CSE 478 Robot Autonomy
Heuristic Search

Abhishek Gupta (abhgupta@cs)
Siddhartha Srinivasa (siddh@cs)

TAs:
Carolina Higuera (chiguera@cs)
Rishabh Jain (jrishabh@cs)
Entong Su (ensu@cs)

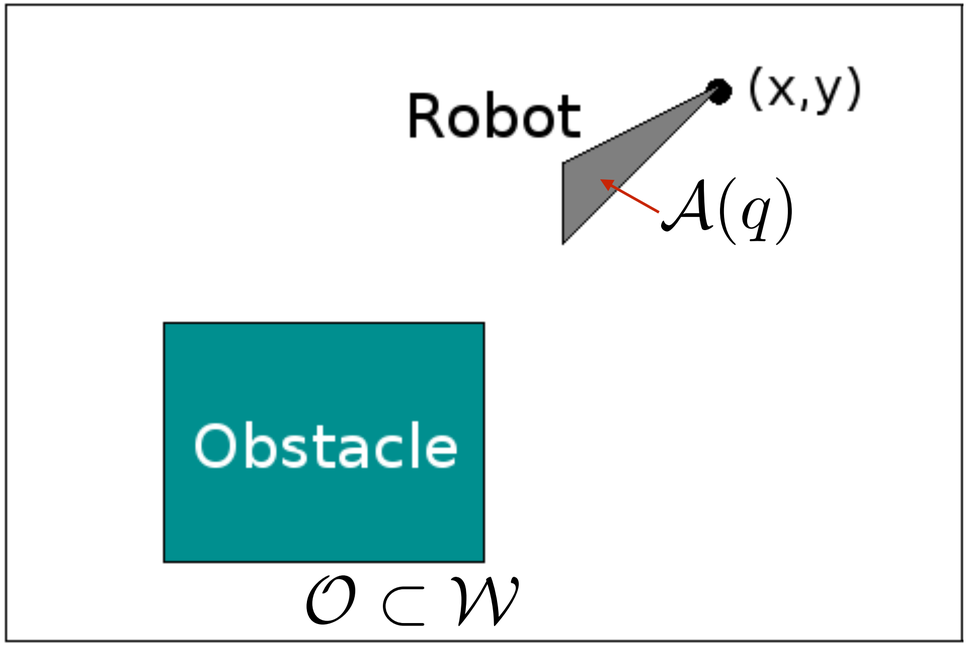
Example 3: Racecar

$$\mathbb{R}^2 \times \mathbb{S}^1$$

special euclidean group $SE(2)$

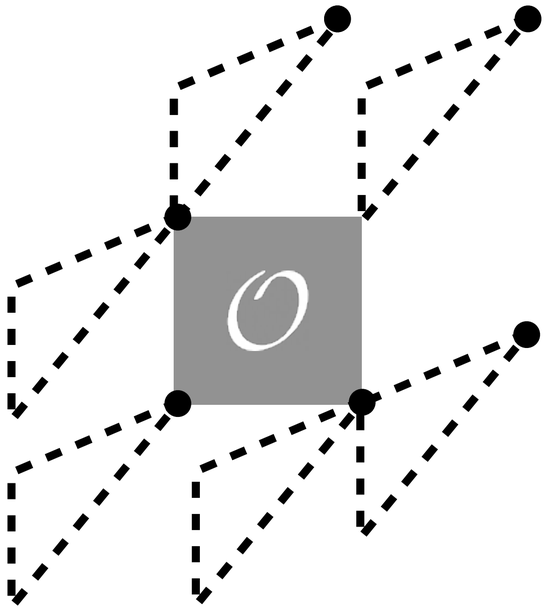
Obstacles

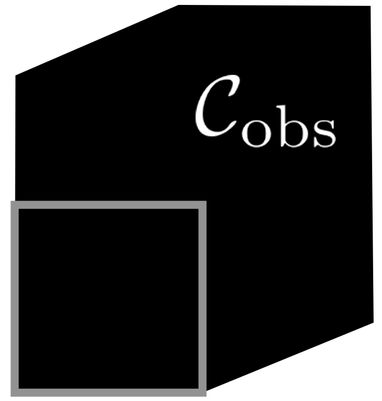
Example 1: Translating triangle



Can be efficiently computed using Minkowski sum

Example: Translating Triangle in Plane

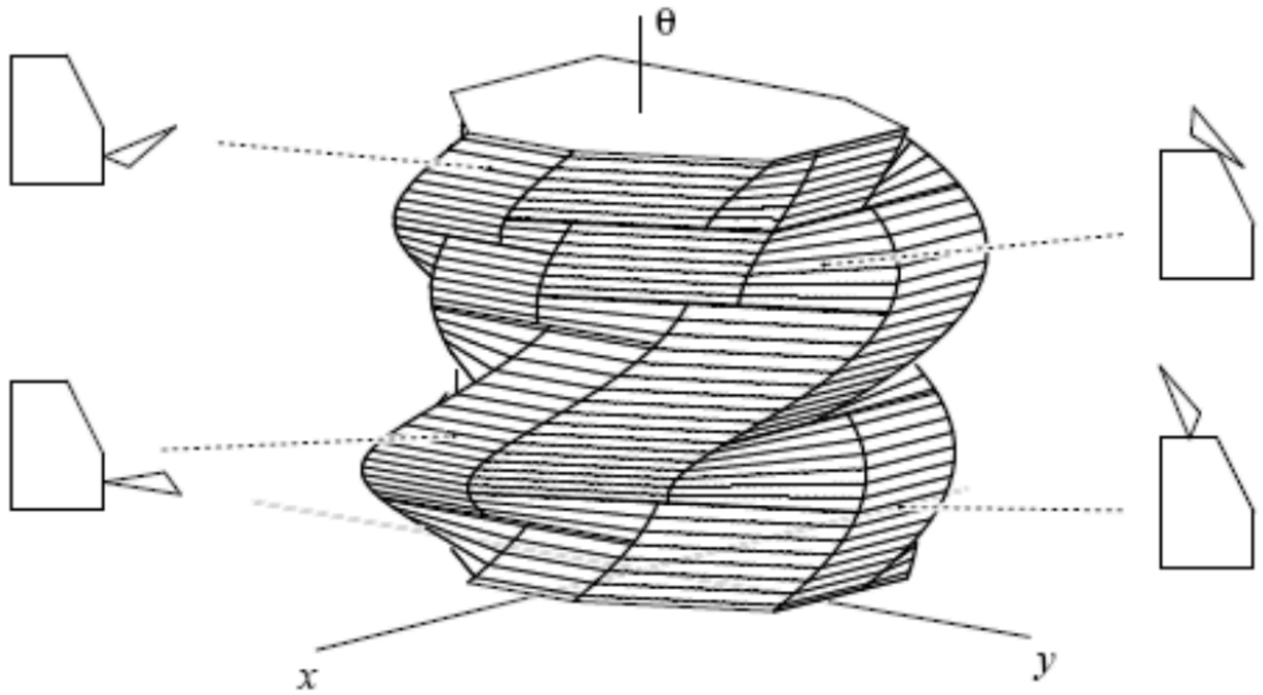




Can be computed for convex polygons (Minkowski sum)

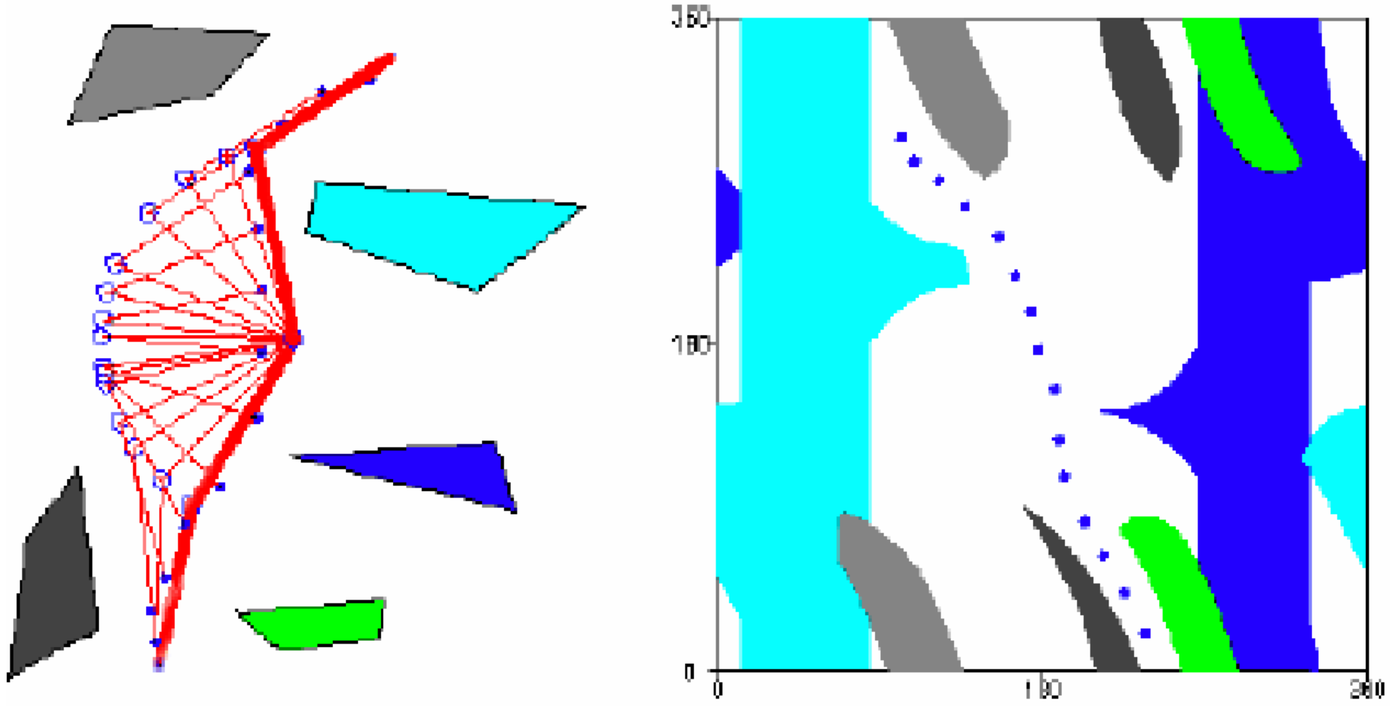
(EXAMPLE FROM LYDIA KAVRAKI AND STEVEN LAVALLE)

Example: Translating and Rotating Triangle



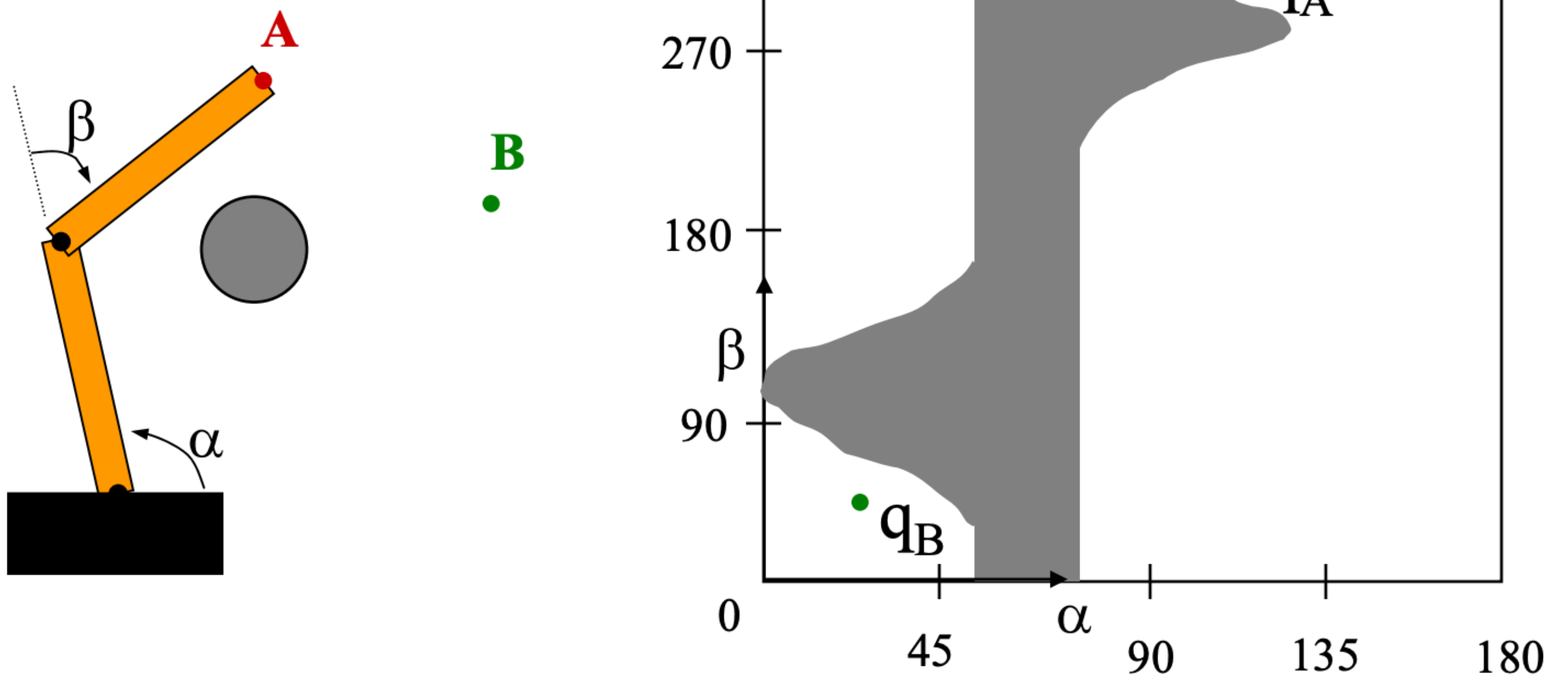
(EXAMPLE FROM HOWIE CHOSET)

Example 3: 2-link planar arm



Courtesy Tapomayukh Bhattacharya

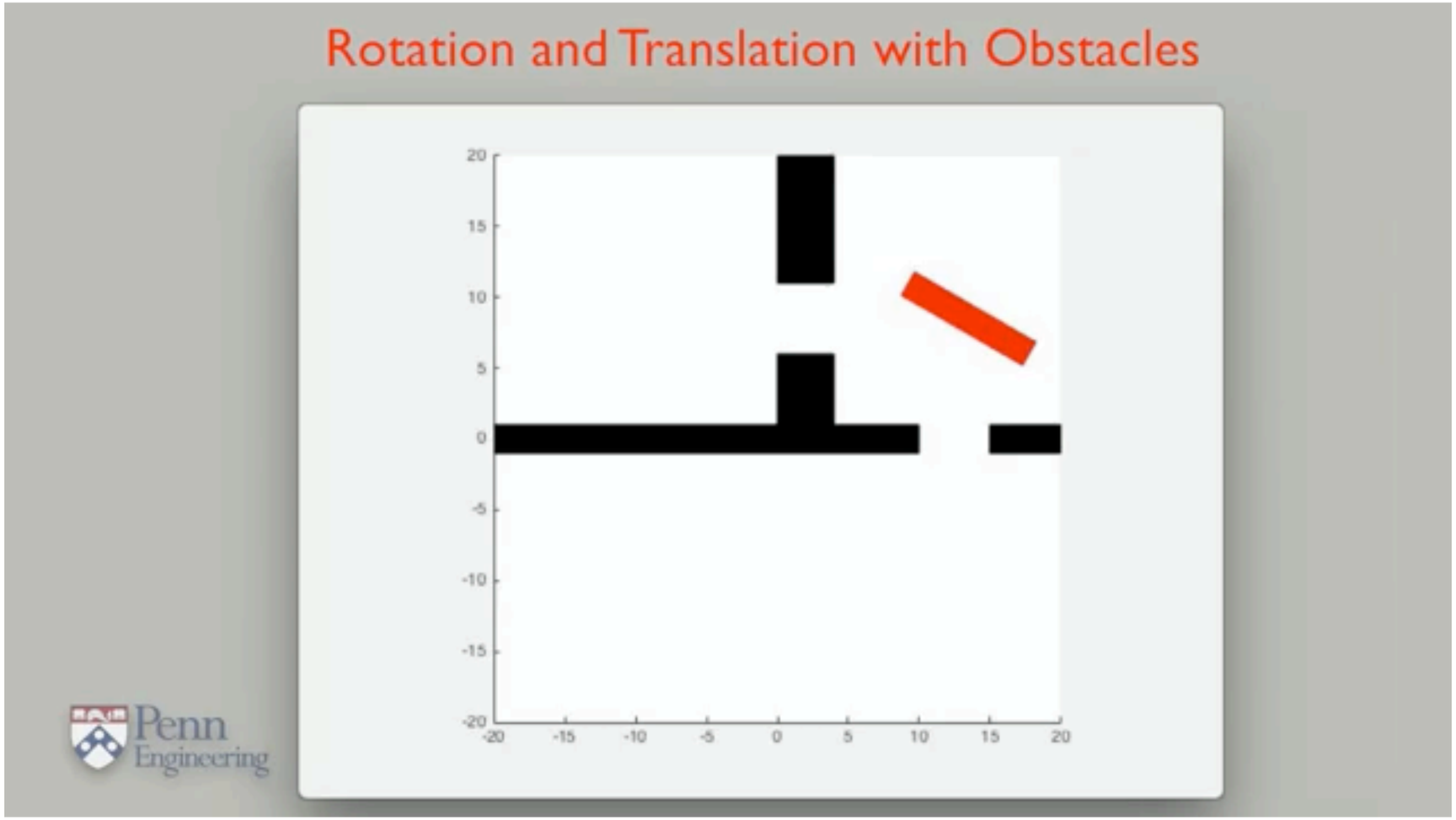
Example 3: 2-link planar arm



Courtesy Tapomayukh Bhattacharya

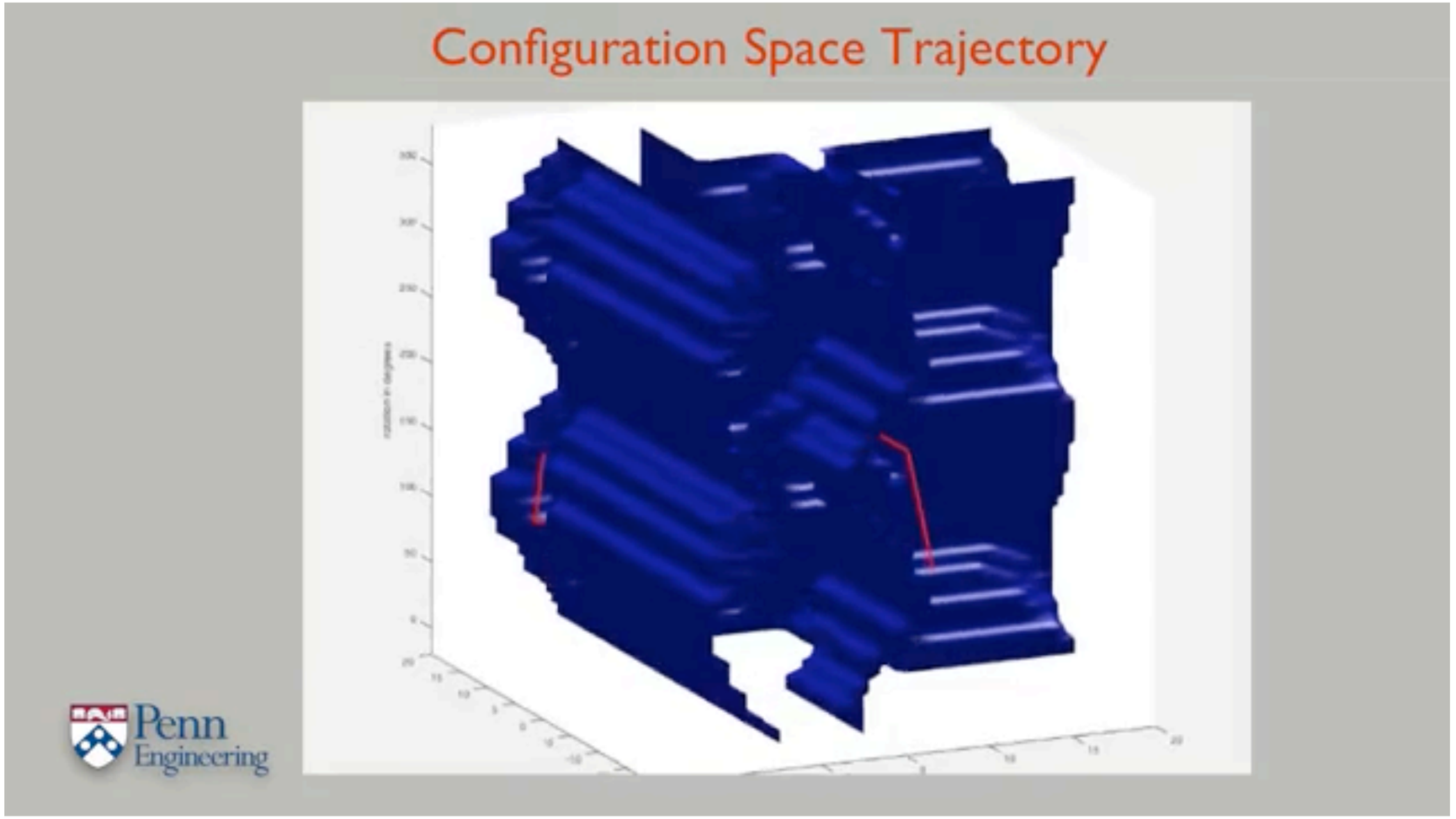
Geometric Path Planning Problem

Planning in Configuration Space



(EXAMPLE FROM CJ TAYLOR)

Planning in Configuration Space



(EXAMPLE FROM CJ TAYLOR)

Geometric Path Planning

WORKSPACE, OBSTACLE

$\mathcal{W}, \mathcal{O} \subset \mathcal{W}$

CONFIGURATIO N SPACE

$\mathcal{C}, \mathcal{C}_{\text{obs}}, \mathcal{C}_{\text{free}}$

START + GOAL CONFIGURATIO

q_s, q_g

$$\xi : [0, 1] \rightarrow \mathcal{C}_{\text{free}}$$

$$\xi(0) = q_s, \xi(1) = q_g$$

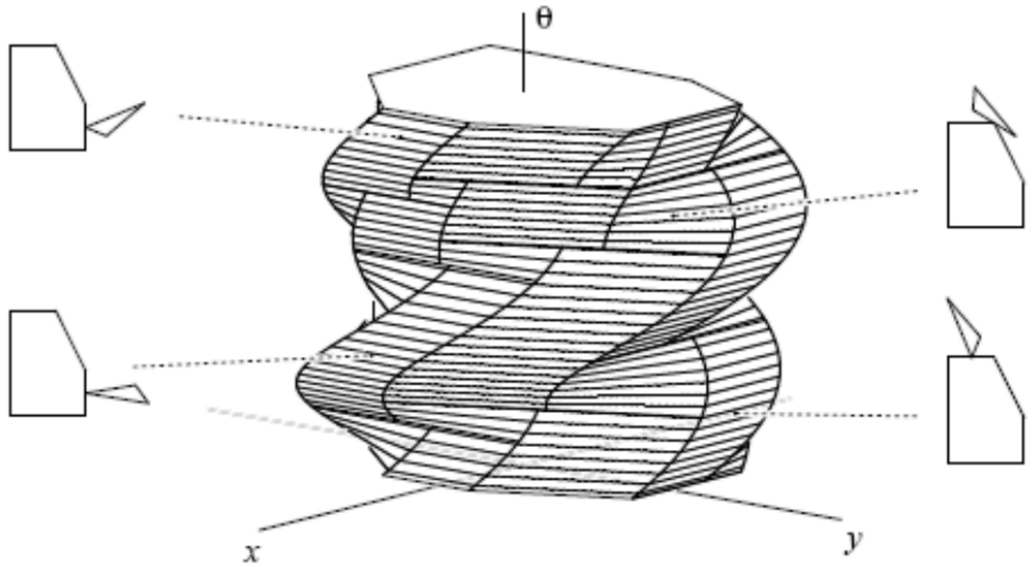
COLLISION-FREE PATH

(WHICH MAY MINIMIZE COST)

Challenges in Motion Planning

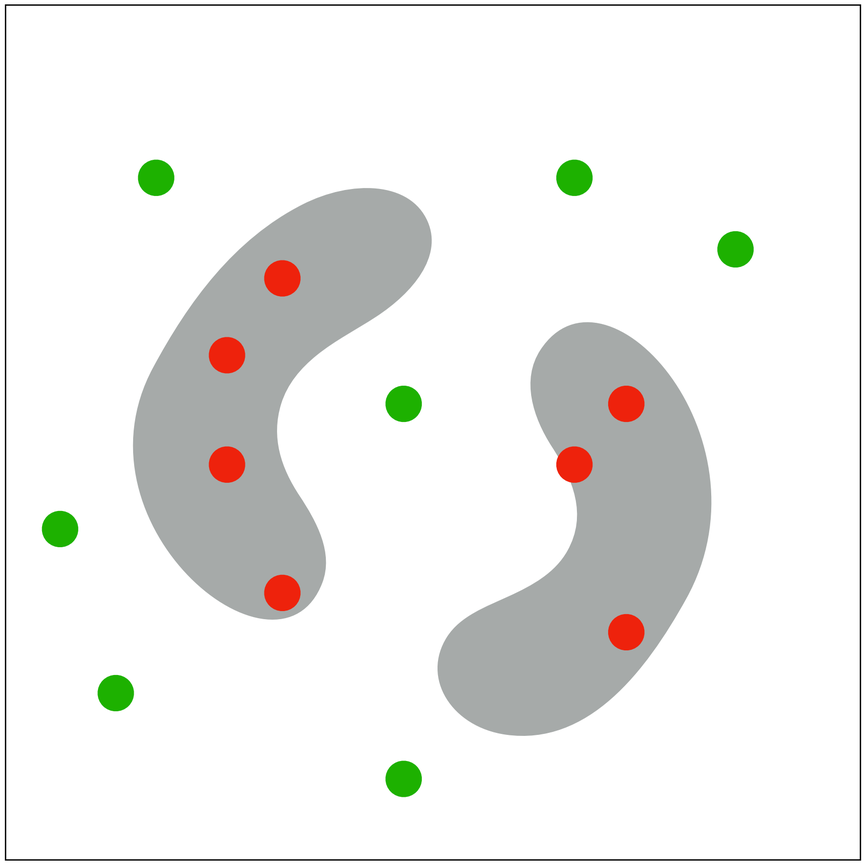
- Computing configuration-space obstacles **HARD!**
- Planning in continuous high-dimensional space **HARD!**

Goal: tractable approximations with provable guarantees!

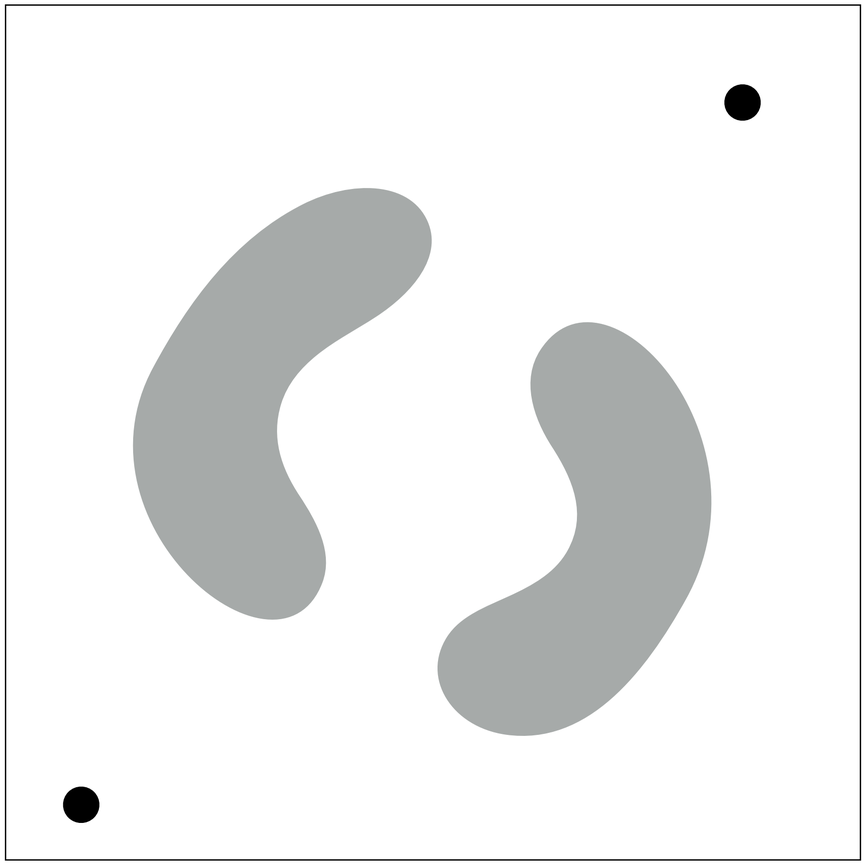


(EXAMPLE FROM HOWIE CHOSET)

Approximate the Configuration Space with Samples



Minimal Cost Path on a Graph

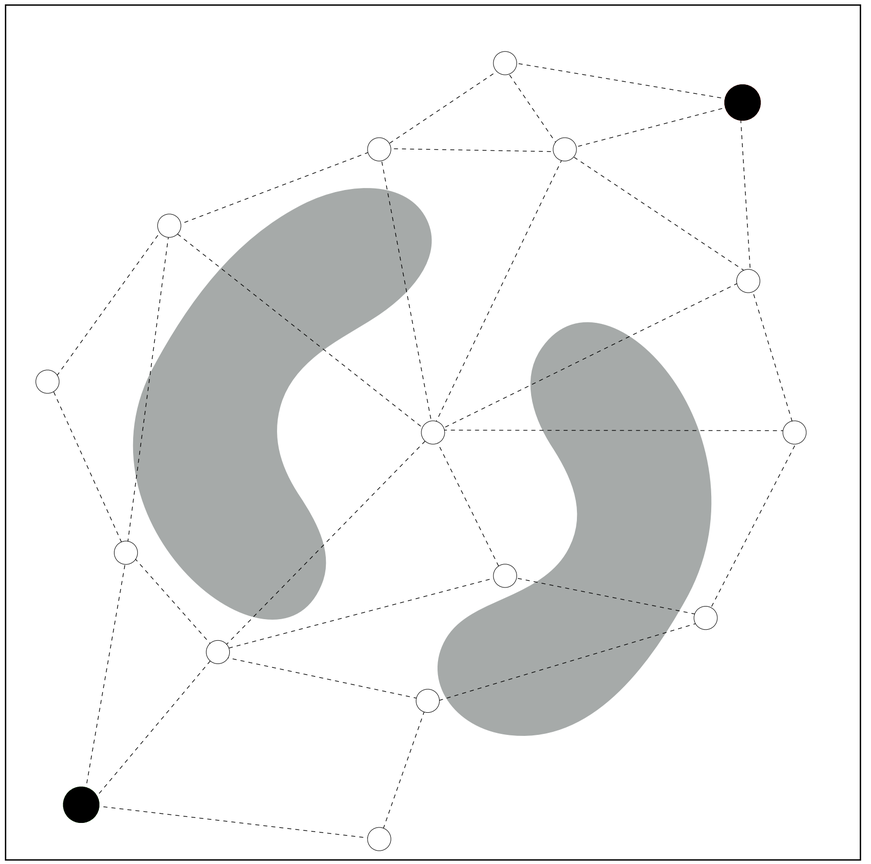


START, GOAL

COST (E.G. LENGTH)

Minimal Cost Path on a Graph

GRAPH IS IMPLICIT

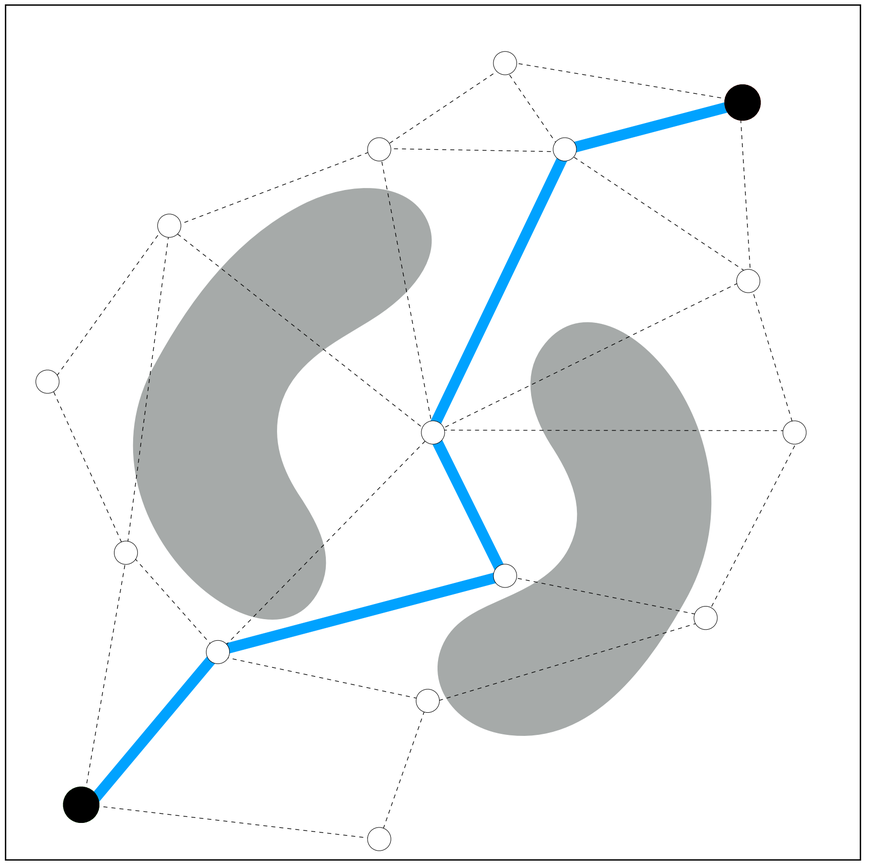


START, GOAL

COST (E.G. LENGTH)

GRAPH (VERTICES, EDGES)

Minimal Cost Path on a Graph



START, GOAL

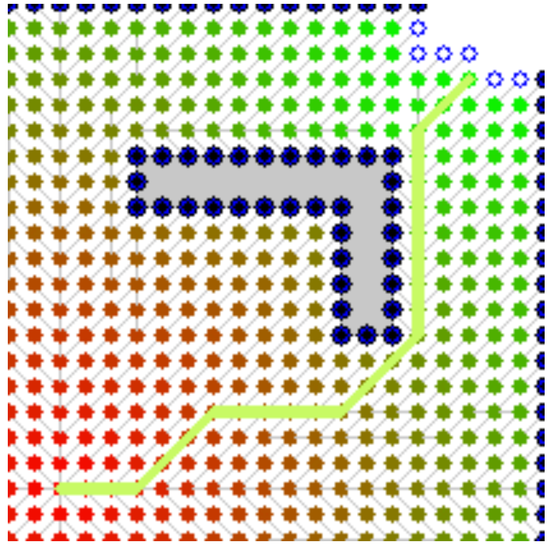
COST (E.G. LENGTH)

GRAPH (VERTICES, EDGES)

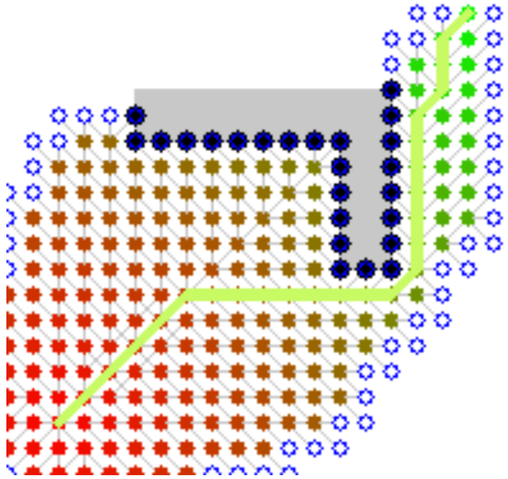
Best-First Search Meta-Algorithm

- Key insight: maintain a **priority queue of promising nodes**, ranked by f(s)
- Initialize queue with start node
- While queue is not empty
  - Pop the most promising node from the queue
  - If it's the goal, compute path by backtracking to the start
    - Return path
  - If it's not the goal, do some bookkeeping, add its successors to the queue
- Return failure

Best-First Search Meta-Algorithm



DIJKSTRA

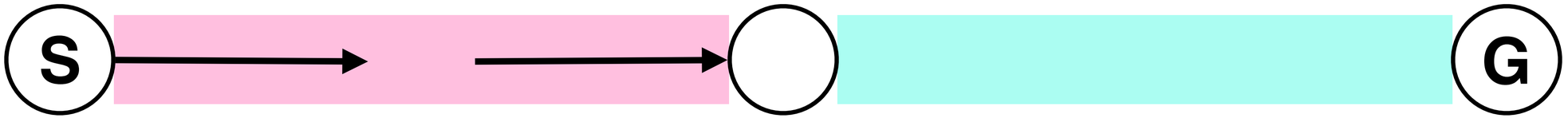


A*

(WIKIPEDIA)

Dijkstra

- Prioritize by cost-to-come



COST-TO-COME
ESTIMATED COST OF
SHORTEST PATH FROM

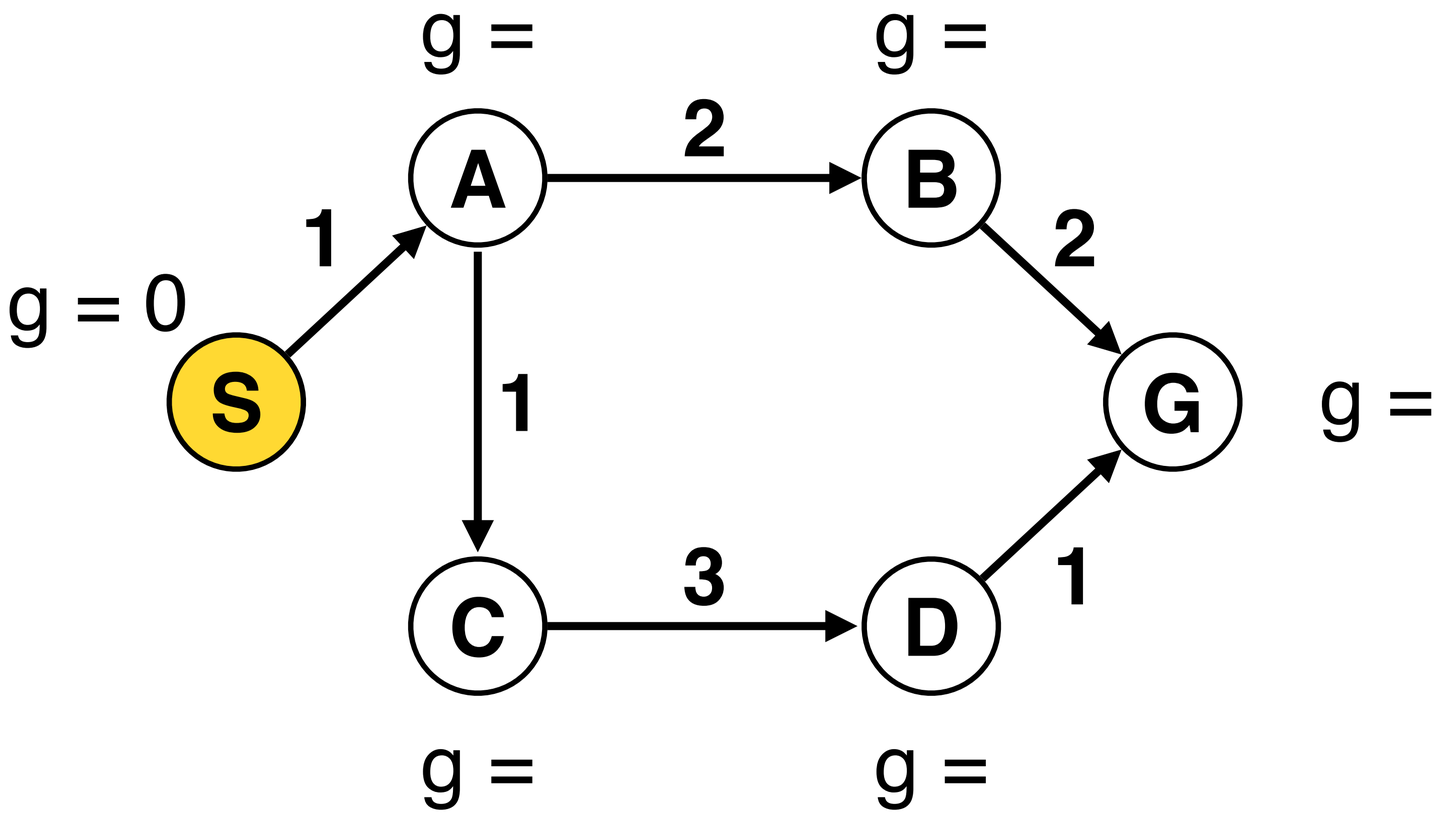
$$f(s) = g(s)$$

Dijkstra's Shortest Path Algorithm

- Best-first search with f(s) = g(s)

OPEN (NODE, F-VALUE)

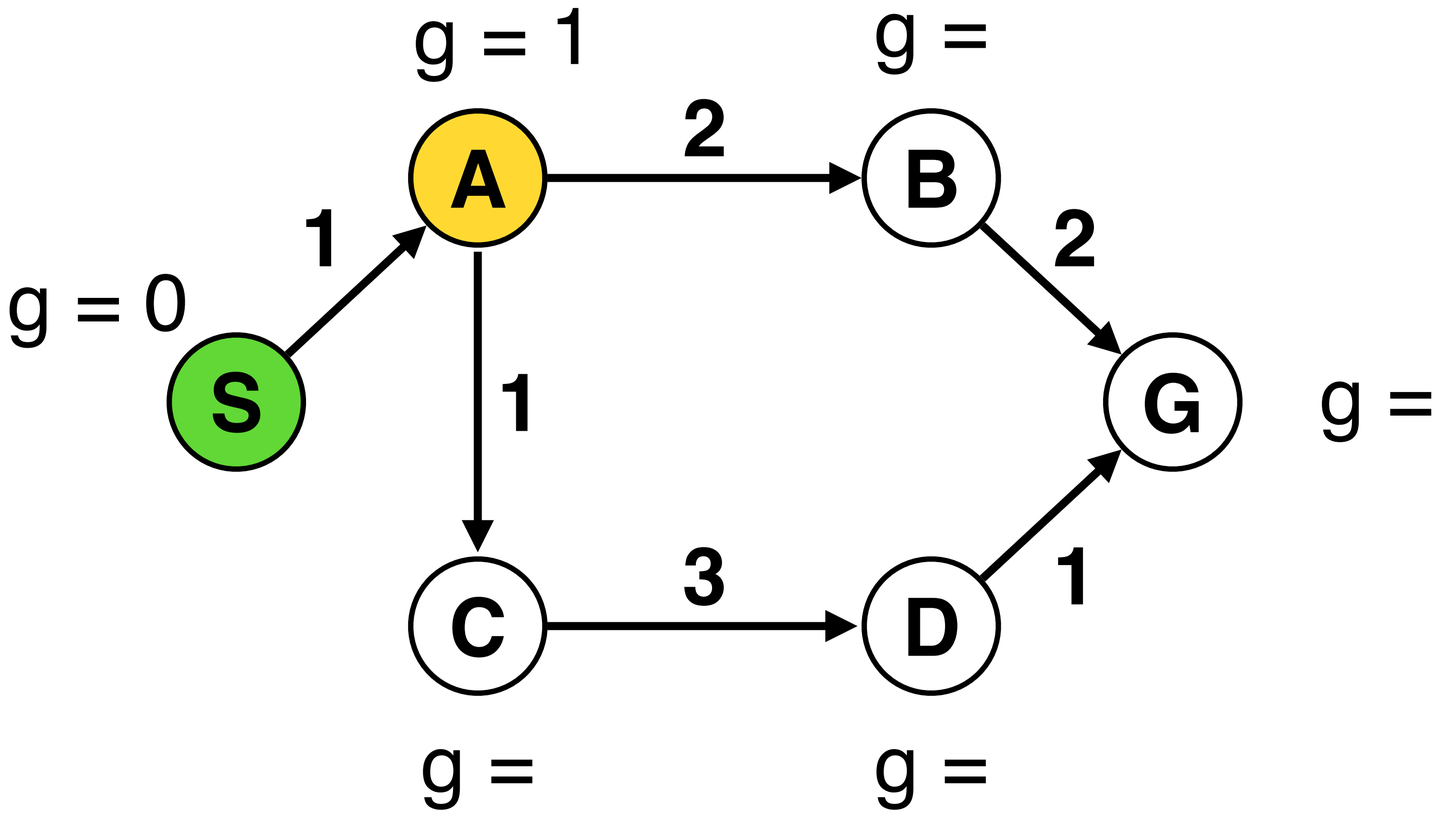
S, 0



CLOSED

Dijkstra's Shortest Path Algorithm

- Best-first search with f(s) = g(s)



OPEN (NODE, F-VALUE)

A, 1

CLOSED

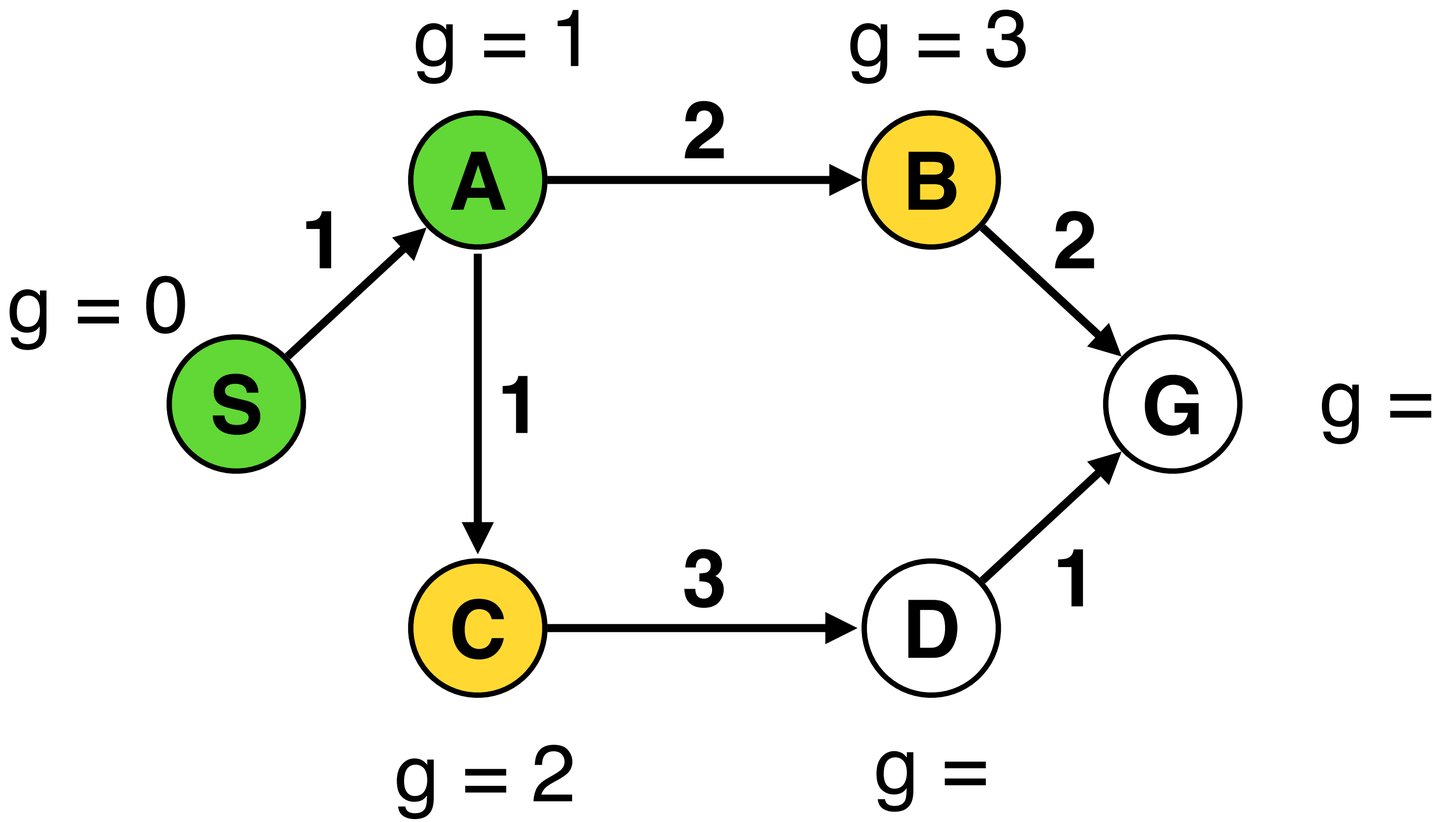
S

Dijkstra's Shortest Path Algorithm

- Best-first search with f(s) = g(s)

OPEN (NODE, F-VALUE)

C, 1+1=2
B, 1+2=3



CLOSED

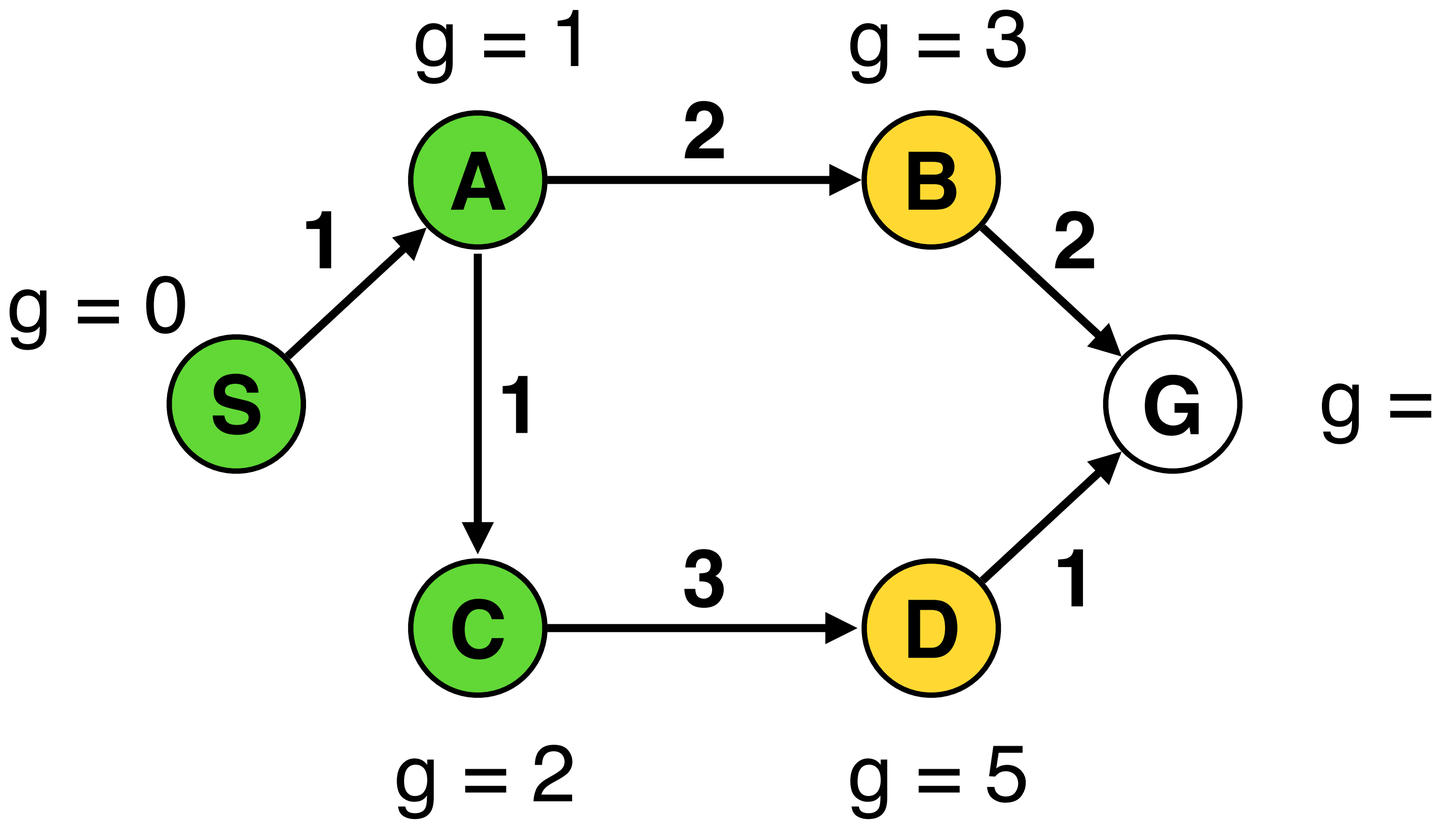
SA

Dijkstra's Shortest Path Algorithm

- Best-first search with f(s) = g(s)

OPEN (NODE, F-VALUE)

B, 1+2=3
D, 2+3=5



CLOSED

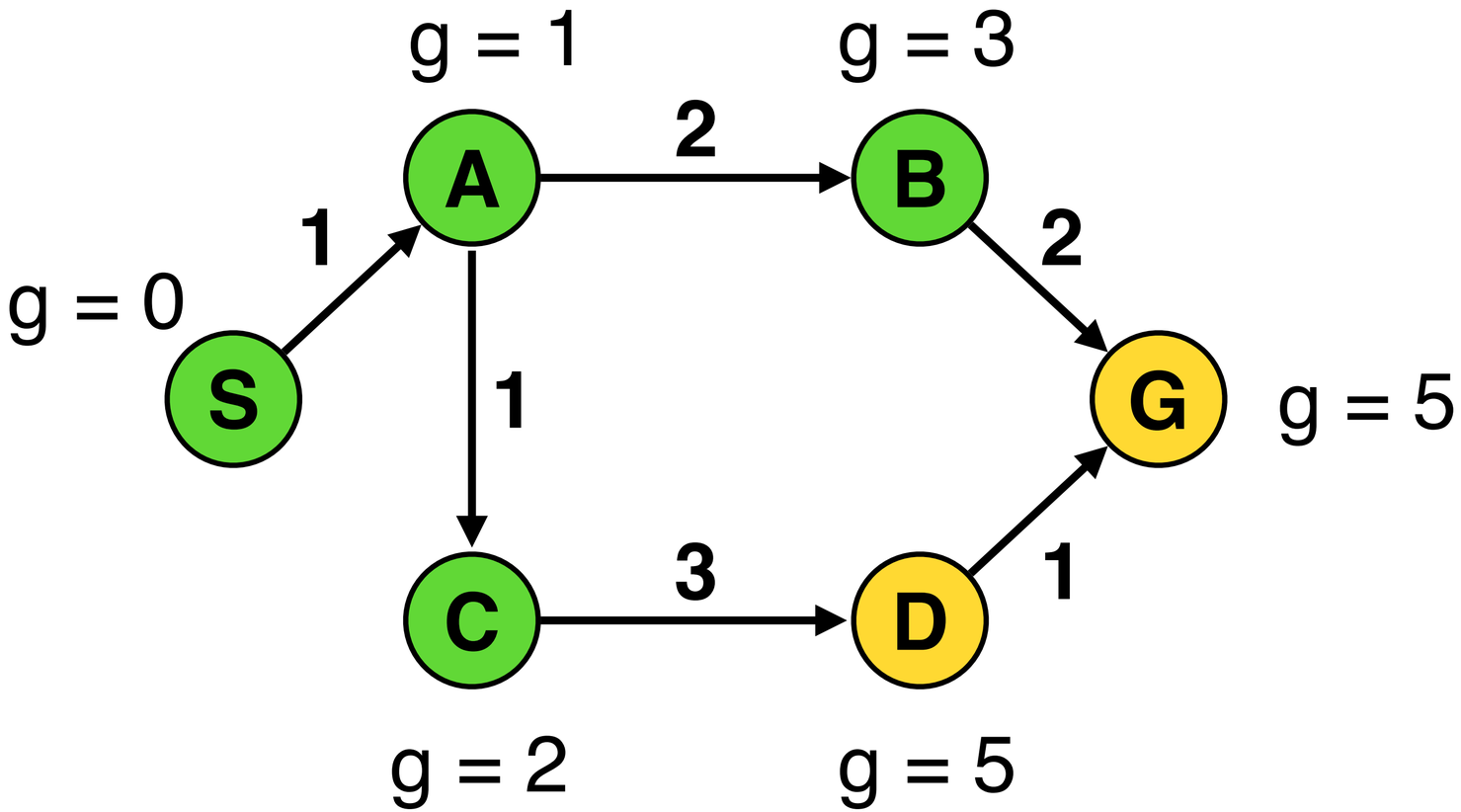
SAC

Dijkstra's Shortest Path Algorithm

- Best-first search with f(s) = g(s)

OPEN (NODE, F-VALUE)

D, 2+3=5
G, 3+2=5



CLOSED

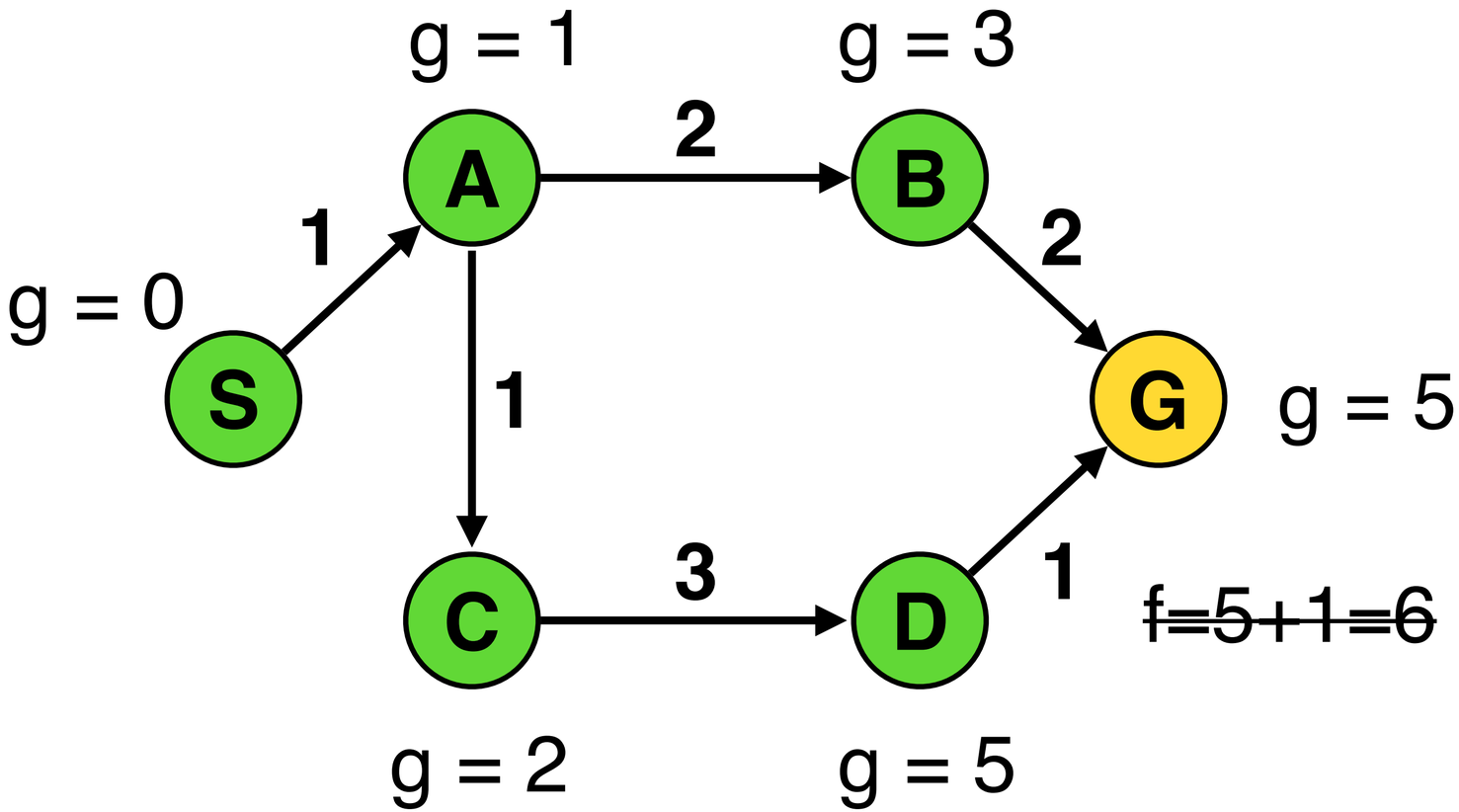
SACB

Dijkstra's Shortest Path Algorithm

- Best-first search with f(s) = g(s)

OPEN (NODE, F-VALUE)

G, 3+2=5



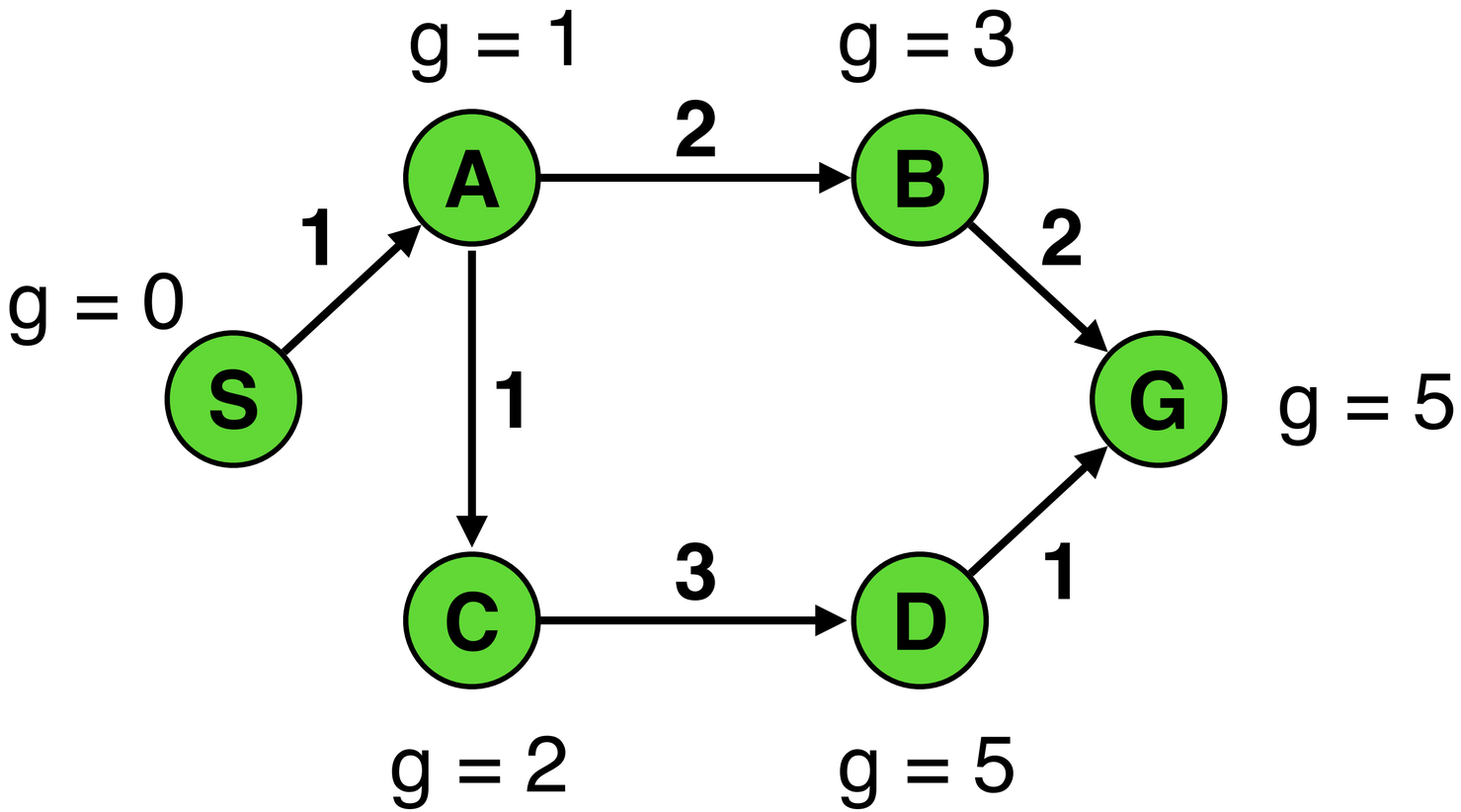
CLOSED

SACBD

Dijkstra's Shortest Path Algorithm

- Best-first search with f(s) = g(s)

OPEN (NODE, F-VALUE)



CLOSED

SACBDG

Best-First Search Implementation

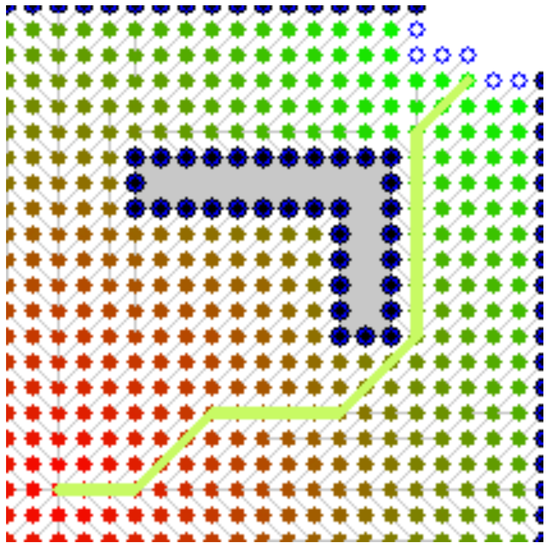
- Inputs: graph **G = (V, E)**; cost **c(s, s') = c(e)**; start and goal
- Data structures maintained
  - **OPEN**: priority queue of nodes that may be expanded (with priority f)
  - **CLOSED**: set of nodes that have been expanded
  - **g(s)**: estimated minimum cost from start to node s ("cost-to-come")

Best-First Search Implementation

- Initialize **g(start) = 0** and all other g-values to infinity
- Insert **start** into **OPEN**
- While **goal** not in **CLOSED**
  - Remove **s** with smallest f(s) from **OPEN**
  - Add **s** to **CLOSED**
  - For every neighbor **s'** that isn't **CLOSED**
    - If **g(s) + c(s, s') < g(s')**, update **g(s')** and add **s'** to **OPEN** (with parent **s**)

Dijkstra’s Shortest Path Algorithm

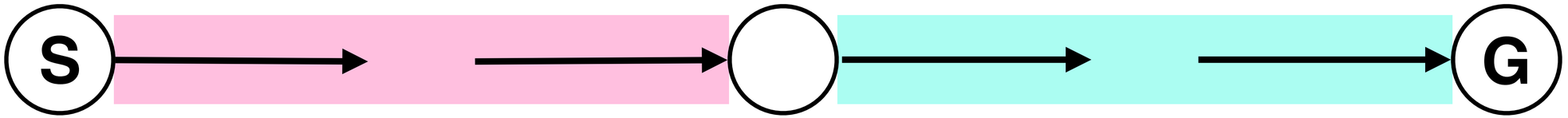
- Best-first search with f(s) = g(s)
- Only expands nodes with lower cost-to-come than goal!



(WIKIPEDIA)

Heuristics

- What if we had a **heuristic** h(s) that estimated the cost from s to the goal?



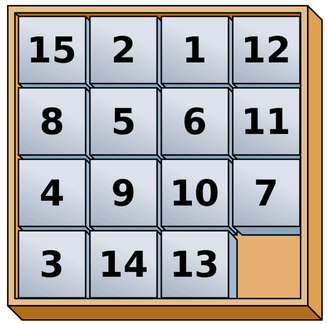
COST-TO-COME
ESTIMATED COST OF
SHORTEST PATH FROM

COST-TO-GO
ESTIMATED COST OF
SHORTEST PATH TO

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

Where Do Heuristics Come From?

- Compute the exact cost-to-go in a relaxed problem
  - If cost is path length, the Euclidean distance to the goal is a heuristic (assumes no more obstacles)
- Other possible sources of heuristics: domain knowledge, learning, etc.
  - Sliding tile puzzle: number of tiles out of place



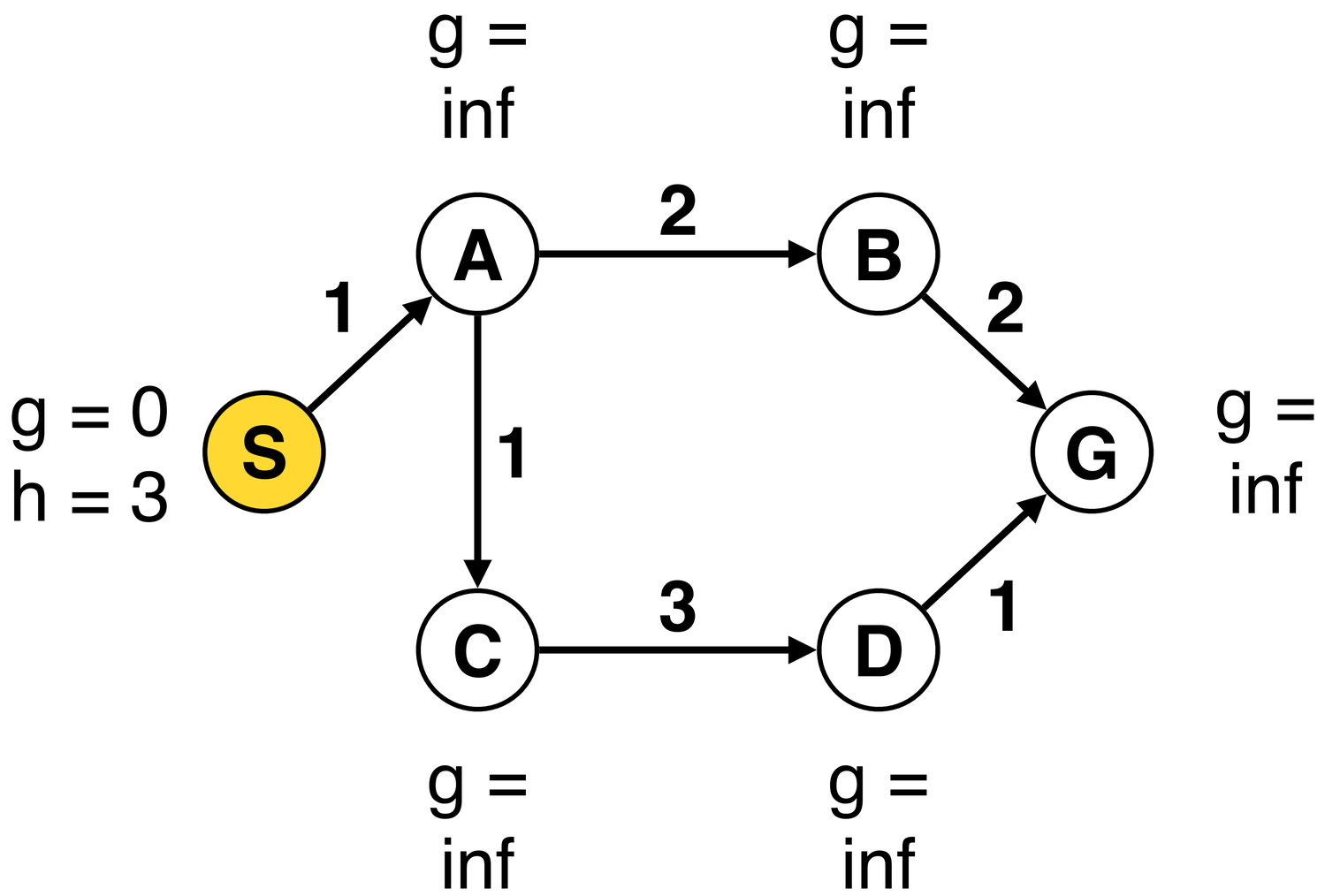
(WIKIPEDIA)

A* Search

- Best-first search with f(s) = g(s) + h(s)
- Combines cost-to-come with estimated cost-to-go
- Only expands nodes with lower f-value than goal!
- Minimizes number of nodes expanded
  (under certain conditions on the heuristic function)

HART ET AL., 1968

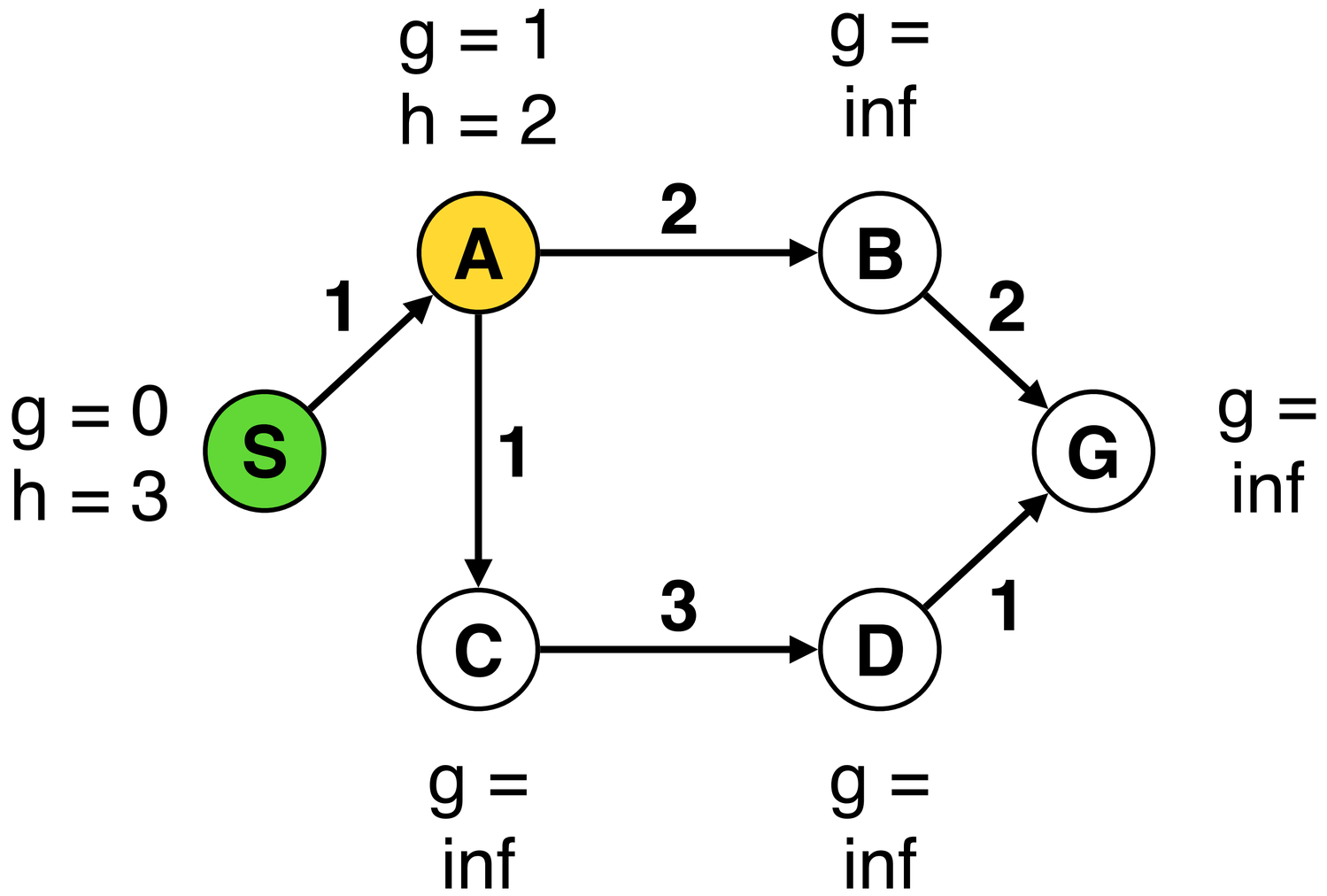
(COMPUTER HISTORY MUSEUM)



OPEN (NODE, F-VALUE)

S, 0 + 3 = 3

CLOSED

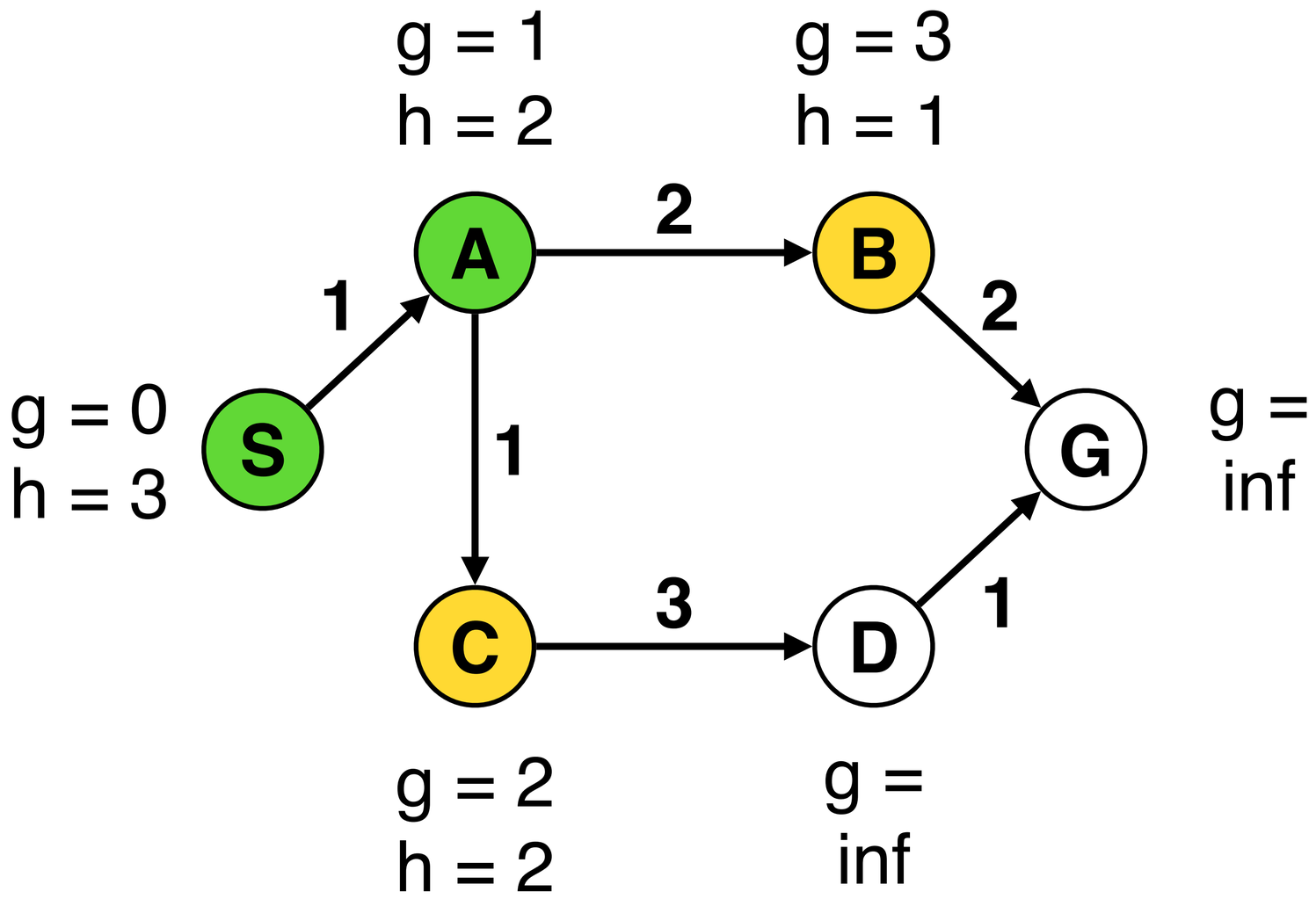


OPEN (NODE, F-VALUE)

A, 1 + 2 = 3

CLOSED

S



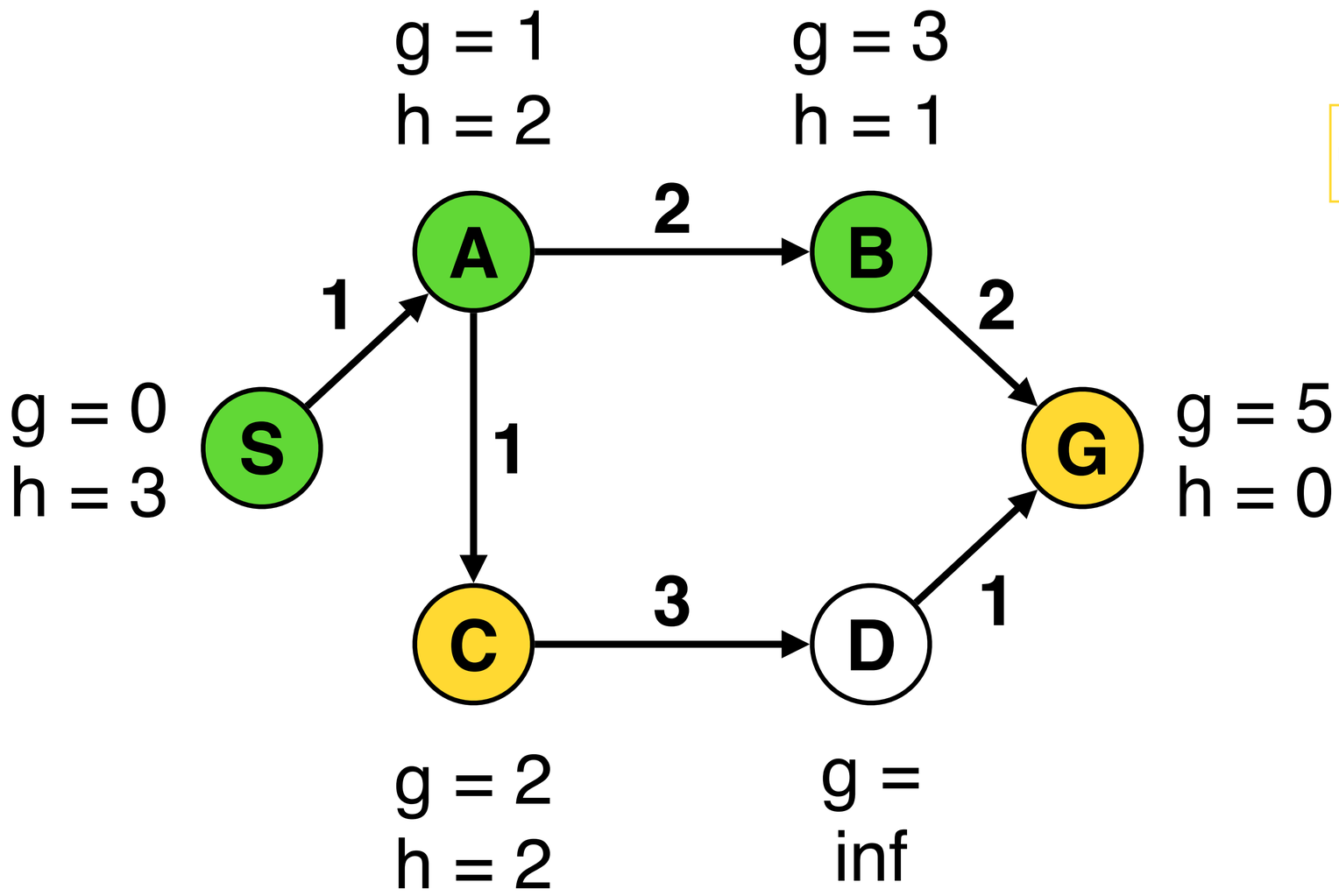
OPEN (NODE, F-VALUE)

B, 3 + 1 = 4

C, 2 + 2 = 4

CLOSED

SA



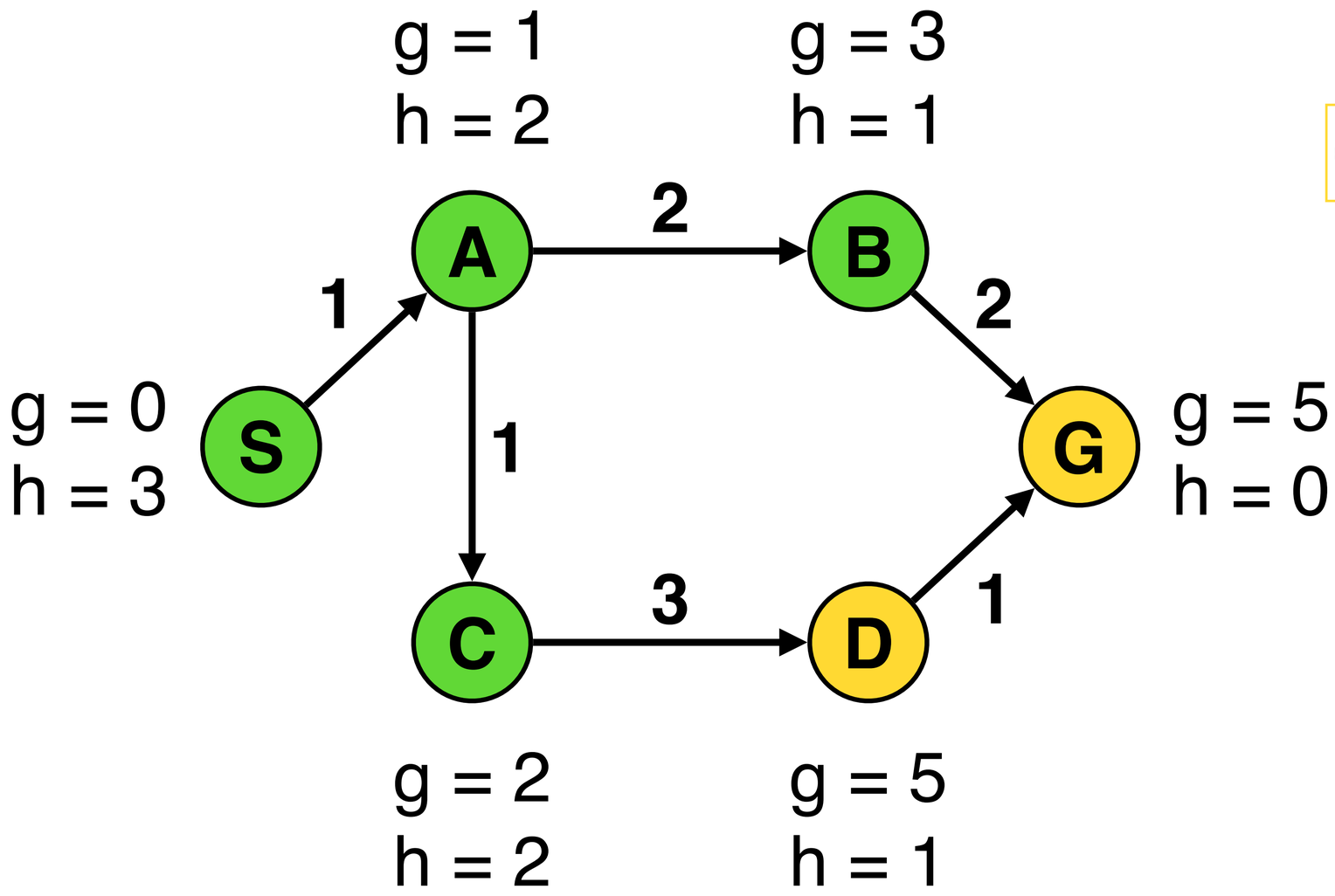
OPEN (NODE, F-VALUE)

C, 2 + 2 = 4

G, 5 + 0 = 5

CLOSED

SAB



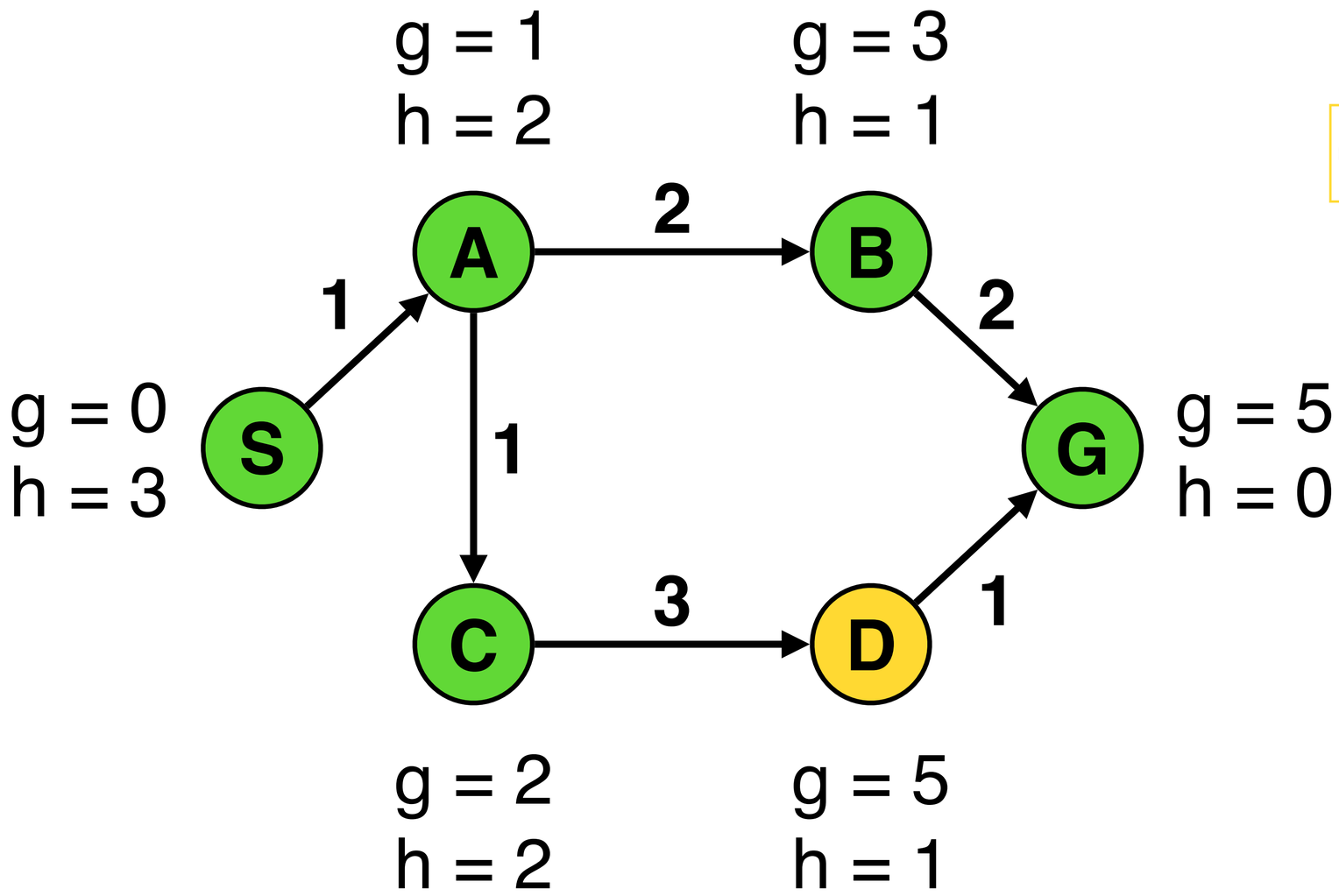
OPEN (NODE, F-VALUE)

G, 5 + 0 = 5

D, 5 + 1 = 6

CLOSED

SABC



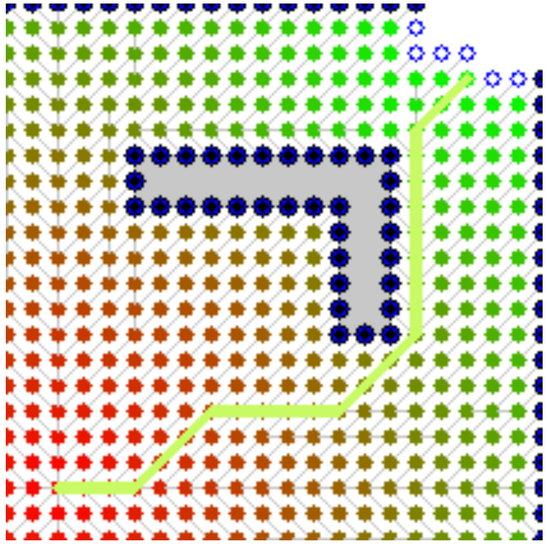
OPEN (NODE, F-VALUE)

D, 5 + 1 = 6

CLOSED

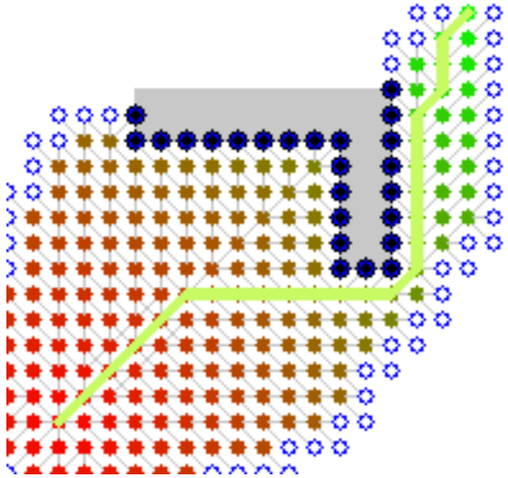
SABCG

Best-First Search



DIJKSTRA

Expands nodes where
$f(s) = g(s) < L$



A*

Expands nodes where
$f(s) = g(s) + h(s) < L$

(WIKIPEDIA)

Properties of Heuristics

Admissible: underestimates cost to the goal, i.e. $h(s) \leq h^*(s)$

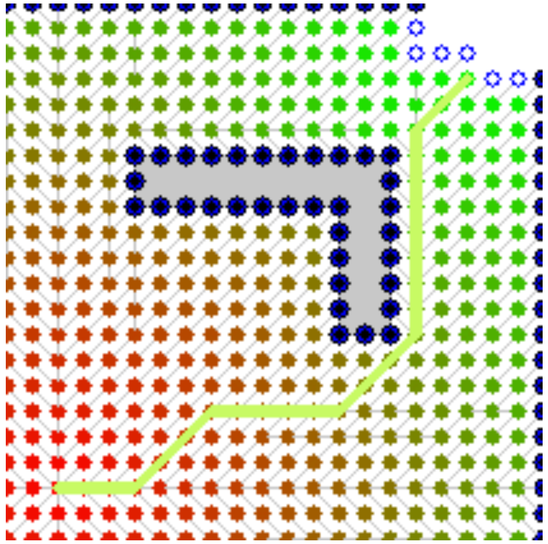
A* with admissible heuristic returns optimal path

Consistent: satisfies triangle inequality, i.e. $h(s) \leq c(s, s') + h(s')$

A* with consistent heuristic returns optimal path
and is efficient (will not re-expand nodes)

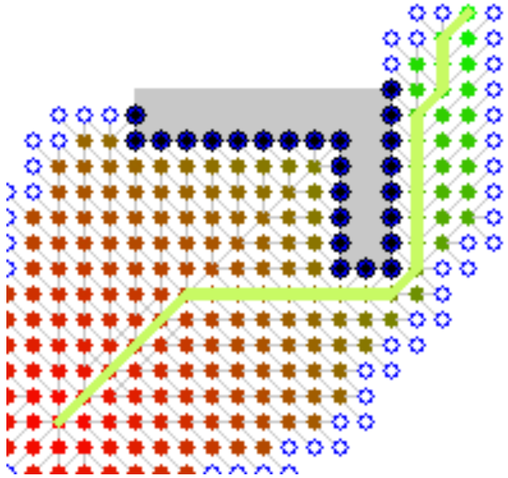
All consistent heuristics are admissible (not vice versa).

Planning with Heuristics



DIJKSTRA

Expands nodes where
$f(s) = g(s) < L$

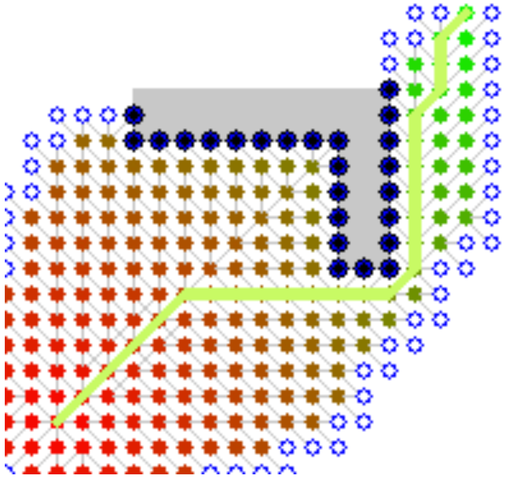


A*

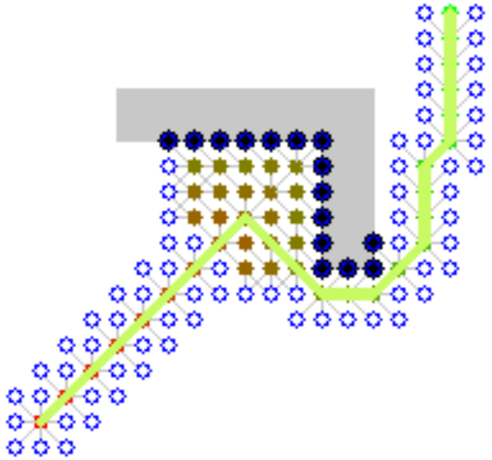
Expands nodes where
$f(s) = g(s) + h(s) < L$

(WIKIPEDIA)

Planning with Inadmissible Heuristics



ADMISSIBLE



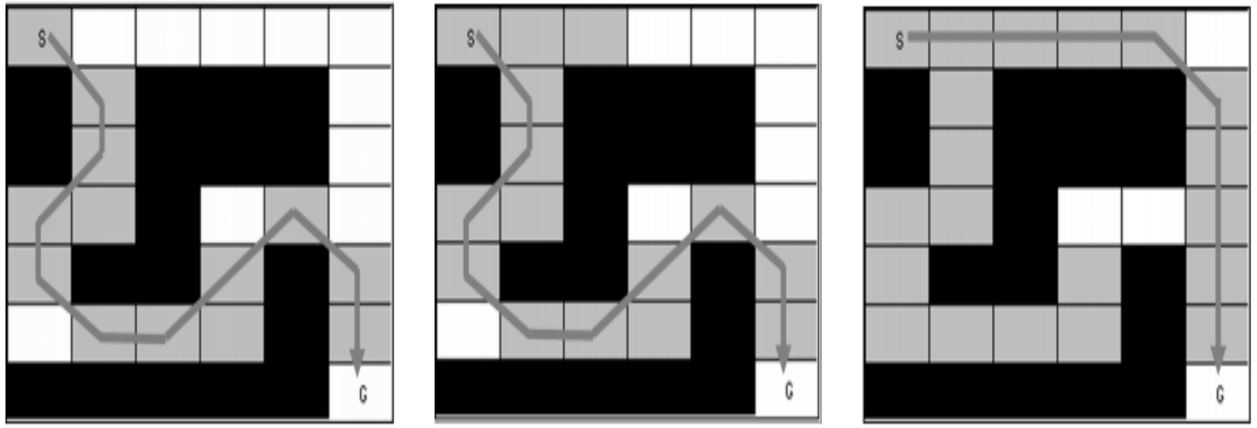
INADMISSIBLE

Can trade off solution quality for number of nodes expanded!

(WIKIPEDIA)

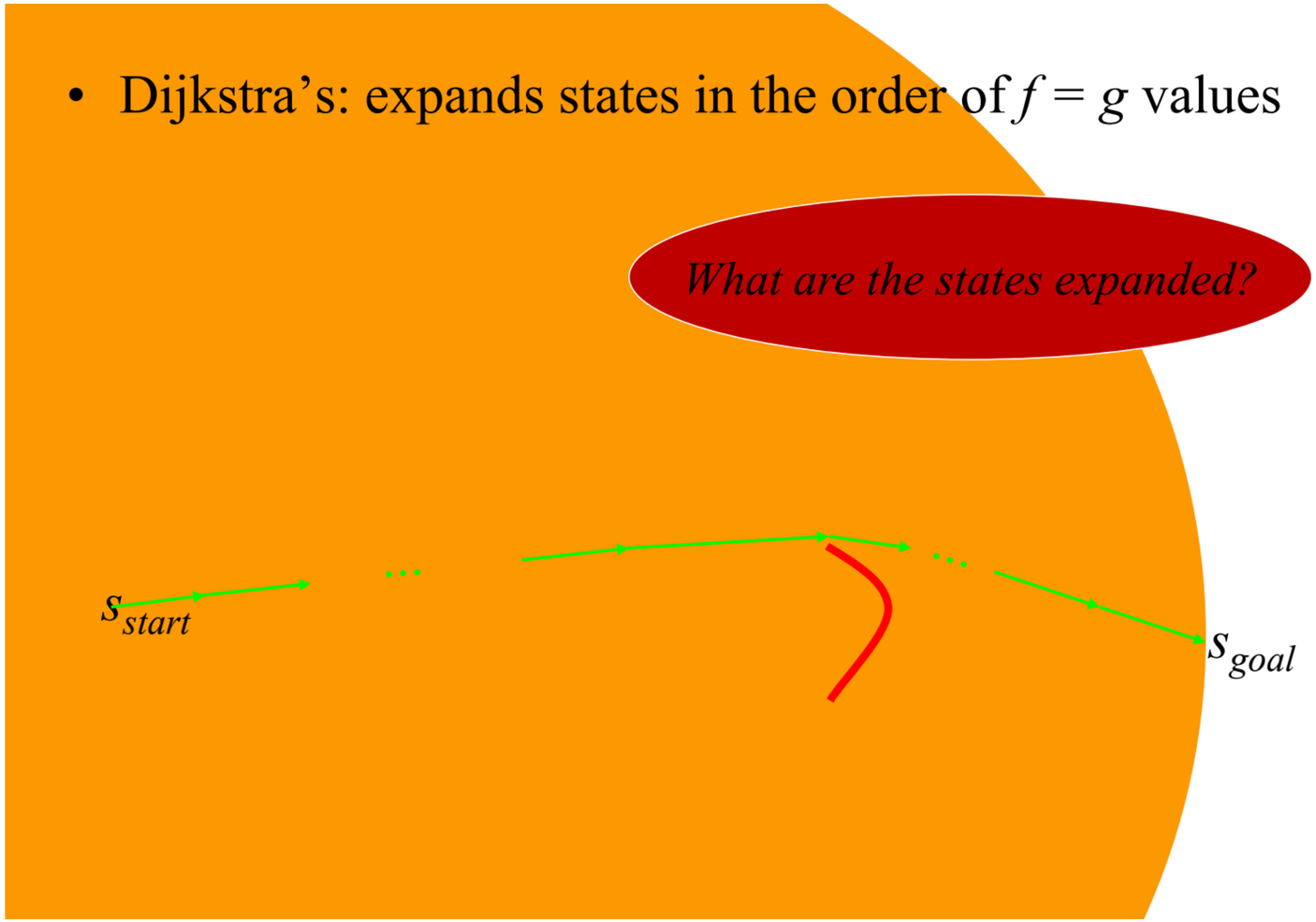
Weighted A* is Best-First Search

- Dijkstra's shortest path algorithm expands nodes in order of f = g
- A* search expands nodes in order of f = g + h
- **Weighted A*** search expands nodes in order of f = g + εh
  - ε > 1 biases toward nodes that are closer to goal!
  - Solution is always ε-optimal (cost is at most ε times larger)



Left to right:
ε = 2.5,
ε = 1.5,
ε = 1.0 (optimal)

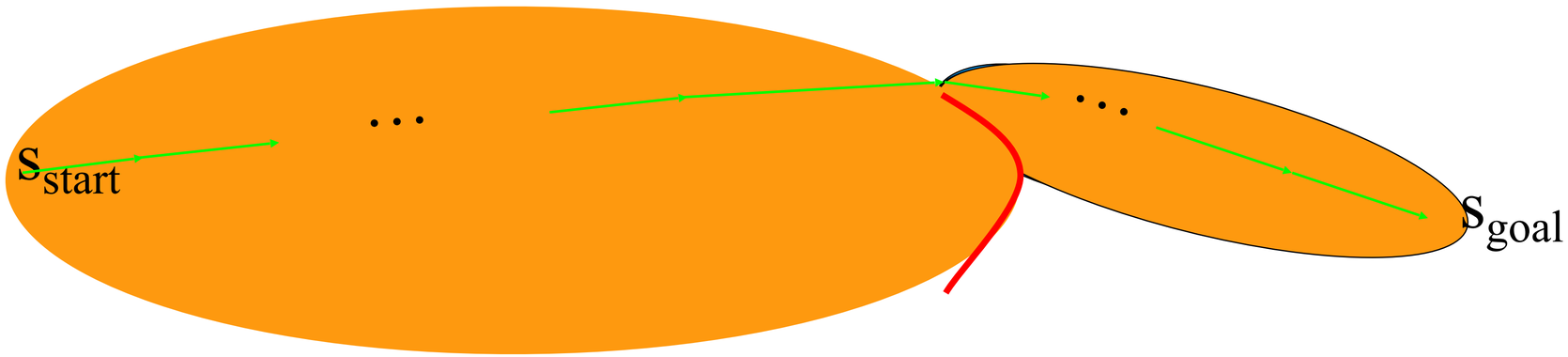
(EXAMPLE FROM MAX LIKHACHEV)



Courtesy Max Likhachev

Effect of the Heuristic Function

A* Search: expands states in the order of $f = g+h$ values

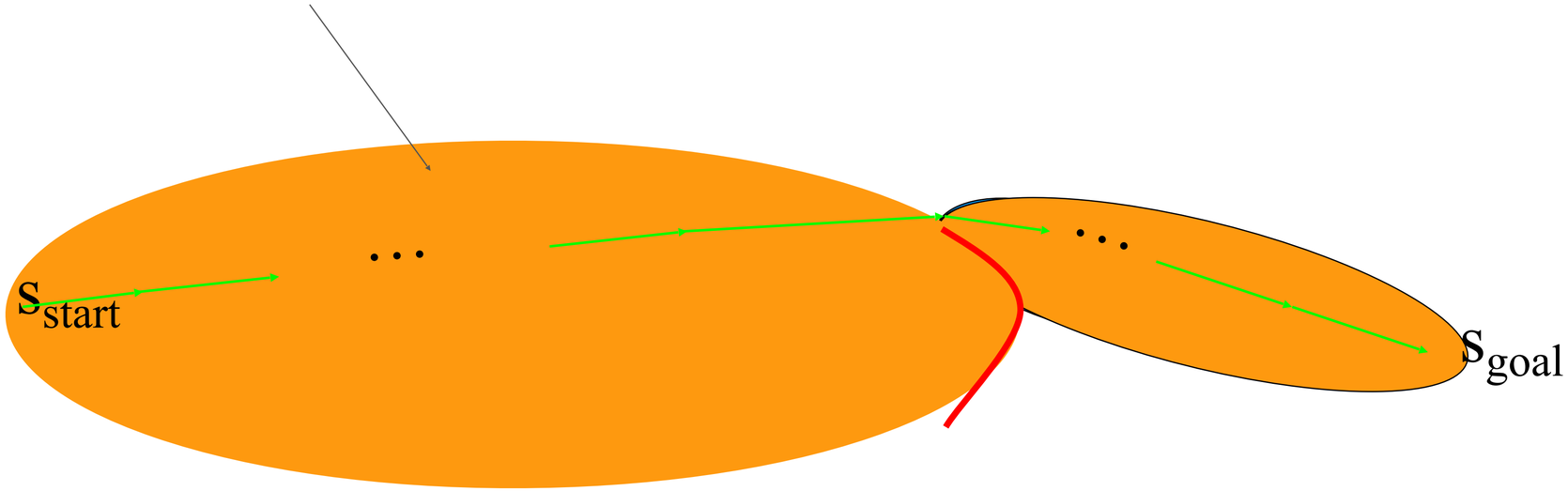


Courtesy Max Likhachev

Effect of the Heuristic Function

A* Search: expands states in the order of $f = g+h$ values

for large problems this results in A* quickly running out of memory (memory: O(n))

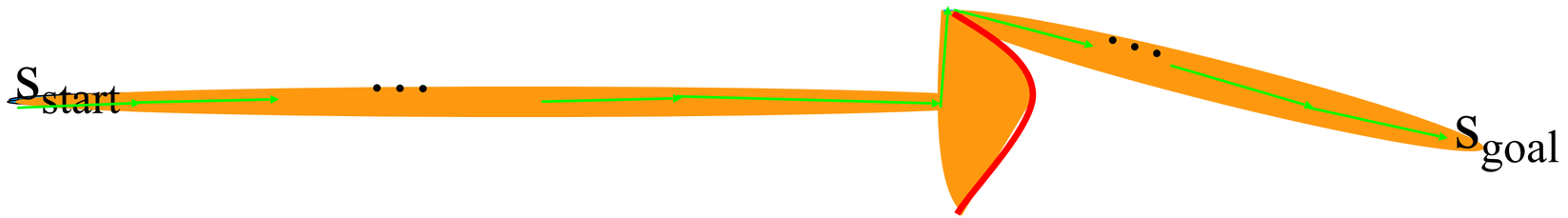


Courtesy Max Likhachev

Effect of the Heuristic Function

Weighted A* Search: expands states in the order of $f = g+\varepsilon h$ values, $\varepsilon > 1$ = bias towards states that are closer to goal

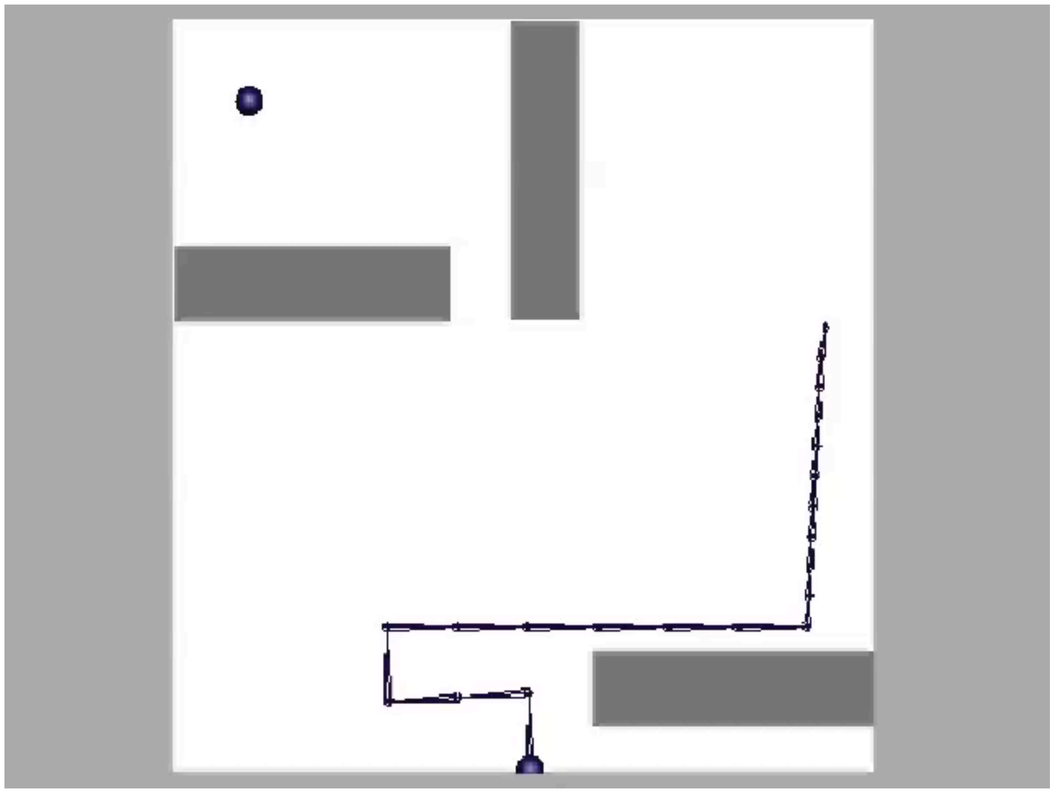
solution is always ε-suboptimal:
cost(solution) ≤ ε·cost(optimal solution)



Courtesy Max Likhachev

Heuristic Search on Huge Graphs

- Biasing toward the goal can dramatically reduce number of nodes expanded
- Can plan with much larger graphs
- This example: simulated 20 DOF robotic arm, with over 10^{26} nodes
  - Uses ARA* to plan, which is an anytime version of weighted A*
  - Plan with large ε, shrink ε toward 1, repeat



(EXAMPLE FROM MAX LIKHACHEV)

Summary

- Best-first search expands nodes in order of their f-values
- Heuristics guide best-first search by estimating the cost-to-go from any node
- A* with an admissible heuristic returns the optimal path
- Weighted A* turns an admissible heuristic into an inadmissible heuristic by scaling with a factor $\varepsilon > 1$, biasing best-first search toward the goal
  - Solution cost is at most ε times larger than optimal solution cost
  - Trades path quality for computation (both time and memory constraints)