

ECE 484: Principles of Safe Autonomy (Fall 2026)
Lecture 16: Search and Planning: A* and heuristics

Professor: Huan Zhang
huanz@illinois.edu

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

Outline

Review: deterministic search

- Uniform-cost search and greedy best-first search

A and A* search

- Admissible heuristics and optimality
- Design of heuristics
- Consistent heuristics and a closed set

Hybrid A*: planning for vehicles



Recap: the same search loop

Both keep a frontier Q of candidate paths.

Start: put the zero-cost start path in Q.

↓

Select and remove the path with the smallest priority.

↓

If it reaches the goal, return that path.

↓

Otherwise extend by one edge, add the new paths to Q, and repeat.

Uniform-cost search
Smallest $g(P)$

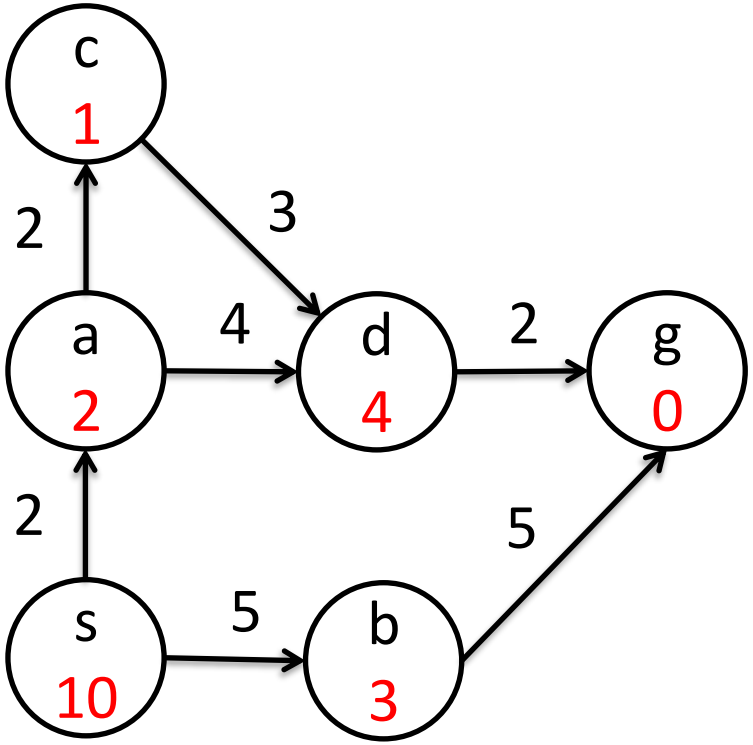
Greedy best-first search
Smallest $h(\text{head}(P))$

$g(P)$: cost so far. $h(v)$: remaining-cost estimate. $\text{head}(P)$: reached vertex of path P.

Recap: UCS versus greedy search

Uniform-cost search
Choose smallest path cost $g(P)$.
Our example: s → a → d → g
Cost: 2 + 4 + 2 = 8 (optimal).

Greedy best-first search
Choose smallest heuristic $h(\text{head}(P))$.
Our example: s → b → g
Cost: 5 + 5 = 10 (not optimal).



Edge labels: costs. Red vertex labels: h.

Today: A* uses $f(P) = g(P) + h(\text{head}(P))$.

g(P) = w(P): path cost so far. h: estimated remaining cost.

A and A* search

The challenges

- Uniform-cost search without a cost-to-go heuristic h 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 neglects the cost to arrive at a node.

The idea of A search is to combine past and future

- Use both the cost of the partial path P and the estimated cost to reach the goal from $v = \mathrm{head}(P)$.
- Sum of the two costs: $f(P) = g(P) + h(v)$.
- $g(P) = w(P)$: cost-to-come along this path.
- $h(v)$: cost-to-go estimate from v to the goal.
- $f(P)$: estimated total cost through this path.

A search

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

If h = 0 then A search = UCS.

A search is complete, but is not necessarily optimal.

A* search

- Choose an **admissible heuristic**: $0 \leq h(v) \leq h^*(v)$ for every vertex v.
- $h^*(v)$: cost of the cheapest path from v to the goal.
- To be admissible, h should never overestimate the true cost-to-go.
- A search with an admissible heuristic is called A* — guaranteed to find an optimal path.

A* search

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

while $Q \neq \emptyset$:

pick (and remove) path P with lowest estimated cost $f(P) = g(P) + h(\text{head}(P))$ 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*

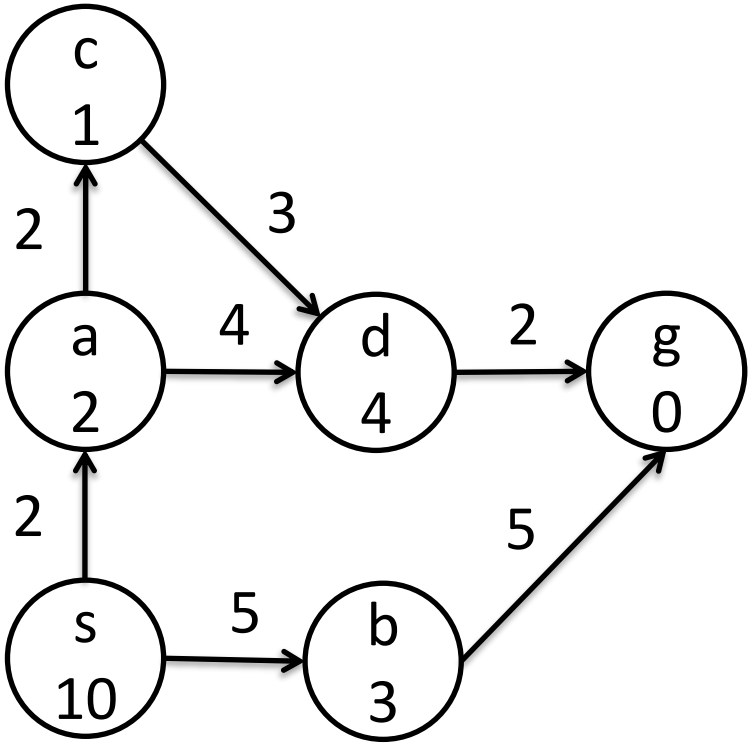
add $\langle v, P \rangle\ to\ Q$; *// Add expanded paths*

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

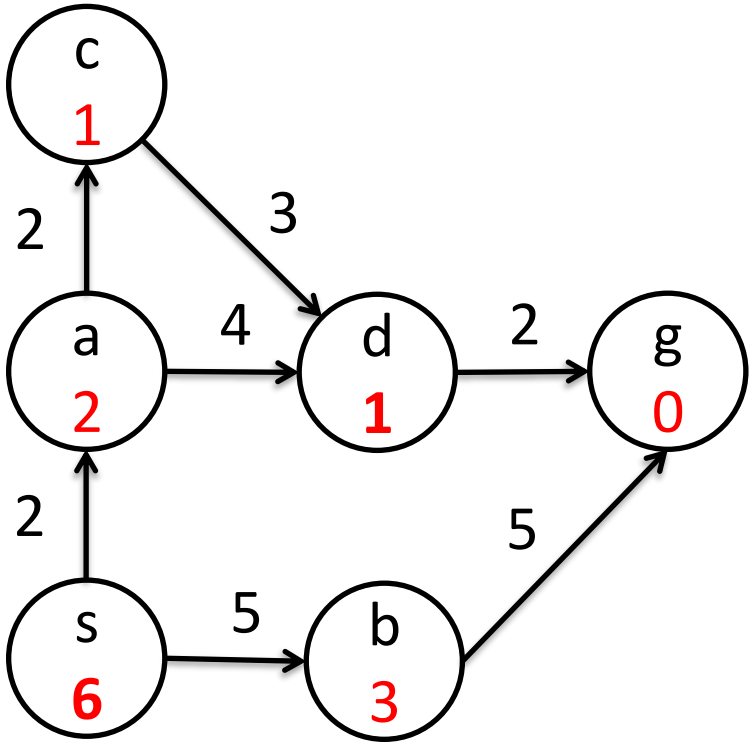
Q: frontier of paths. This version has no closed set (visited list).

A vertex can be reached again through another path.

Check the heuristic before using A*



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



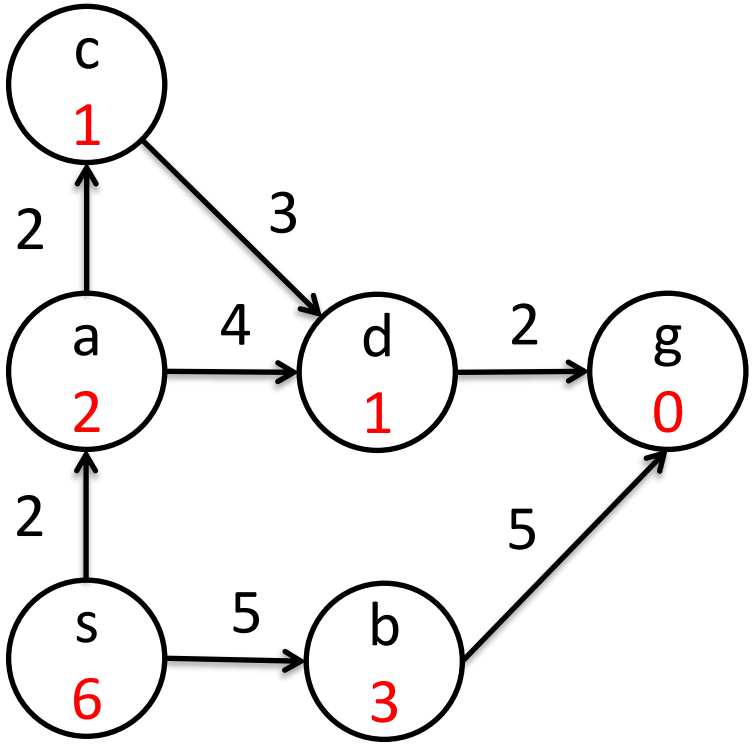
Updated heuristic values

Example of A* search

Q:

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

Rows are not sorted. Select the smallest f.



Admissible heuristic values

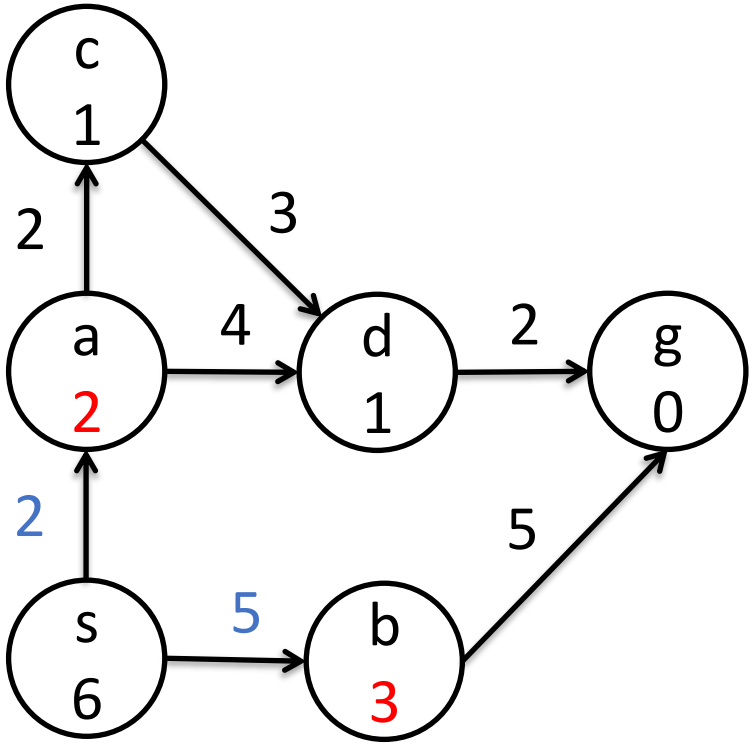
g: path cost • h: estimated remaining cost • f = g + h

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

Rows are not sorted. Select the smallest f.



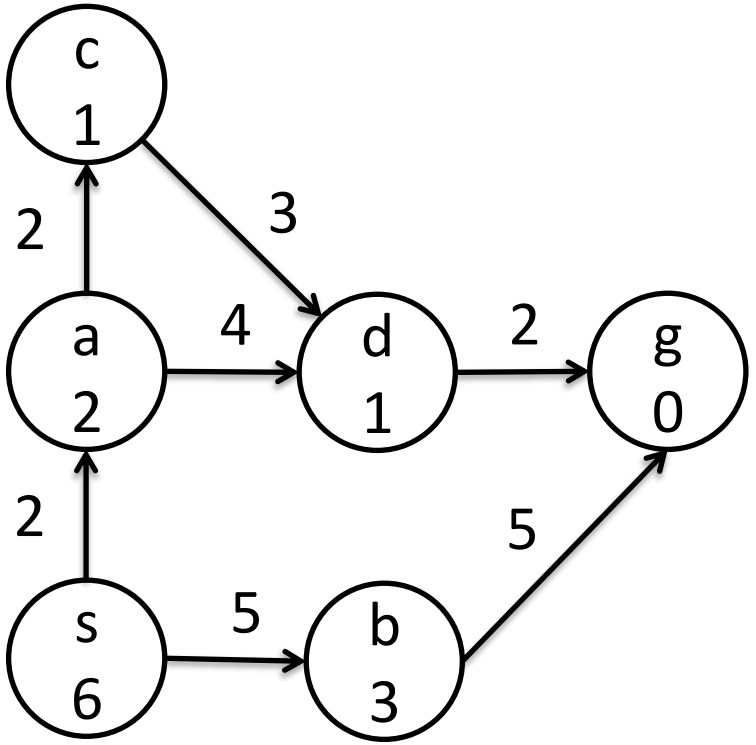
g: path cost • h: estimated remaining cost • f = g + h

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

Rows are not sorted. Select the smallest f.



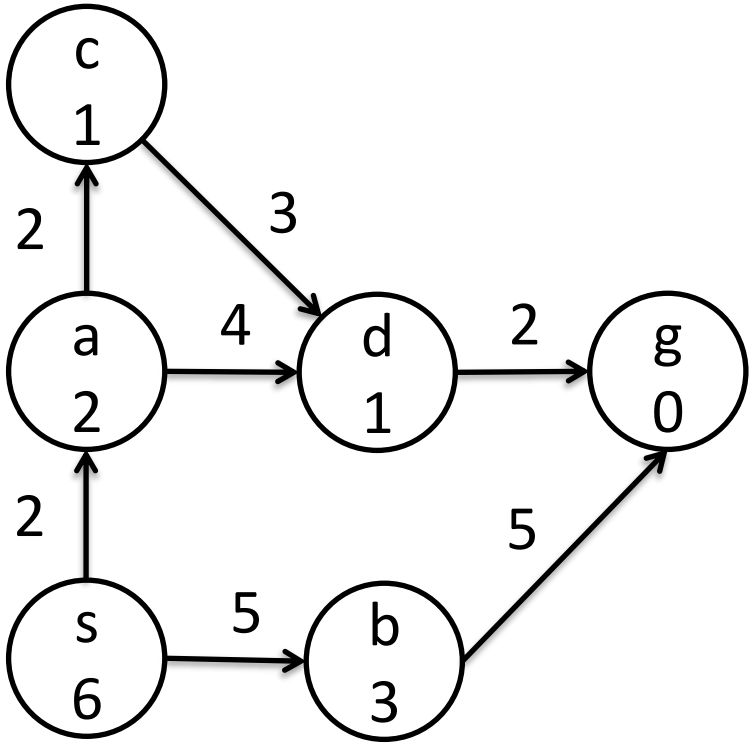
g: path cost • h: estimated remaining cost • f = g + h

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

Rows are not sorted. Select the smallest f.

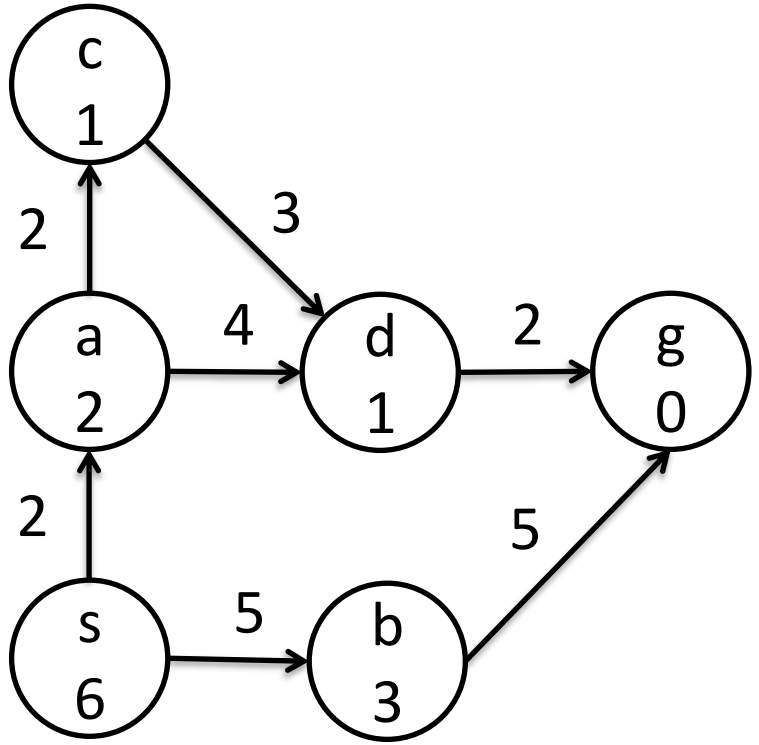


g: path cost • h: estimated remaining cost • f = g + h

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



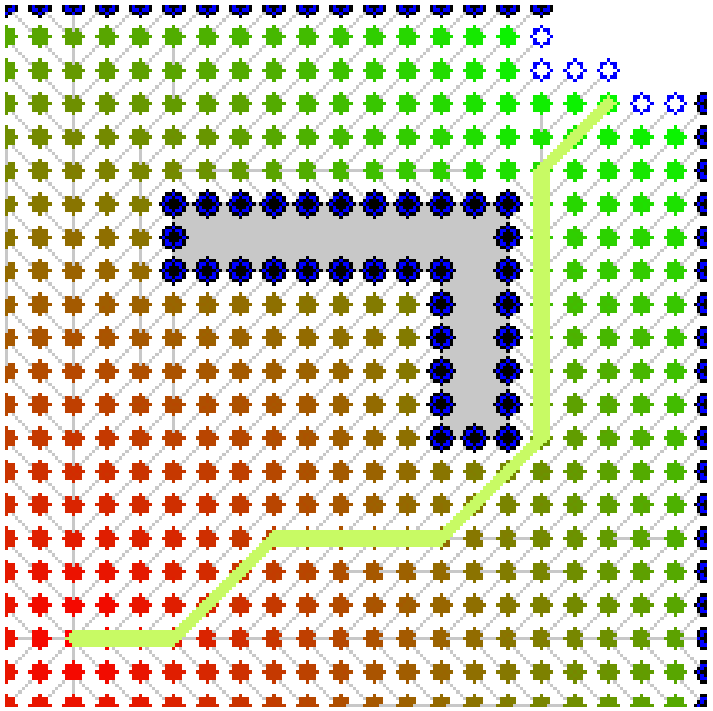
Rows are not sorted. Select the smallest f.

Three paths tie at f = 8. Select the goal path and return cost 8.

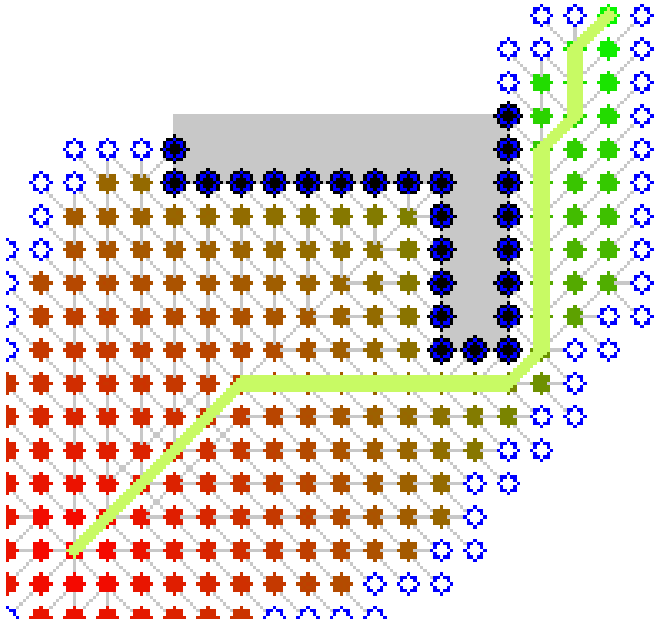
g: path cost • h: estimated remaining cost • f = g + h

Uniform cost search vs A*

Uniform-cost search



A* search



Red: start Green: goal Gray: obstacle Colored marks: explored region

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

A* optimality: the theorem and the key idea

Theorem. Suppose the graph is finite, edge costs are positive, and a path to the goal exists. With an admissible heuristic h, A* **terminates and returns a cheapest path.**

We use A* **without a closed set**: it keeps different paths to the same vertex, selects a path with the smallest f score from the frontier Q, and only then tests whether it reaches the goal.

For the proof, choose a cheapest path P^* and call its cost w^*. A **prefix** R is the initial part of this path, from start to a vertex v. A* **does not need to know** P^*.



Initially, the start path is in Q. Whenever A* expands a prefix of P^*, it adds the next longer prefix to Q. Other expansions do not remove that waiting prefix. **Therefore, some prefix of P^* remains in Q until the algorithm returns.**

A* optimality: compare the two path costs

The same cheapest path P^* has total cost w^*.

Cost already paid: $g(R)$	Actual cost of the remaining part

$h(v)$ cannot be larger than this remaining cost.

1. Bound the waiting prefix. The estimate $h(v)$ cannot exceed the actual remaining cost along P^*. Adding the cost $g(R)$ already paid therefore gives $f(R) = g(R) + h(v) \leq w^*$.

2. Compare it with the selected goal path. Let P be the first goal path selected by A*. Since A* chooses the smallest score in Q and R is still waiting, $f(P) \leq f(R) \leq w^*$.

3. At the goal, the estimate is the actual cost. There is no cost left, so $h(goal) = 0$ and $f(P) = w(P)$. Thus $w(P) \leq w^*$. Since w^* is the cheapest possible cost, **P must be optimal.**

Why must it eventually return? Until then it only selects paths with $f \leq w^*$. Positive edge costs bound their length, so a finite graph has only finitely many such paths.

Admissible heuristics

How to find admissible heuristics? A heuristic should never overestimate the cost-to-go.

Some examples

- $h(v) = 0$: this always works! However, it is not very useful: A* = UCS.
- For path-length costs: $h(v) = \text{distance}(v, goal)$, the straight-line distance.
- More generally, $h(v) = \|v - goal\|_p$ when each edge cost is at least that norm distance.

More generally

- Choose h as the optimal cost-to-go for a relaxed problem that is easy to compute.
- Relaxed problem: ignore some constraints in the original problem.
  - Ignore some obstacles, or 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 are the vertices on the graph?
- How to define the edges?
- What are the start and goal vertices?
- How is the cost 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 — uniform-cost search; can be slow.
h = 1	NO — h(goal) must be 0.
h = number of numbered tiles in the wrong position	YES — “teleport” each tile in one move.
h = sum of Manhattan distances of numbered tiles from their goal positions	YES — move each tile, ignoring other tiles.

Counting a simple 8-puzzle heuristic

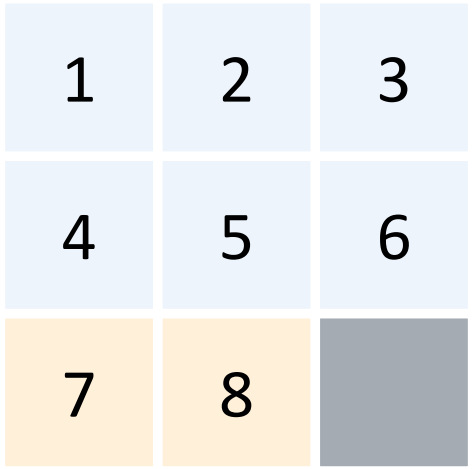
Current state



Count tiles 7 and 8.

Each is one square from its goal.

Goal state



Misplaced tiles: 2

Manhattan distance: 1 + 1 = 2

One legal move shifts one numbered tile by one square.

Which heuristic is better?

Some heuristics are better than others

- $h = 0$ is admissible, but is not very useful.
- $h = h^*$ gives the exact remaining cost, but can be expensive to compute.

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; 0 is dominated by all of them.

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 computing h and the complexity of the search.

Consistent heuristics

An additional useful property for A* heuristics is **consistency**.

A nonnegative heuristic on V is consistent if $h(goal) = 0$ and

$h(u) \leq w(u, v) + h(v)$ for every edge $(u, v) \in E$.

In other words, a consistent heuristic satisfies a triangle inequality.

If P ends at u and P^+ extends it to v, then

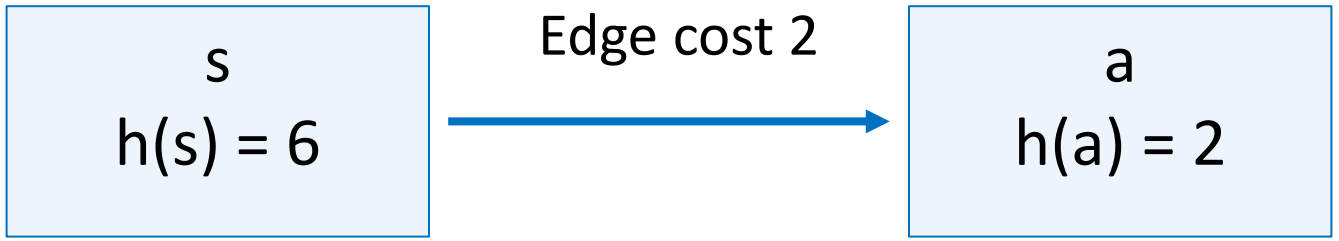
$f(P^+) = g(P) + w(u, v) + h(v) \geq g(P) + h(u) = f(P)$.

The first path expanded to a vertex has minimum cost to that vertex.

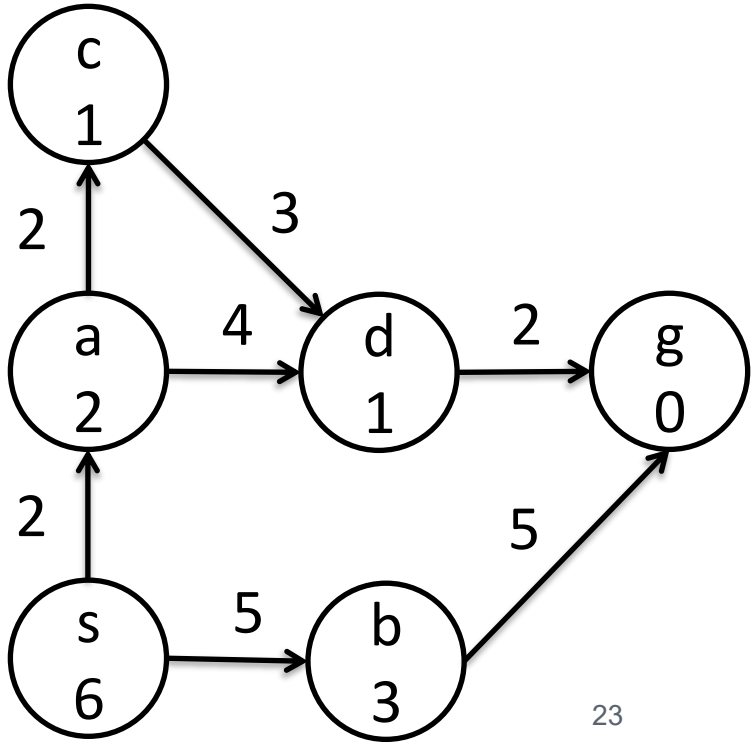
This makes a **closed set** safe: **expand each vertex only once**.

Admissibility and consistency are different

Admissible does not always mean consistent.



From our example:
s → a → d → g costs 8.
a → d → g costs 6.



Consistency would require $6 \leq 2 + 2$.

False: the estimate drops by 4 after paying only 2.

The earlier A* trace kept all paths, so admissibility was enough.

A* search with a closed set (requires consistency)

$Q \leftarrow \{\langle start \rangle\}$; $C \leftarrow \emptyset$
while $Q \neq \emptyset$:
  remove path P with smallest $f(P) = g(P) + h(\text{head}(P))$
  $u \leftarrow \text{head}(P)$
  if $u \in C$: **continue**
  if $u = goal$: **return** P
  add u to C
  for each edge $(u, v) \in E$ with $v \notin C$:
    add $\langle v, P \rangle$ to Q
return FAILURE

Use a consistent h. Q: frontier of paths. C: expanded vertices.

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 into 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

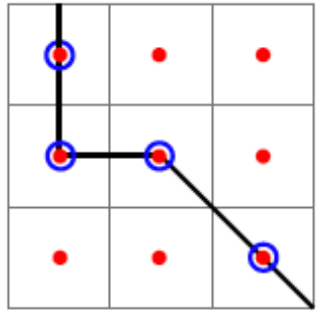
We could represent vehicle state in a *uniform* discrete grid, for example, a 4D grid: $x, y, \theta\ (heading), dir$ (fwd,rev)

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

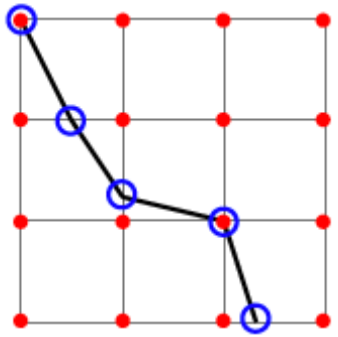
But, the discrete path (a) may not be executable by the vehicle dynamics

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

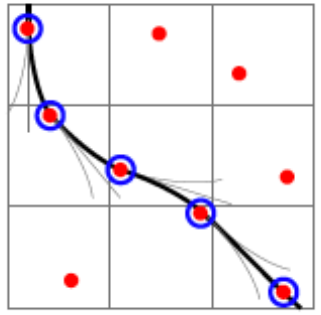
(a)



(b)



(c)

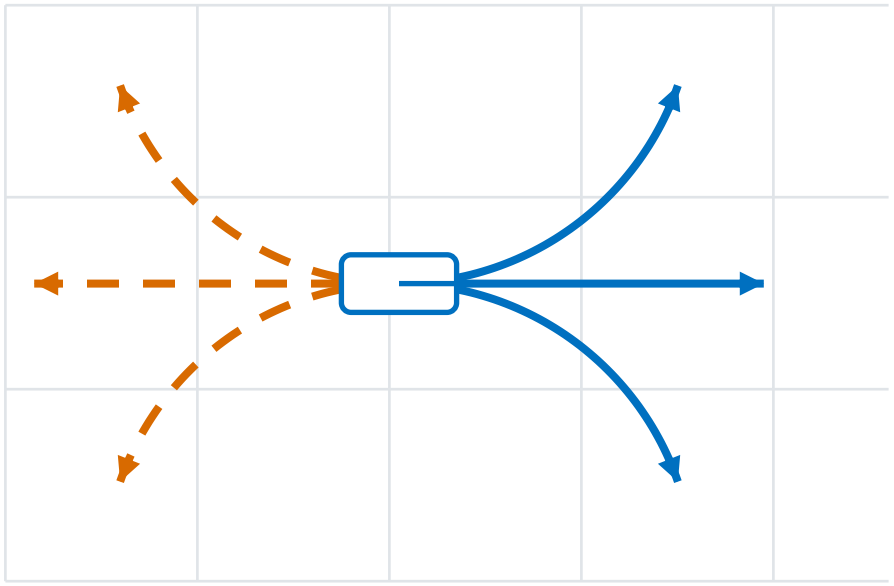


Montemerlo et al., "Junior: The Stanford Entry in the Urban Challenge," J. Field Robotics 25(9), 2008, Fig. 15.

Hybrid A*: states and motion primitives

A state keeps position, heading, and direction.

$n = (x, y, \theta, dir)$; dir: forward or reverse.



Reverse Forward

Motion primitives: short simulated motions.
Try left, straight, and right in each direction.

Cells: bins of position, heading, and direction.
Keep a continuous pose inside each cell.

Hybrid A* Search

$O \leftarrow \{start\}$ with $g(start) = 0$, $h(start) = \text{heuristic}(start)$; $C \leftarrow \emptyset$

while $O \neq \emptyset$:

  remove n with smallest $f(n) = g(n) + h(n)$ from O

  if n is in the goal region: **return** its path

  add $\text{cell}(n)$ to C

  for each motion primitive m:

    simulate the motion from n to a successor s

    if the motion collides, or $\text{cell}(s) \in C$: **continue**

    $g(s) = g(n) + \text{cost}(n, s)$; $h(s) = \text{heuristic}(s)$

    if the cell has no open state, or $g(s)$ is lower:

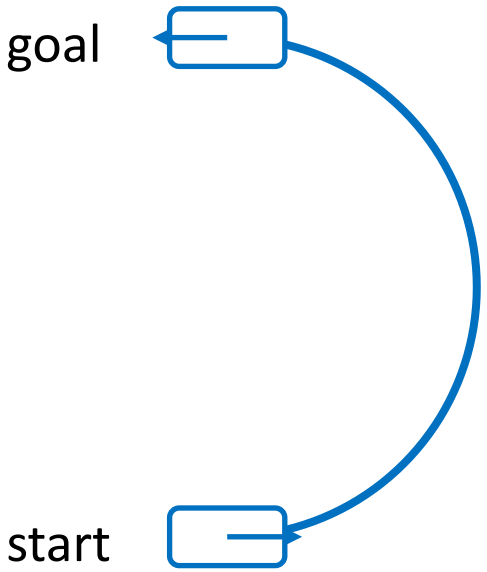
      store s and its parent in that cell; add/update it in O

return FAILURE

O: frontier (same role as Q). C: closed cells. g(n): cost of the stored path.

Two relaxed problems guide hybrid A*

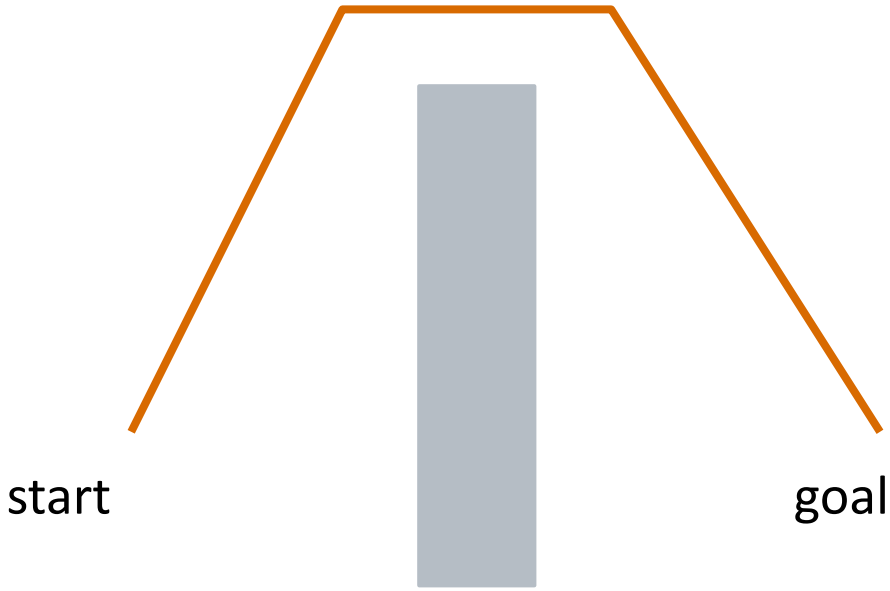
Keep turning limits; ignore obstacles



A car cannot turn in place.

h_{turn}: shortest relaxed path cost

Keep obstacles; ignore turning limits



A point must still go around the wall.

h_{obs}: shortest relaxed path cost

Use $h = \max(h_{turn}, h_{obs})$: the stronger lower bound.

Junior: The Stanford Entry in the Urban Challenge

Shaded areas: expanded states. Thick curves: returned paths.

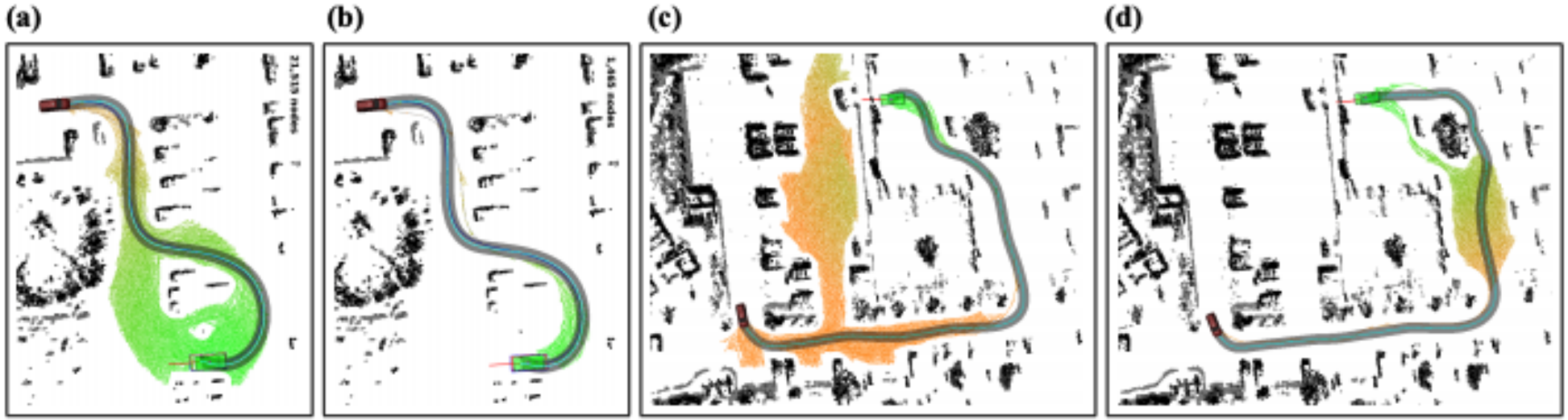


Figure 16: Hybrid-state A* heuristics. (a) Euclidean distance in 2-D expands 21,515 nodes. (b) The non-holonomic-without-obstacles heuristic is a significant improvement, as it expands 1,465 nodes, but as shown in (c), it can lead to wasteful exploration of dead-ends in more complex settings (68,730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

Montemerlo et al. (2008), Fig. 16 • robots.stanford.edu/papers/junior08.pdf

Properties of Hybrid A*

Soundness / feasibility

- The returned path joins feasible, collision-checked motion primitives.
- Soundness can be compromised if the primitives do not represent feasible trajectories.

Completeness

- Hybrid A* is not guaranteed complete: merging states in a cell can discard a useful route.
- Finer discretization can help, but gives no general guarantee.

Optimality

- Hybrid A* is not guaranteed optimal: it uses a finite set of primitives and merges continuous states.
- The quality of the solution depends on the motion primitives and heuristics.

Summary

A* combines cost-to-come $g(P)$ and a cost-to-go estimate $h(\mathrm{head}(P))$.

- Informed search: choose the smallest estimated total cost.

For the optimality guarantee, use an **admissible** heuristic: $h(v) \leq h^*(v)$.

- Never overestimate the actual cost-to-go.
- Dominating heuristics give stronger bounds; consistency makes a closed set safe.

Hybrid A* extends A* to continuous vehicle states and incorporates kinematic constraints.

- It produces feasible motions, but merging states loses completeness and optimality guarantees.