



上海科技大学
ShanghaiTech University

CS283: Robotics Spring 2025: Planning

Sören Schwertfeger / 师泽仁

ShanghaiTech University

ADMIN

Midterm; April 17; during class hours

- Midterm: “test-run” for the final...
- Content:
 - Everything till (including) today’s lecture.
 - Take a look at facts, algorithms, concepts
 - Take a look at the homeworks again
 - Sample exam: https://robotics.shanghaitech.edu.cn/sites/default/files//files/final_Example.pdf
- You are allowed to bring 3 A4 sheets (so 6 pages) of info to the exams (including Final – so for midterm maybe use 1.5 or 2 A4 sheets). You can write/ print anything on those sheets. On top of every page (so 6 times) there needs to be your name (pinyin), student ID and ShanghaiTech email address. We will check every cheat sheet before the exam and confiscate every sheet without name or with a name that is not yours.
- No electronics/ calculator/ smartwatch allowed

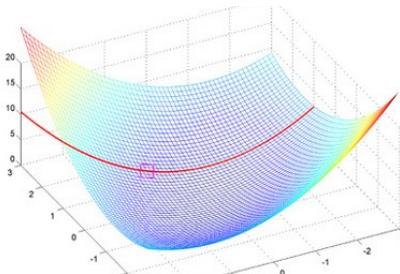
Presentation

- Upload your pdf/ ppt to your group git latest April 7 23:59!
- If you want to use your own pdf/ ppt later you'll loose 30% of that score.
- Be present at all 3 presentation slots – listen carefully – everybody needs to ask in total at least 3 questions (for 3 different presentations)

Path Planning: Overview of Algorithms

1. Optimal Control

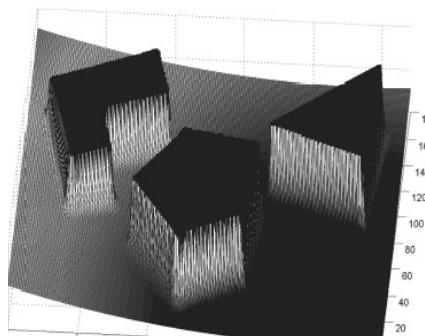
- Solves truly optimal solution
- Becomes intractable for even moderately complex as well as nonconvex problems



Source:
<http://mitocw.udsm.ac.tz>

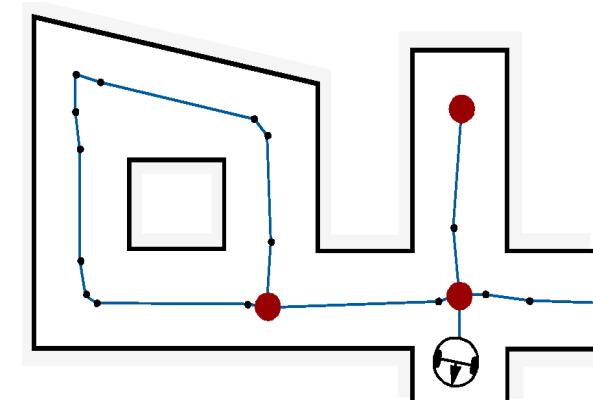
2. Potential Field

- Imposes a mathematical function over the state/configuration space
- Many physical metaphors exist
- Often employed due to its simplicity and similarity to optimal control solutions

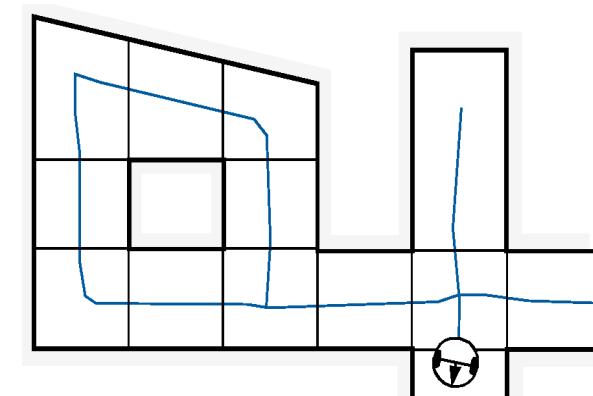


3. Graph Search

- Identify a set edges between nodes within the free space



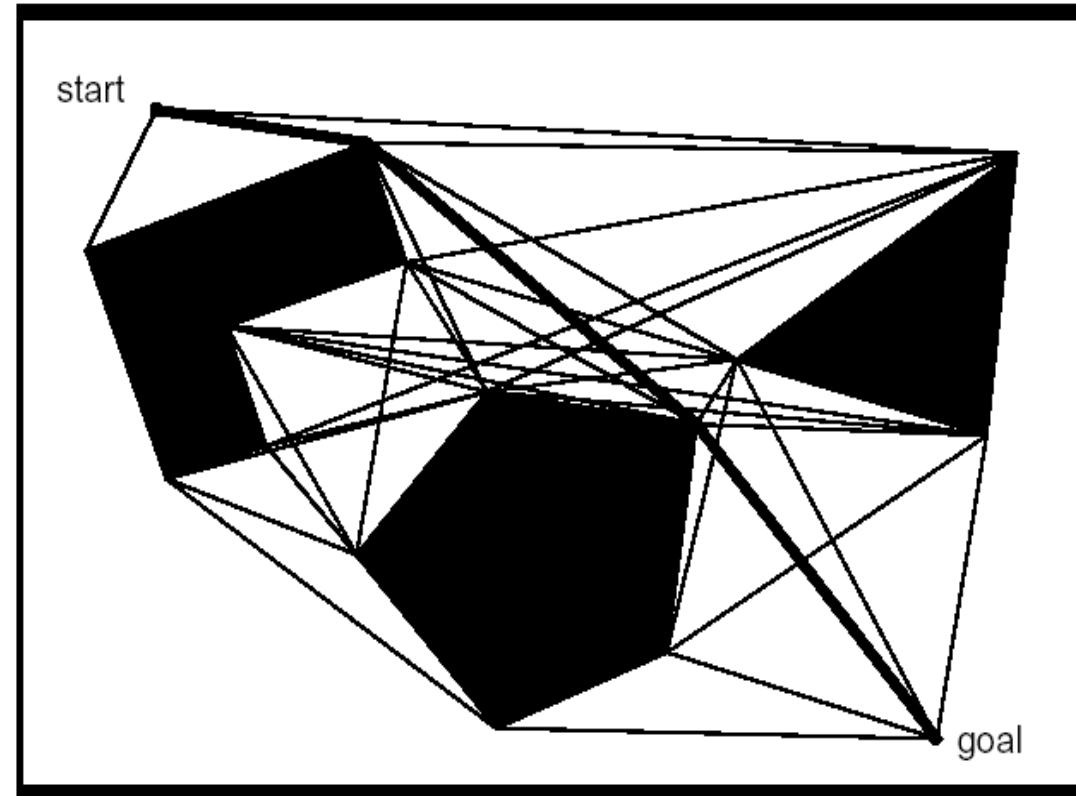
- Where to put the nodes?



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

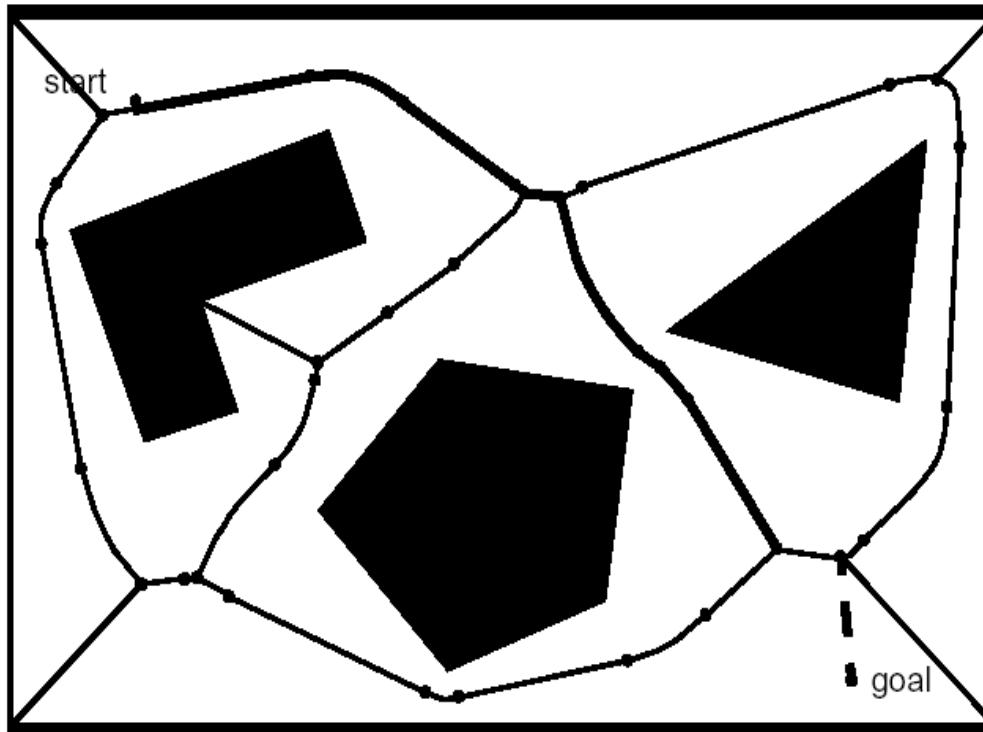


- Particularly suitable for polygon-like obstacles
- Shortest path length
- Grow obstacles to avoid collisions

Graph Construction: Visibility Graph

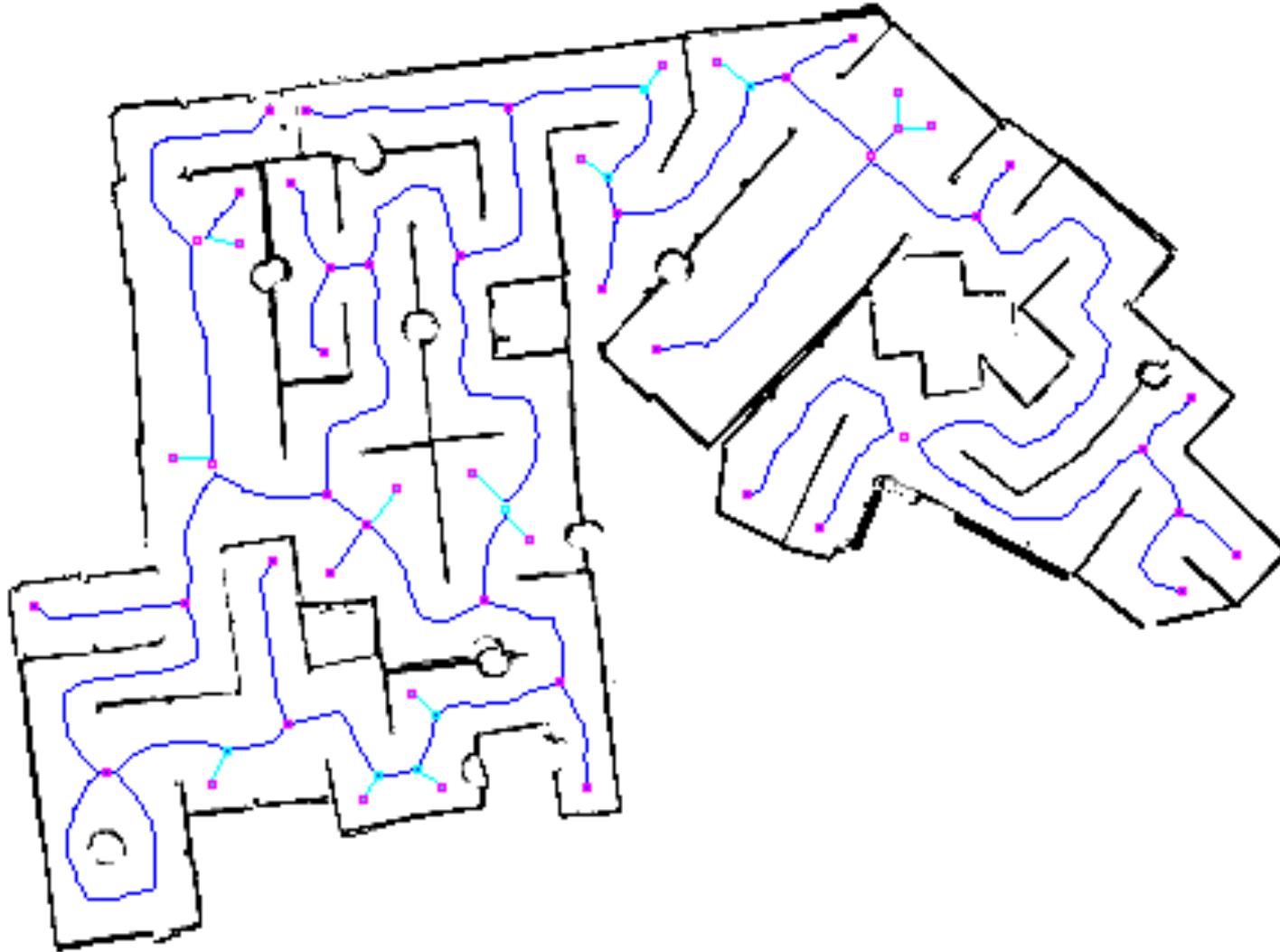
- Pros
 - The found path is optimal because it is the shortest length path
 - Implementation simple when obstacles are polygons
- Cons
 - The solution path found by the visibility graph tend to take the robot as close as possible to the obstacles: the common solution is to grow obstacles by more than 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

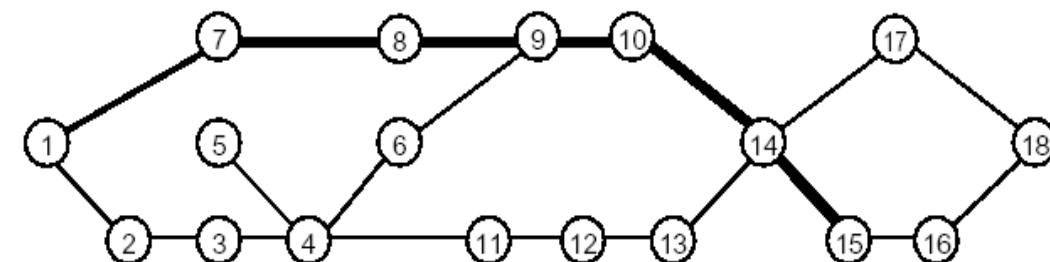
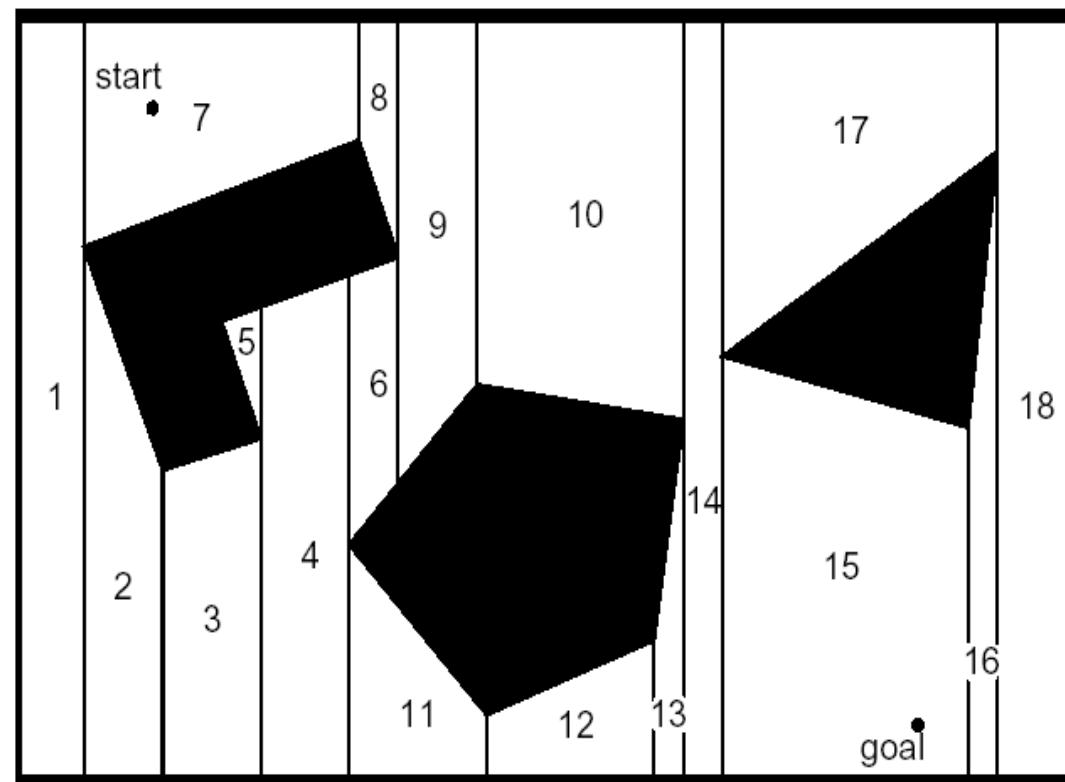
Topology Graph



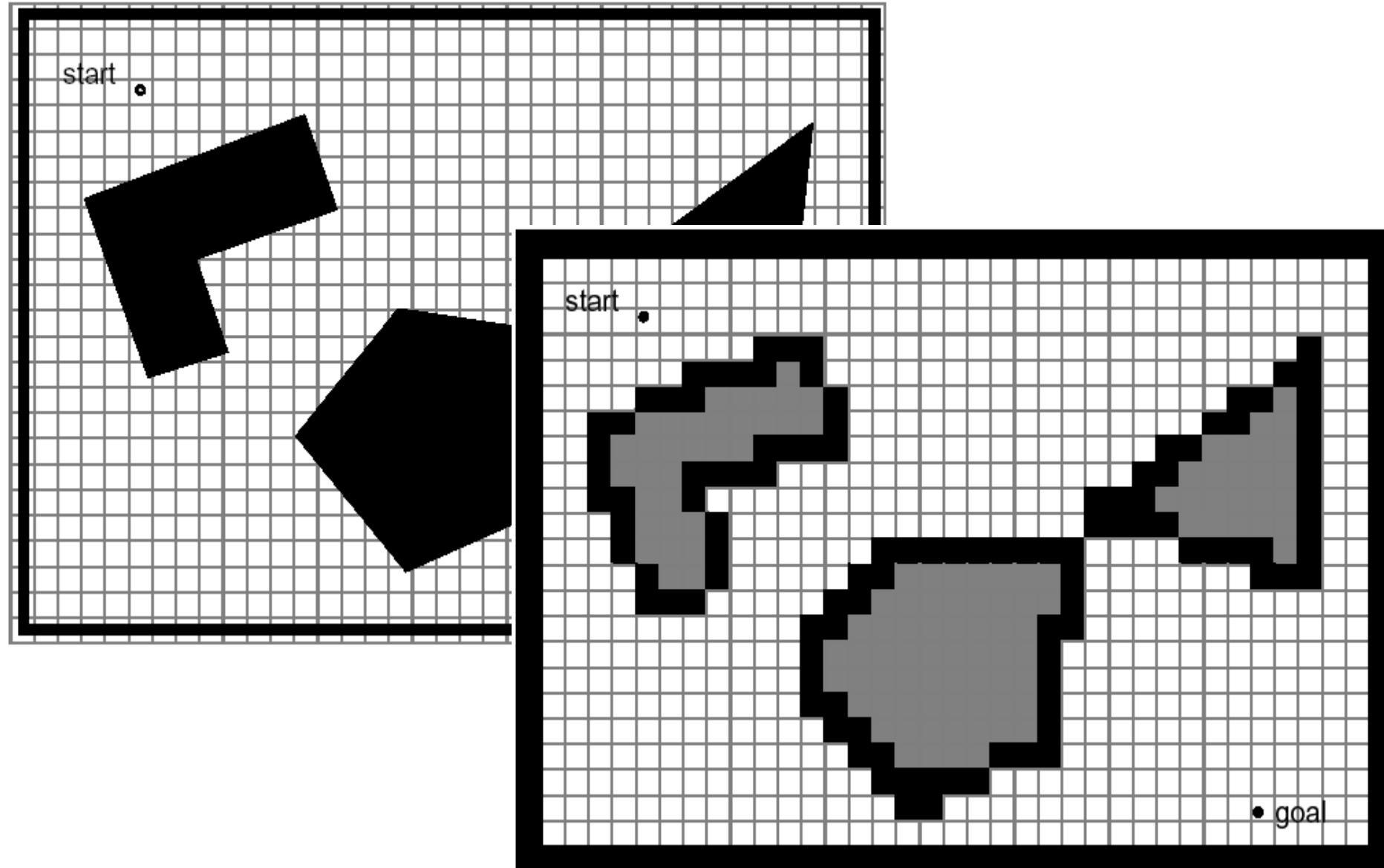
Graph Construction: Voronoi Diagram

- 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

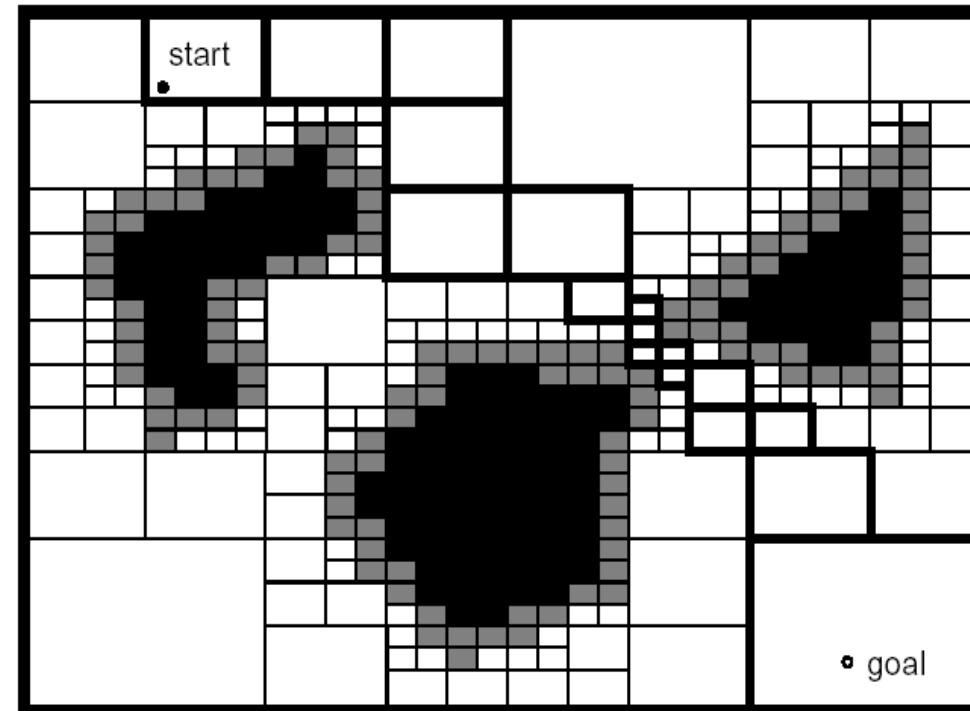
Graph Construction: Exact Cell Decomposition (2/4)



Graph Construction: Approximate Cell Decomposition (3/4)



Graph Construction: Adaptive Cell Decomposition (4/4)



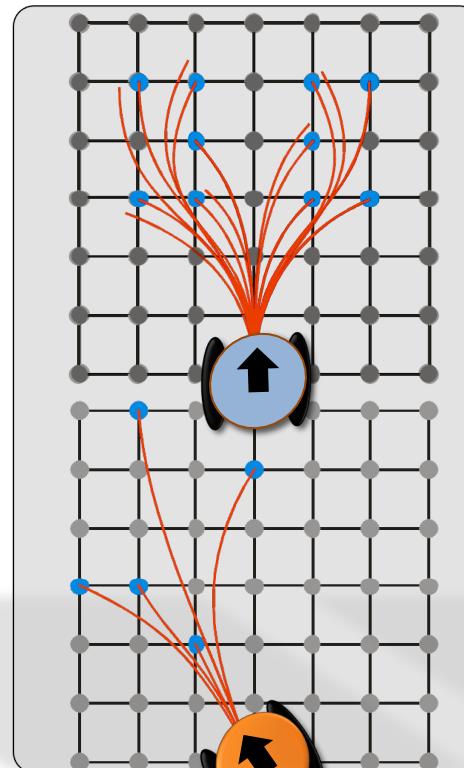
- Close relationship with map representation (Quadtree)!

Graph Construction: State Lattice Design (1/2)

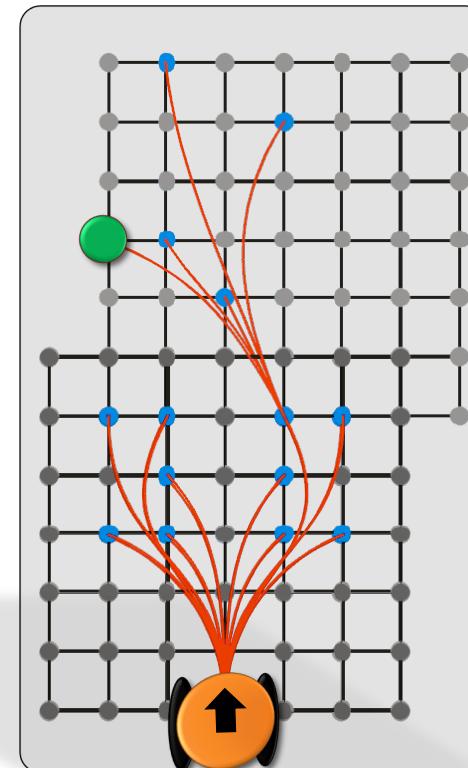
- Enforces **edge feasibility**



Offline:
Motion Model



Offline:
Lattice Gen.

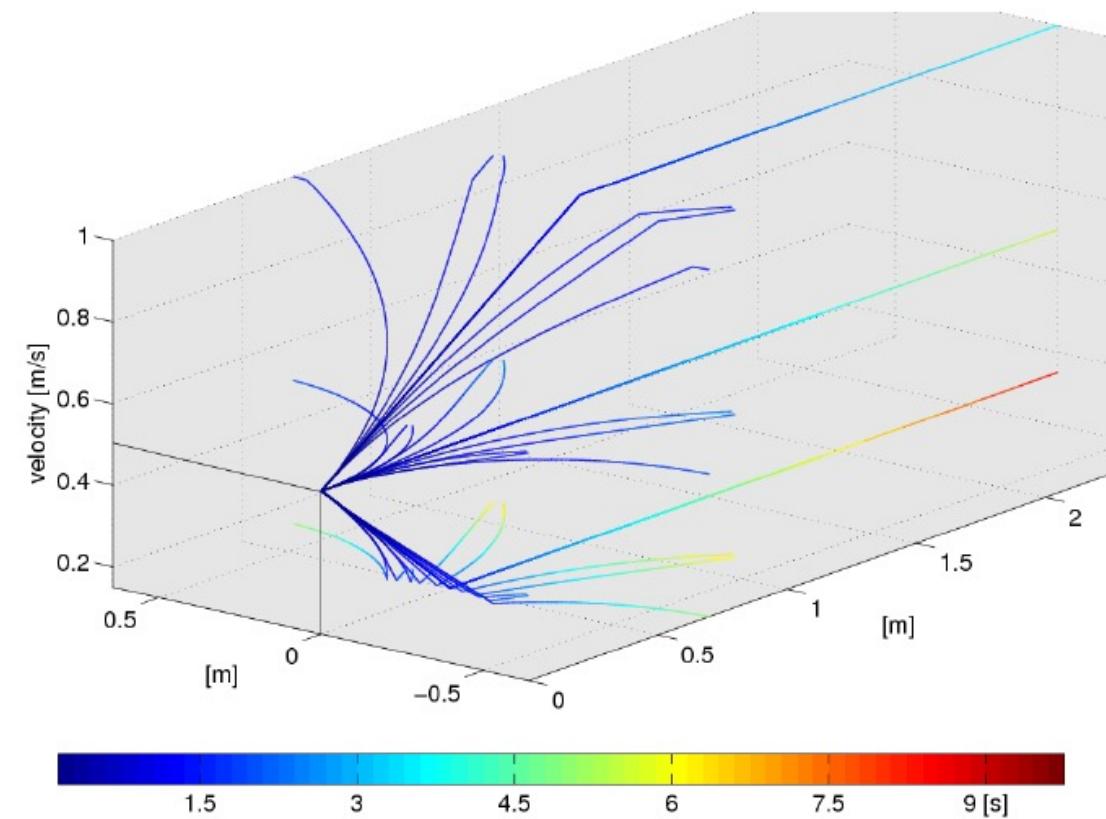
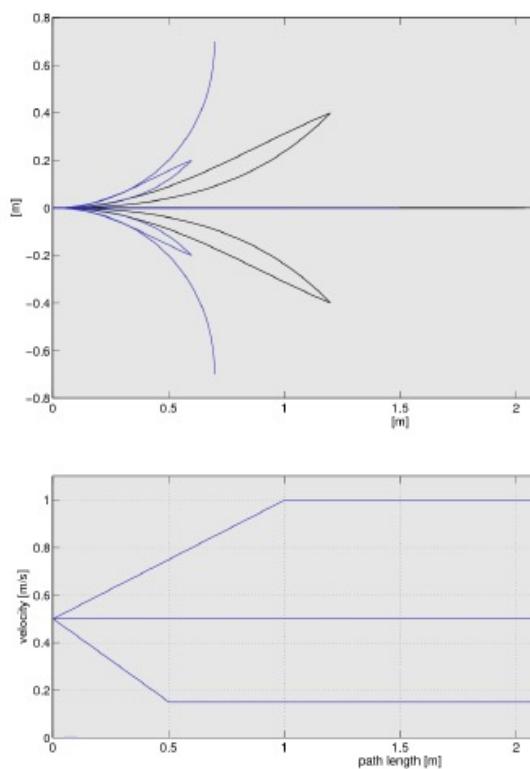


Online:
Incremental Graph
Constr.

Graph Construction: State Lattice Design (2/2)

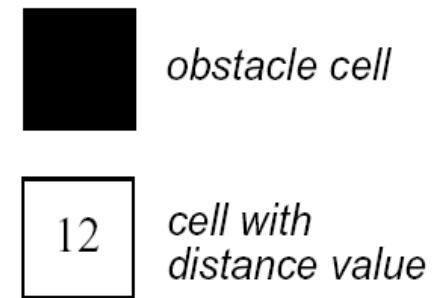
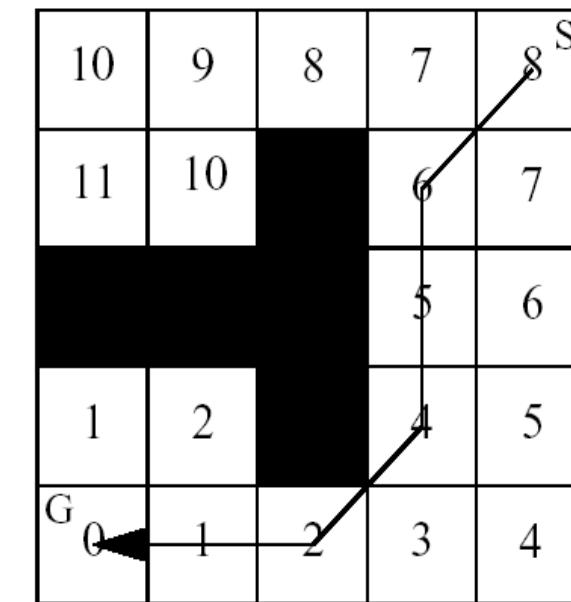
Martin Rufli

- State lattice encodes only kinematically feasible edges



Deterministic Graph Search

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



ADMIN

HW4 is out

- Start early ...
- First 4 tasks are config only – build a Gazebo simulation and use ROS2 Navigation 2 to perform simple autonomy
- Task 5: Implement frontier exploration yourself 😊

Project

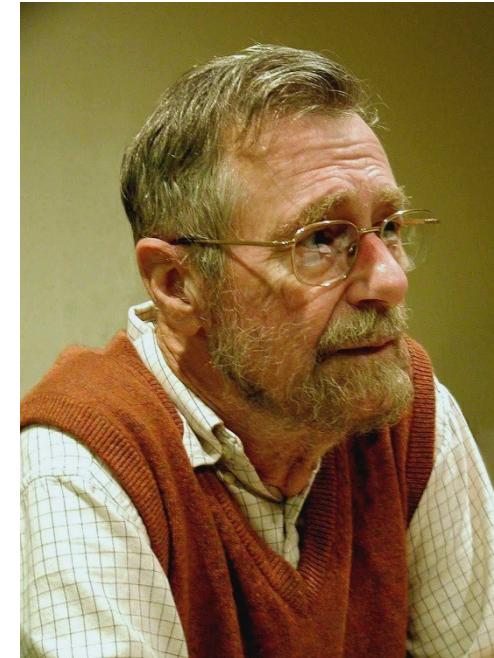
- May 26th, 22:00: due date for the intermediate project report
- June 25th, 22:00: due date for the final demo. (You are welcome to do it earlier!)
- June 28th, 22:00: due date for the final report. (You are welcome to finish it earlier!)
- June 28th, 22:00: due date for the webpage. (You are welcome to finish it earlier!)
- Schedule Project meeting with me for tomorrow, Monday or Tuesday (April 26, April 29 or April 30)!
 - You contact me!
 - Counts towards your “attend weekly project meeting” score!

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!

DIJKSTRA'S ALGORITHM

EDSGER WYBE DIJKSTRA



1930 - 2002

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

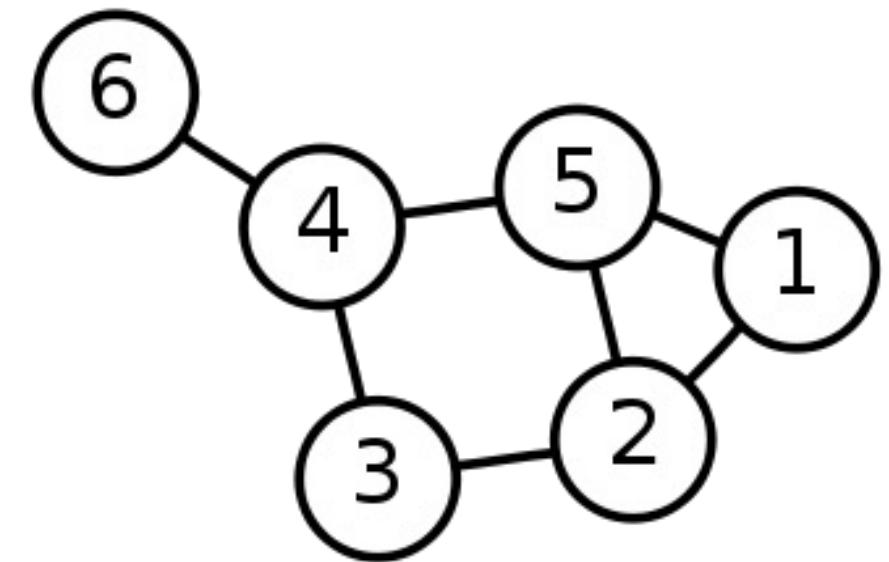
<http://www.cs.utexas.edu/~EWD/>

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  
for all v ∈ V-{s}  
    do dist[v] ← ∞  
S ← ∅  
Q← V  
while Q ≠ ∅  
do u ← mindistance(Q, dist)  
    S←S∪{u}  
    for all v ∈ neighbors[u]  
        do if dist[v] > dist[u] + w(u, v)  
            then d[v] ←d[u] + w(u, v)  
(if desired, add traceback code)  
return dist
```

(distance to source vertex is zero)

(set all other distances to infinity)

(S , the set of visited vertices is initially empty)

(Q , the queue initially contains all vertices)

(while the queue is not empty)

(select the element of Q with the min. distance)

(add u to list of visited vertices)

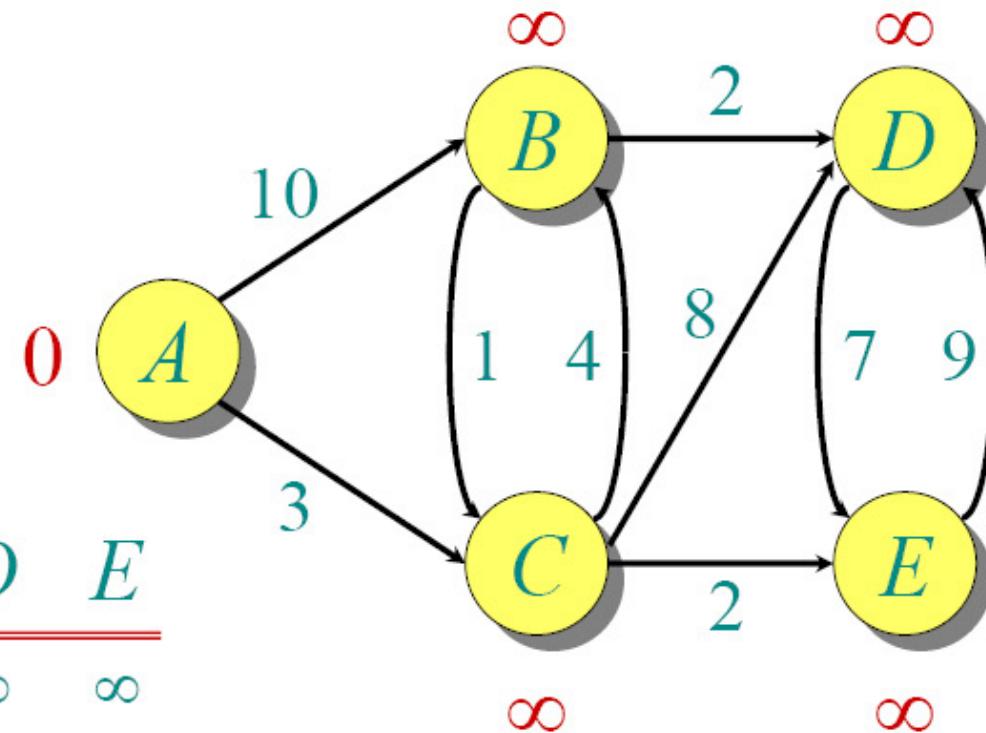
(if new shortest path found)

(set new value of shortest path)

Dijkstra Example

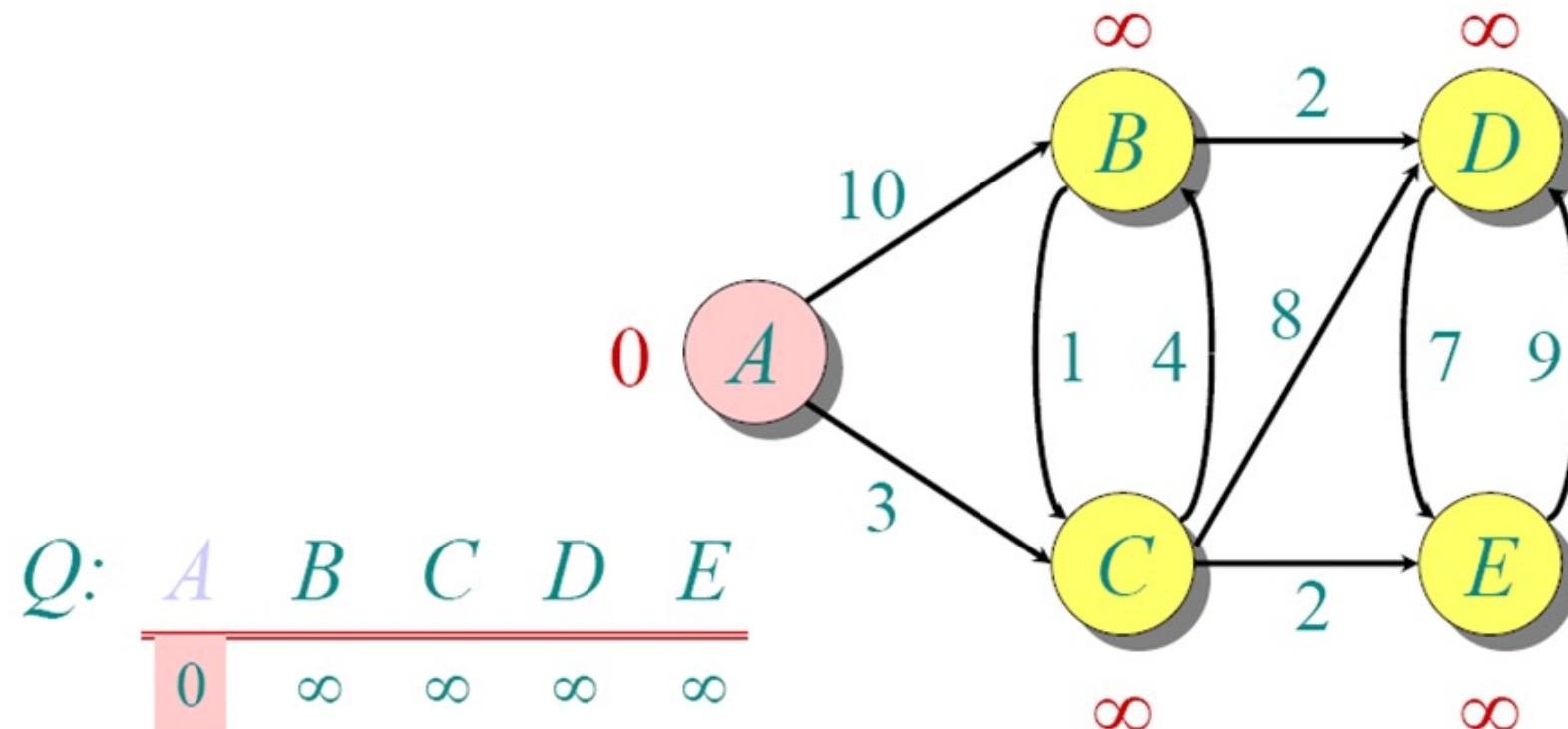
Initialize:

$$Q: \begin{array}{ccccc} A & B & C & D & E \\ \hline 0 & \infty & \infty & \infty & \infty \end{array}$$

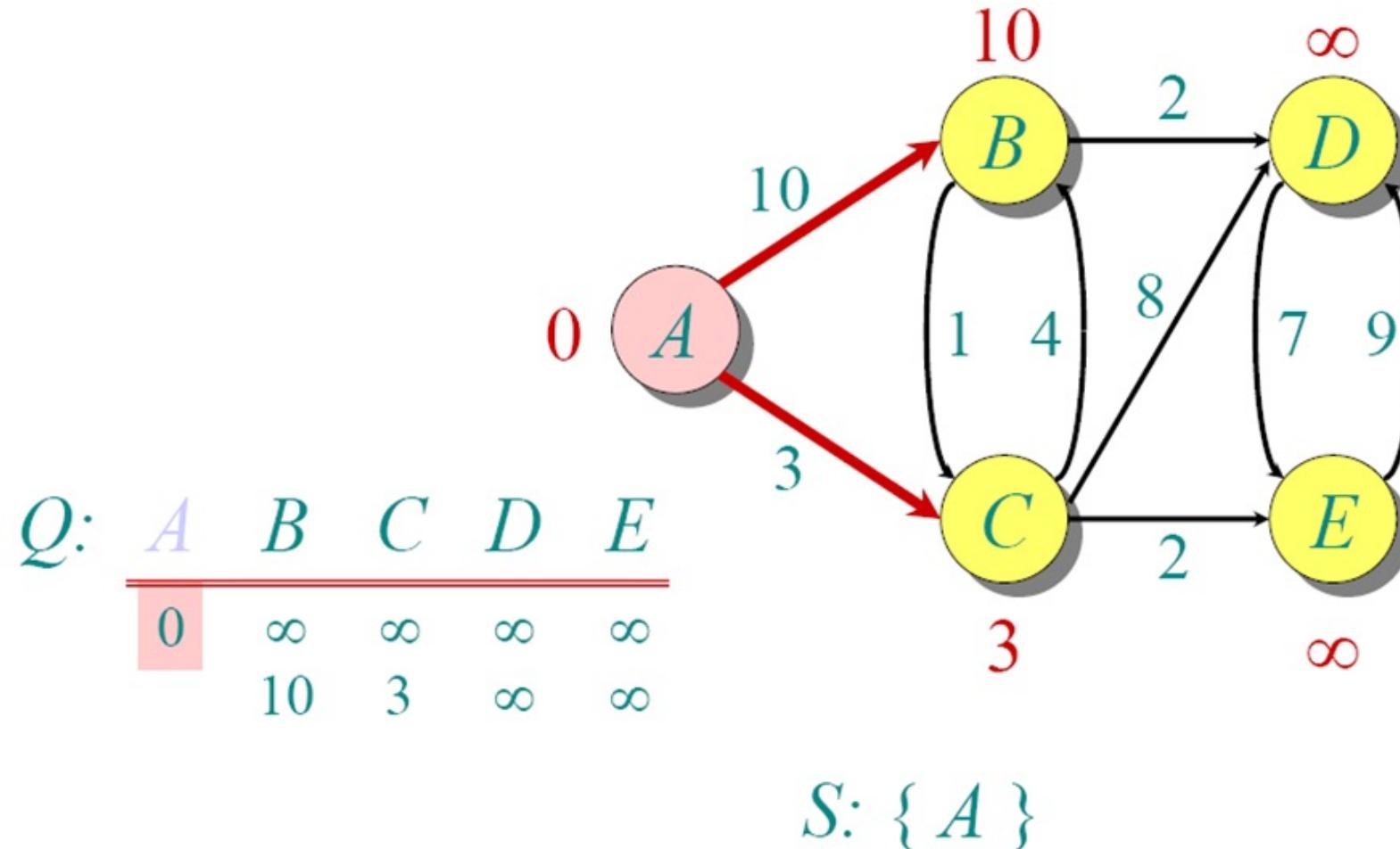


$$S: \{\}$$

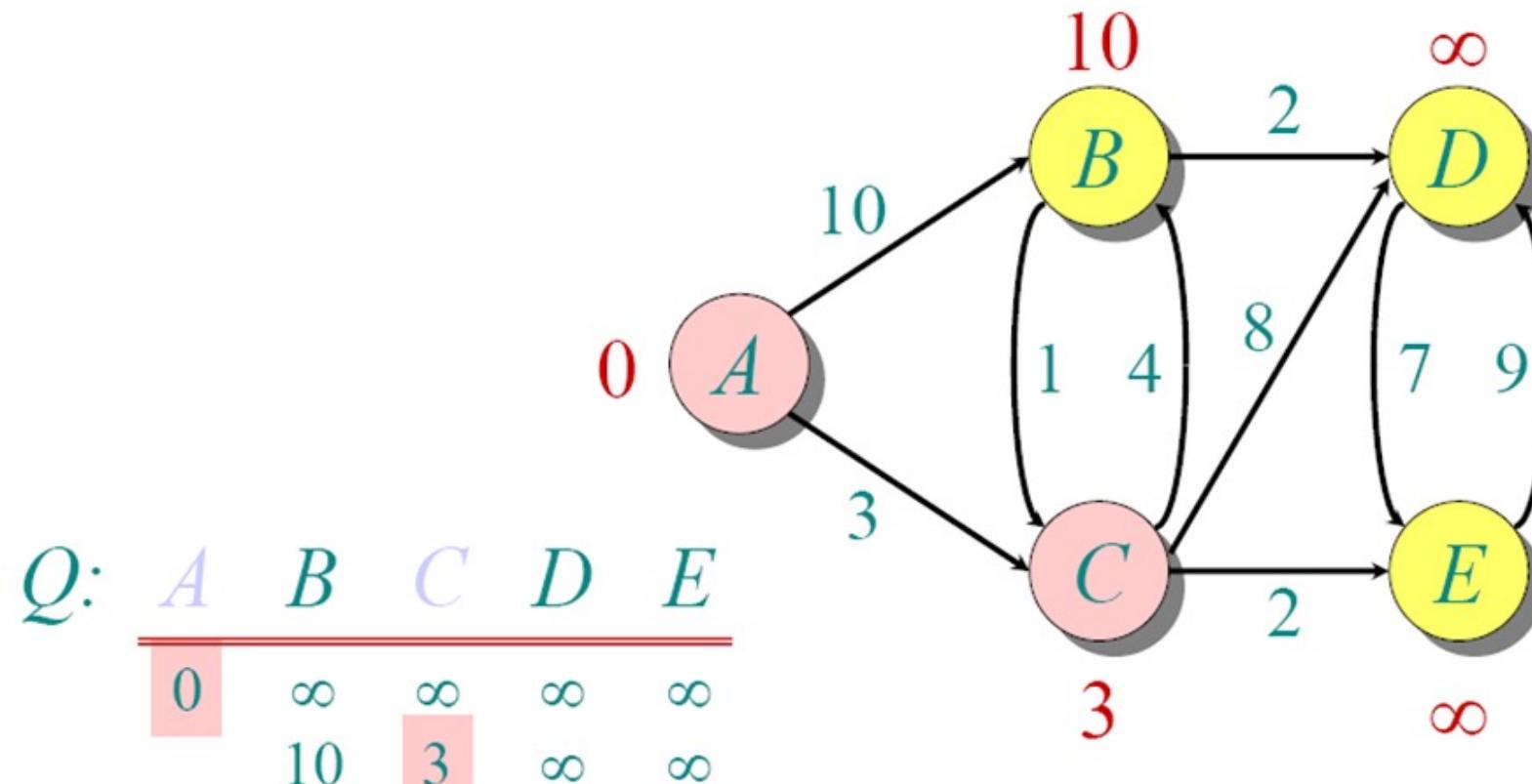
Dijkstra Example



Dijkstra Example

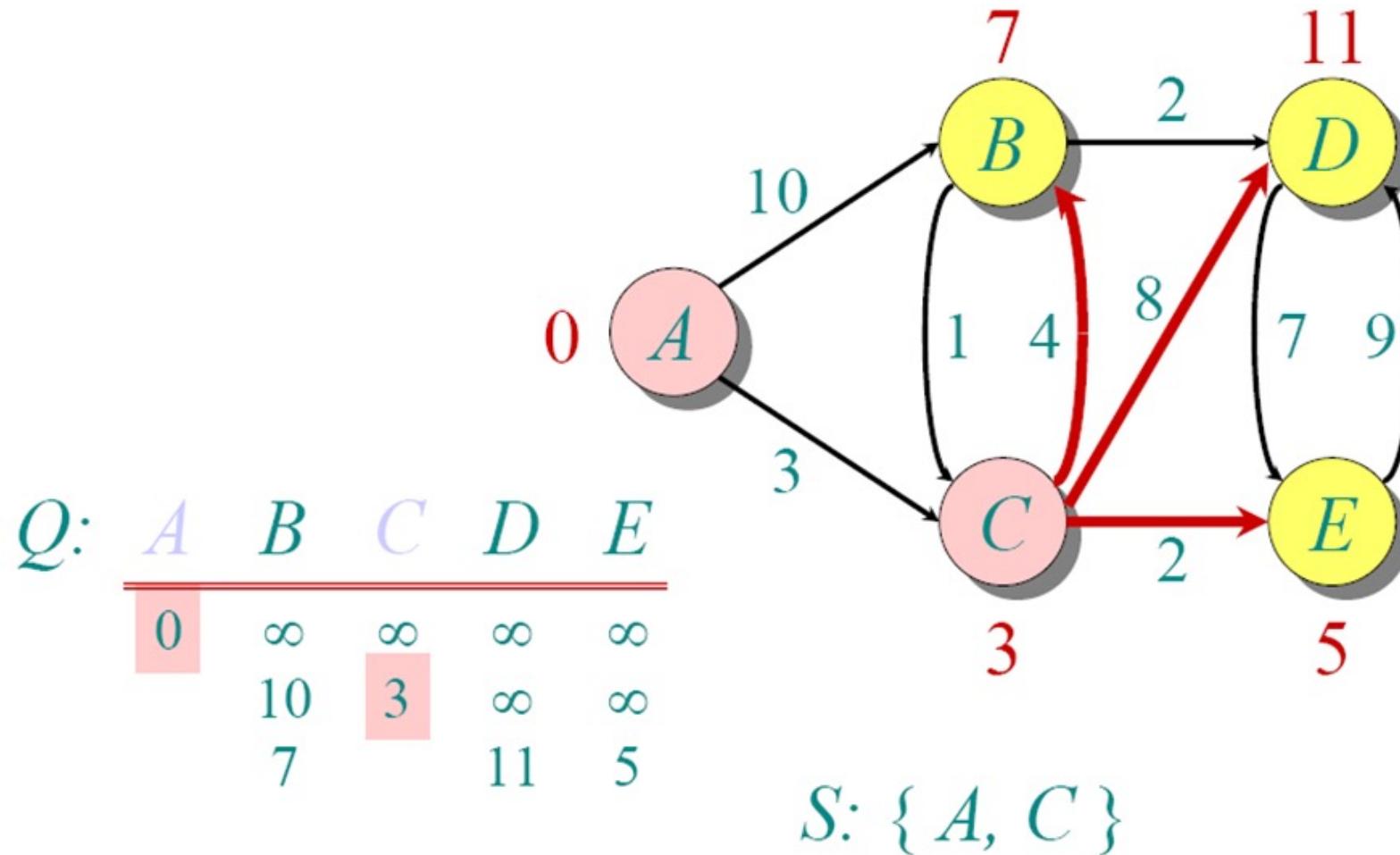


Dijkstra Example

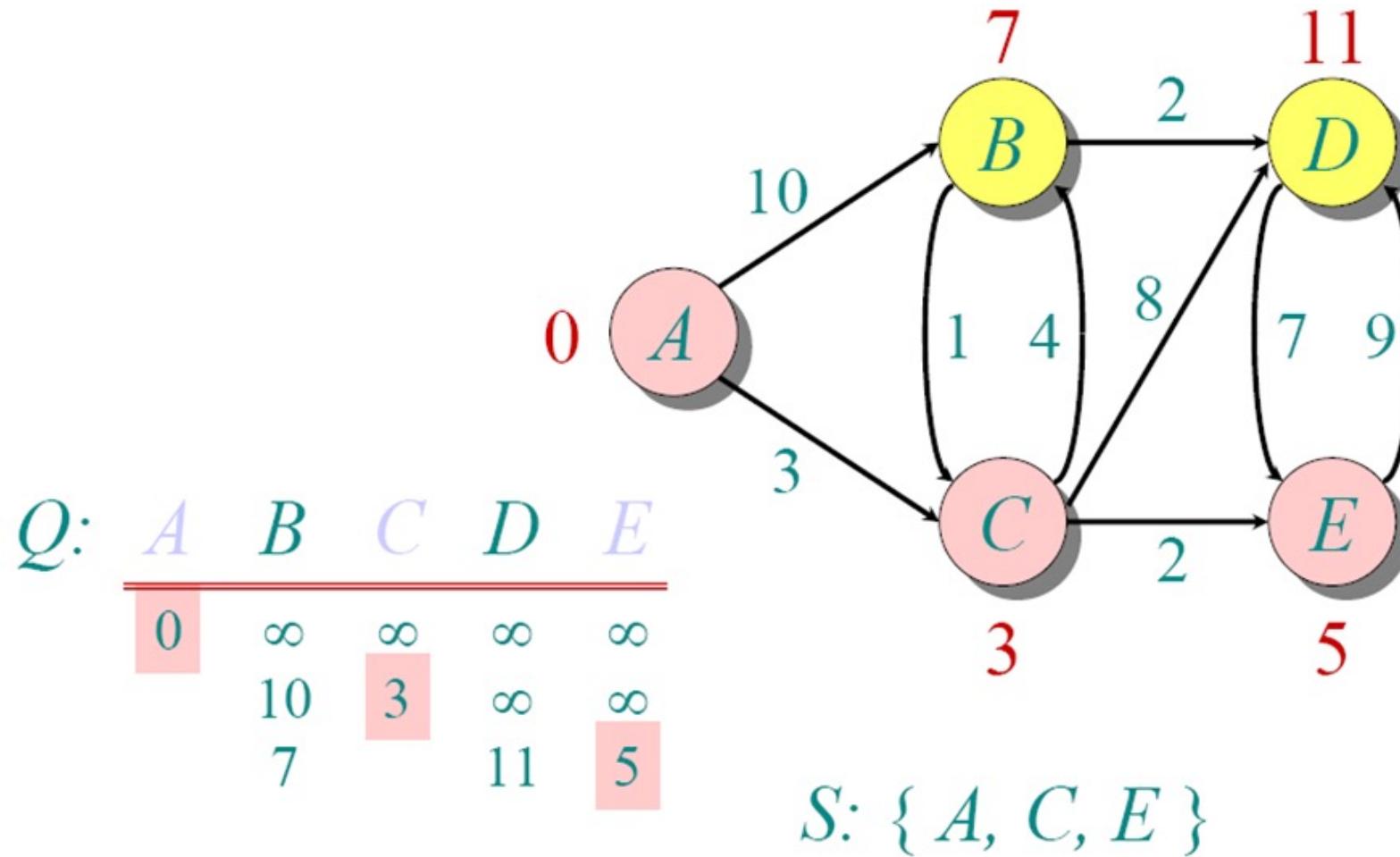


$S: \{ A, C \}$

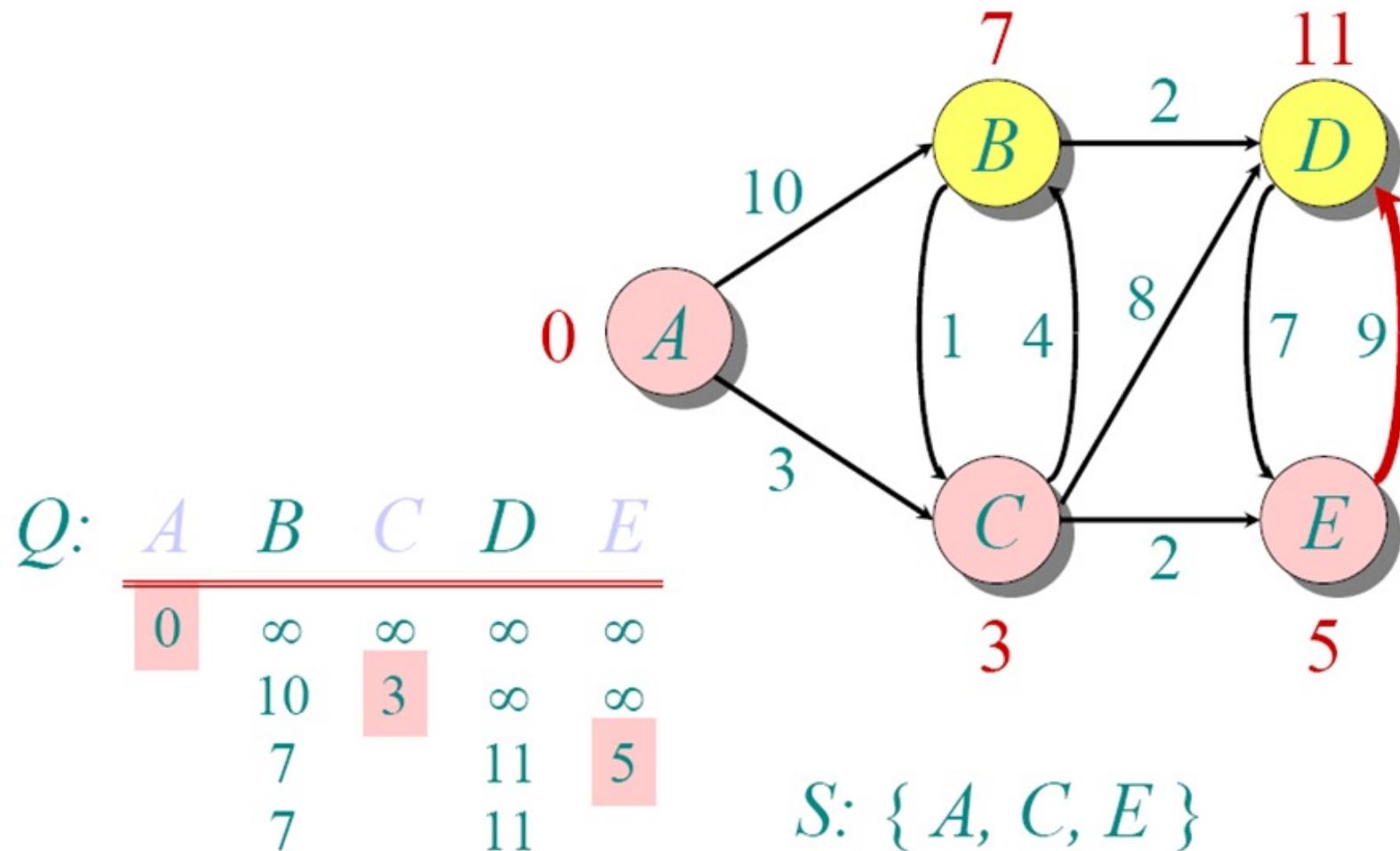
Dijkstra Example



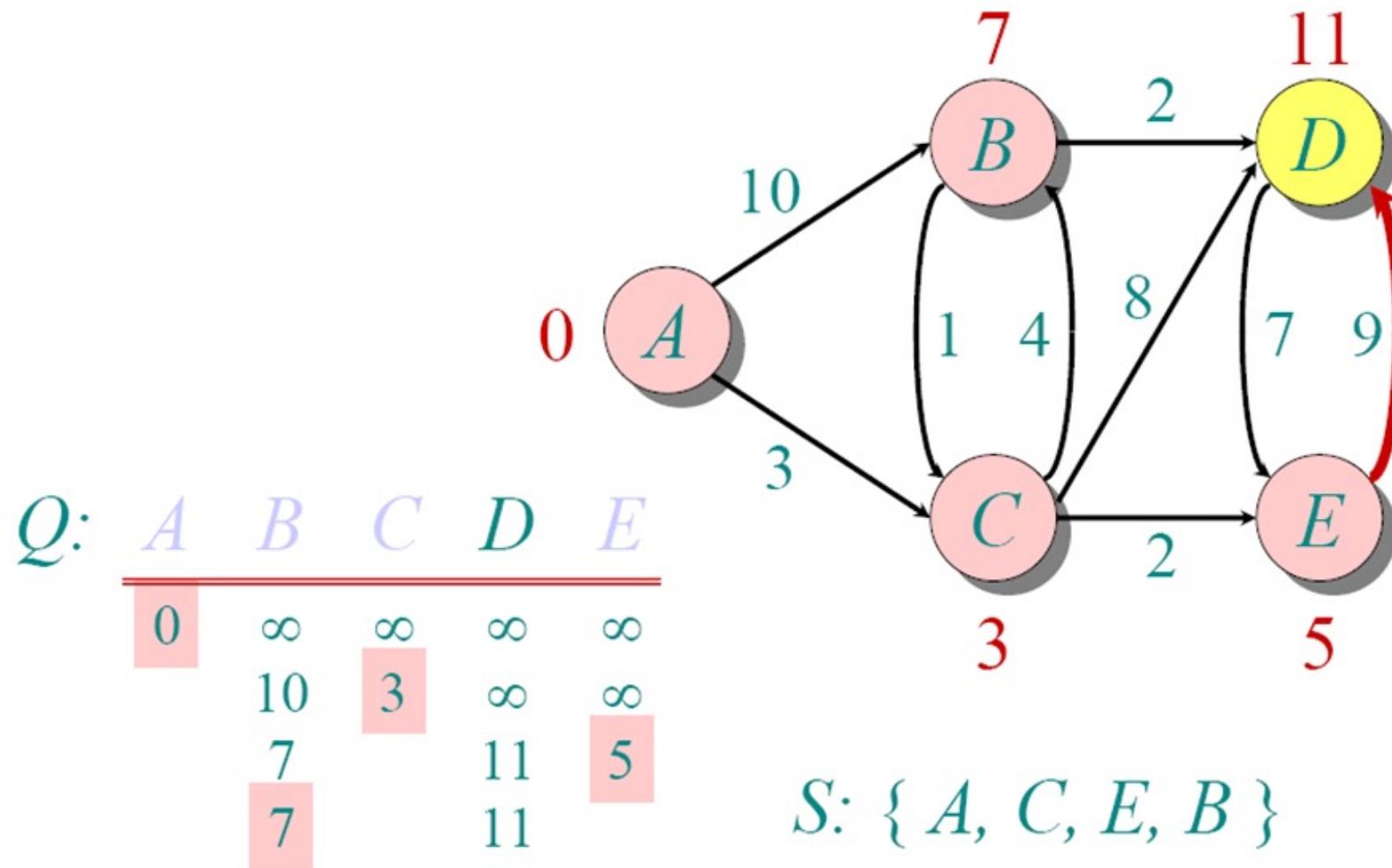
Dijkstra Example



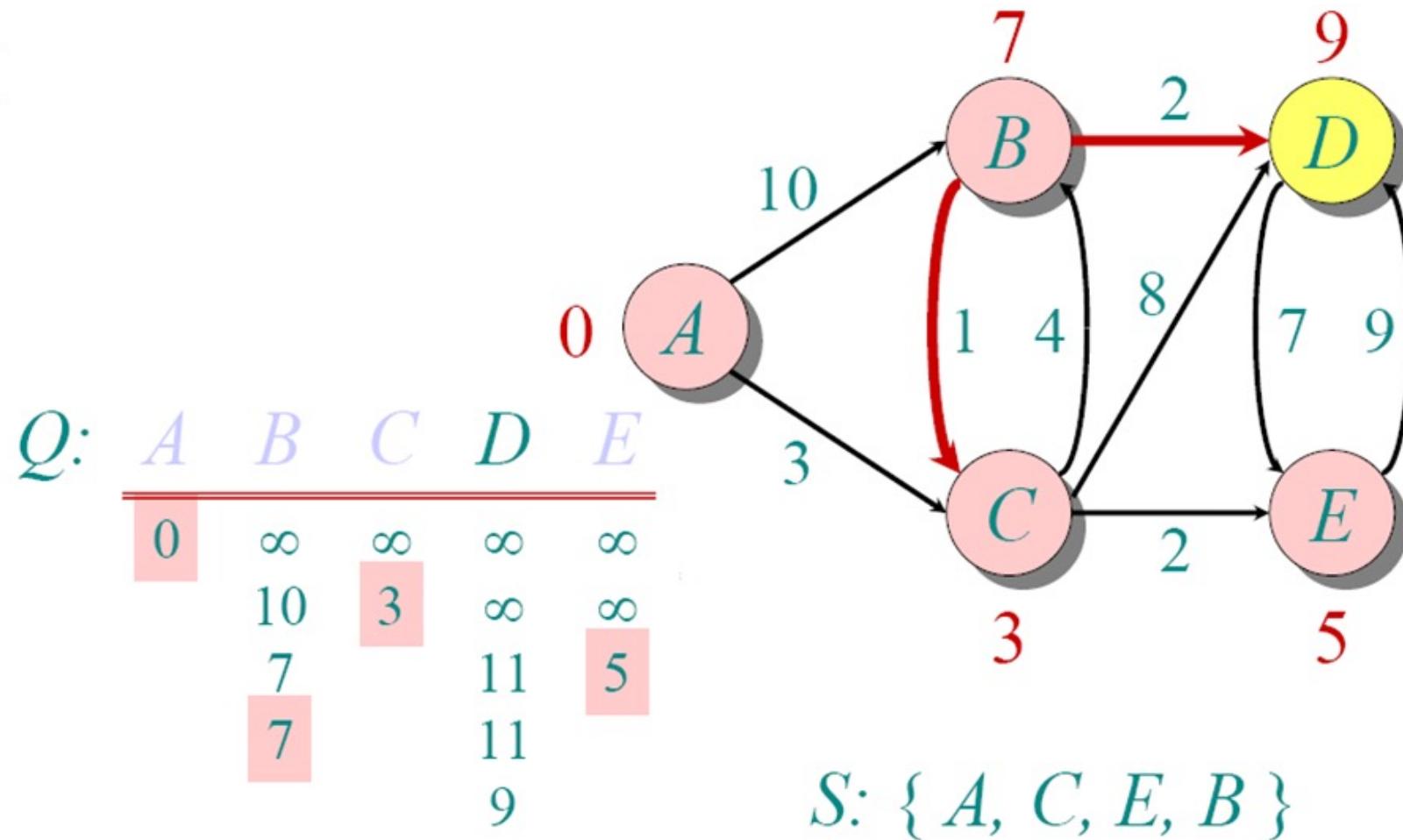
Dijkstra Example



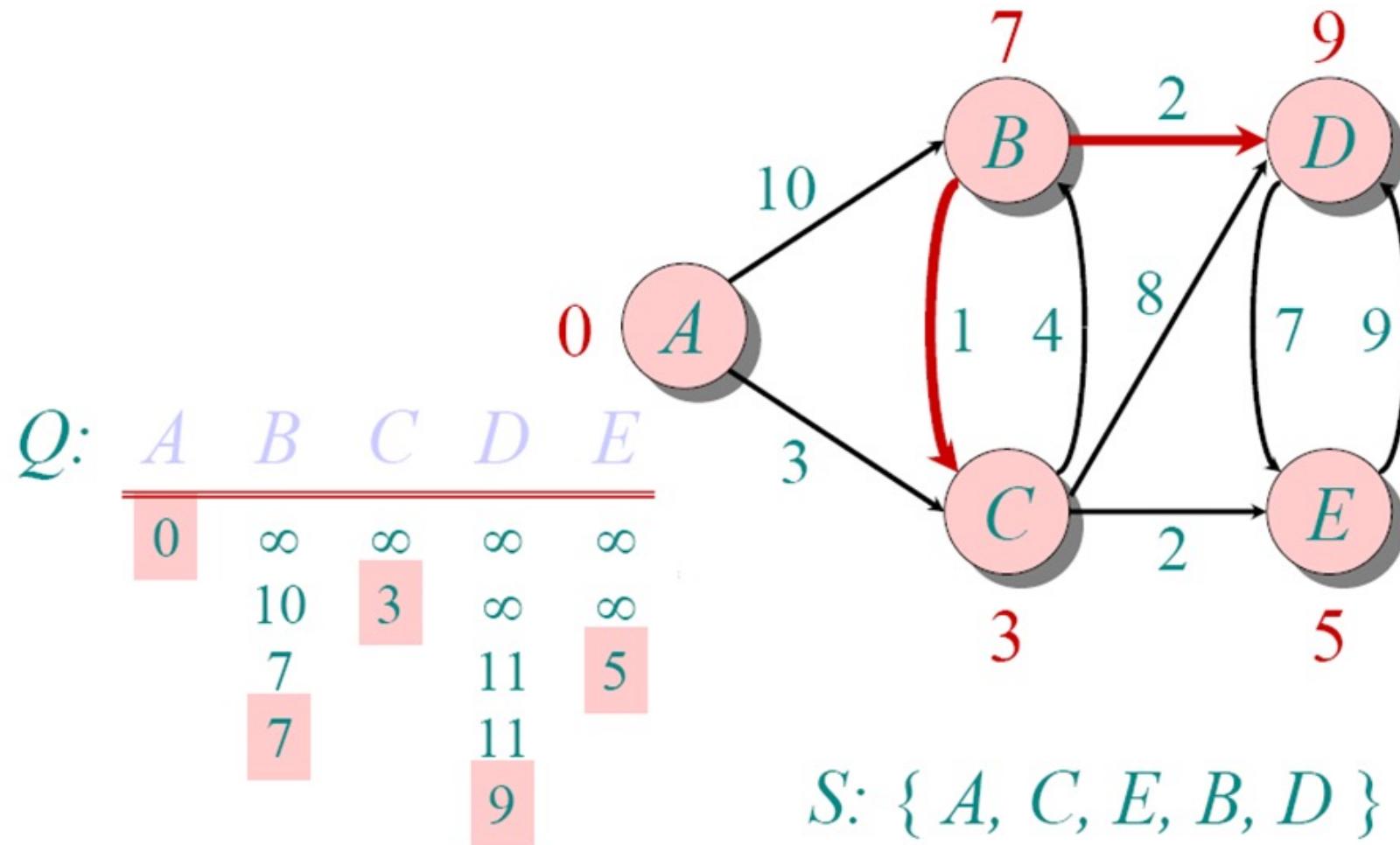
Dijkstra Example



Dijkstra Example



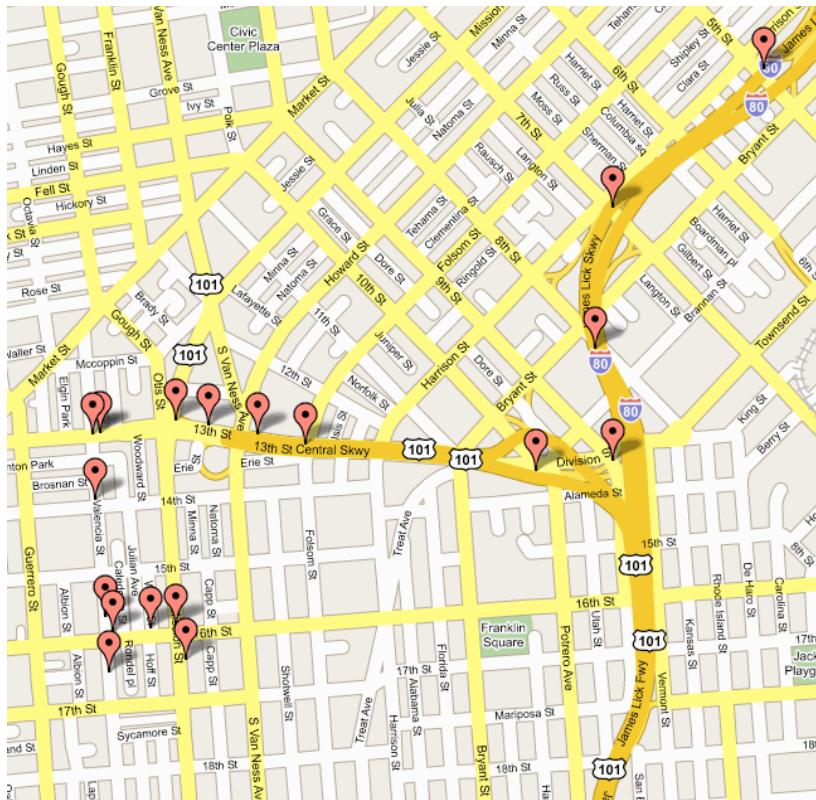
Dijkstra Example



APPLICATIONS OF DIJKSTRA'S ALGORITHM

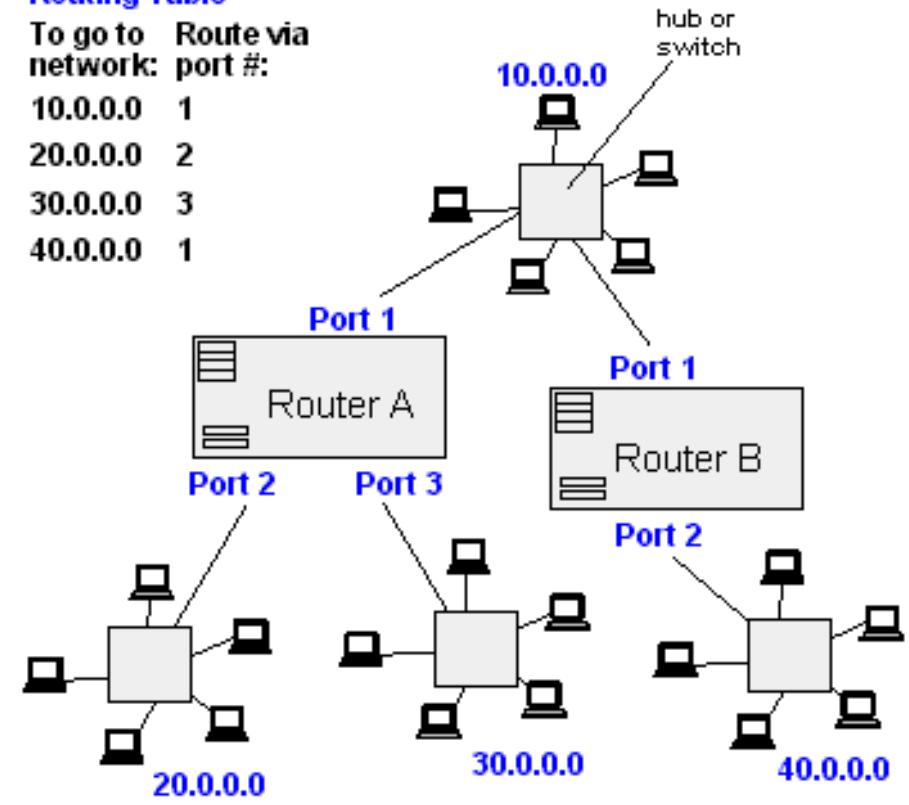
- Navigation Systems
 - Internet Routing

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



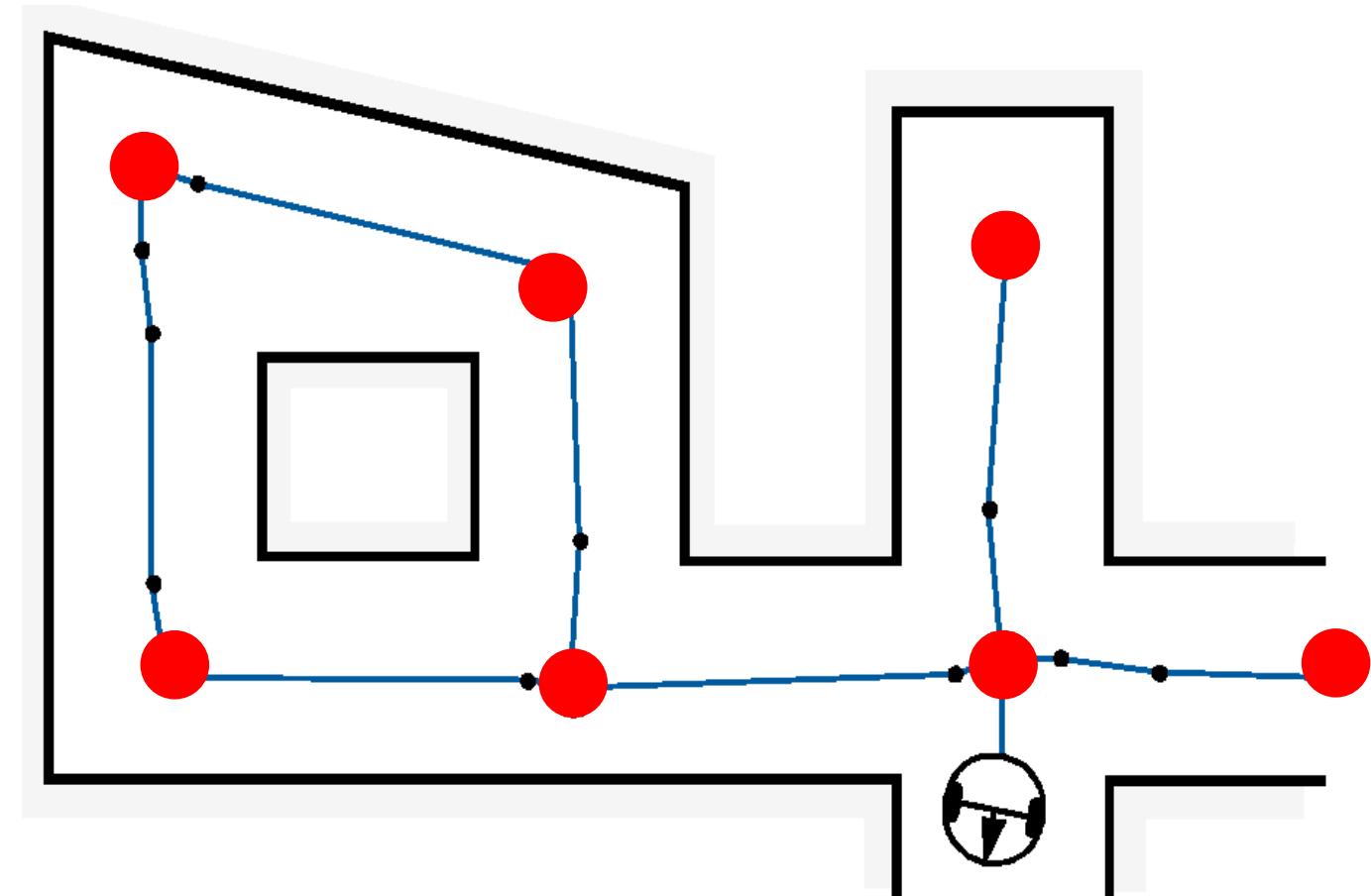
Router A Routing Table

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



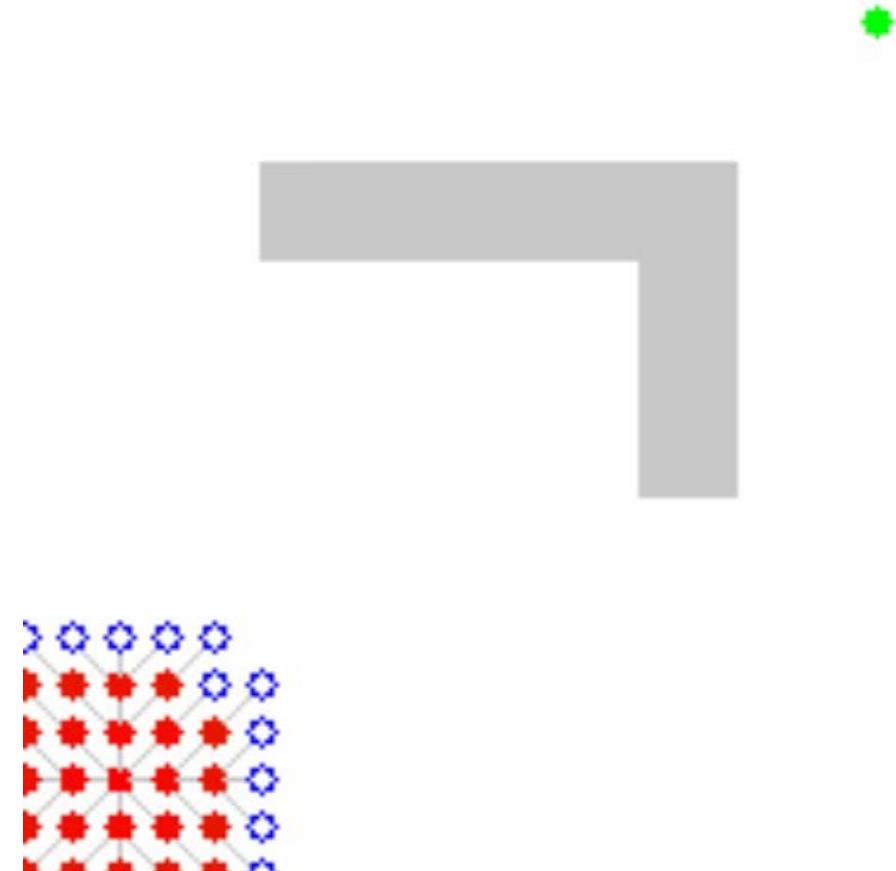
Dijkstra's Algorithm for Path Planning: Topological Maps

- 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



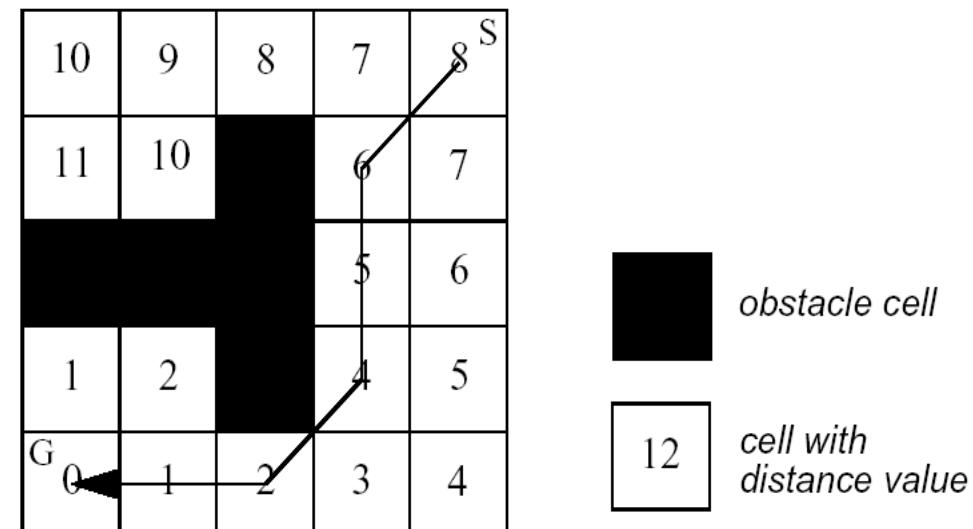
Dijkstra's Algorithm for Path Planning: Grid Maps

- 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

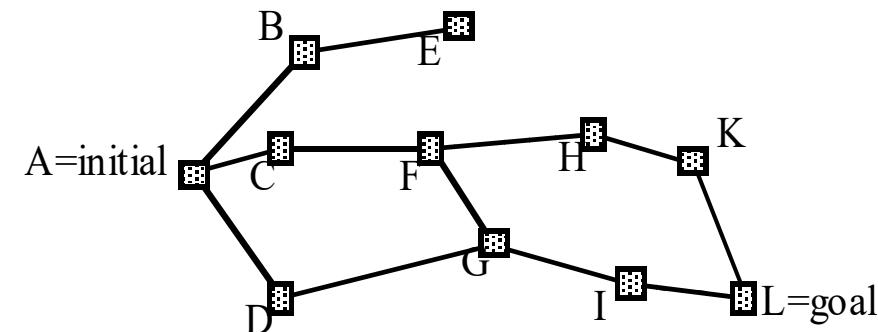
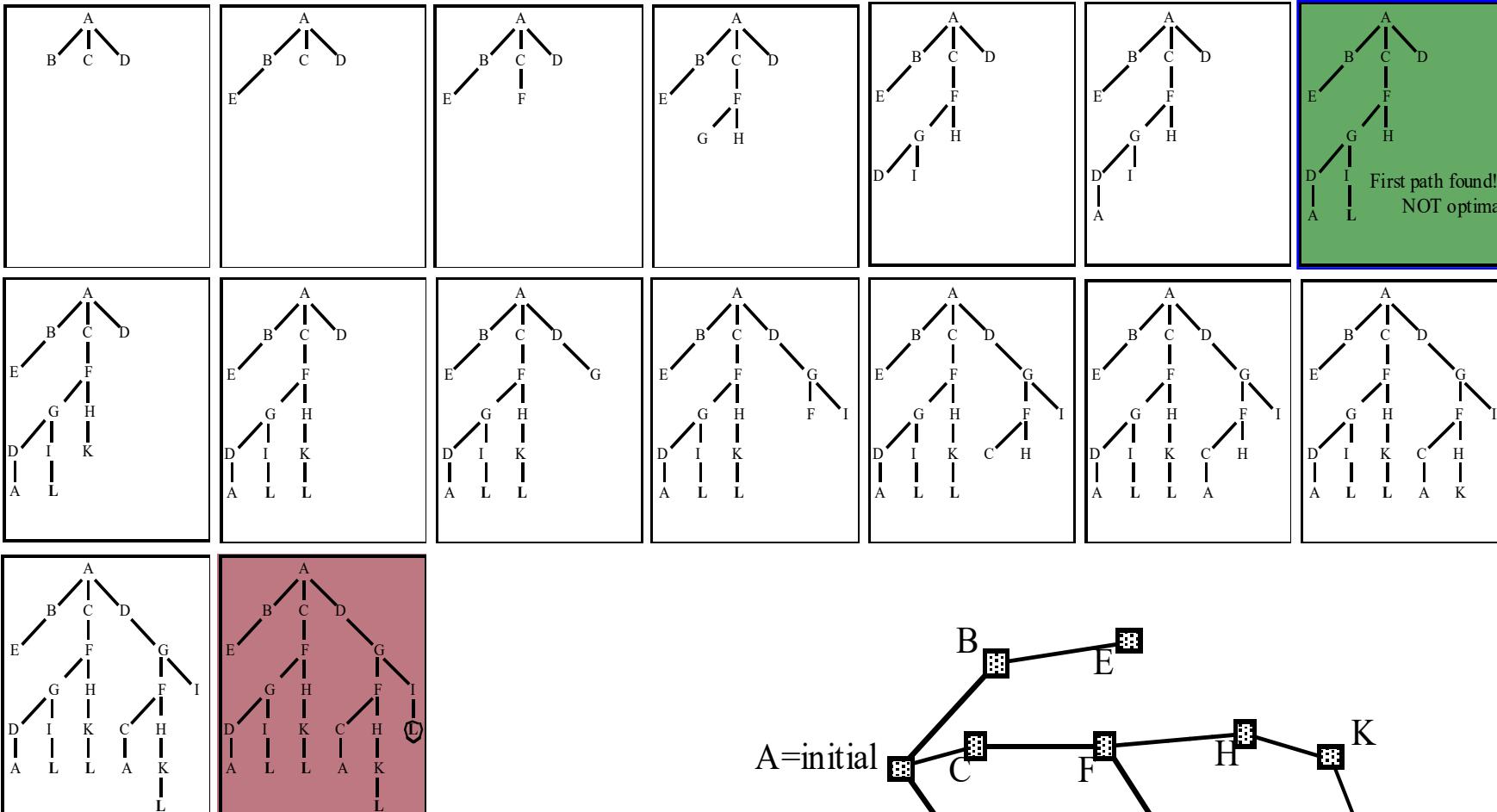


Graph Search Strategies: Breath-First Search

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

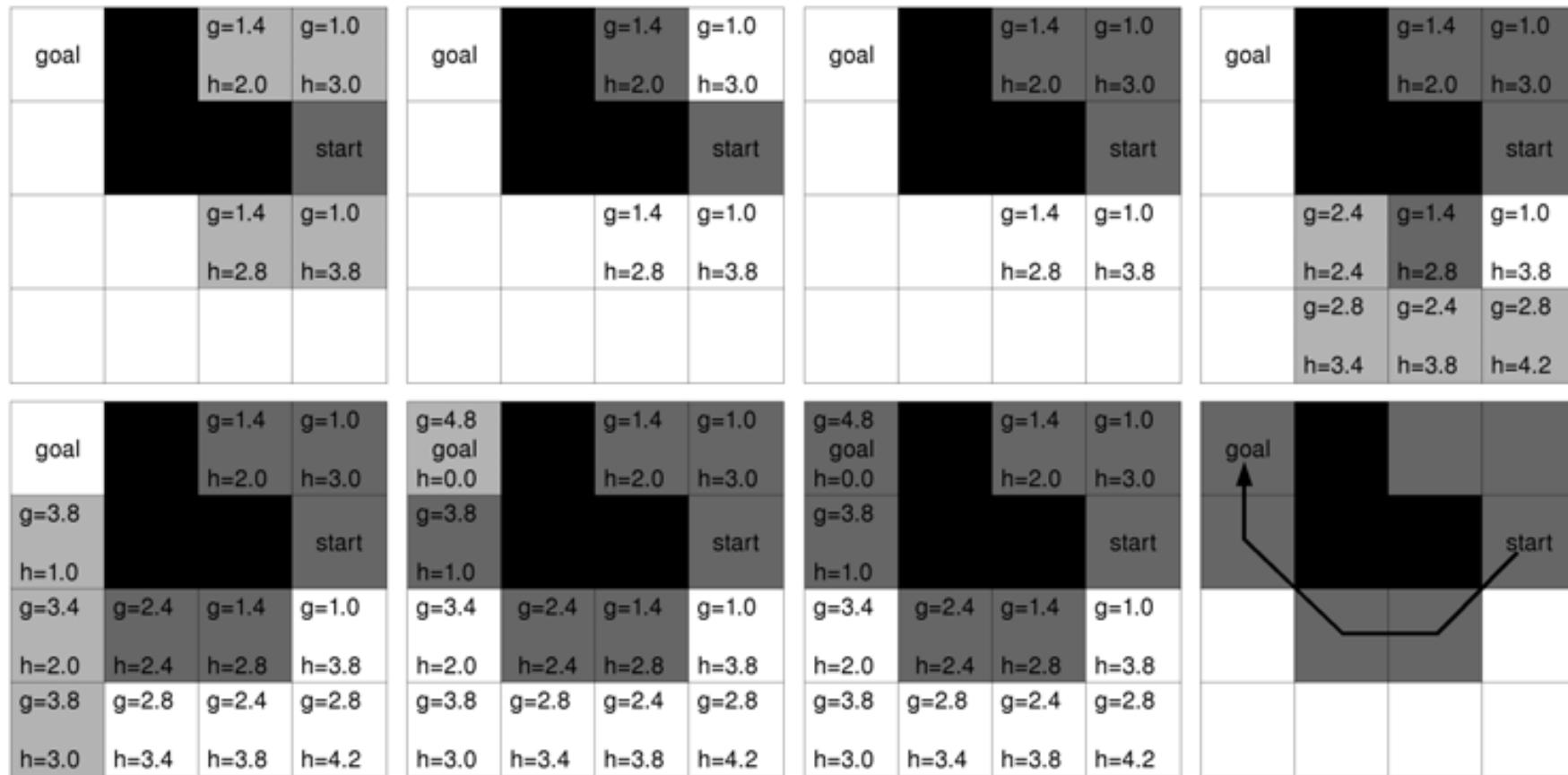


Graph Search Strategies: Depth-First Search



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 $h(n)$:

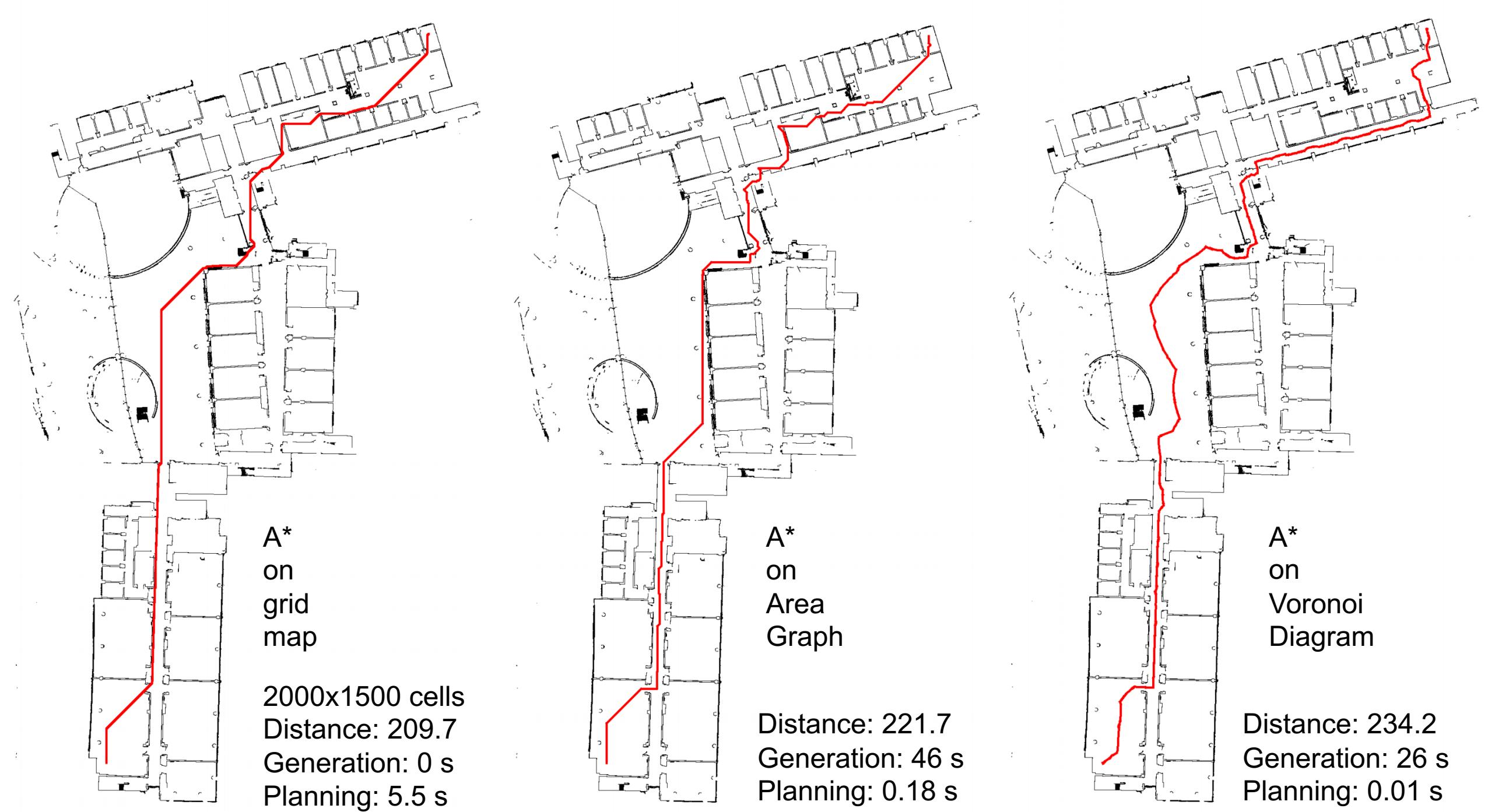
$$h(n) = \sqrt{(x_{goal} - x)^2 + (y_{goal} - y)^2} .$$

- Example: heuristic 5x geometric distance
- $h(n) = 0 \Rightarrow$ Dijkstra's Algorithm

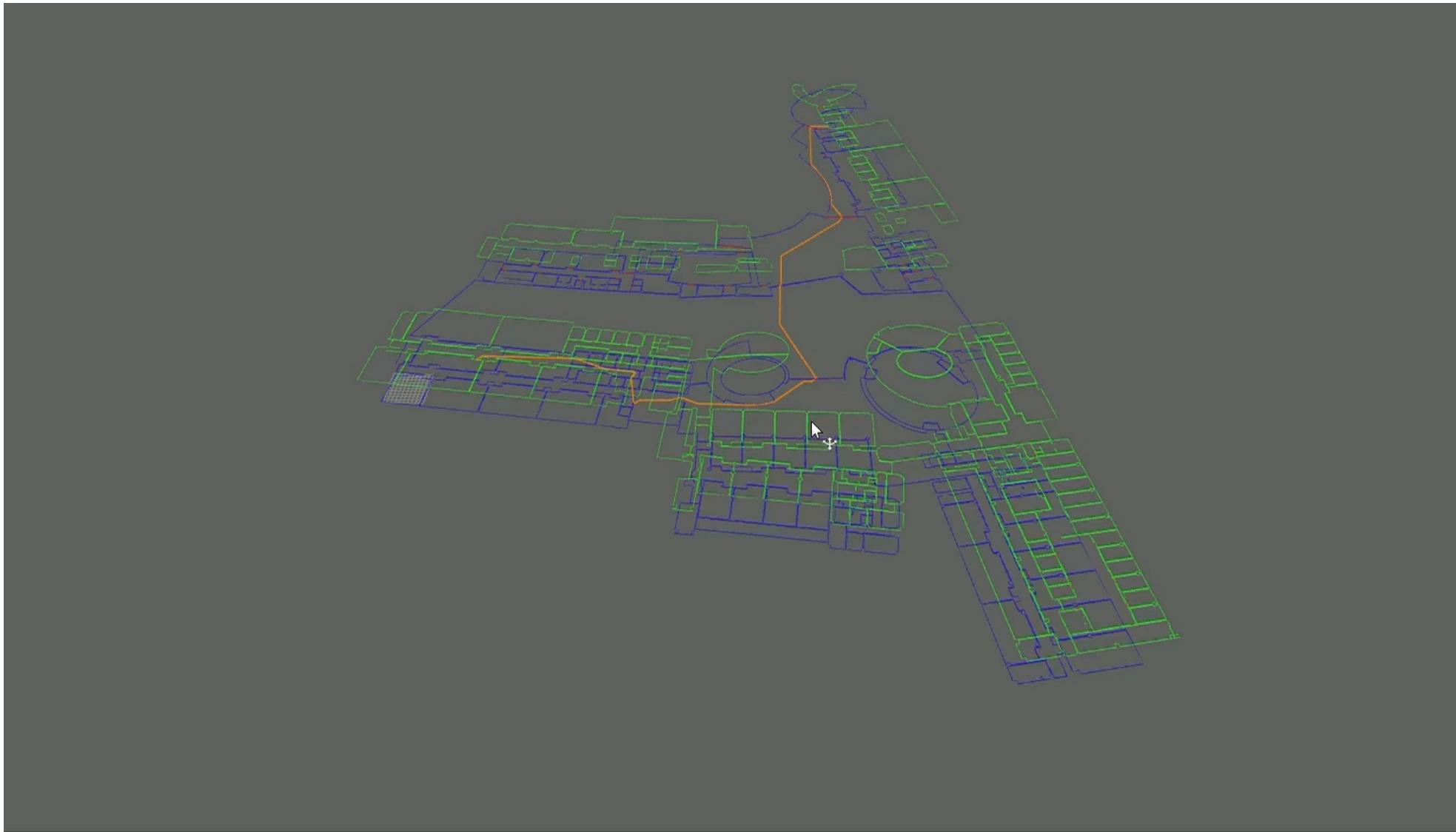


A*

- 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; ...

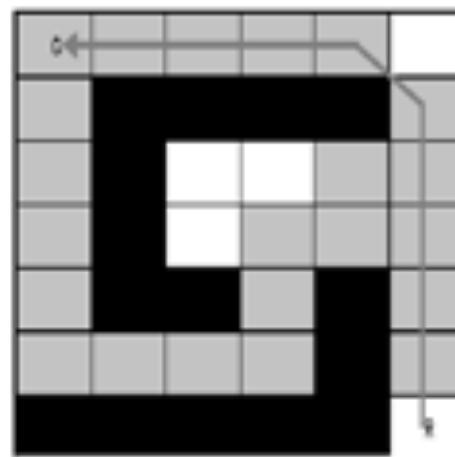
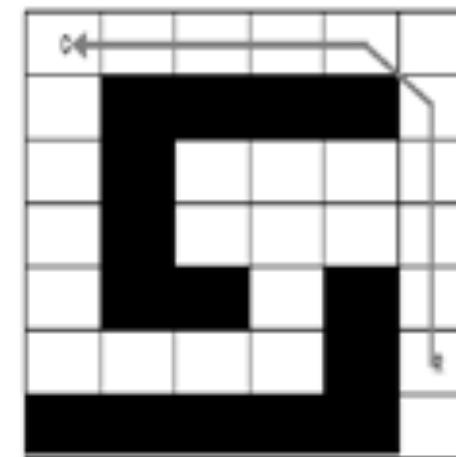
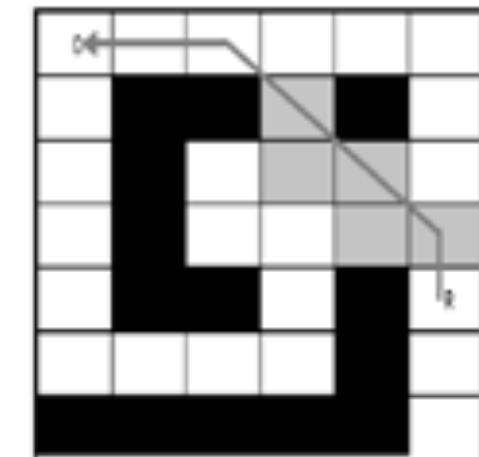


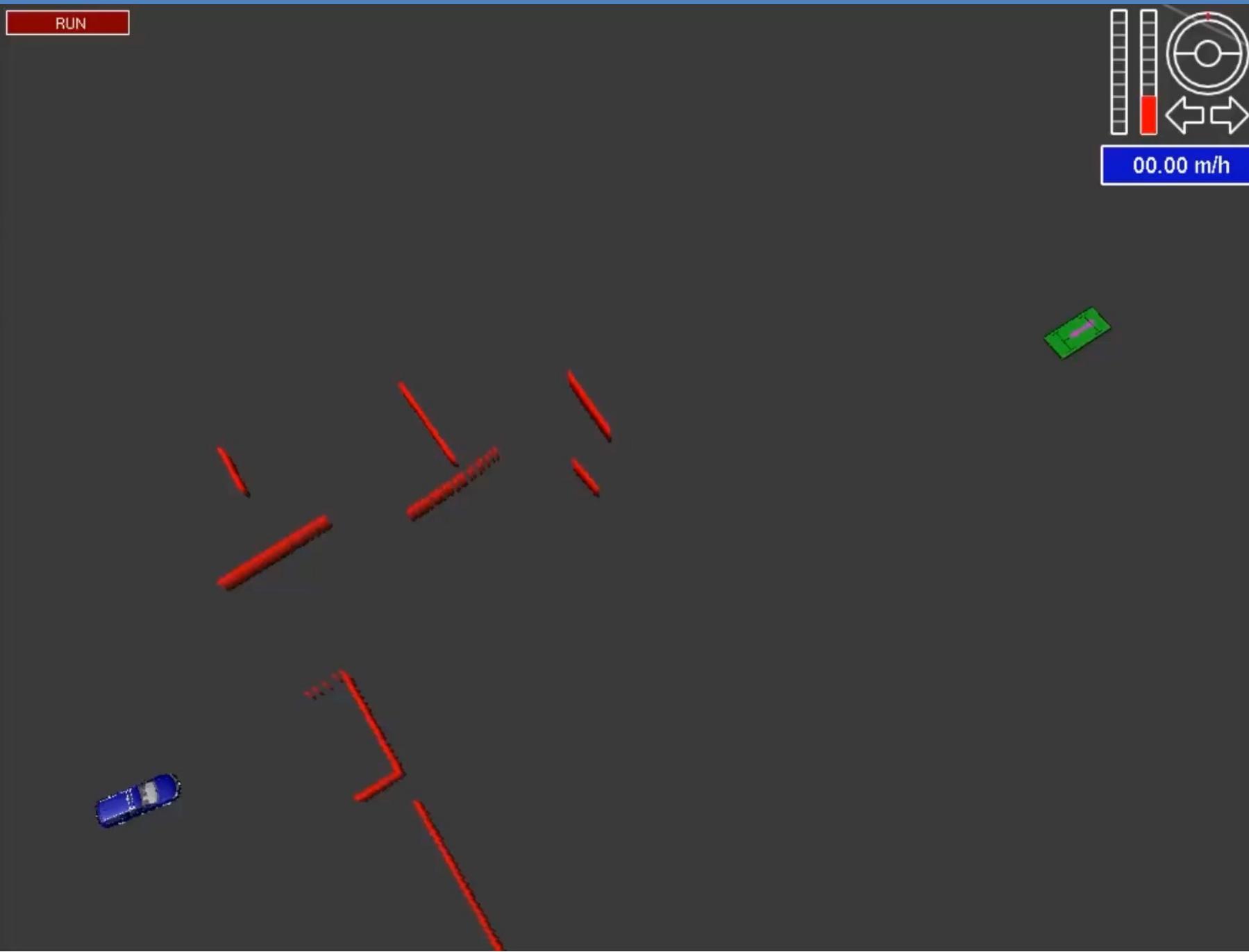
3D Path planned with A* in Area Graph of 2 stories of
SIST &
SEM
buildings



Graph Search Strategies: D* Search

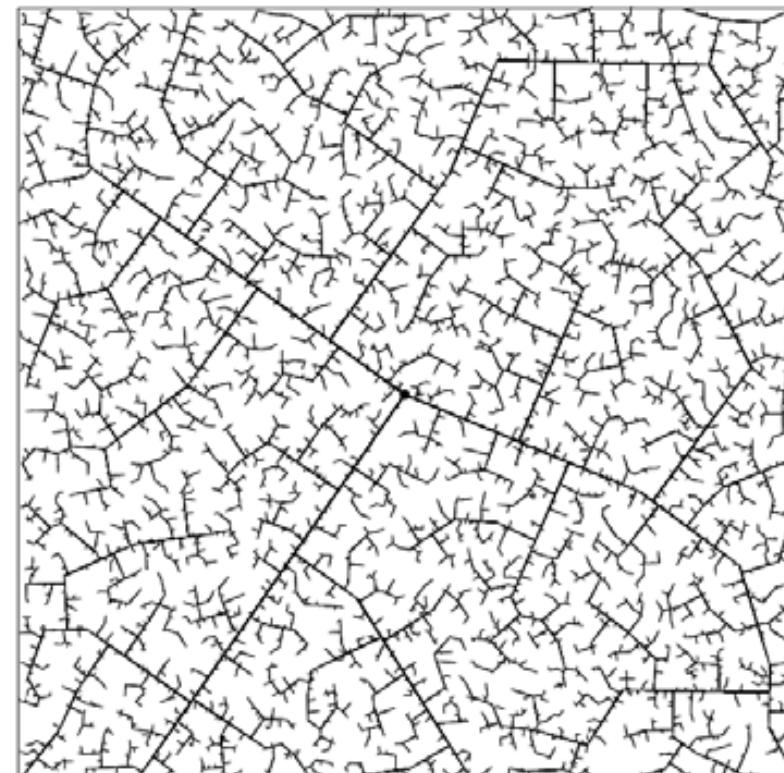
- Similar to A* search, except that the search starts from the goal outward
- $f(n) = g(n) + \epsilon h(n)$
- First pass is identical to A*
- Subsequent passes reuse information from previous searches


$$\epsilon = 1.0$$

$$\epsilon = 1.0$$

$$\epsilon = 1.0$$



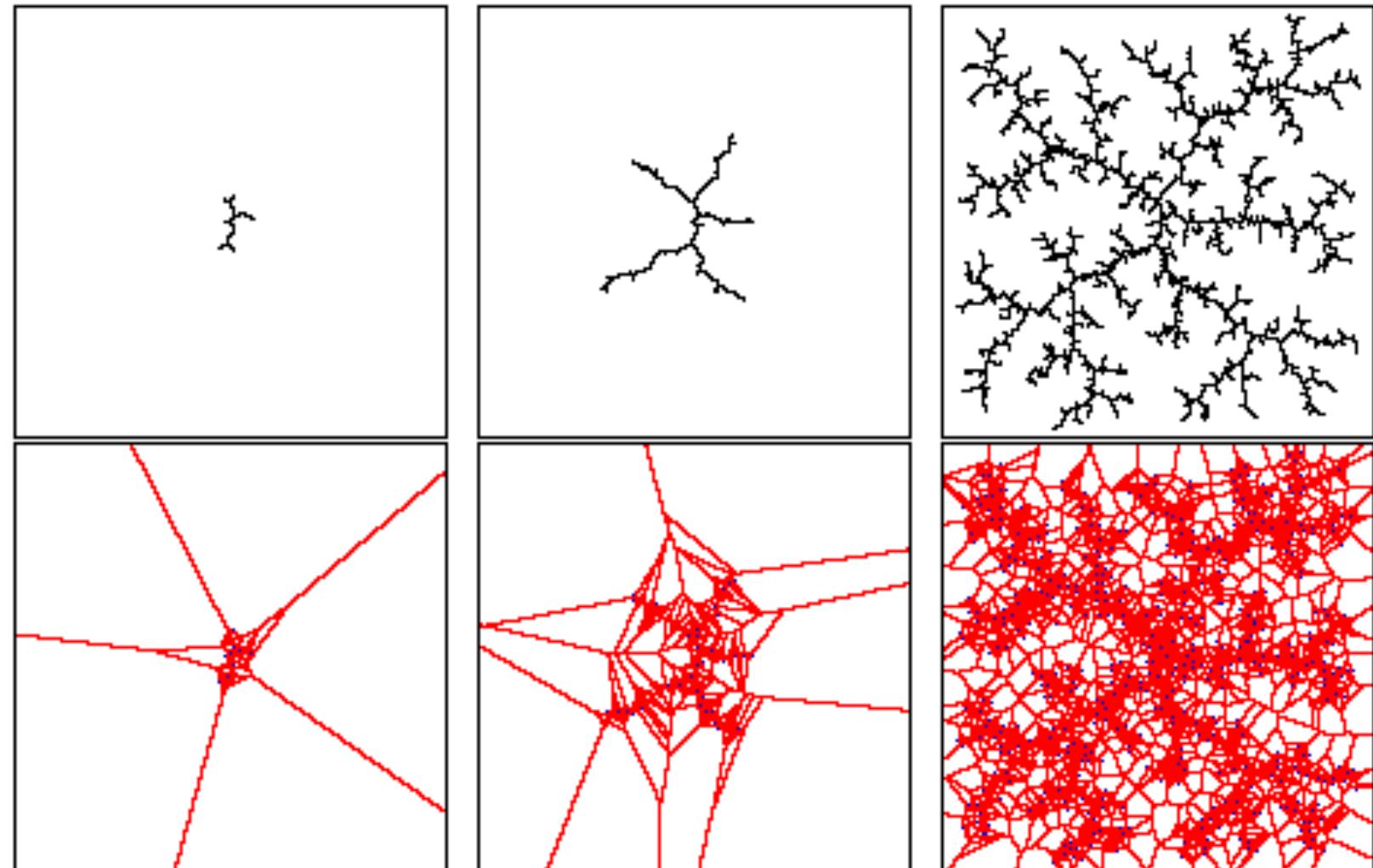
Graph Search Strategies: Randomized Search

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



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



#inclu
#inclu
#inclu
rrtstar.cpp baodi

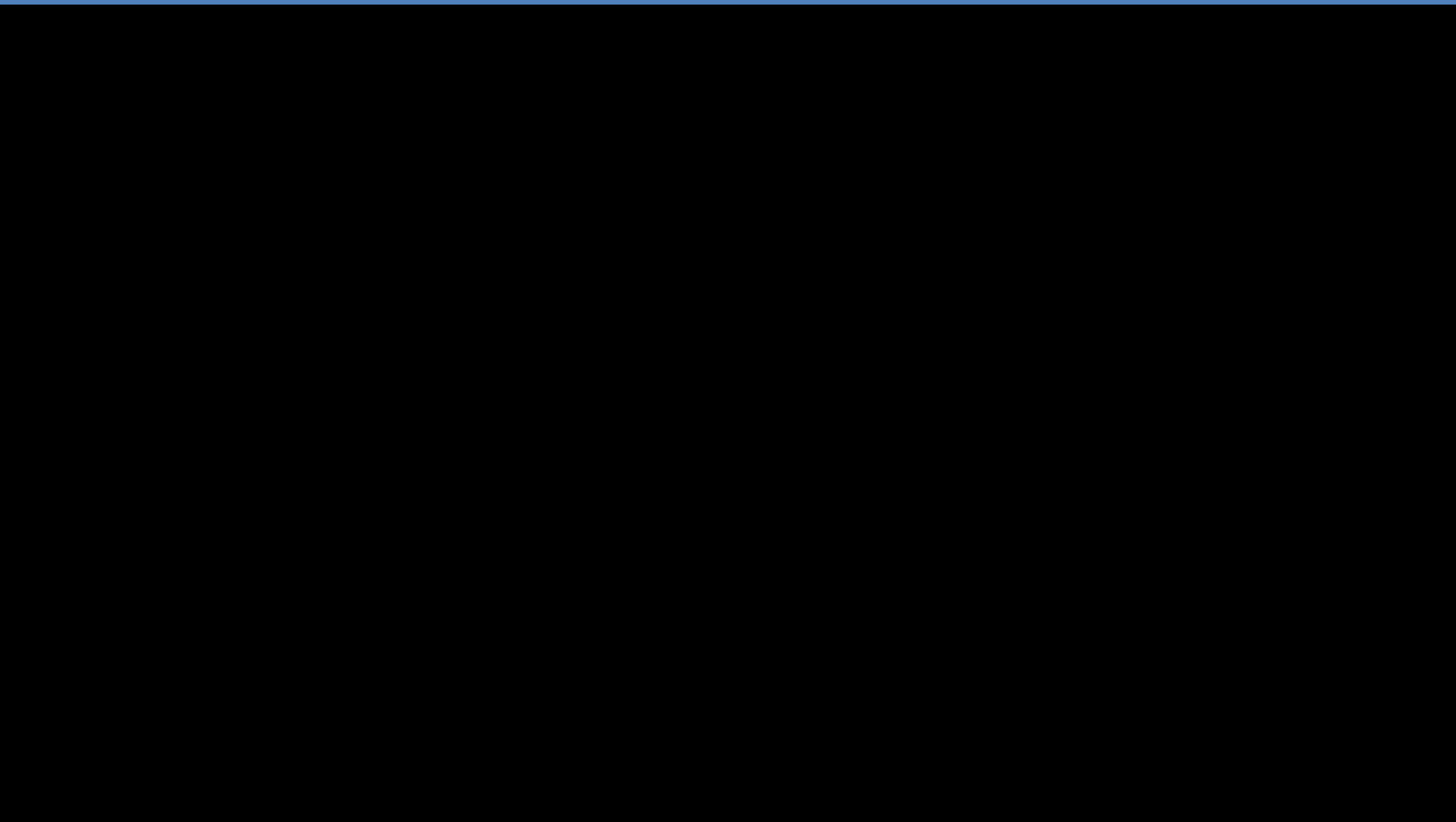
#inclu
#inclu
#inclu
rrtstar_planner.cpp

#inclu
#defin
rrtstar.h

#inclu
#inclu
#inclu
rrtstarplan.cpp

#inclu
#defin
rrtstarplan.h

```
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_smusher-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