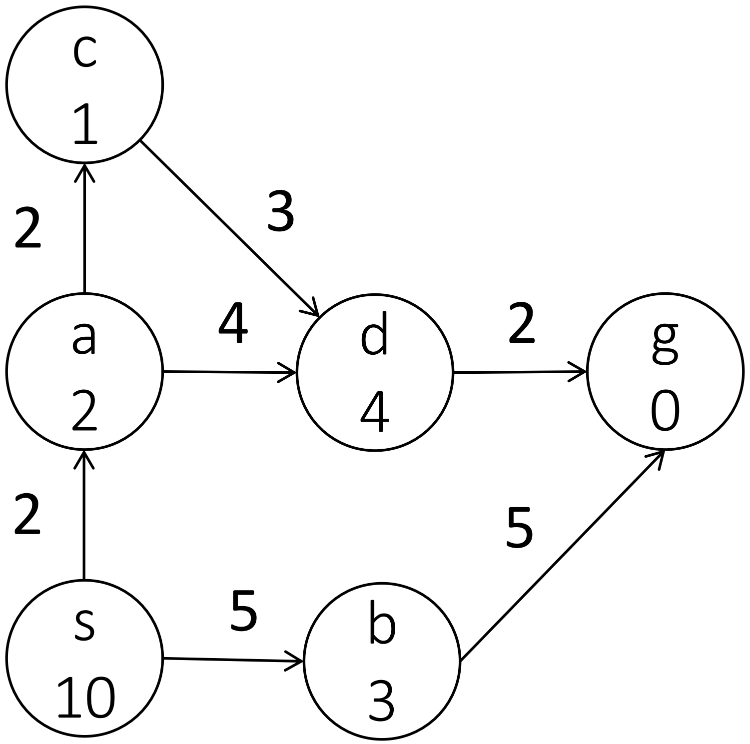
Path	Cost	h
$\langle a, s \rangle$	2	2
$\langle b, s \rangle$	5	3



Example of Greedy search

Q:

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



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 and not optimal, but is often fast and efficient, depending on the heuristic function h
  - Exercise: Construct an example where greedy can get stuck
- Worst-case time and space complexity $O(b^m)$

A search

The problems

UCS is optimal and complete	Best First Search can be fast
UCS may be slow; wander around before finding the goal.	Not optimal and not complete Neglects 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 search

open set and closed set

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

$Q \leftarrow \langle start \rangle$ *// 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*

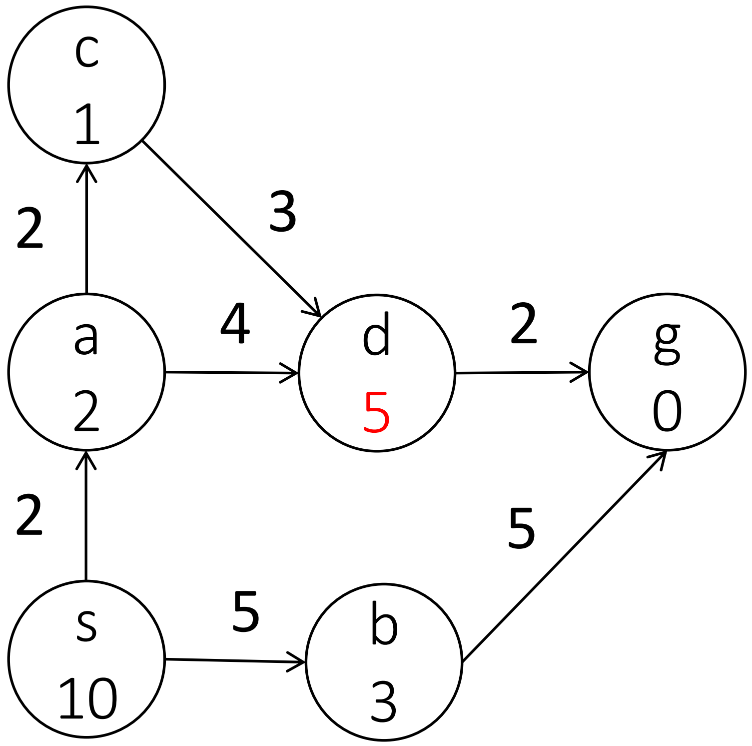
    add $\langle v, P \rangle\ to\ Q$; *// Add expanded paths*

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

Example of A search

Q:

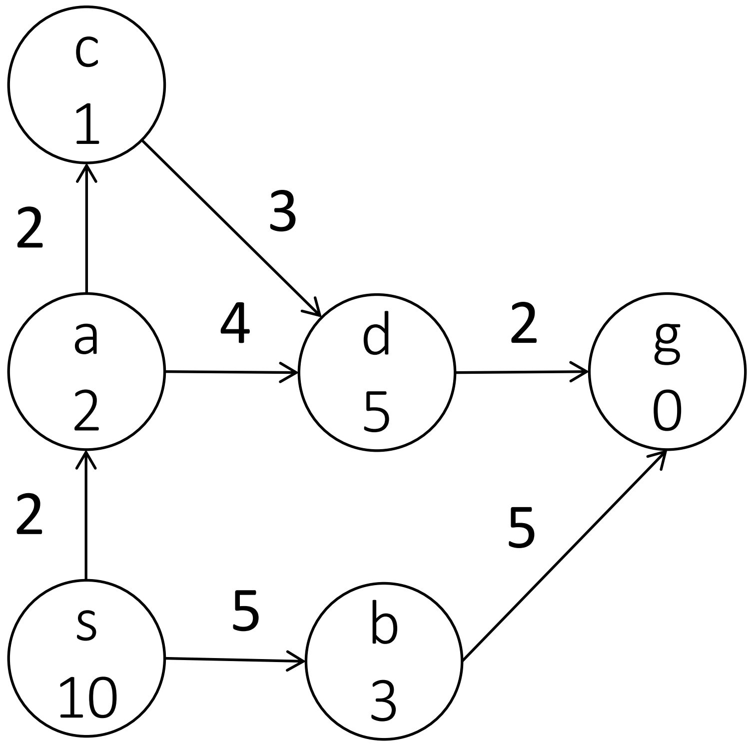
Path	g	h	f
$\langle s \rangle$	0	10	10



Example of A search

Q:

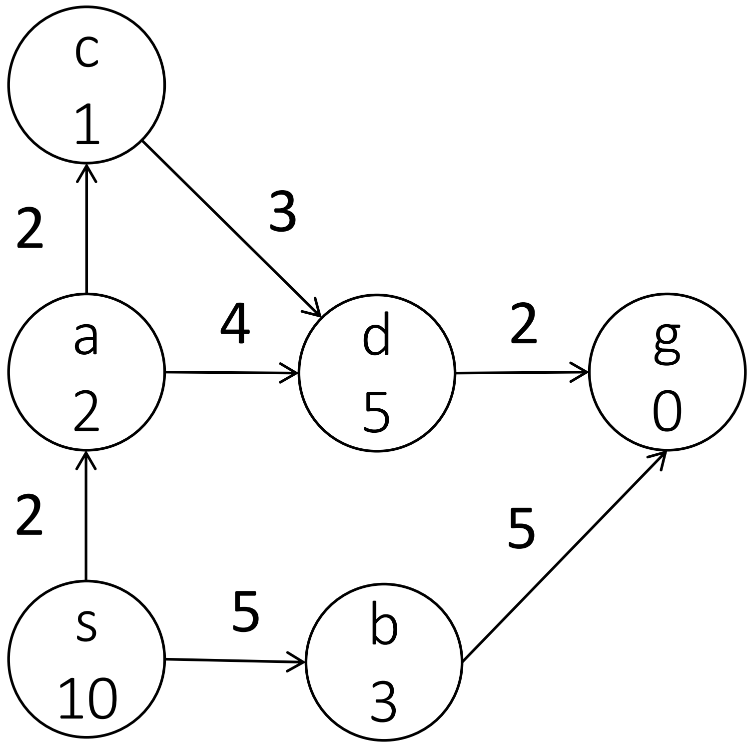
Path	g	h	f
$\langle a, s \rangle$	2	2	4
$\langle b, s \rangle$	5	3	8



Example of A search

Q:

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



Remarks on A search

- A search is similar to UCS, with a bias induced by the heuristic h
- If h = 0, A = UCS.
- The A search is complete, but is *not optimal*
  - What is wrong? (Recall that if h = 0 then A = UCS, and hence optimal...)

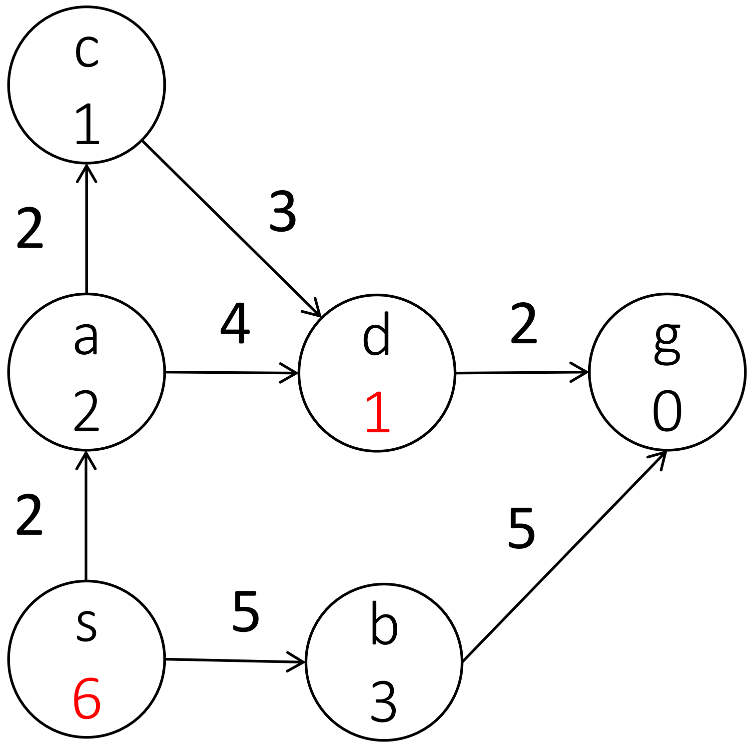
A* Search

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

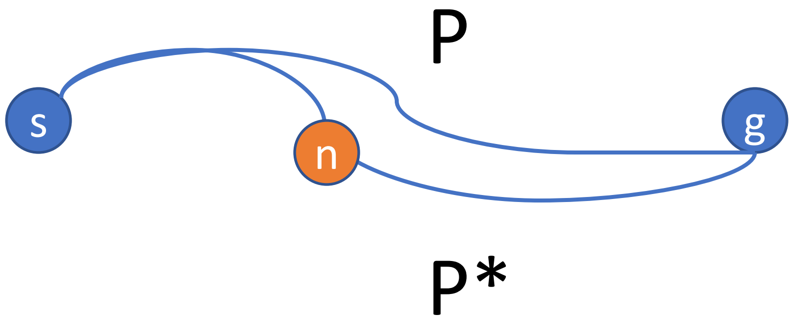
Example of A* search

Q:

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



Proof of optimality of A*



- Let w^* be the cost of the optimal path
- Suppose for the sake of contradiction, that A* returns P with $w(P) > w^*$
- Find the first unexpanded node on the optimal path P*; call it n
- $f(n) > w(P)$, otherwise n would have been expanded
- $f(n) = g(n) + h(n)$

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

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

    $= f^*(n) = w^*$ [by def. of f, and since w^* is the cost of the optimal path]
- Hence $w^* >= f(n) = w(P)$, which is a contradiction

Admissible heuristics

- How to find an admissible heuristic? i.e., a heuristic that never overestimates the cost-to-go.
- Examples of admissible heuristics
  - $h(v) = 0$: this always works! However, it is not very useful, A∗ = UCS
  - $h(v) = distance(v, g)$ when the vertices of the graphs are physical locations
  - $h(v) = ||v - g||_p$, when the vertices of the graph are points in a normed vector space
- A general method
  - 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

Admissible heuristics for the 8-puzzle

Initial state:

1		5
2	6	3
7	4	8

Goal state:

1	2	3
4	5	6
7	8	

Which of the following are admissible heuristics?

- h = 0 YES, always good
- h = 1 NO, not valid in goal state
- h = number of tiles in the wrong positon 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.

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 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

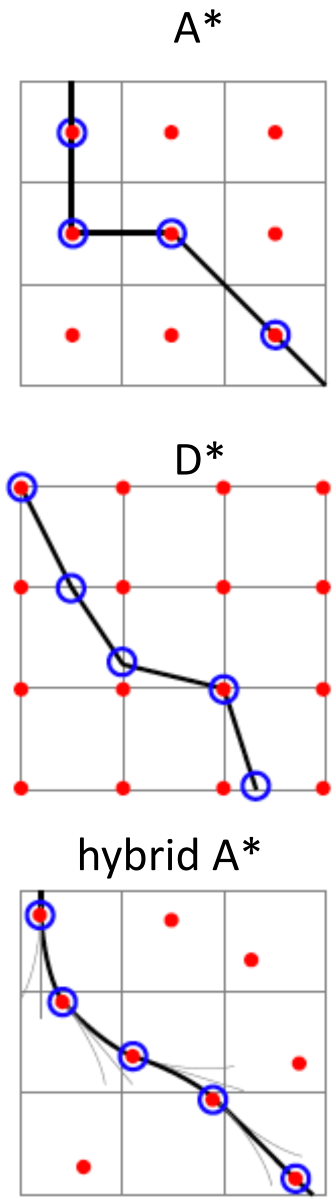
Consistent heuristics

- An additional useful property for A$*$ heuristics is called consistency
  - A heuristic $h : X \rightarrow \mathbb{R}_{\geq 0}$ is said consistent if $\forall (u, v) \in E$
    $h(u) \leq w\,(e = (u, v)) + h(v)$
  - 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: $f\,(v) = g(v) + h(v) = g(u) + w(u, v) + h(v) \geq f\,(u)$
- Hence, the values of f on the sequence of nodes expanded by A$*$ is non-decreasing: the first path found to a node is also the optimal path $\Rightarrow$ no need to compare costs!

A* to hybrid A*

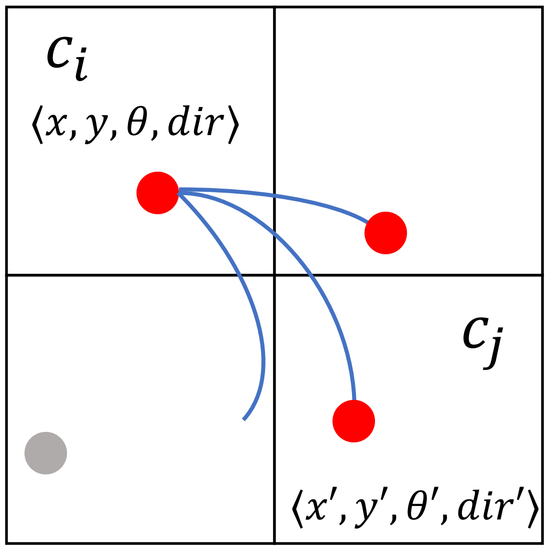
Read Junior paper Sec 6.3: http://robots.stanford.edu/papers/junior08.pdf

- Recall free-form planning problem as search
  - Vertices = discretized state/cell; edges to neighbors except obstacles
- A* associates costs with cell center
  - Problem: Resulting discrete plan cannot be executed by a vehicle
- Field D* (Ferguson and Stentz, 2005) associates cost with cell corners and allows arbitrary linear paths between cells
- Hybrid A* associates a continuous state with each cell
  - Such that the continuous coordinate can be realized by the vehicle



Hybrid A*

- Let x, y, θ $(heading), dir$ (fwd,rev) be the current state of the vehicle
- Suppose these coordinates lie in cell c_i in the A* representation
- Then we will associate c_i with coordinates $x_i = x, y_i = y, \theta_i = \theta, dir_i = dir$
- Next, suppose the vehicle applies control input u and the resulting state is $\langle x', y', \theta', dir' \rangle$ and this falls in cell c_j
  - If this is the first time c_j is visited then it is assigned coordinates x', y', θ', dir'
- Always constructs realizable paths, but it is not complete
  - Coarser the discretization, more likely hybrid A* will fail



Heuristic functions in hybrid A*

- Euclidean distance
- Nonholonomic without obstacles
  - Ignores obstacles but takes into account the non-holonomic dynamics
  - Can be computed offline
  - Fails in U-shaped dead-ends
- Holonomic with obstacles
  - Ignores the non-holonomic dynamics but includes obstacles
  - Computed online using 2D grid
- Both are admissible, could use the max of the two

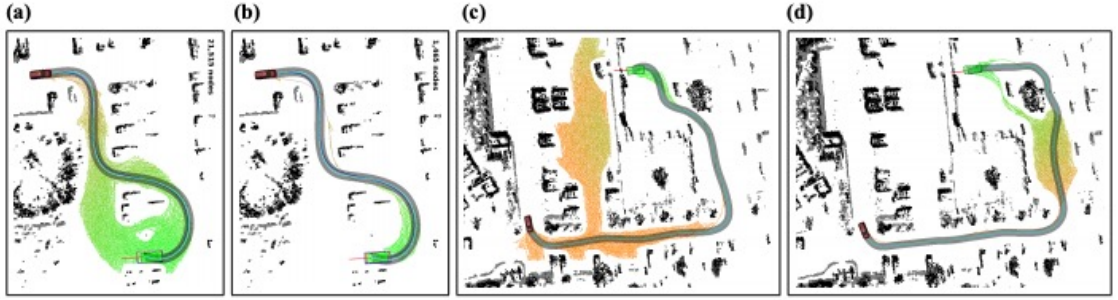
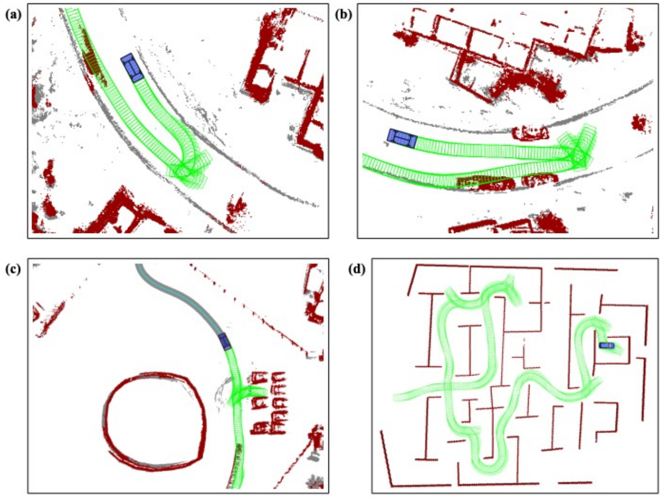


Figure 16: Hybrid-state A* heuristics. (a) Euclidean distance in 2-D expands 21,515 nodes. (b) The non-holonomic-without-obstacles heuristic is a significant improvement, as it expands 1,465 nodes, but as shown in (c), it can lead to wasteful exploration of dead-ends in more complex settings (68,730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



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* $h(v) \le h^*(v)$
  - Never over-estimate the actual cost to go
  - Are all $h(v)$ values needed ?
  - What if h is not admissible
  - How to find heuristics

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* $h(v) \leq h^*(v)$
  - Are all $h(v)$ values needed ?
  - What if h is not admissible
  - How to find heuristics
- Hybrid A* ensures that computed paths are realizable by actual vehicle dynamics