Search and Planning 2

Principles of Safe Autonomy ECE498SMA

Sayan Mitra

Based on some lectures by Emilio Frazzoli

4 March

Outline

- A* and Hybrid A*
- Dynamic programming

Review of last lecture

- Search for collision free trajectories can be converted to graph search
- Uniform cost search uses cost-to-come
- Greedy/best-first search
- A search

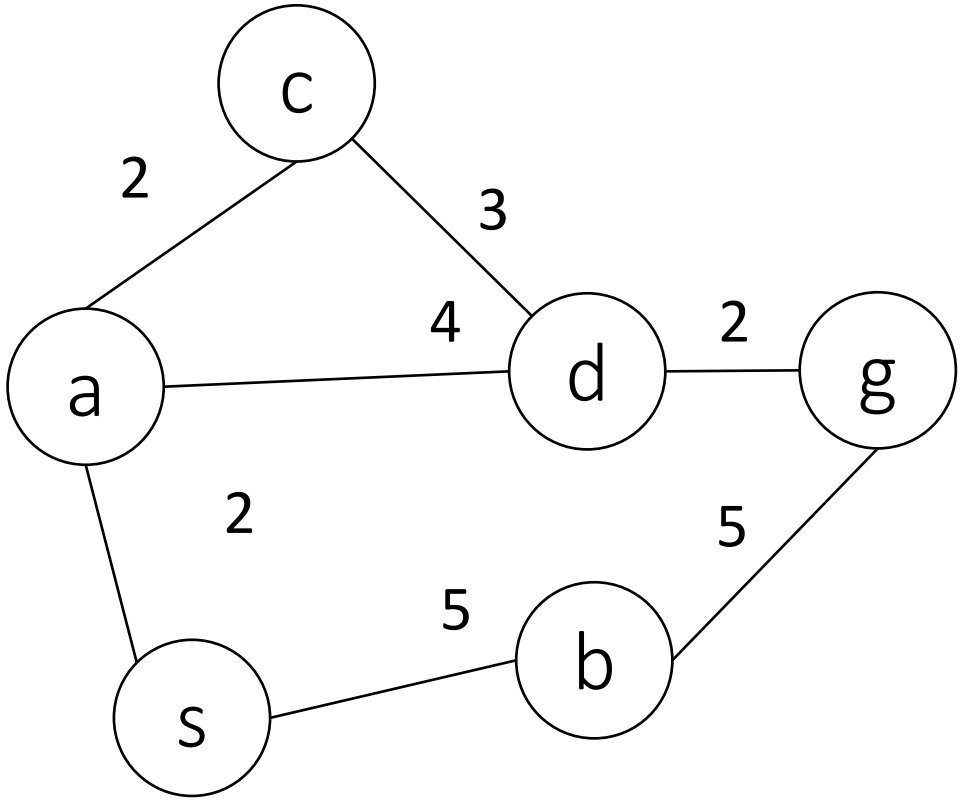
Shortest path problems

- Input: $\langle V, E, w, start, goal \rangle$
  - V: (finite) set of vertices
  - $E \subseteq V \times V$: (finite) set of edges
  - $w : E \to \mathbb{R}_{>0}$: 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 (starting in start and ending in goal, such that its weight $w(P)$ is minimal among all such paths
  - The weight of a path is the sum of the weights of its edges.

Example: Find the minimal path from s to g:

a simple path P: {g, d, a, s}

w(P): 8



Search Performance Metrics

- Soundness: when a solution is returned, is it guaranteed to be correct?
- Completeness: – is the algorithm guaranteed to find a solution when there is one?
- 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?

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*

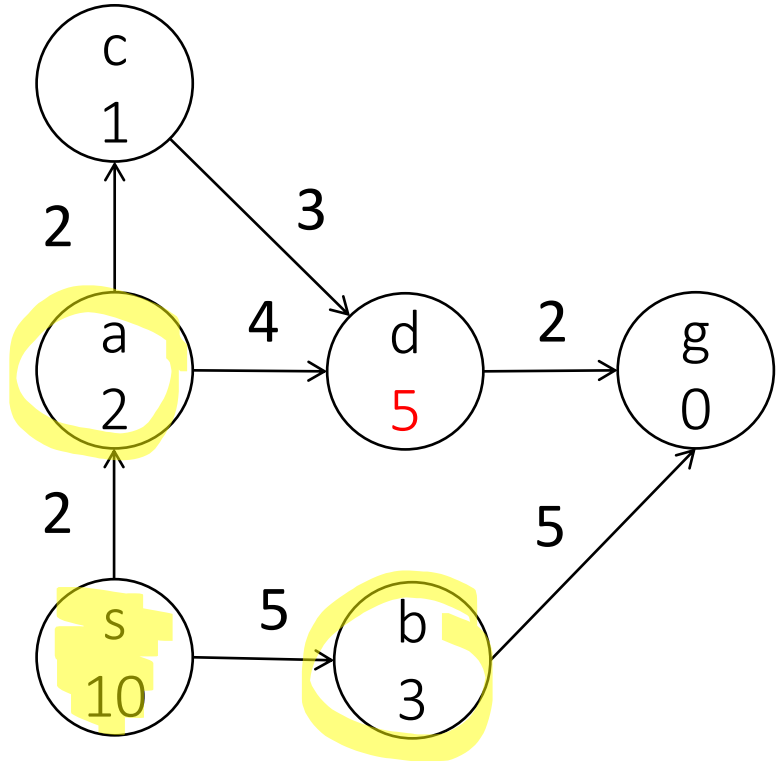
add $\langle v, P \rangle\ to\ Q$; *// Add expanded paths*

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

Example of A search

Q:

Path	g	h	f
⟨s⟩	0	10	10
⟨a,s⟩	2	2	4
⟨b,s⟩	5	3	8



Example of A search

Q:

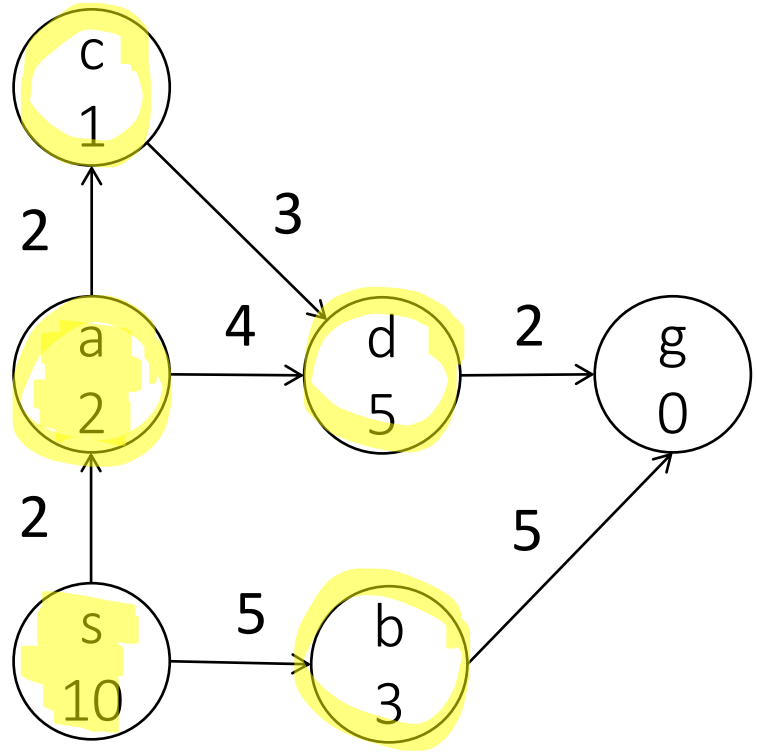
Path	g	h	f
~~$\langle a, s \rangle$~~	~~2~~	~~2~~	~~4~~
→ $\langle b, s \rangle$	5	3	8

→ $\langle c, q, s \rangle$ 4 1 5

$\langle d, q, s \rangle$ 6 5 11

$\langle d, c, q, s \rangle$ 7 5 12

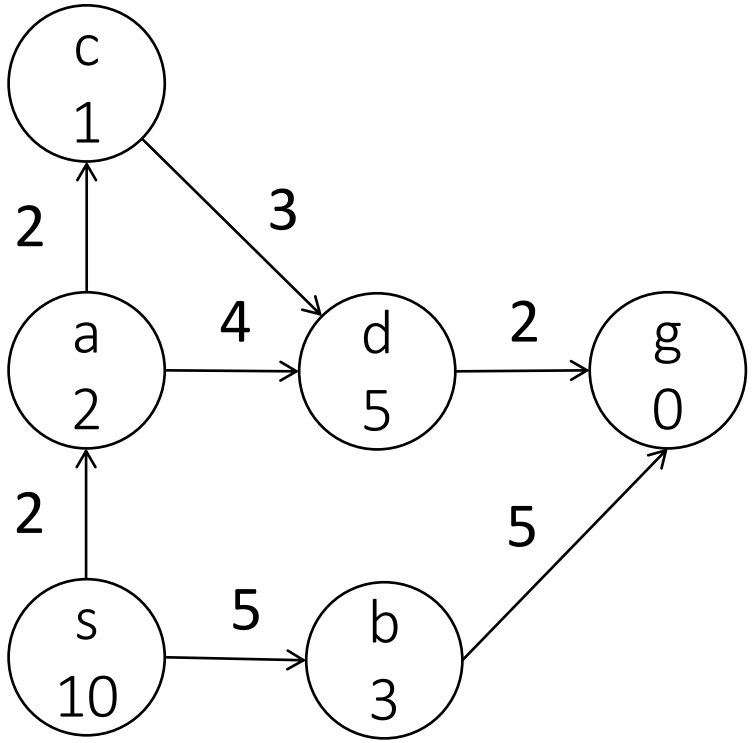
$\langle g, b, s \rangle$ 10 0 10 not optimal



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 $that\ h(v) \leq h^*(v)$
  - $h^*(v)$ 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

Example of A* search

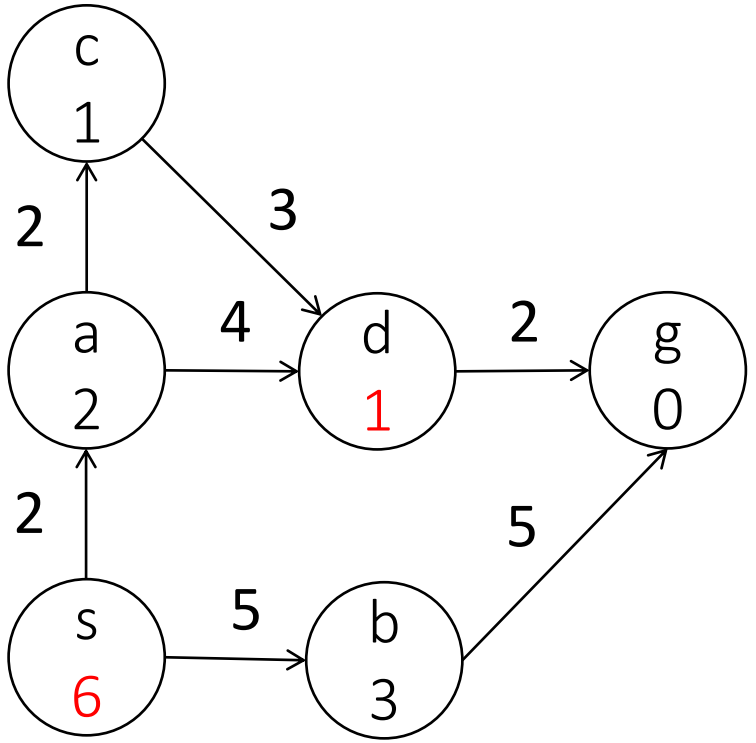
Q:

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

changed h

finds $\langle g, d, a, s \rangle$

optimal path



Proof of optimality of A*

- Let P* be the optimal path with cost w*
- 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
  - $h(v) = \left|\left|v - g\right|\right|_p$, 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.

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

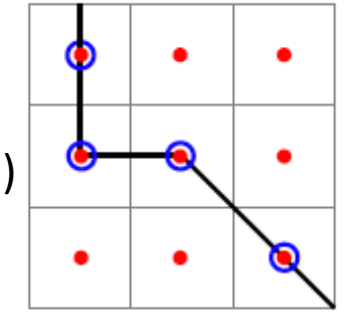
Consistent heuristics

- 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!

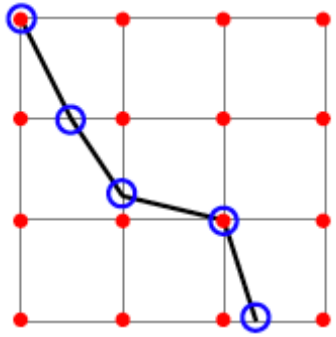
Hybrid A*

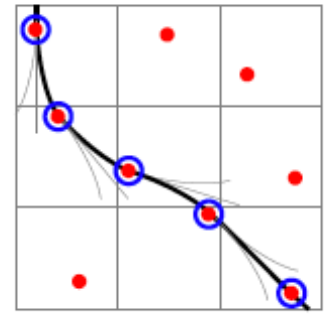
open cells: cells that are accessible from root and closed cells

(a)



- One problem with regular (non-hybrid) A* is that the resulting discrete plan cannot be executed by a vehicle
- Represent vehicle state in a ~~*uniform*~~ discrete grid
  - 4D grid: $x, y, \theta\ (heading), dir$ (fwd,rev)
- If the current coordinate is $\langle x, y, \theta \rangle$ and those coordinates lie in cell c_i then the *representative continuous state* for cell c_i will be $x_i = x, y_i = y, \theta_i = \theta$
- After applying control input u to vehicle, suppose the predicted state is x', y', θ'
  - $x', y', \theta' = f(x, y, \theta, u); \dot{x} = \cdots$
  - representative for $c_j = x', y', \theta'$
  - This defines a transition from c_i to c_j





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

Junior: The Stanford Entry in the Urban Challenge

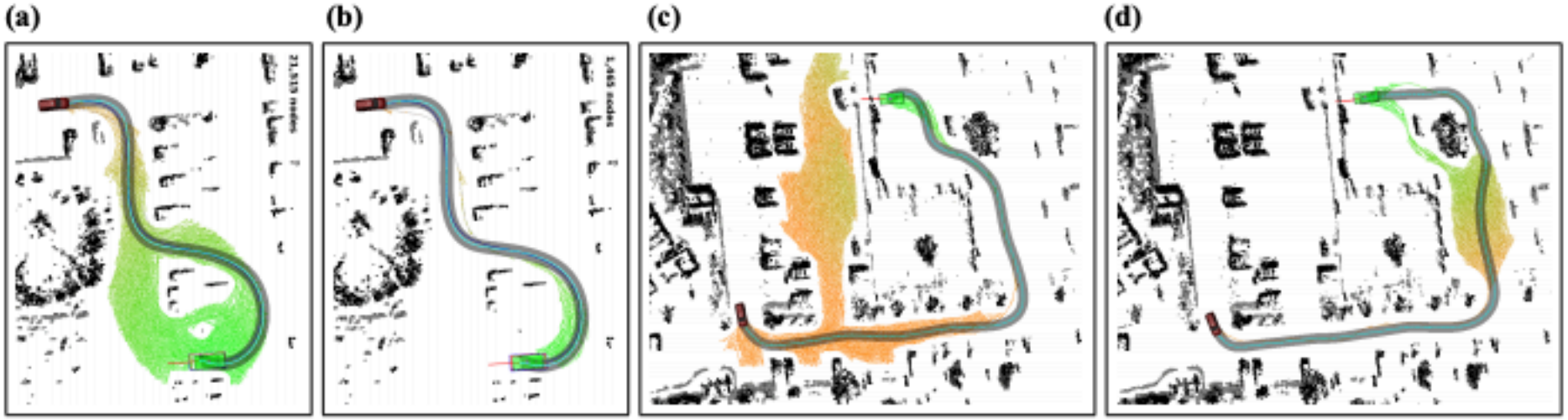


Figure 16: Hybrid-state A* heuristics. (a) Euclidean distance in 2-D expands 21,515 nodes. (b) The non-holonomic-without-obstacles heuristic is a significant improvement, as it expands 1,465 nodes, but as shown in (c), it can lead to wasteful exploration of dead-ends in more complex settings (68,730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

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
  - How to find heuristics

Dynamic programming/Dijkstra

- The optimality principle
  - Let P = (s, . . . , v, . . . g) be an optimal path (from s to g).
  - Then, for any v ∈ P, the sub-path S = (v, . . . , g) is itself an optimal path (from v to g)

- Using the optimality principle
  - Essentially, optimal paths are made of optimal paths. Hence, we can construct long complex optimal paths by putting together short optimal paths, which can be easily computed.
  - Fundamental formula in dynamic programming:

$$h^*(u) = \min_{(u,v)\in E} [w((u,v)) + h^*(v)]$$

Typically, it is convenient to build optimal paths working backwards from the goal.

Dijkstra's algorithm

$Q \leftarrow V$ *// all vertices in queue queue*

for all $v \in V, \bar{h}(v) = (\infty$ *if* $v \notin G, 0$ *otherwise)*

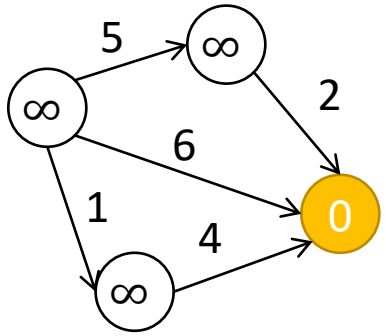
while $Q \neq \emptyset$ do

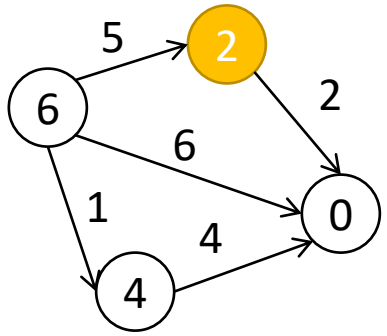
$u \leftarrow argmin_{v \in Q}$ {pick *minimum cost vertex in* Q}

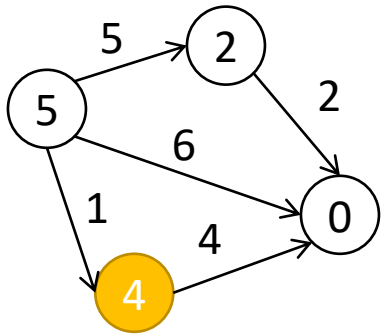
remove u from Q

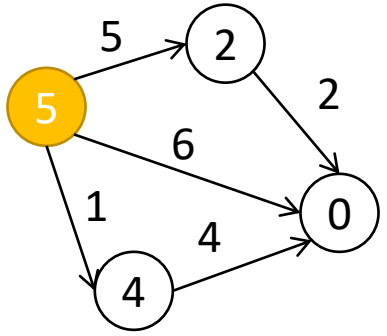
foreach $(v, u) \in E$ do *// for all edges*

$\bar{h}(v) \leftarrow \min(\bar{h}(v), \bar{h}(u) + w(e))$ *// Relax costs*









Dynamic programming/Dijkstra

- The output of Dijkstra's algorithm is the optimal cost-to-go function h*
- For any vertex we can compute the optimal outgoing edge via the dynamic programming equation
- Dynamic programming requires the computation of all optimal sub-paths, from all possible initial states (curse of dimensionality)

Concluding remarks

- A $*$ optimal and very effective in many situations.
- However, in some applications, it requires too much memory. Some possible approaches to address this problem include
  - Branch and bound
  - Anytime A $*$
- Other search algorithms
  - D $*$ and D $*$ -lite: versions of A $*$ for uncertain graphs
  - Hill search: move to the neighboring state with the lowest cost