



上海科技大学  
ShanghaiTech University

## CS283: Robotics Spring 2026: Planning

---

Sören Schwertfeger / 师泽仁

ShanghaiTech University

# ADMIN

---

# HW3 is out

- Start early ...
- Use off the shelf SLAM
- Implement ICP by hand
- Evaluate the results!

# Project

- May 20<sup>th</sup>, 23:59: due date for the intermediate project report
- June 22<sup>nd</sup>: due date for the final demo. (You are welcome to do it earlier!)
- June 22<sup>nd</sup>, 23:59: due date for the final report. (You are welcome to finish it earlier!)
- June 24<sup>th</sup>, 23:59: due date for the webpage. (You are welcome to finish it earlier!)
  
- Project Meetings:
  - Optional: meetings before that to discuss the progress – contact Prof. Schwertfeger
  - May 21 and 26: meetings to discuss the intermediate report

# Project Safety!

- You work with real robotic hardware => can be dangerous!
- Don't stick your fingers where they don't belong!
- Be aware of the moving robot!
- Be careful when handling/ charging the batteries! If you are unsure, ask one of my students for help!
- Be careful with electric components.
- Be nice to the robot!

# PATH PLANNING

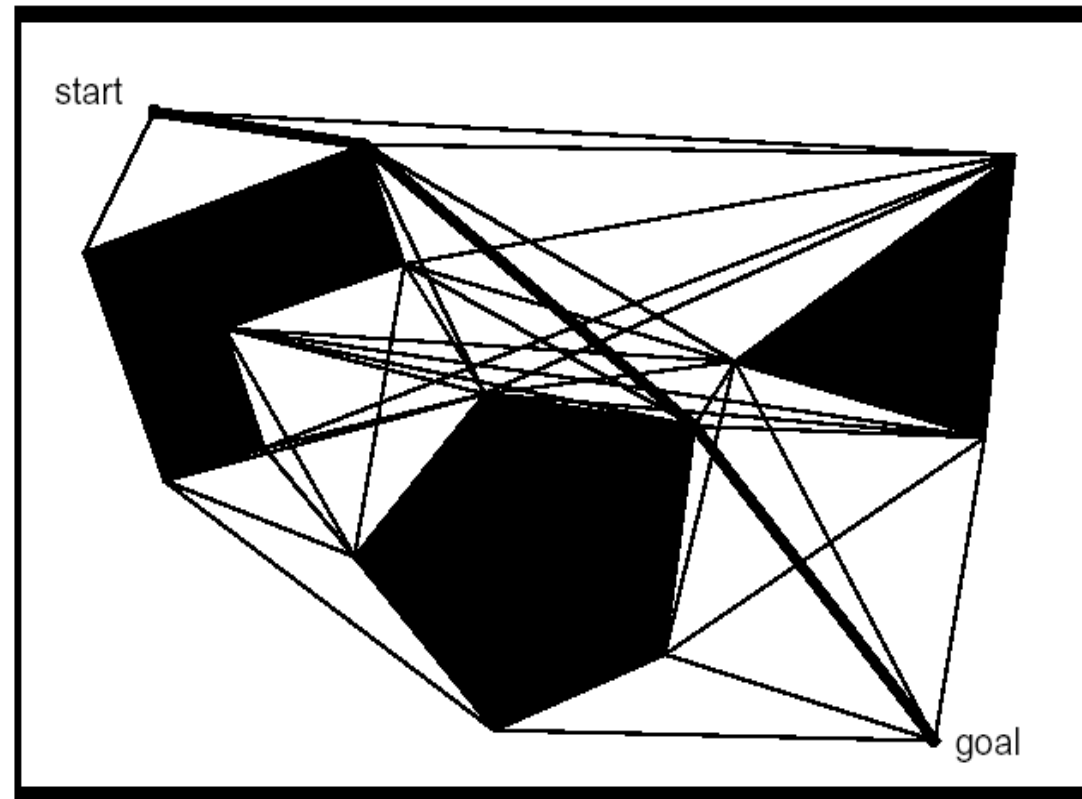
---



# Graph Search

- Overview
  - Solves a least cost problem between two states on a (directed) graph
  - Graph structure is a discrete representation
- Limitations
  - State space is discretized → completeness is at stake
  - Feasibility of paths is often not inherently encoded
- Algorithms
  - (Preprocessing steps)
  - Breath first
  - Depth first
  - Dijkstra
  - A\* and variants
  - D\* and variants

# Graph Construction: Visibility Graph



- Particularly suitable for polygon-like obstacles
- Shortest path length
- Grow obstacles to avoid collisions

# Graph Construction: Visibility Graph

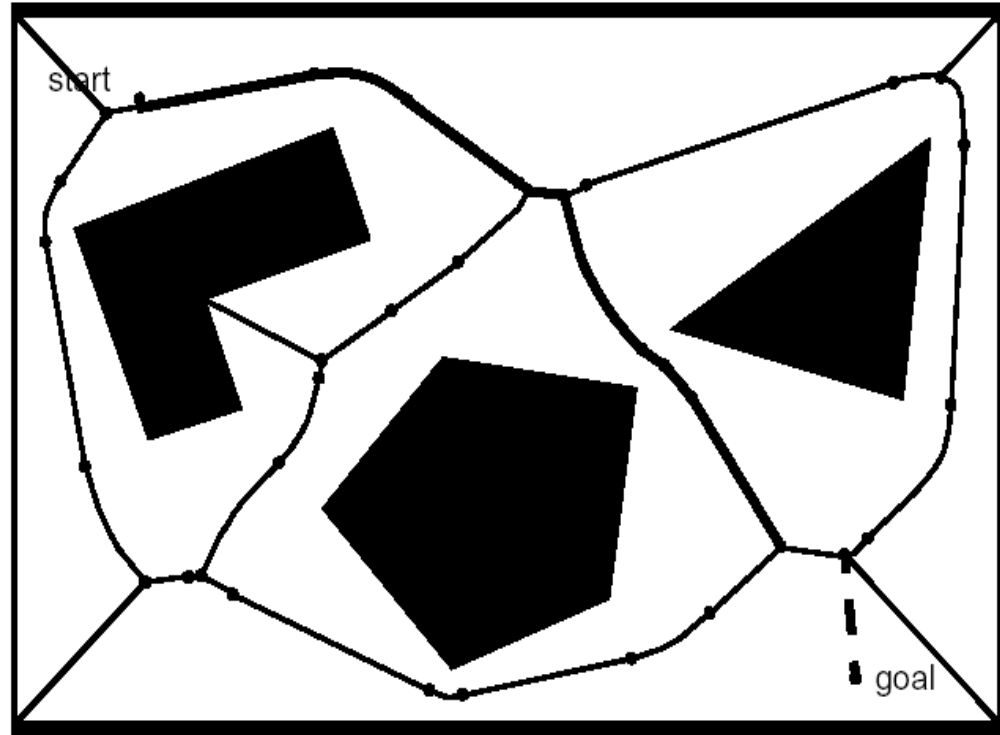
- Pros

- The found path is optimal because it is the shortest length path
- Implementation simple when obstacles are polygons

- Cons

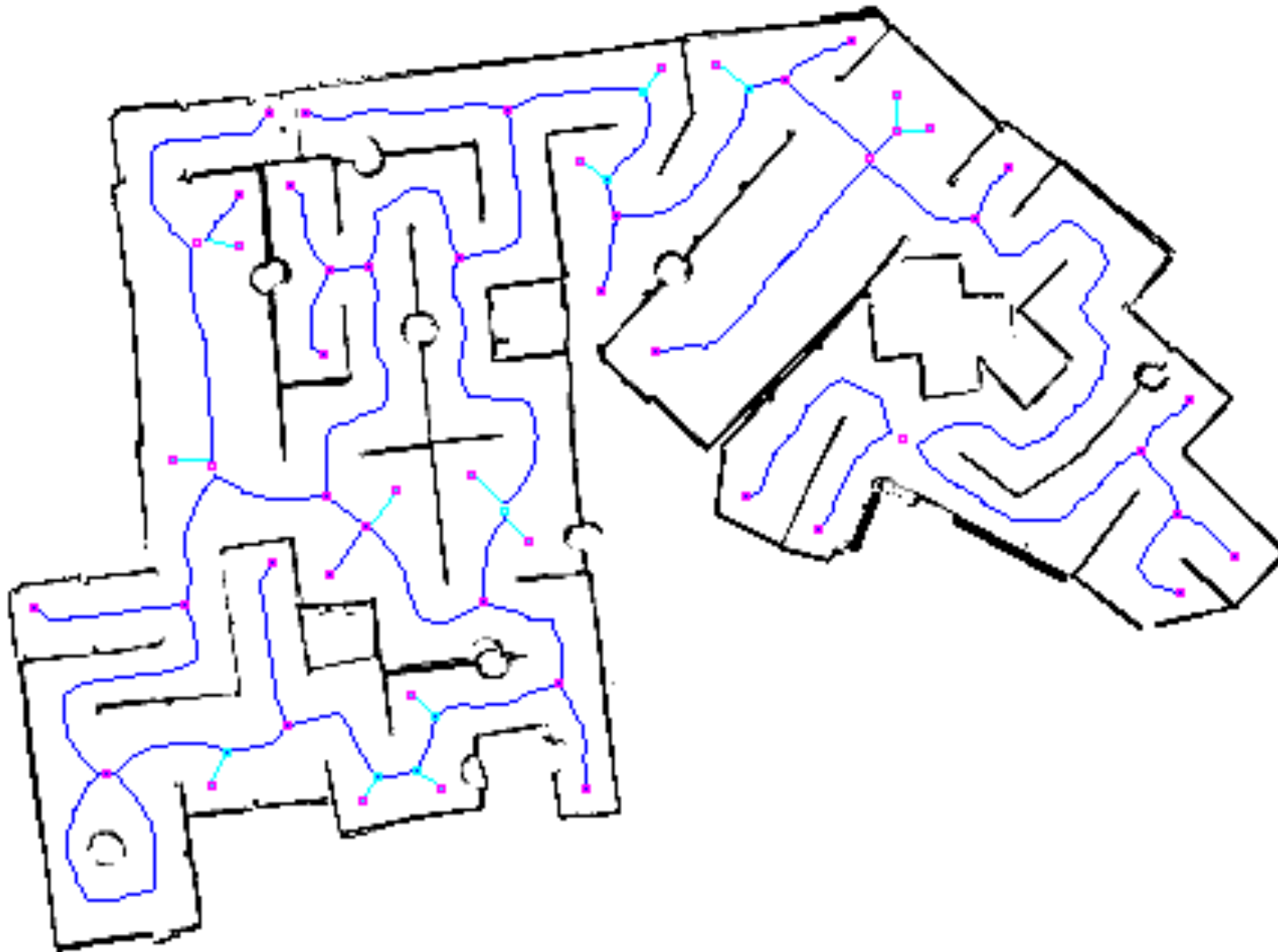
- The solution path found by the visibility graph tend to take the robot as close as possible to the obstacles: the common solution is to grow obstacles by more than robot's radius
- Number of edges and nodes increases with the number of polygons
- Thus it can be inefficient in densely populated environments

# Graph Construction: Voronoi Diagram



- Tends to maximize the distance between robot and obstacles

# Topology Graph



# Graph Construction: Voronoi Diagram

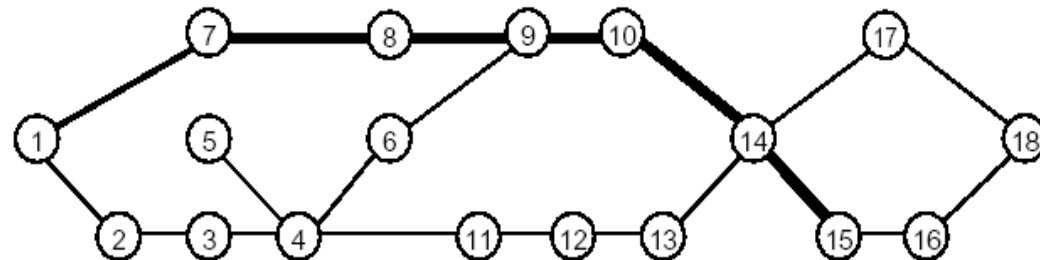
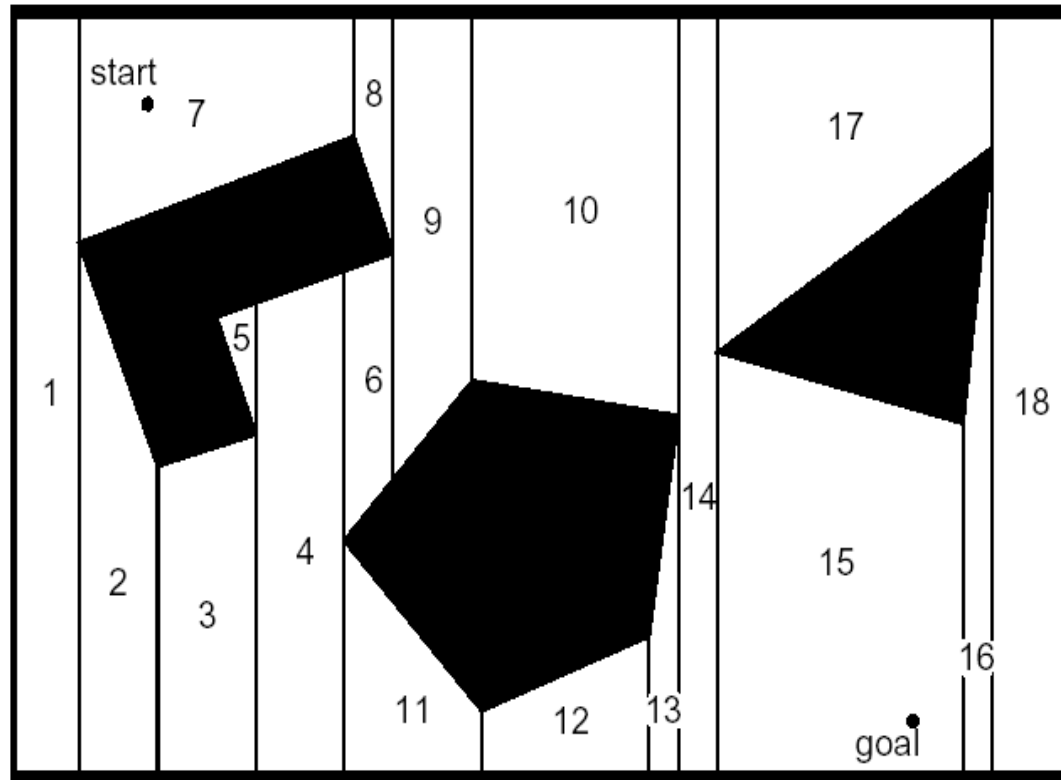
- Pros

- Using range sensors like laser or sonar, a robot can navigate along the Voronoi diagram using simple control rules

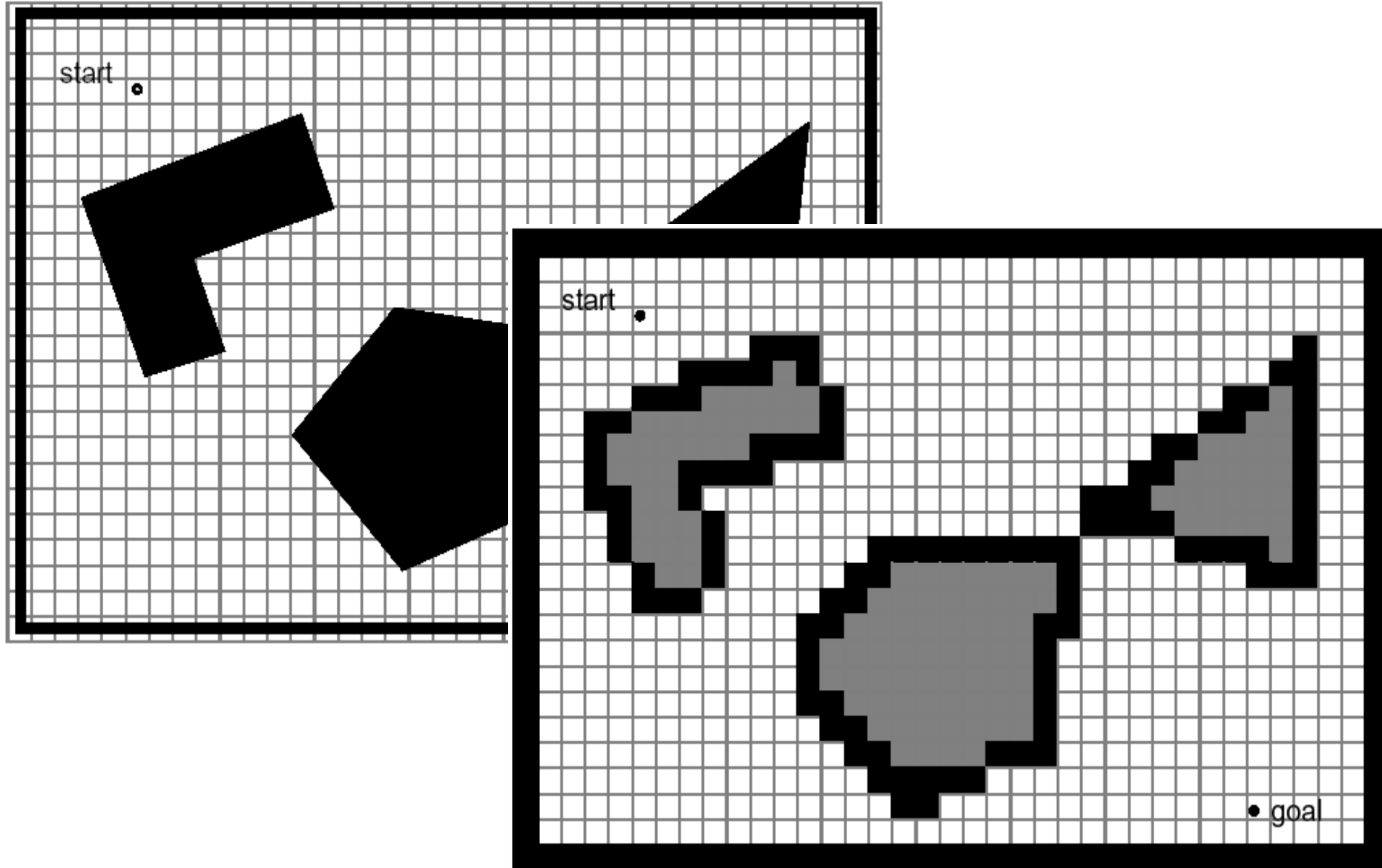
- Cons

- Because the Voronoi diagram tends to keep the robot as far as possible from obstacles, any short range sensor will be in danger of failing
- Voronoi diagram can change drastically in open areas

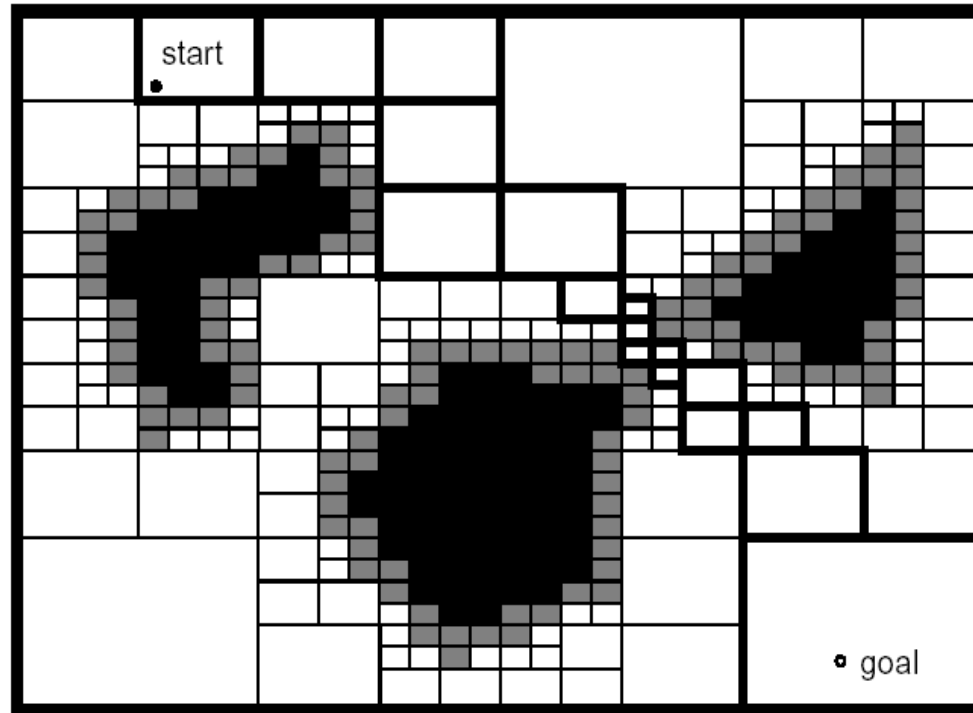
# Graph Construction: Exact Cell Decomposition (2/4)



## Graph Construction: Approximate Cell Decomposition (3/4)



# Graph Construction: Adaptive Cell Decomposition (4/4)



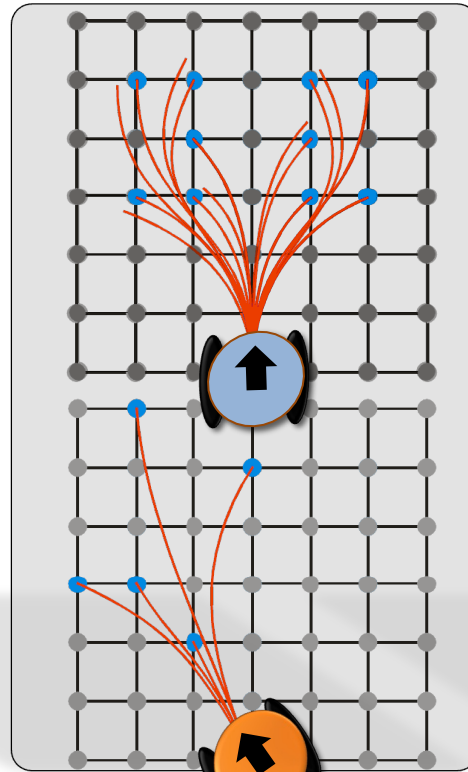
- Close relationship with map representation (Quadtree)!

# Graph Construction: State Lattice Design (1/2)

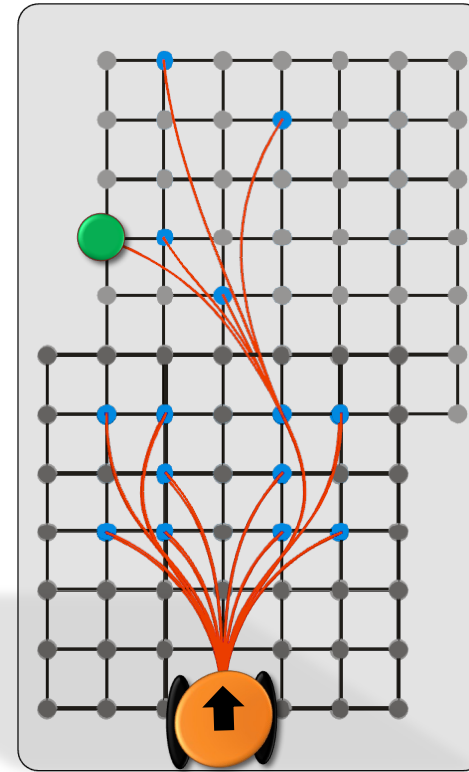
- Enforces **edge feasibility**



Offline:  
Motion Model



Offline:  
Lattice Gen.

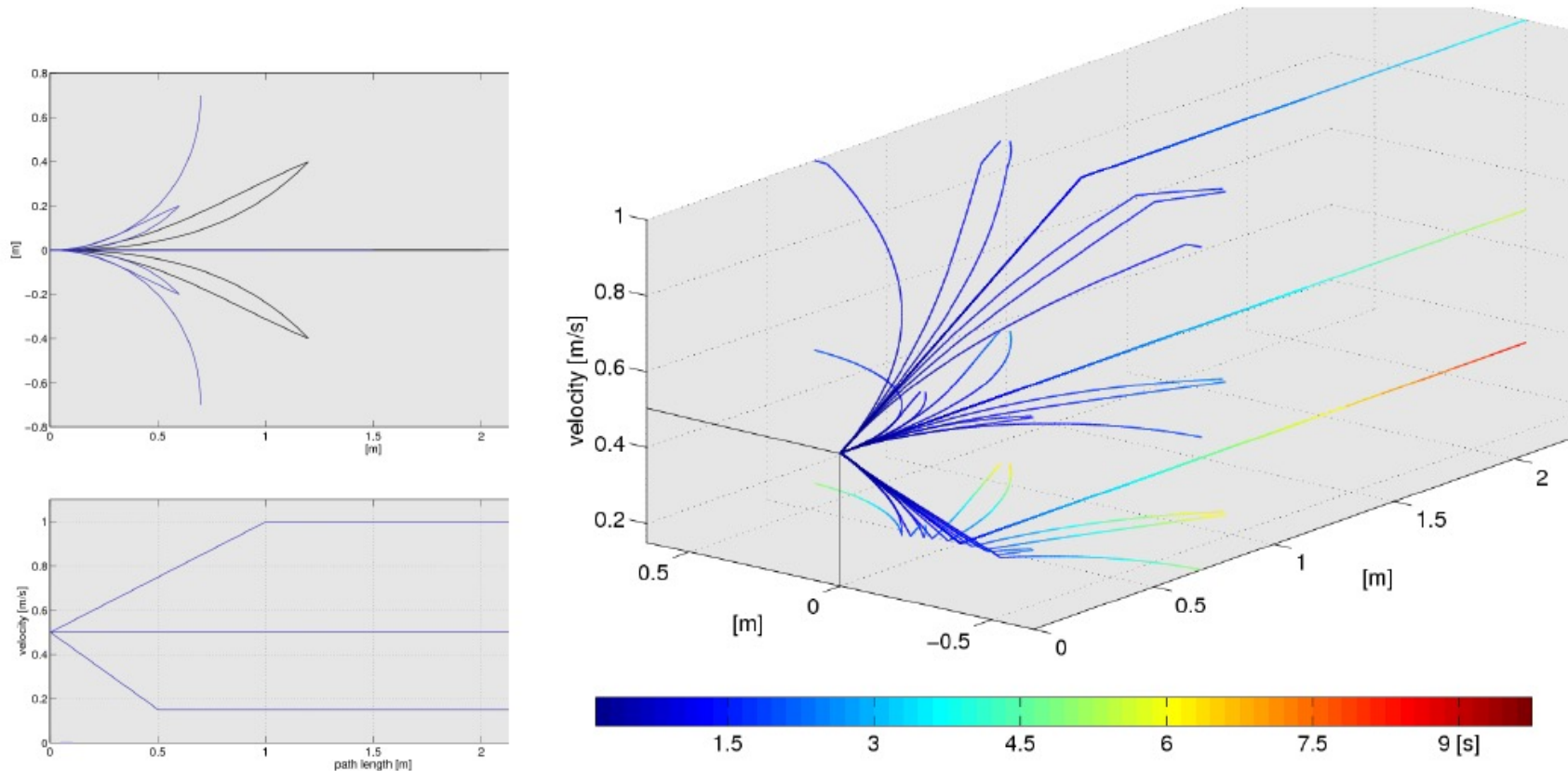


Online:  
Incremental Graph  
Constr.

# Graph Construction: State Lattice Design (2/2)

Martin Rufli

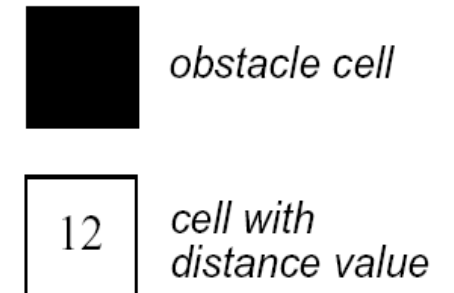
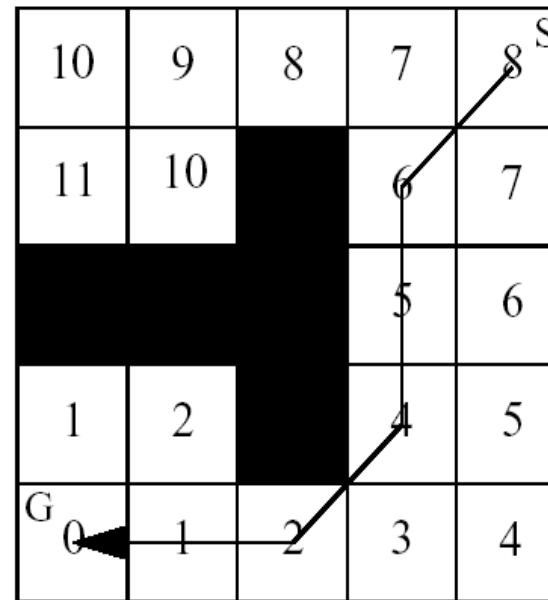
- State lattice encodes only kinematically feasible edges



# Deterministic Graph Search

- Methods

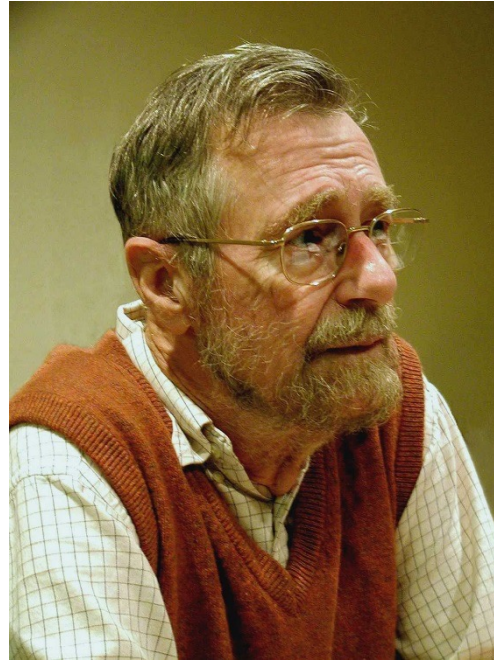
- Breath First
- Depth First
- **Dijkstra**
- A\* and variants
- D\* and variants
- ...



# DIJKSTRA'S ALGORITHM

---

# EDSGER WYBE DIJKSTRA



1930 - 2002

"Computer Science is no more about computers than astronomy is about telescopes."

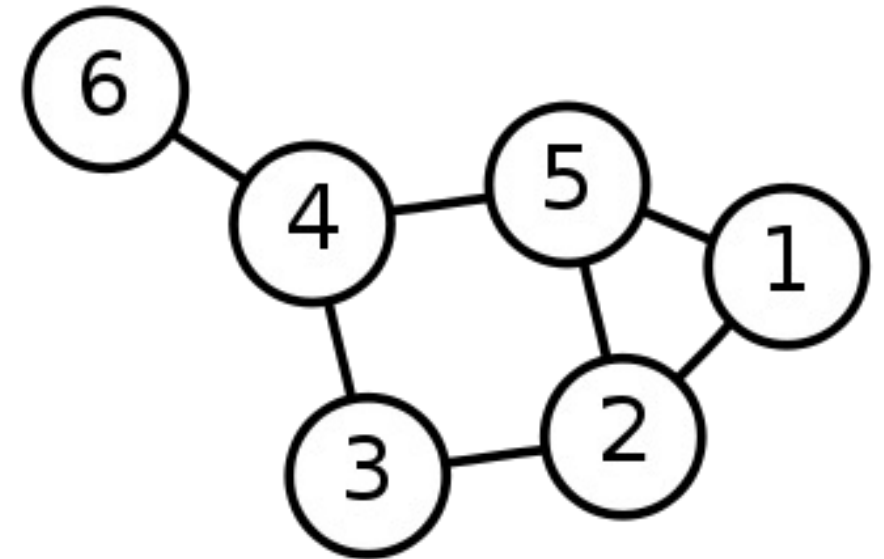
<http://www.cs.utexas.edu/~EWD/>

# SINGLE-SOURCE SHORTEST PATH PROBLEM

- **Single-Source Shortest Path Problem** - The problem of finding shortest paths from a source vertex  $v$  to all other vertices in the graph.

- **Graph**

- Set of vertices and edges
- Vertex:
  - Place in the graph; connected by:
- Edge: connecting two vertices
  - Directed or undirected (undirected in Dijkstra's Algorithm)
  - Edges can have weight/ distance assigned



# Dijkstra's Algorithm

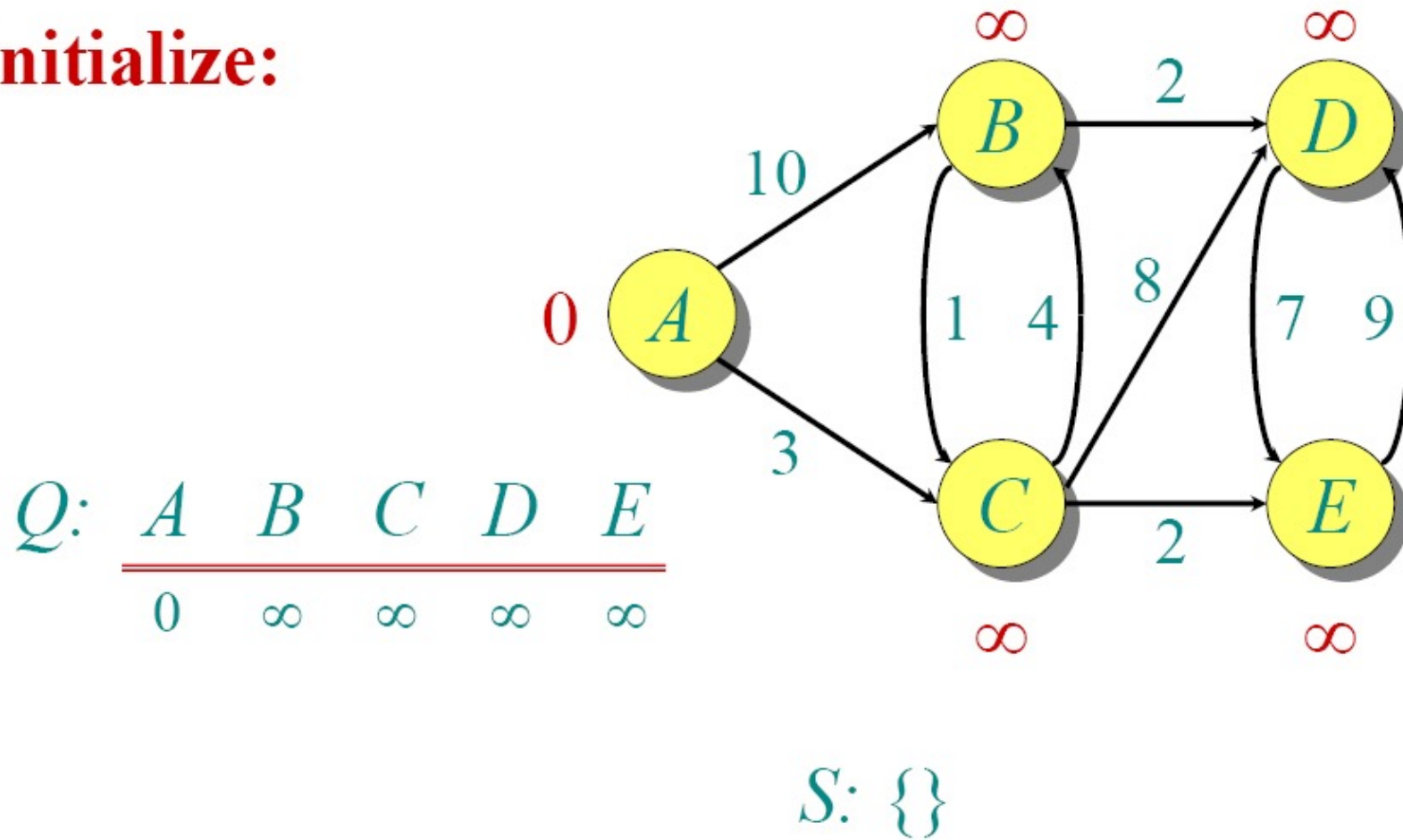
- Assign all vertices infinite distance to goal
- Assign 0 to distance from start
- Add all vertices to the queue
  
- While the queue is not empty:
  - Select vertex with smallest distance and remove it from the queue
  - Visit all neighbor vertices of that vertex,
  - calculate their distance and
  - update their (the neighbors) distance if the new distance is smaller

# Dijkstra's Algorithm - Pseudocode

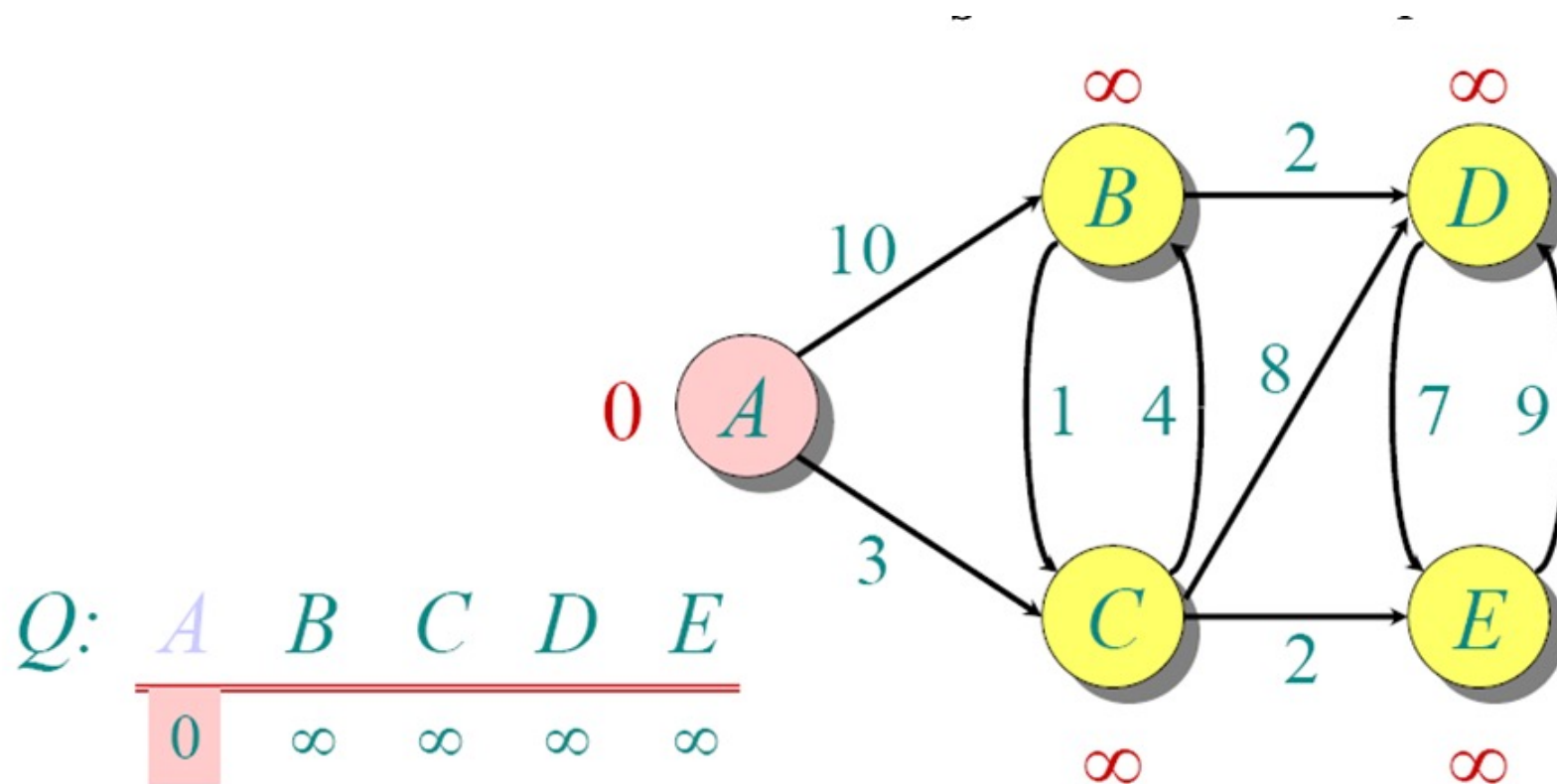
```
dist[s] ← 0                                (distance to source vertex is zero)
for all v ∈ V - {s}
  do dist[v] ← ∞                            (set all other distances to infinity)
S ← ∅                                       (S, the set of visited vertices is initially empty)
Q ← V                                       (Q, the queue initially contains all vertices)
while Q ≠ ∅                                 (while the queue is not empty)
do u ← mindistance(Q, dist)                (select the element of Q with the min. distance)
  S ← S ∪ {u}                               (add u to list of visited vertices)
  for all v ∈ neighbors[u]
    do if dist[v] > dist[u] + w(u, v)       (if new shortest path found)
       then d[v] ← d[u] + w(u, v)         (set new value of shortest path)
      (if desired, add traceback code)
return dist
```

# Dijkstra Example

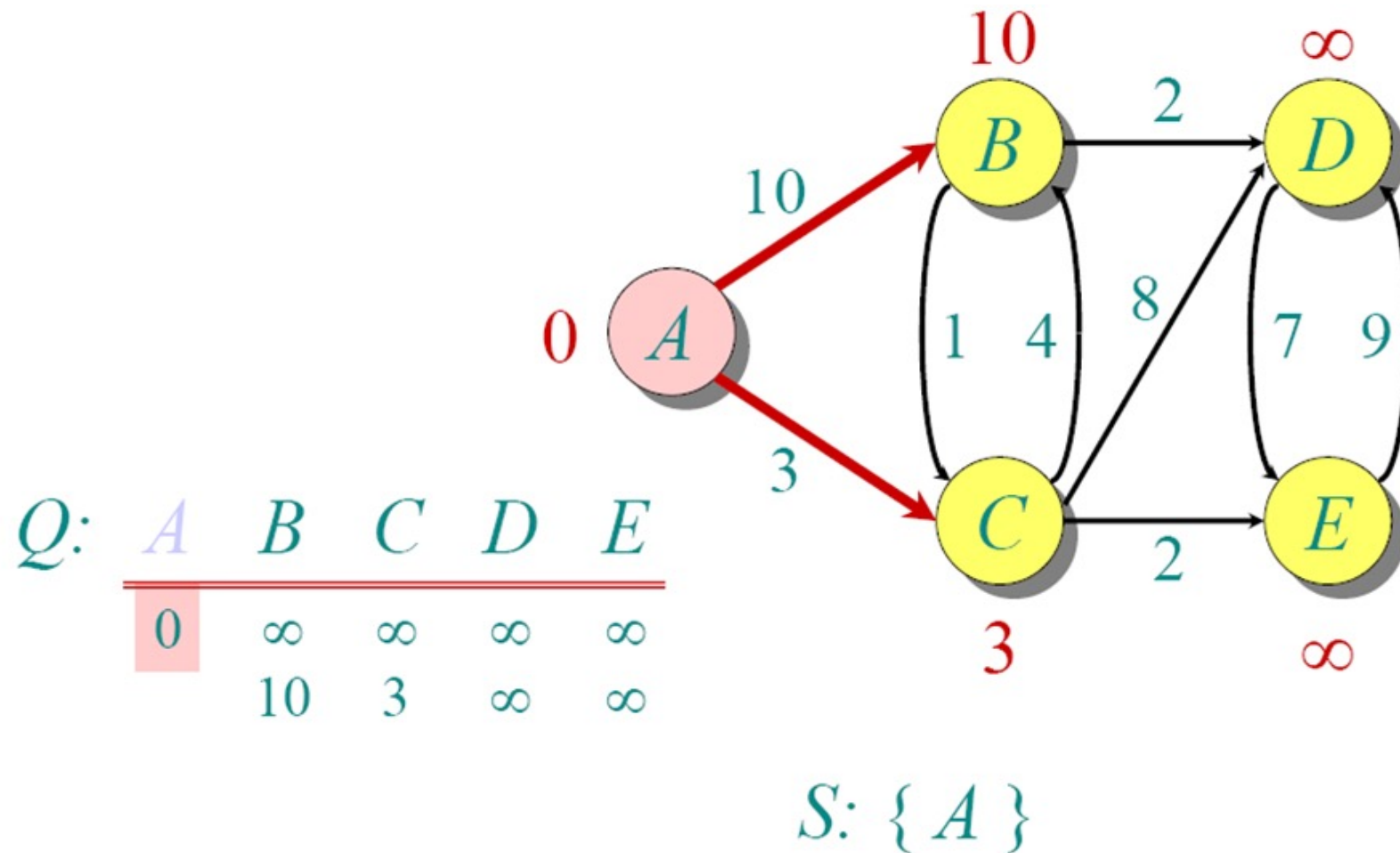
**Initialize:**



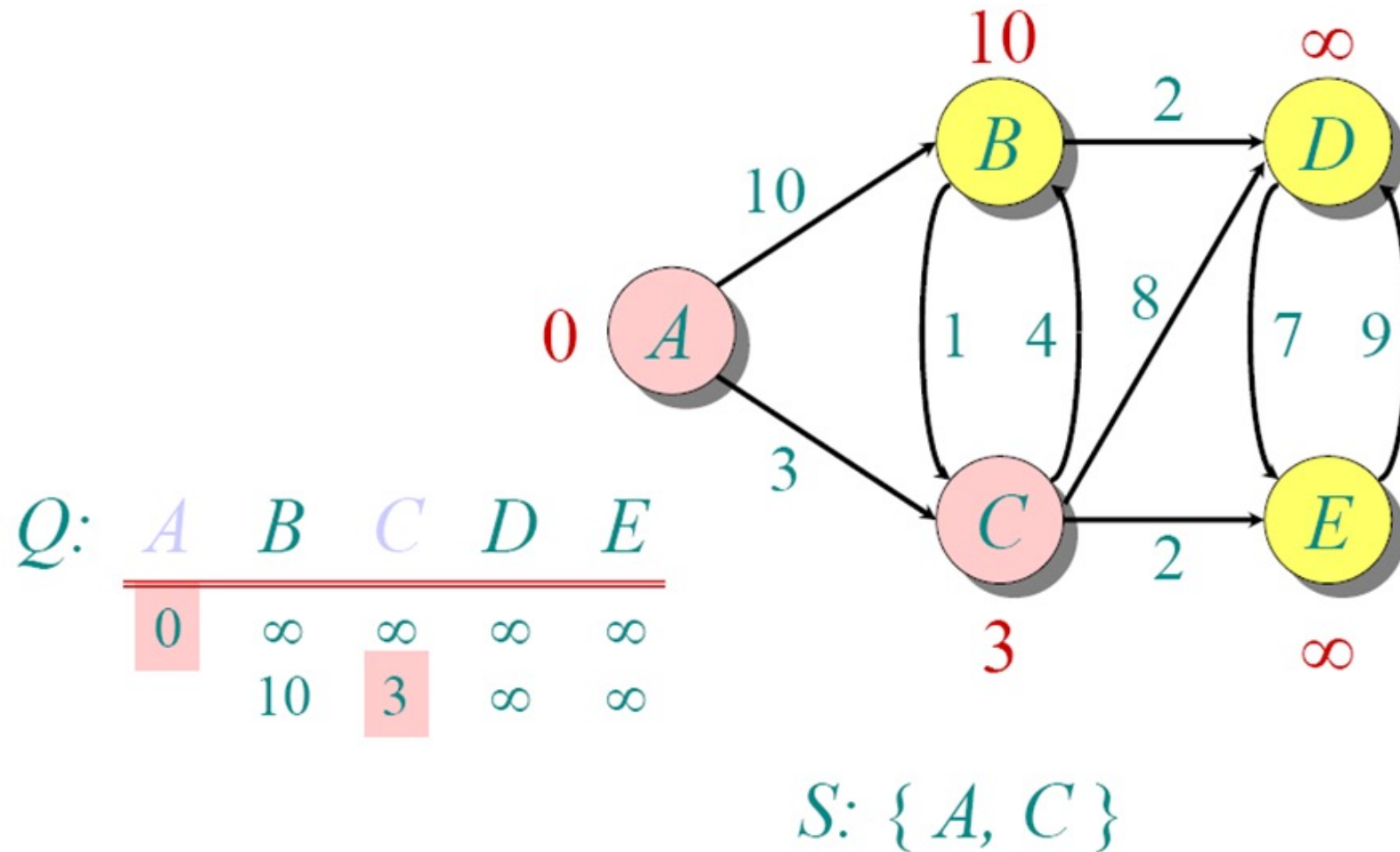
# Dijkstra Example



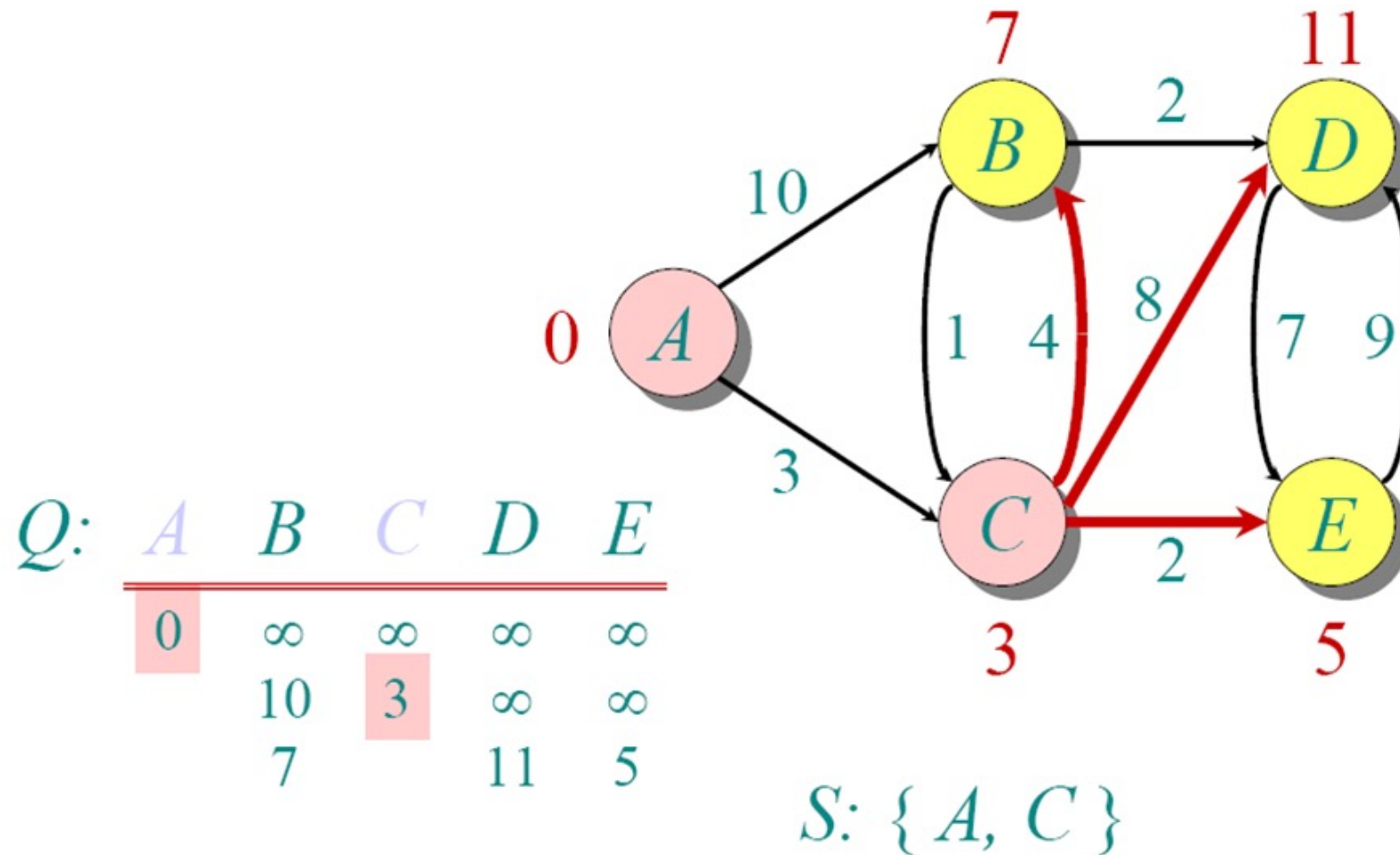
# Dijkstra Example



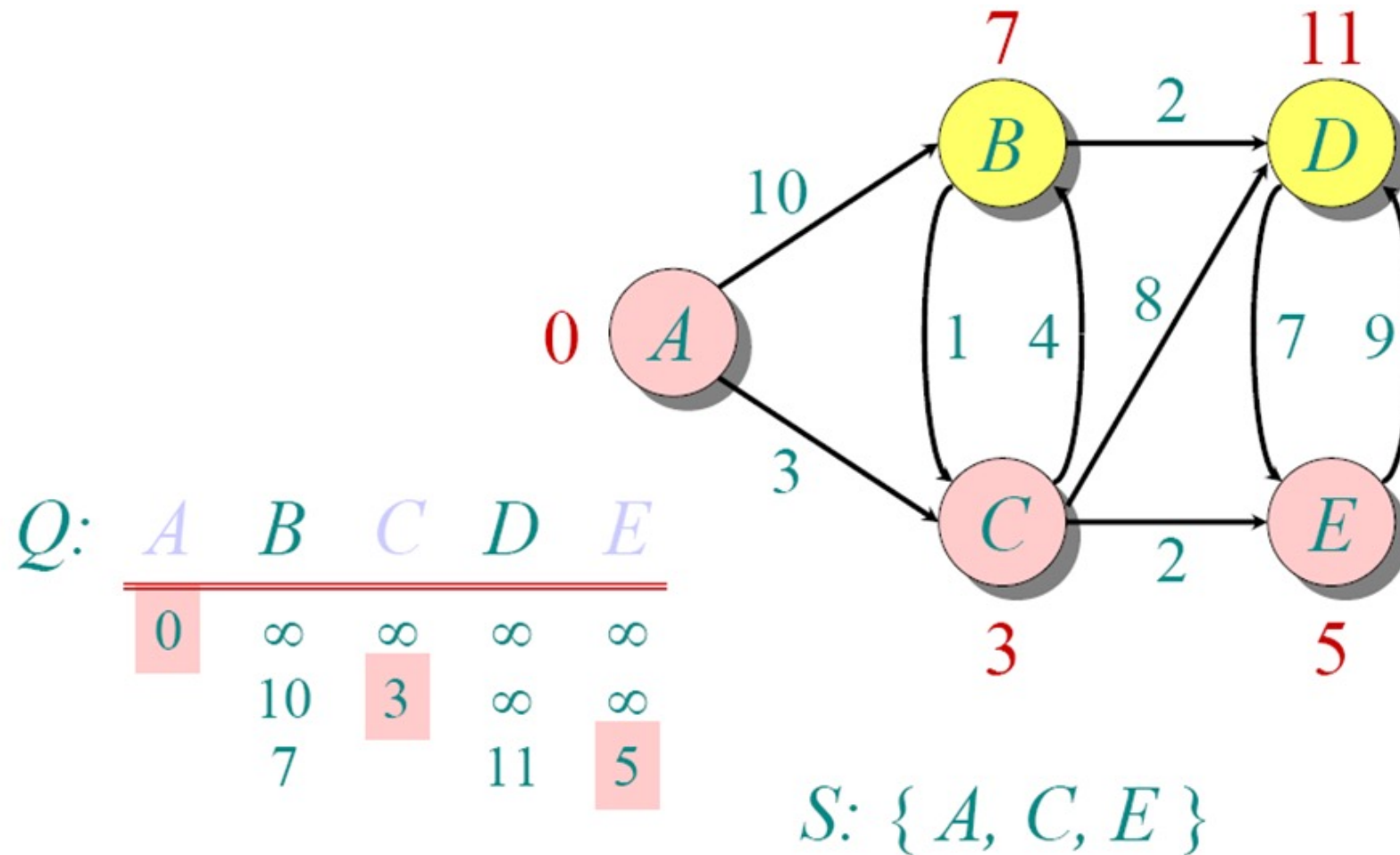
# Dijkstra Example



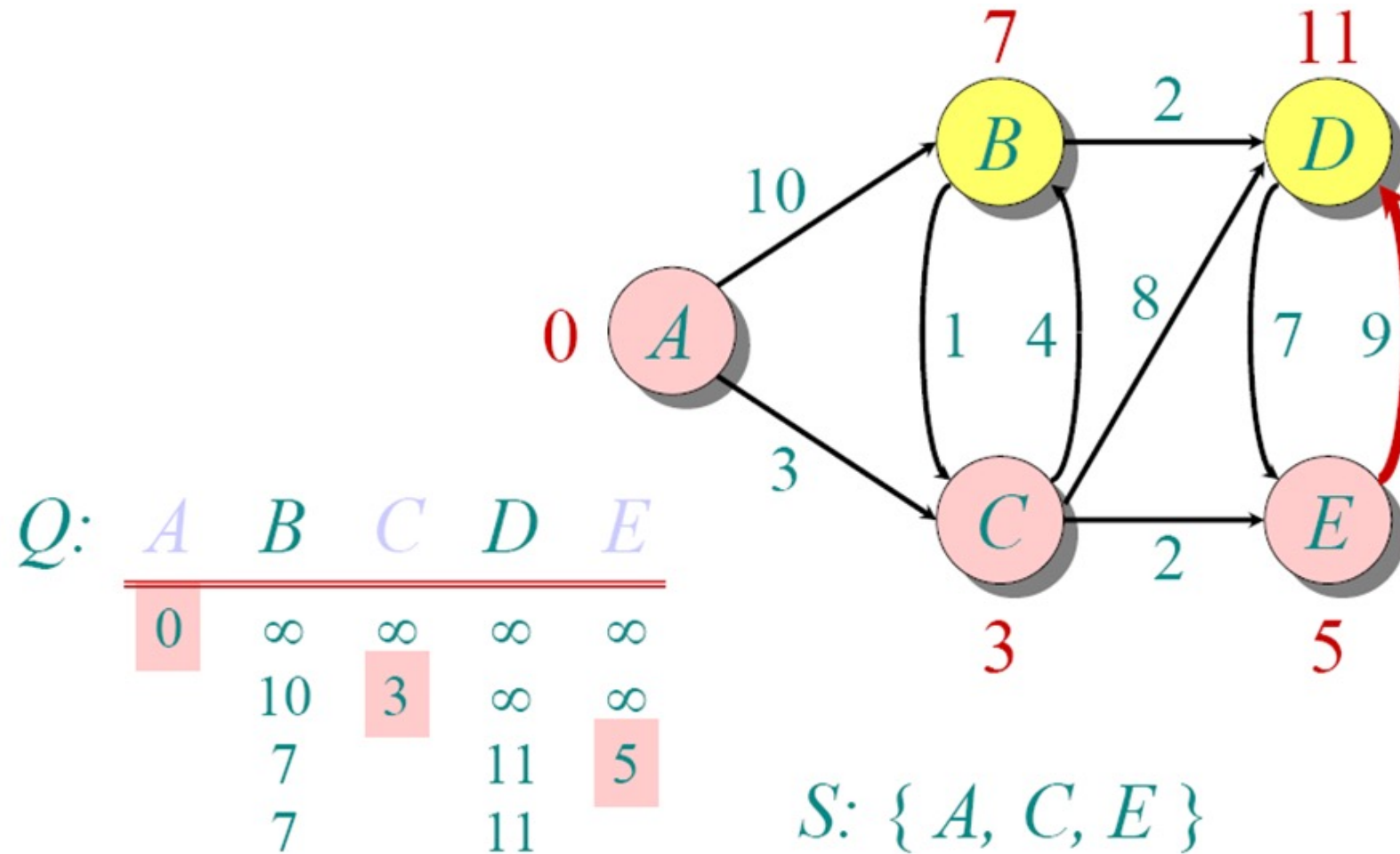
# Dijkstra Example



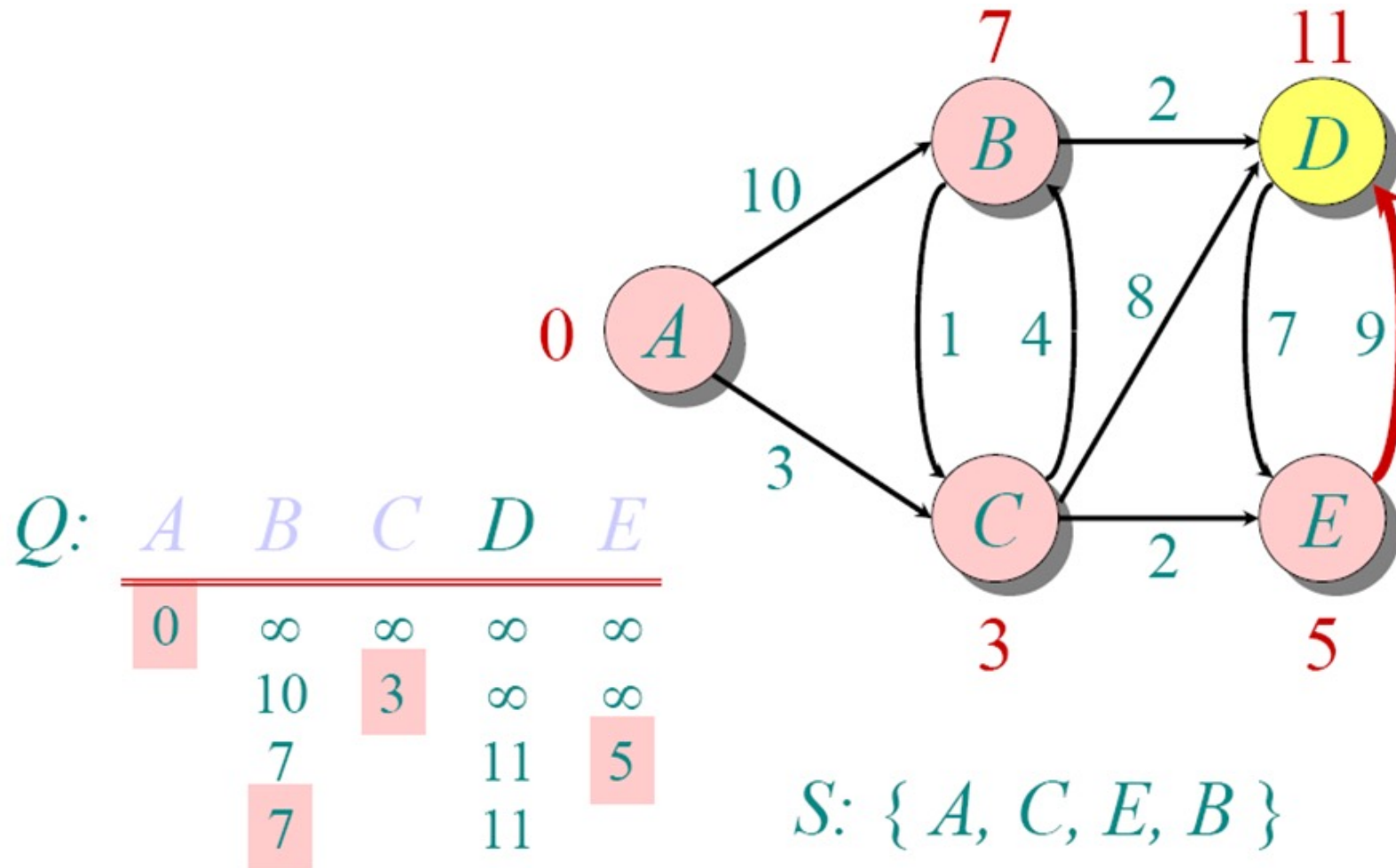
# Dijkstra Example



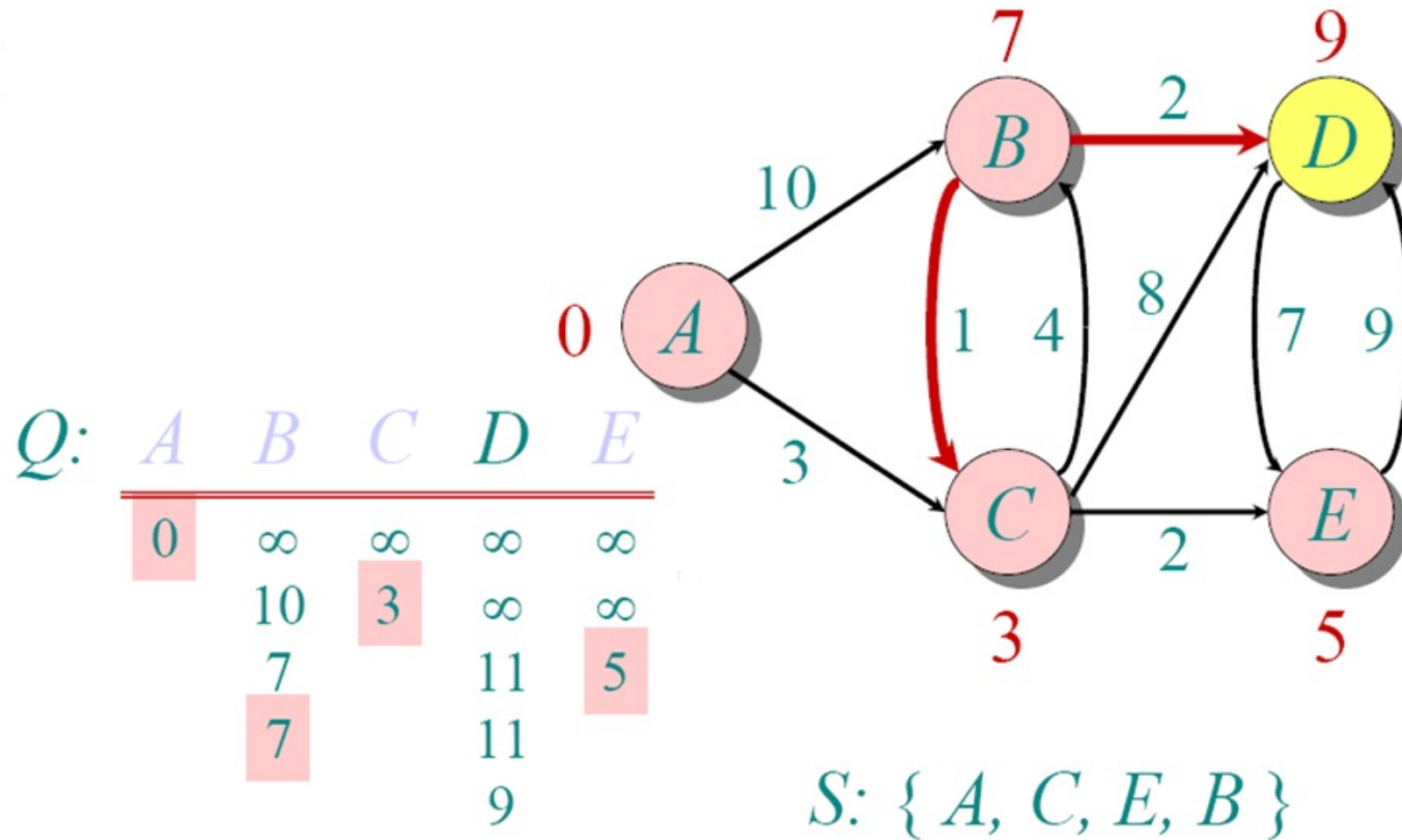
# Dijkstra Example



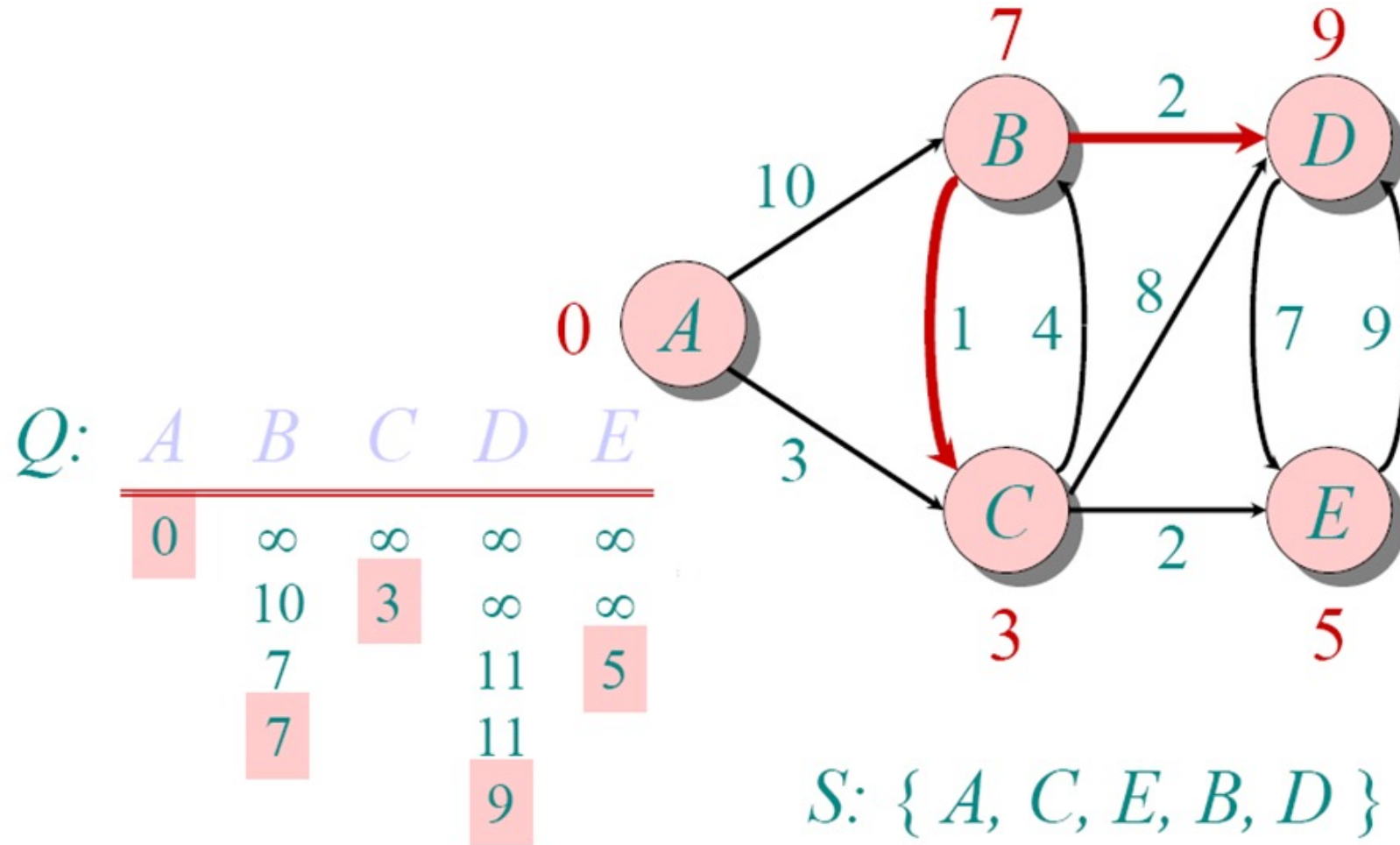
# Dijkstra Example



# Dijkstra Example



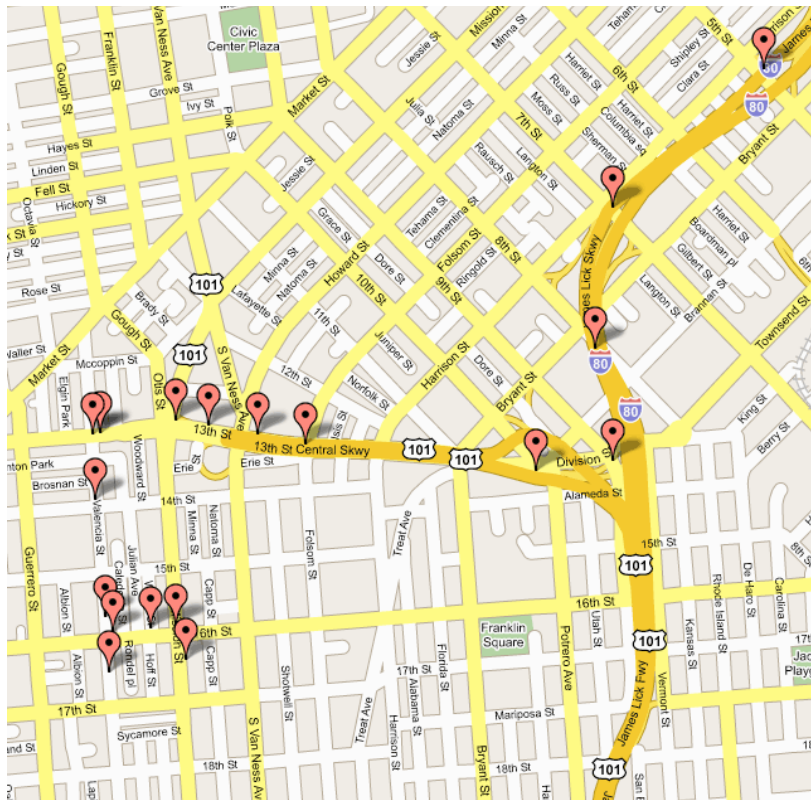
# Dijkstra Example



# APPLICATIONS OF DIJKSTRA'S ALGORITHM

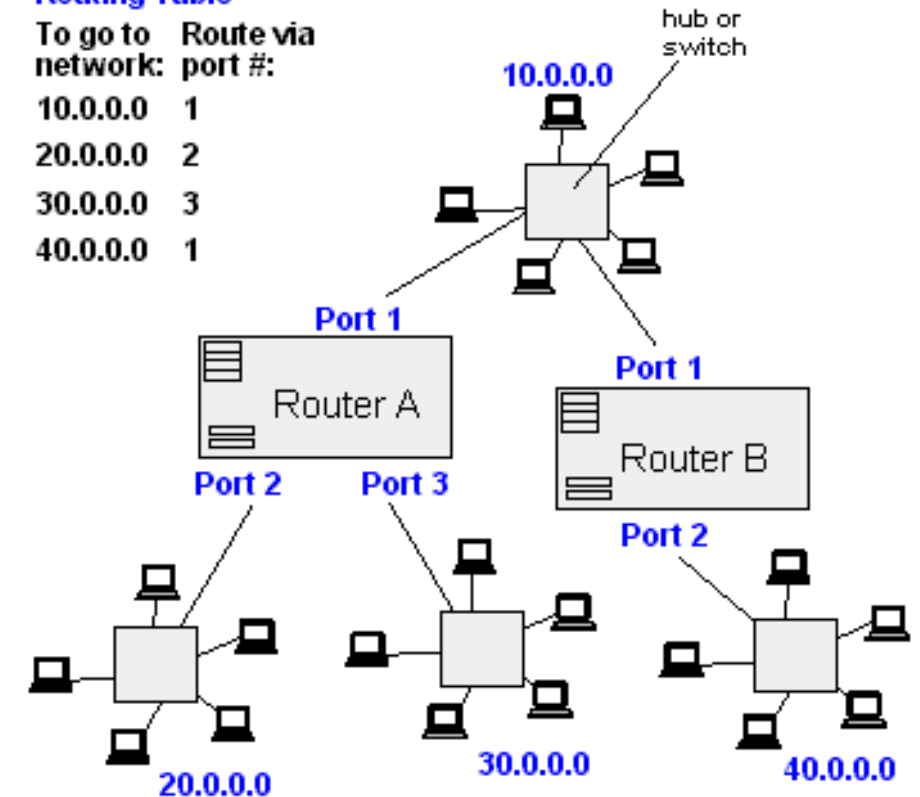
- Navigation Systems
- Internet Routing

From Computer Desktop Encyclopedia  
© 1998 The Computer Language Co. Inc.



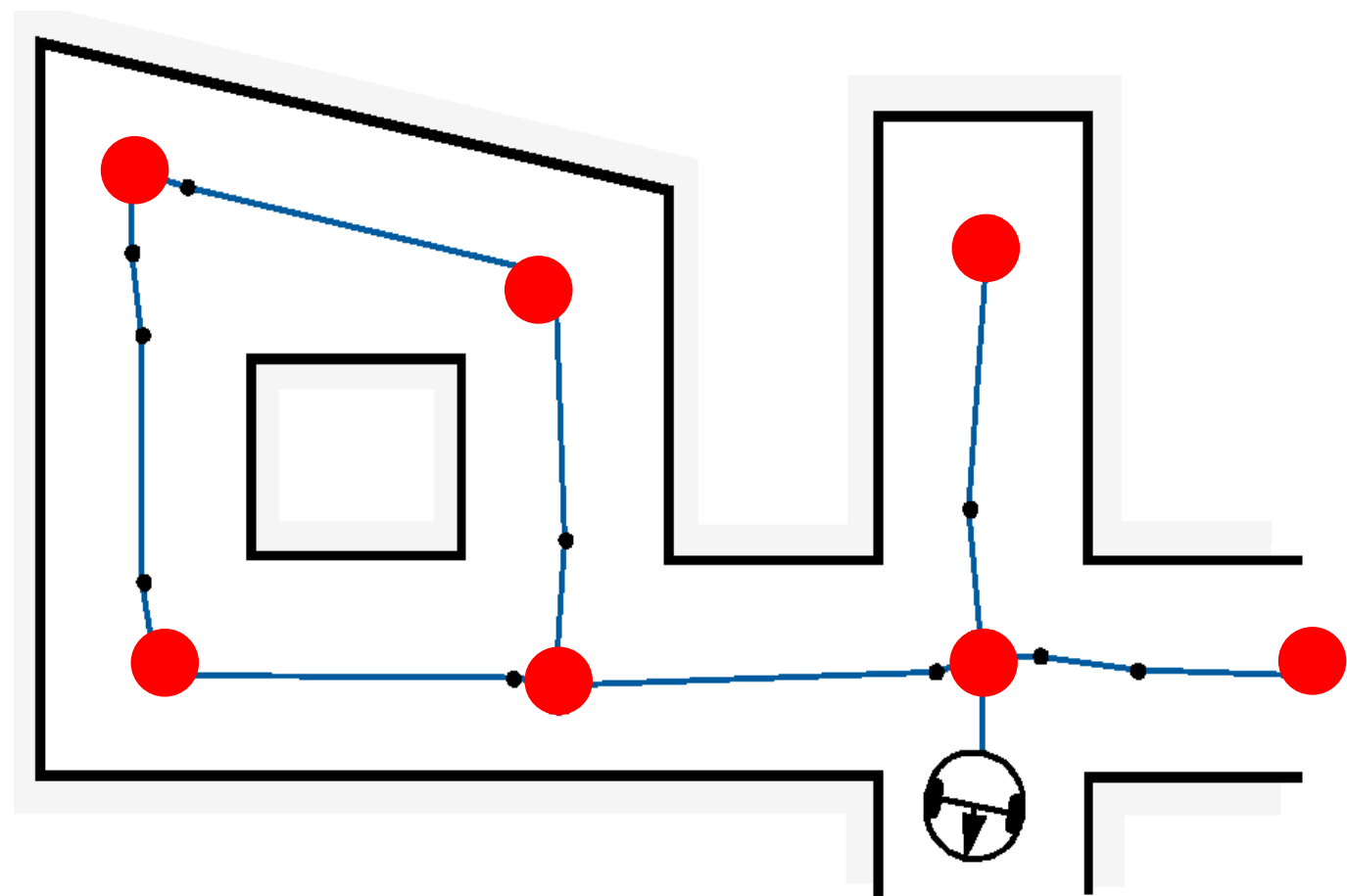
## Router A Routing Table

| To go to network: | Route via port #: |
|-------------------|-------------------|
| 10.0.0.0          | 1                 |
| 20.0.0.0          | 2                 |
| 30.0.0.0          | 3                 |
| 40.0.0.0          | 1                 |



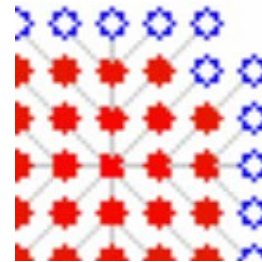
# Dijkstra's Algorithm for Path Planning: Topological Maps

- Topological Map:
  - Places (vertices) in the environment (red dots)
  - Paths (edges) between them (blue lines)
  - Length of path = weight of edge
- => Apply Dijkstra's Algorithm to find path from start vertex to goal vertex



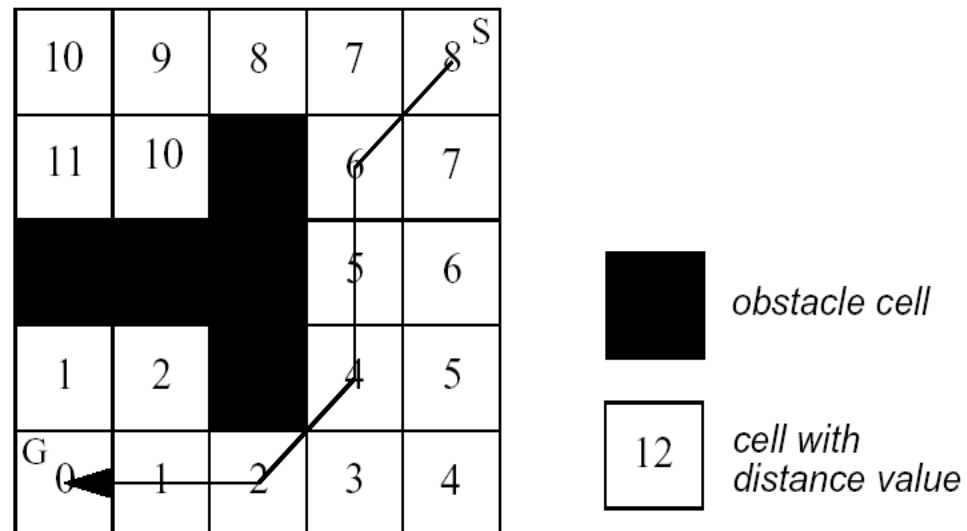
# Dijkstra's Algorithm for Path Planning: Grid Maps

- Graph:
  - Neighboring free cells are connected:
    - 4-neighborhood: up/ down/ left right
    - **8-neighborhood**: also diagonals
  - All edges have weight 1
- Stop once goal vertex is reached
- Per vertex: save edge over which the shortest distance from start was reached => Path

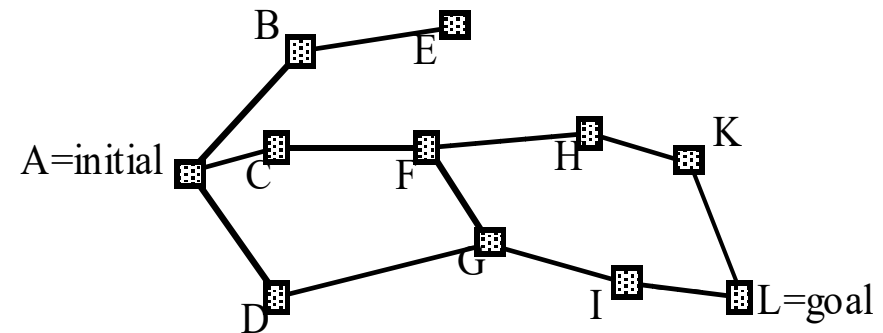
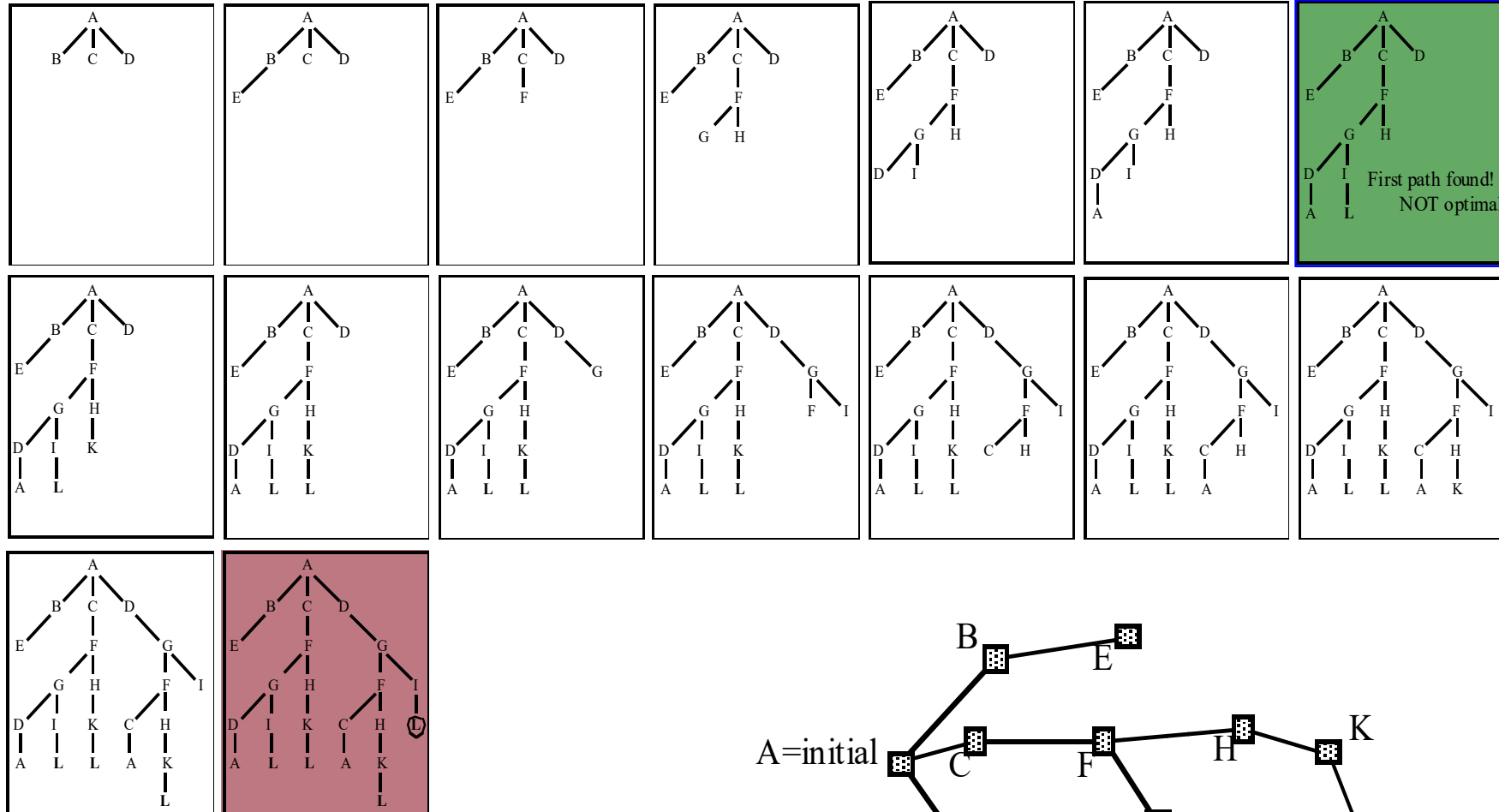


# Graph Search Strategies: Breath-First Search

- Corresponds to a wavefront expansion on a 2D grid
- Breath-First: Dijkstra's search where all edges have weight 1

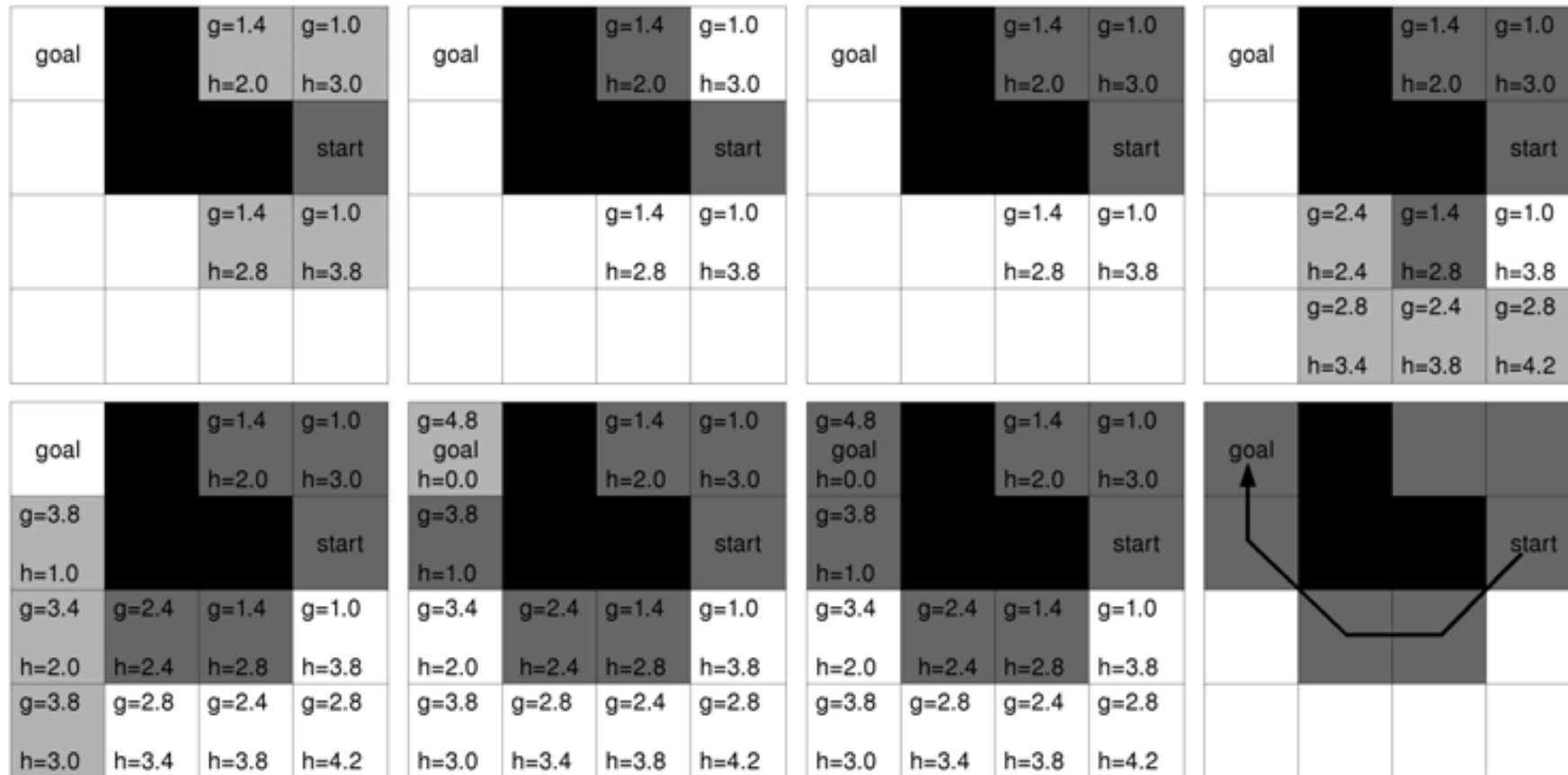


# Graph Search Strategies: Depth-First Search



# Graph Search Strategies: A\* Search

- Similar to Dijkstra's algorithm, except that it uses a heuristic function  $h(n)$
- $f(n) = g(n) + h(n)$



# A\*

- Developed 1986 as part of the Shakey project!
- Complexity:

**Worst-case performance**  $O(|E|) = O(b^d)$

**Worst-case space complexity**  $O(|V|) = O(b^d)$

b: branching factor

d: depth

- Good heuristic => small branching factor



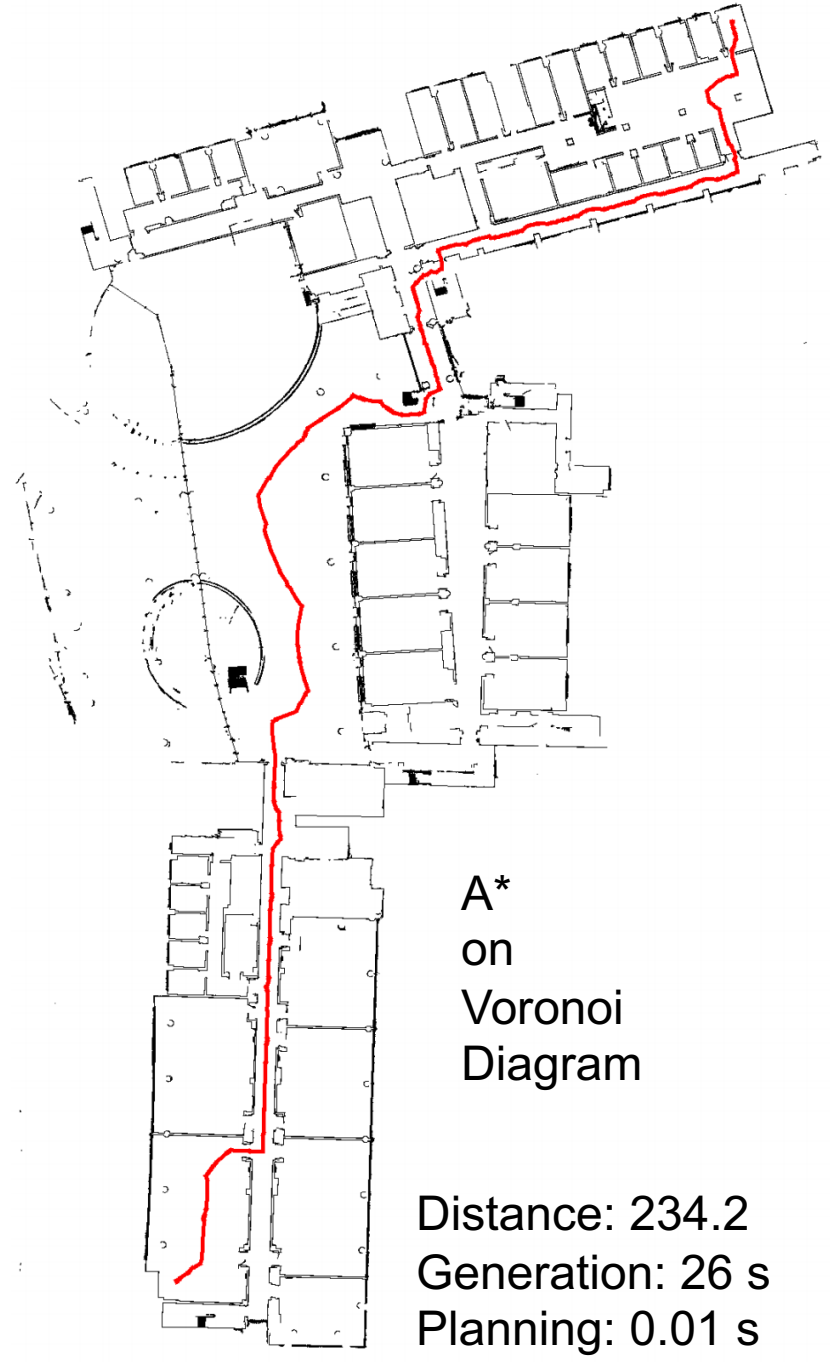
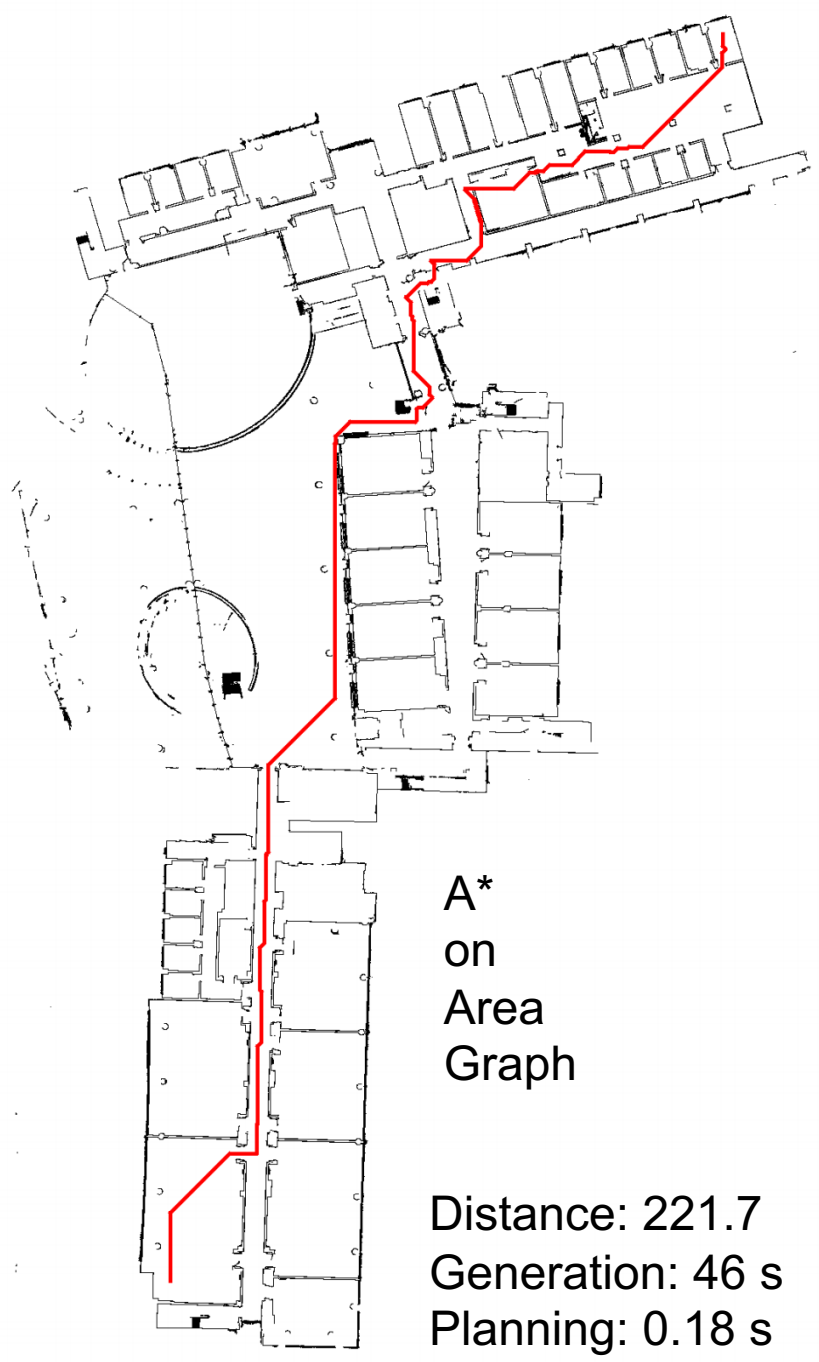
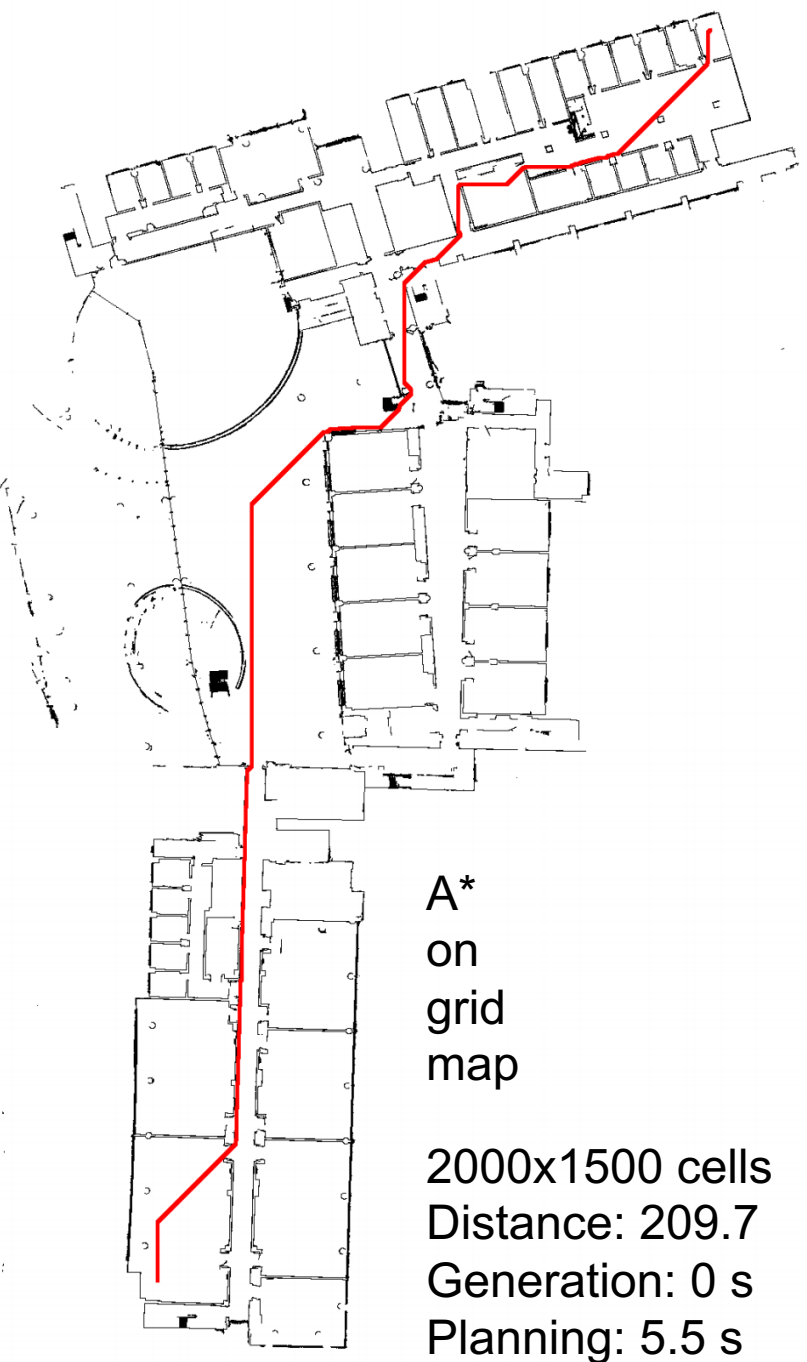
# Optimal Planning

- Dijkstra finds the optimal path
- What about A\*?
  - Find admissible heuristic  $h(n)$
  - Admissible: do not overestimate the true cost-to-go
  - A\* is optimal (finds optimal/ shortest path) if  $h(n)$  is admissible for all  $n$
  - Admissible example: use geometric distance for  $h(n)$ :
$$h(n) = \sqrt{(x_{goal} - x)^2 + (y_{goal} - y)^2}.$$
  - Example: heuristic 5x geometric distance
  - $h(n) = 0 \Rightarrow$  Dijkstra's Algorithm

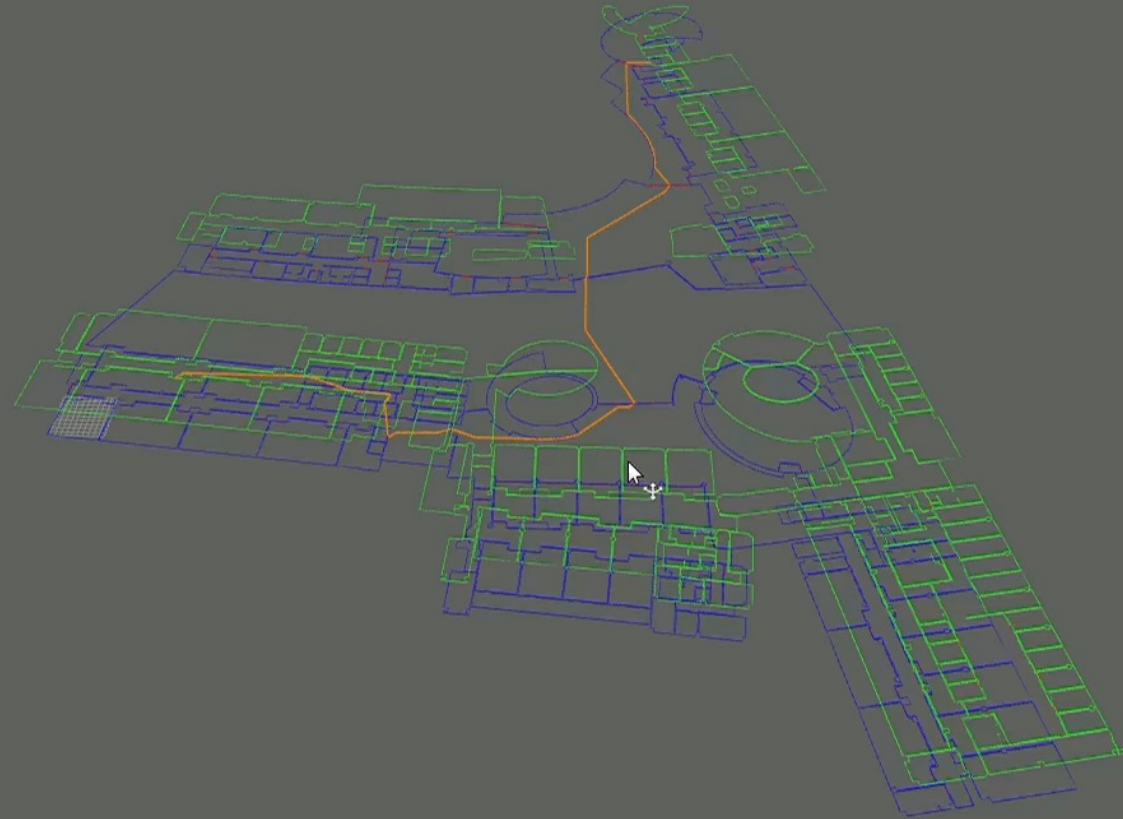


# A\*

- Hierarchical planning possible e.g.: Go to the library:
  - First plan how to get from SIST building to library
  - Then plan how to get from entrance of library to goal room (campus level vs. library level)
- Many variants of A\* algorithms exist – with different properties
- A\* as graph search: applications outside of robotics/ path planning
  - Video games
  - Parsing with stochastic grammars in natural language processing
- Graph on which planning is done matters!
  - E.g.: Grid map; Pose Graph; Topological Graph; Open Street Map; Lattice Graphs; ...

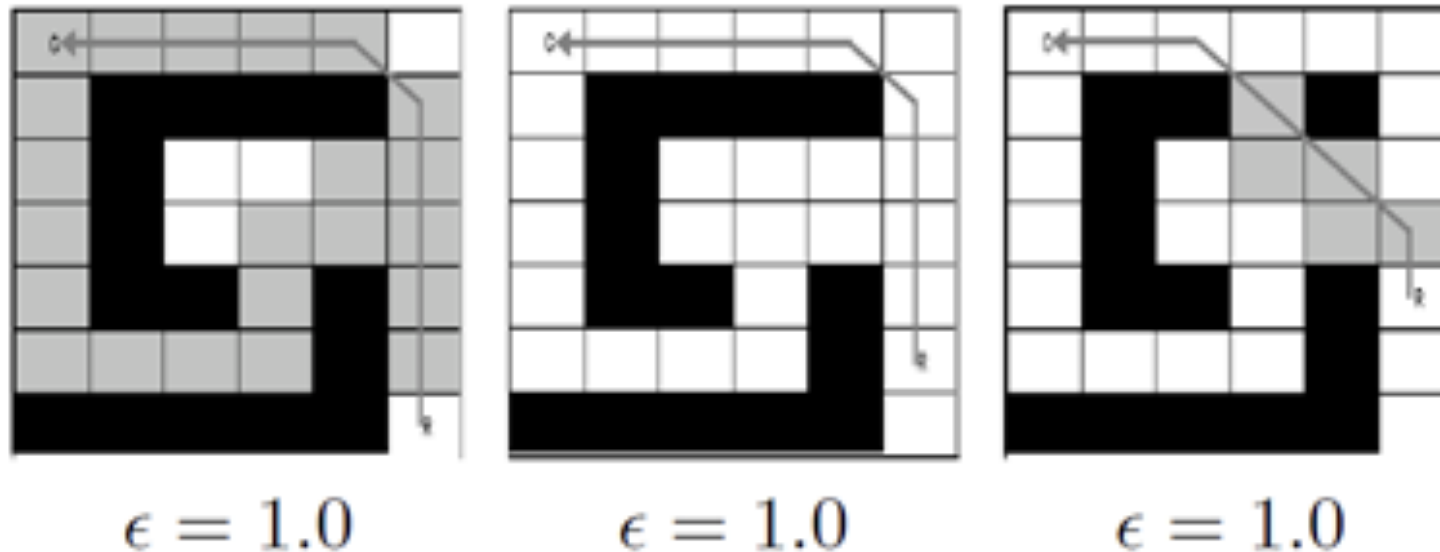


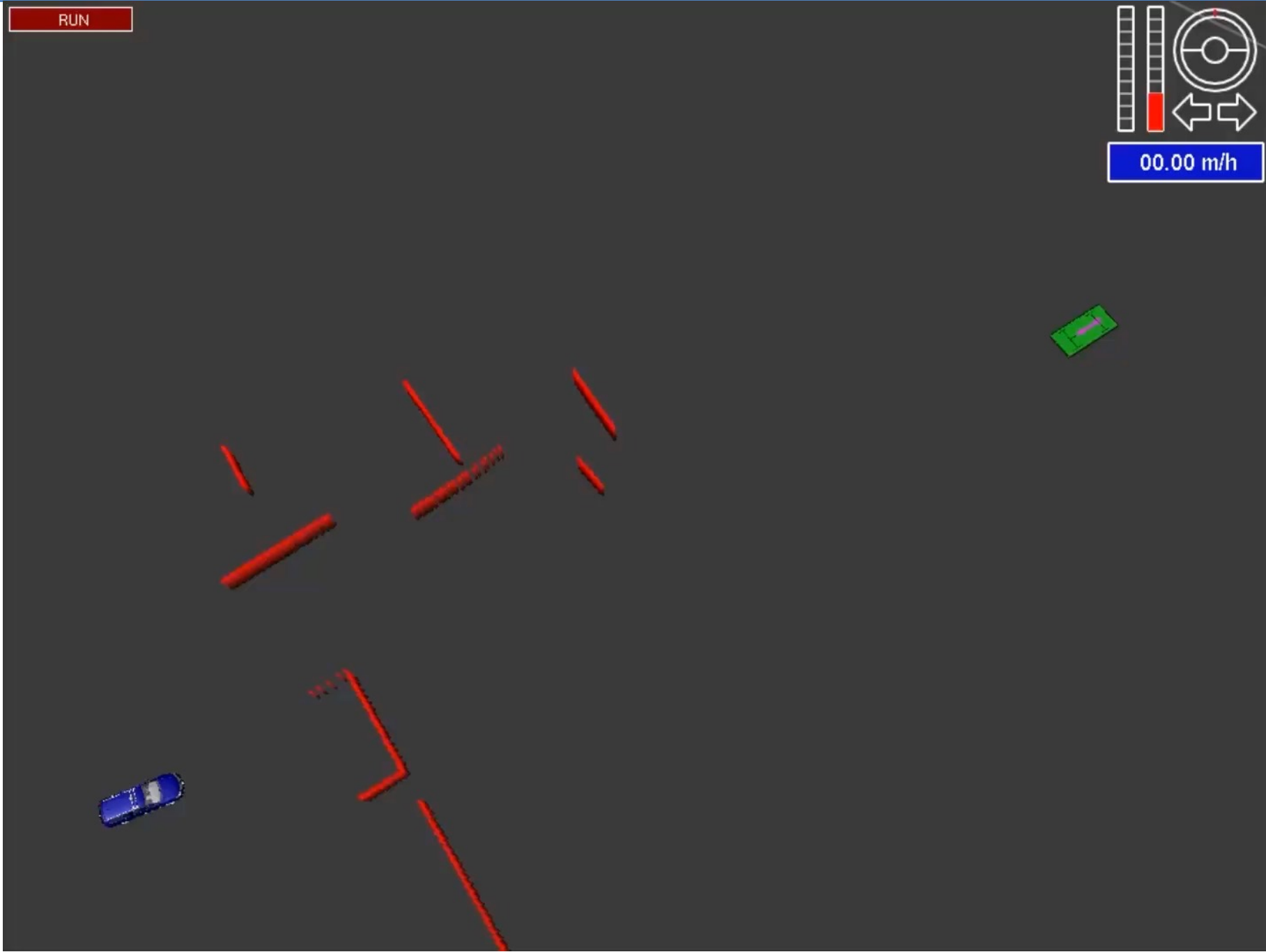
# 3D Path planned with A\* in Area Graph of 2 stories of SIST & SEM buildings



# Graph Search Strategies: D\* Search

- Similar to A\* search, except that the search starts from the goal outward
- $f(n) = g(n) + \epsilon h(n)$
- First pass is identical to A\*
- Subsequent passes reuse information from previous searches



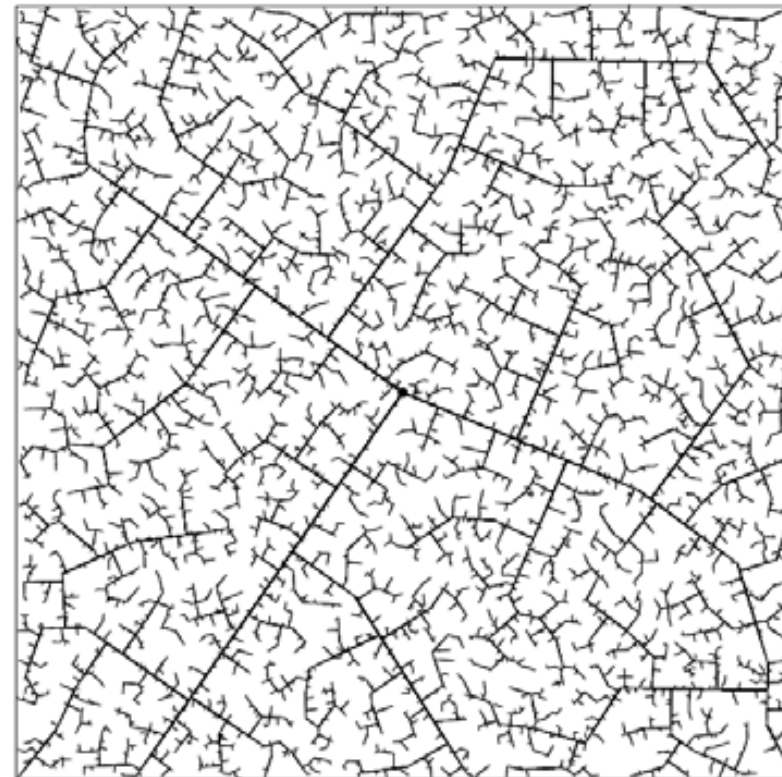


# Graph Search Strategies: Randomized Search

- Most popular version is the rapidly exploring random tree (RRT)
  - Well suited for high-dimensional search spaces
  - Often produces highly suboptimal solutions



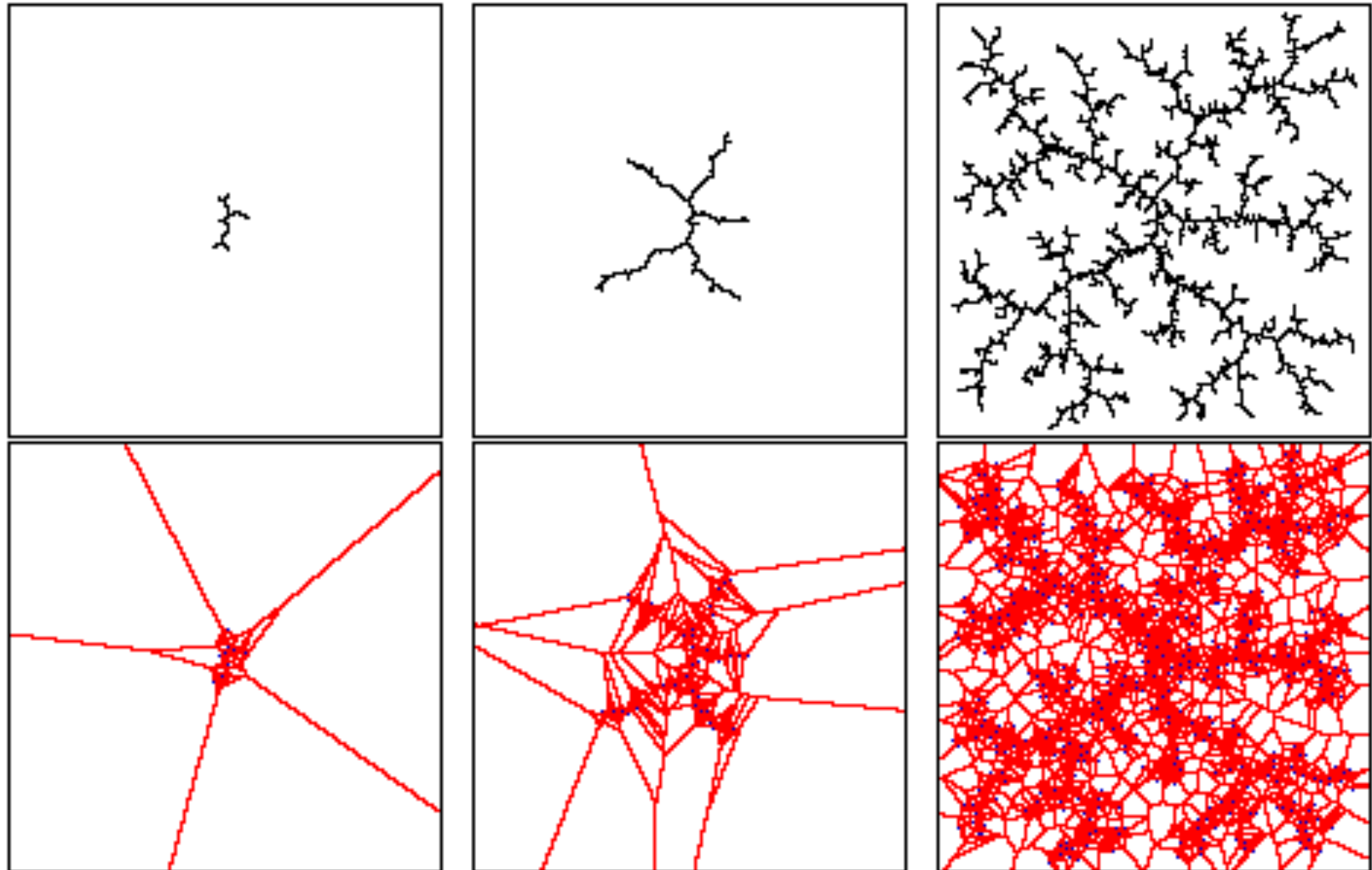
45 iterations

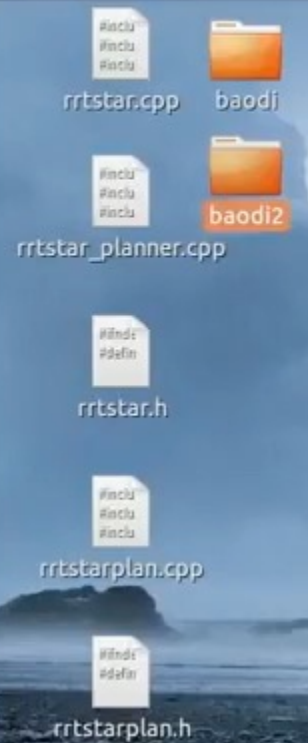


2345 iterations

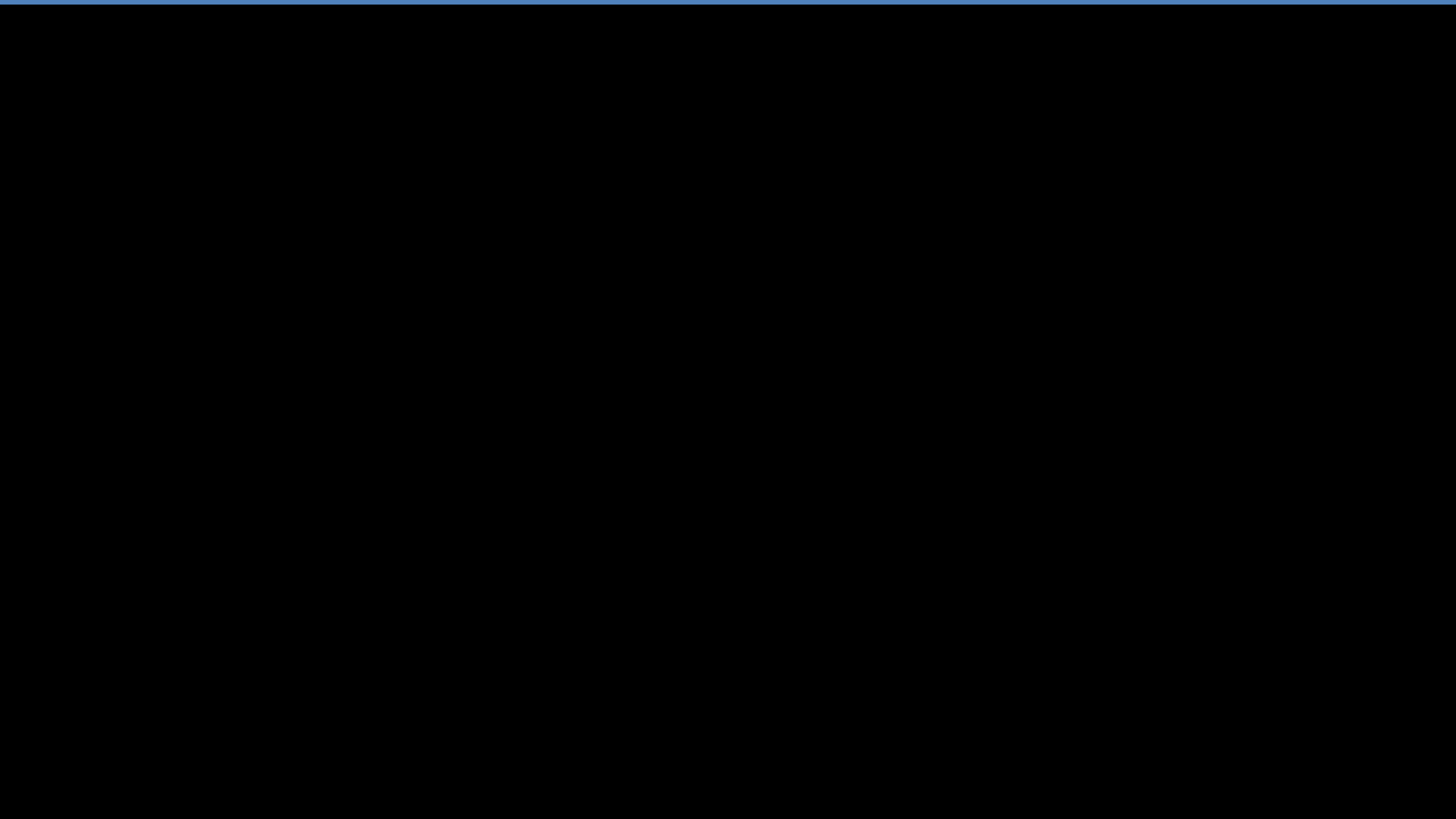
# Why are RRT's rapidly exploring?

The probability of a node to be selected for expansion is proportional to the area of its Voronoi region





```
lizhi@lizhi-HP-EliteBook-8460w: ~/download codes/rrtstar_planner
goal: 5.96963 7.10589
1
New Path Found. Total paths 1
Finding Optimal Path
[ INFO] [1497922923.557025739, 592.300000000]: Got new plan
[ INFO] [1497922923.957553192, 592.700000000]: Goal reached
^C[rviz-13] killing on exit
[amcl-12] killing on exit
[map_server-11] killing on exit
[move_base-10] killing on exit
[kobuki_safety_controller-9] killing on exit
[navigation_velocity_smoother-8] killing on exit
[cmd_vel_mux-7] killing on exit
[mobile_base_nodelet_manager-6] killing on exit
[joint_state_publisher-5] killing on exit
[diagnostic_aggregator-4] killing on exit
[robot_state_publisher-3] killing on exit
[stageros-2] killing on exit
[rosout-1] killing on exit
[master] killing on exit
shutting down processing monitor...
... shutting down processing monitor complete
done
lizhi@lizhi-HP-EliteBook-8460w:~/download codes/rrtstar_planner$
```





# teb\_local\_planner

An optimal trajectory planner for mobile robots based on Timed-Elastic-Bands