Search and Planning
Part II

ECE 484 Fall 2023

Sayan Mitra

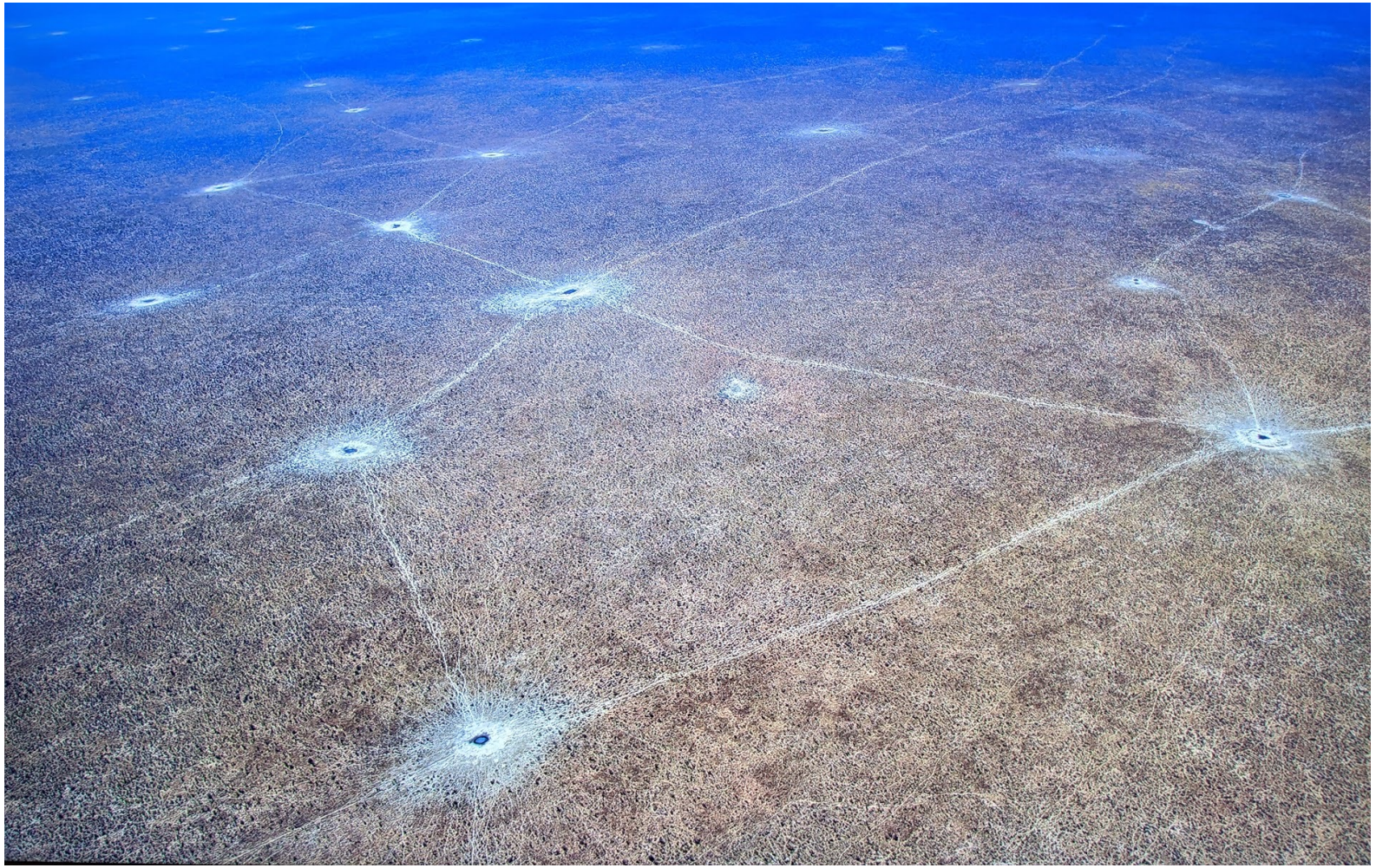
Announcements

- Pitch presentation feedback sent out
- Upcoming events
- Guest lecture: Nov 9th Dr. Qinru Li, Waymo, ECE Alumni (must attend)
- Intermediate check-in with TAs
  - How much progress have you made?
  - How well is the team working?
- Midterm 2: Nov 14th
  - Filtering and planning

Planning Outline

- Review
  - Uninformed search
  - Informed search
  - Optimal search: A
- A* Search
- Python Code for search, planning and much more: https://github.com/sayanmitracode/PythonRobotics#d-algorithm

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

// initialize queue with start

while $Q \neq \emptyset$:

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

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

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

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)

- Exercise: Prove this

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 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*

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 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*

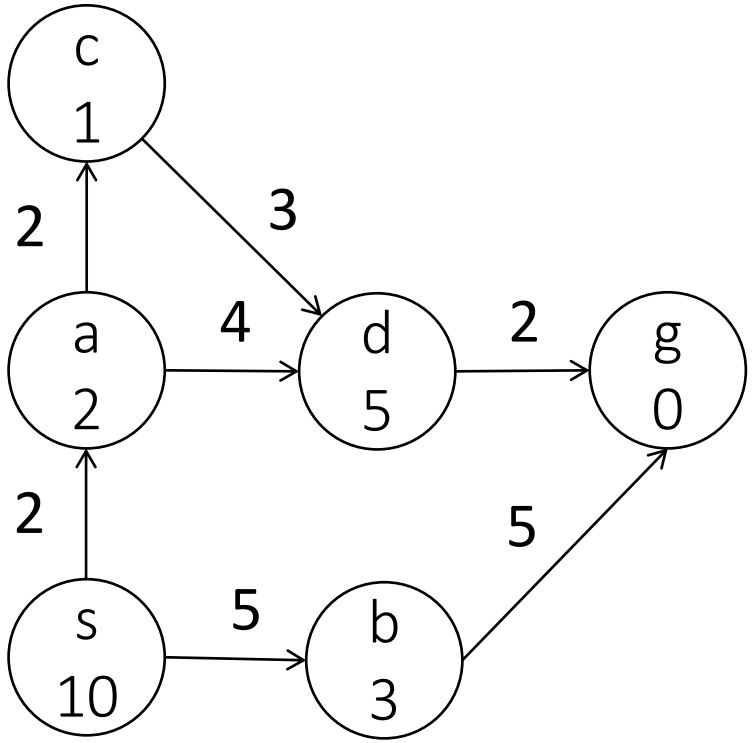
add $\langle v, P \rangle\ to\ Q$; *// Add expanded paths*

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

Example of A search

Q:

Path	g	h	f
$\langle a, s \rangle$	2	2	4
$\langle b, s \rangle$	5	3	8
c,a,s	4	1	5
d,a,s	6	5	11
d,c,a,s	7	5	12
g,b,s	10	0	10



A and A* search

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

If h = 0, A = UCS. No benefit of goal-orientedness

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

What is wrong? We just saw an example where h value was too large and biased the search away from some good paths

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---we do not know this

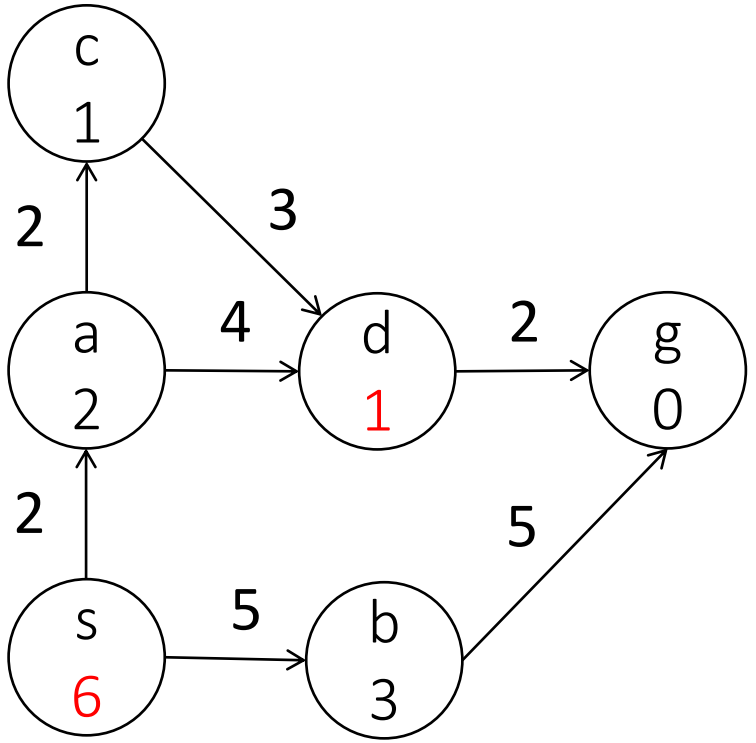
To be admissible $h(v)$ should be at most $h^*(v)$

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

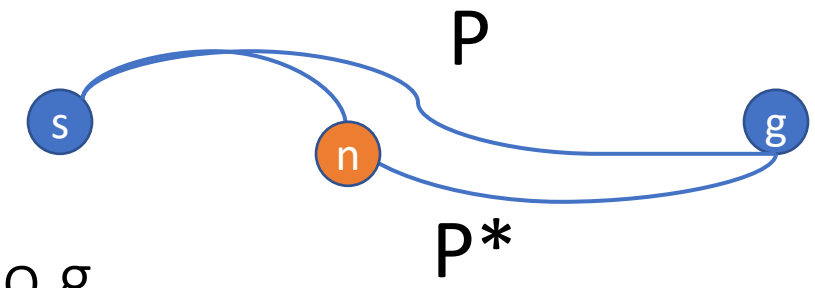
Example of A* search

Q:

Path	g	h	f
$\langle s \rangle$	0	6	6
a,s	2	2	4
b,s	5	3	8
c,a,s	4	1	5
d,a,s	6	1	7
d,c,a,s	7	1	8
g,b,s	10	0	10
g,d,a,s	8	0	8
g,d,c,a,s	9	0	9



Proof of optimality of A*



- Let w* be the cost of the optimal path from s to g
- Suppose for the sake of contradiction, that A* returns P with w(P) > w*
- Find the first *added but 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, since w* is the cost of 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

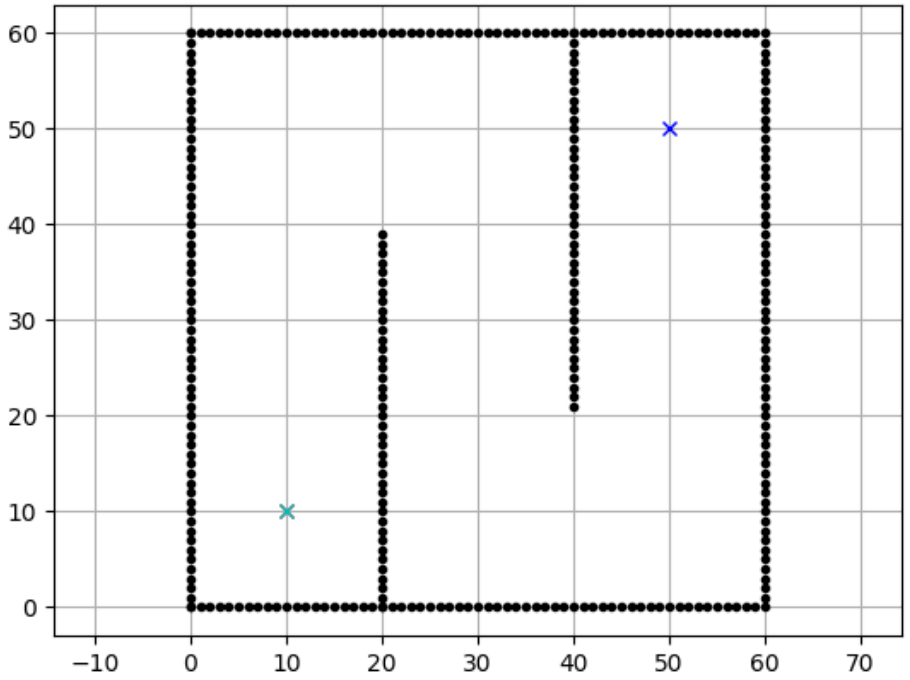
Goal state:

1	2	3
4	5	6
7	8	

Which of the following are admissible heuristics?

- h = 0 — YES, always good
- h = 1 — 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.

Applying A* for shortest path path planning



https://github.com/AtsushiSakai/PythonRobotics/tree/master

- Graph: Grid 2D plane, edges to neighboring vertices except obstacles
- Note vertices are not known ahead of time
- More interesting motion models could be added
- Heuristic: Euclidean distance to goal
- Cyan points are the searched nodes
- Try other heuristics, UCS
- Run code

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 $h: X \to \mathbb{R}_{\geq 0}$ 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:

$$f(v) = g(v) + h(v) = g(u) + w(u,v) + h(v) \geq f(u)$$

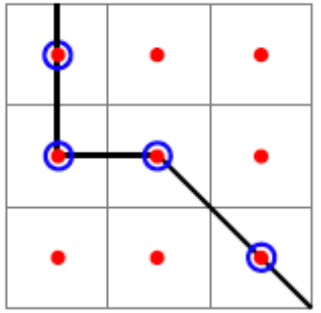
- Hence, the values of f on the sequence of nodes expanded by A∗ is non-decreasing: the first path found to a node is also the optimal path ⇒ no need to compare costs!

A* to hybrid A*

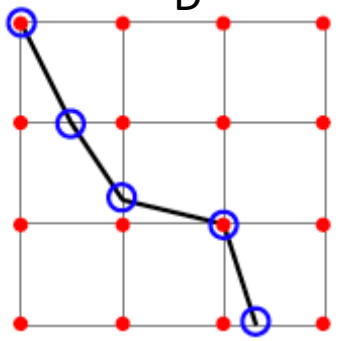
Read Junior paper Sec 6.3: http://robots.stanford.edu/papers/junior08.pdf

- 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

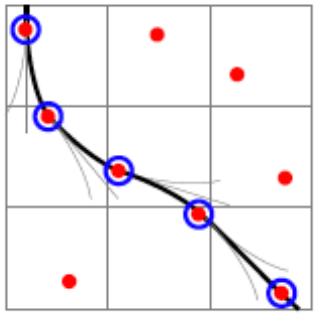
A*



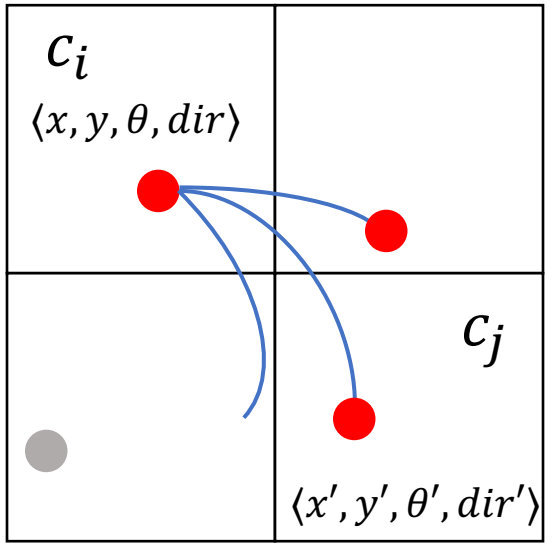
D*



hybrid A*



Hybrid A*



- Let $x, y, \theta\ (heading), dir$ (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
  - That is $\langle x', y', \theta', dir' \rangle = f(\langle x', y', \theta', dir' \rangle, u)$ where f is the dynamic vehicle model
  - 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, it is 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

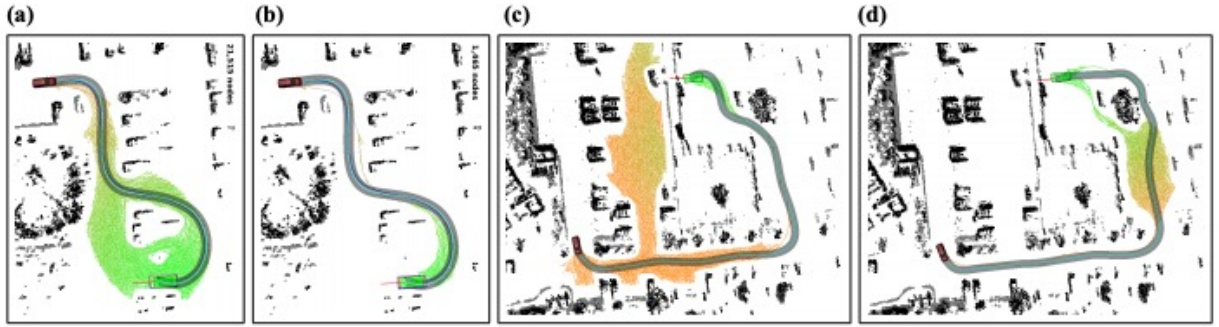
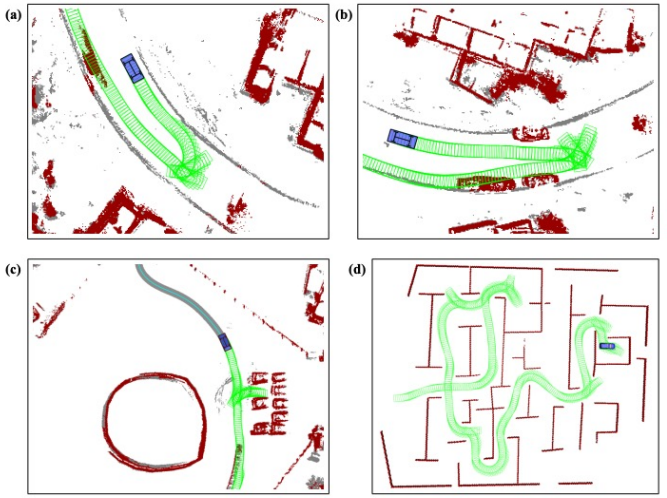


Figure 16: Hybrid-state A* heuristics. (a) Euclidean distance in 2-D expands 21,515 nodes. (b) The non-holonomic-without-obstacles heuristic is a significant improvement, as it expands 1,465 nodes, but as shown in (c), it can lead to wasteful exploration of dead-ends in more complex settings (68,730 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* $h(v) \leq h^*(v)$
  - Never over-estimate the actual cost to go
  - Are all $h(v)$ values needed ?
  - What if h is not admissible
  - When does one heuristic dominate another
  - What are consistent heuristics
- Hybrid A* to account for infeasible paths