上海科技大学
ShanghaiTech University

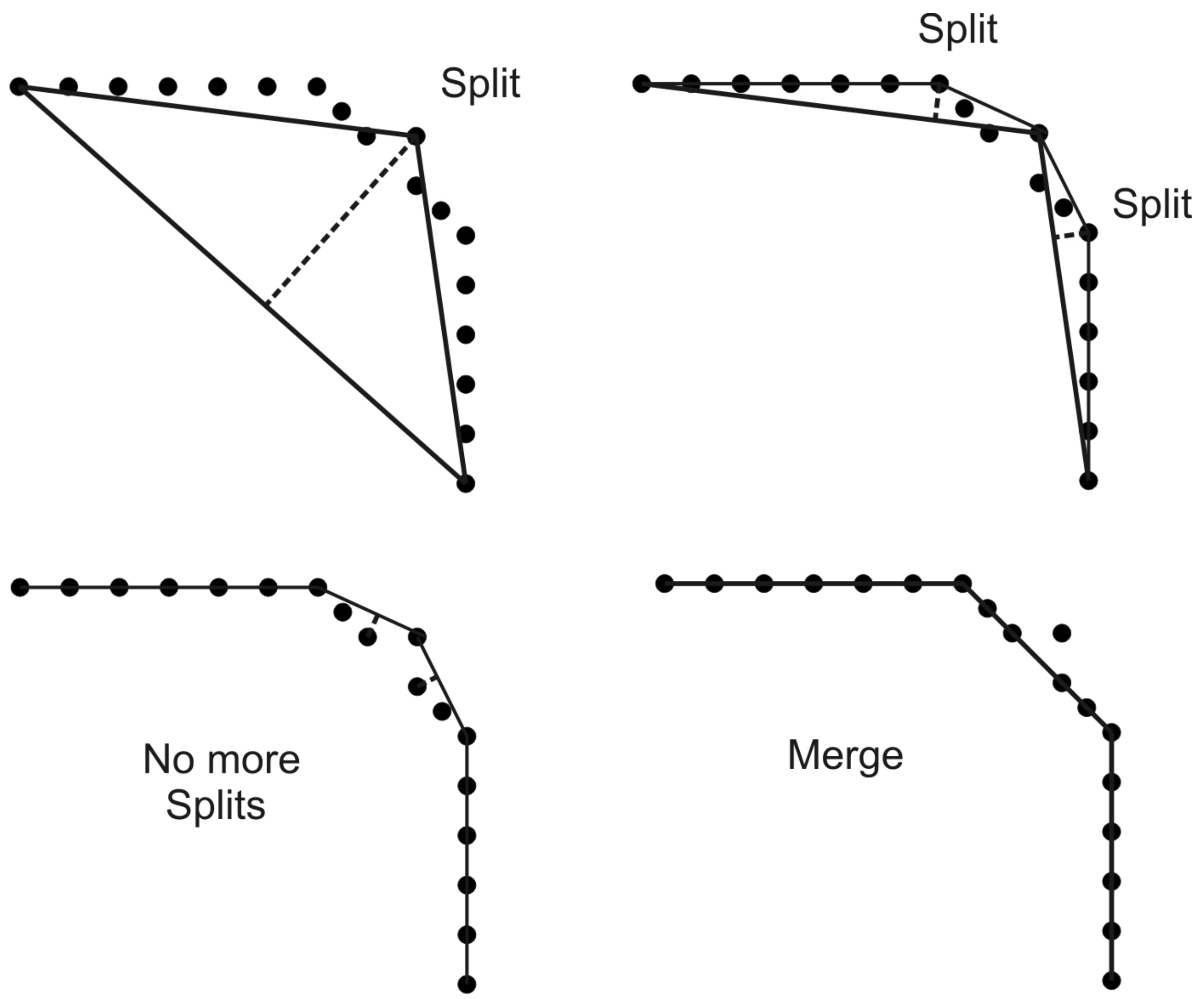
CS283: **Robotics Fall 2017: Planning**

Sören Schwertfeger / 师泽仁

ShanghaiTech University

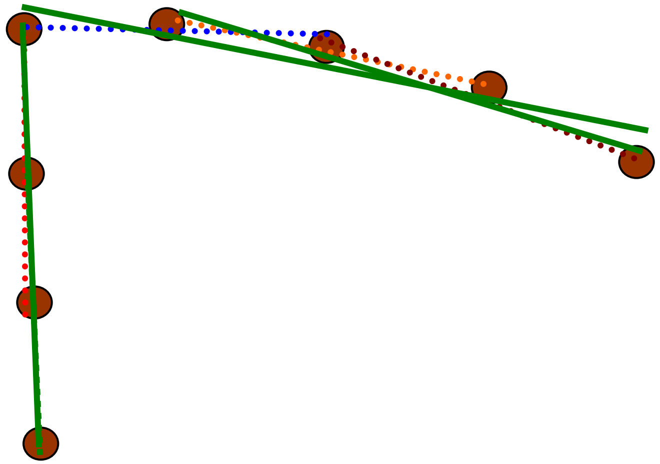
REVIEW

Algorithm 1: Split-and-Merge (Iterative-End-Point-Fit)



Algorithm 2: Line-Regression

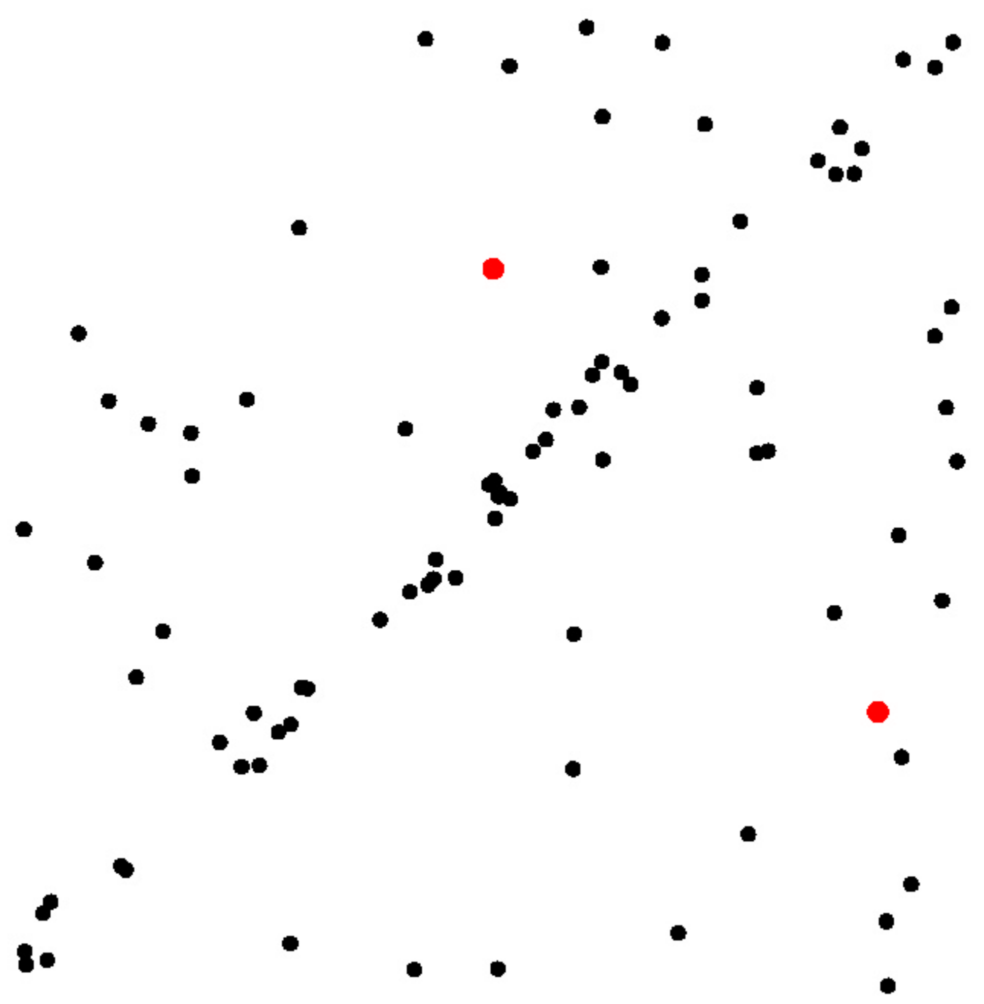
- Uses a “sliding window” of size *Nf*
- The points within each “sliding window” are fitted by a segment
- Then adjacent segments are merged if their line parameters are close



$Nf = 3$

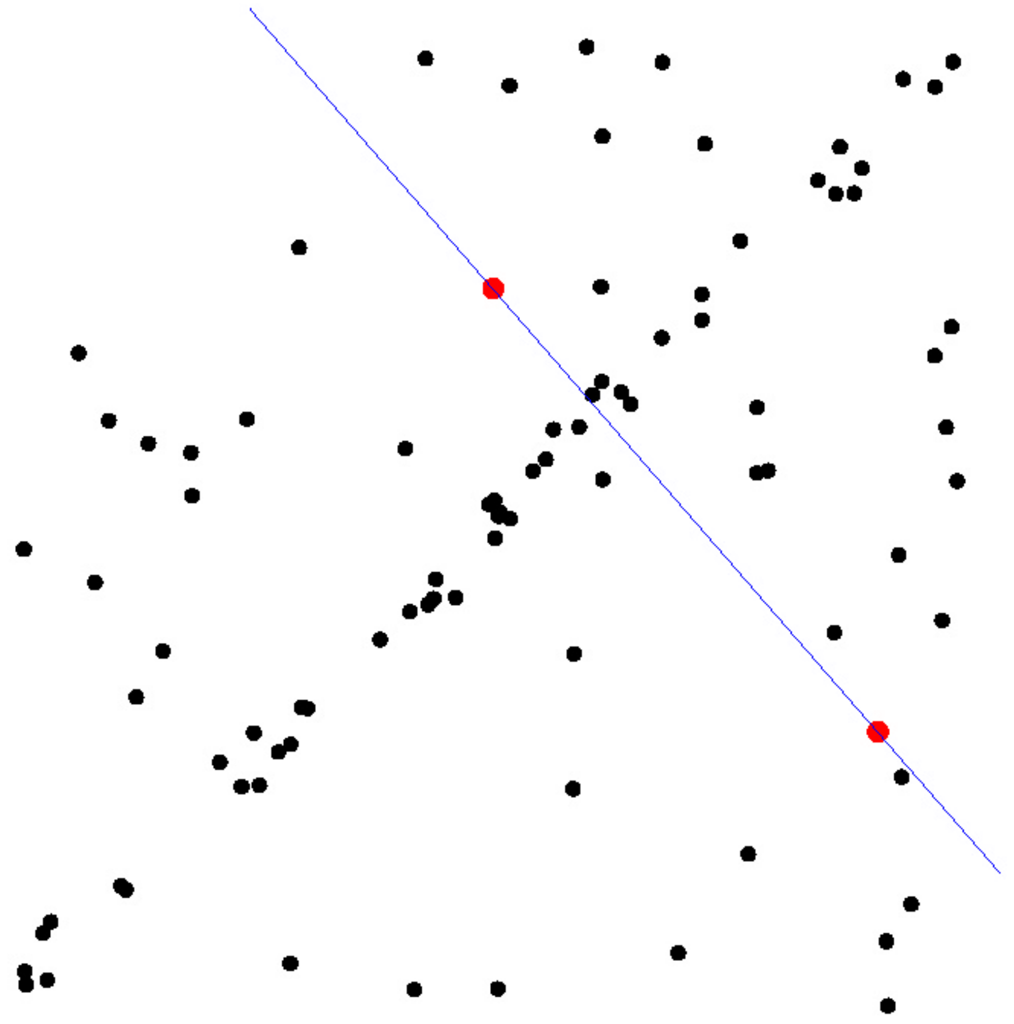
Algorithm 3: RANSAC

Algorithm 3: RANSAC



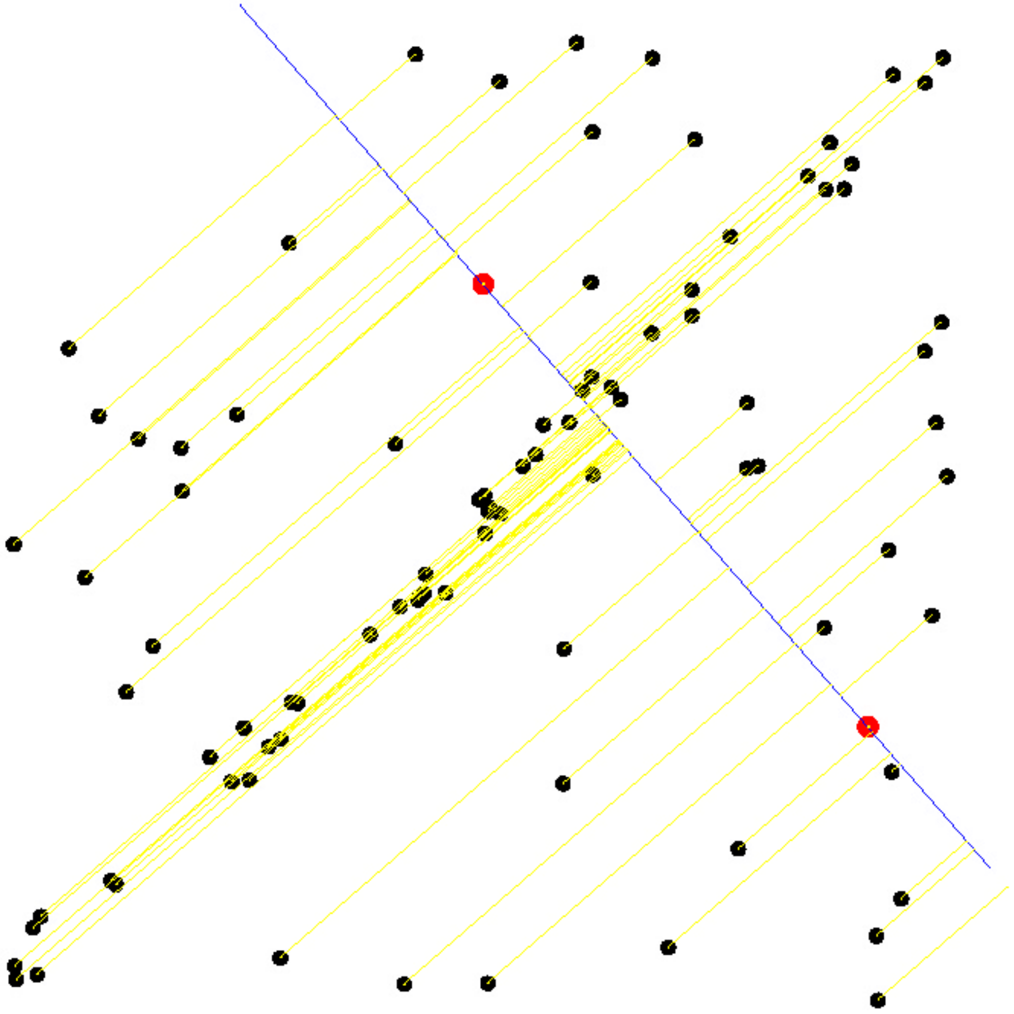
- **Select sample of 2 points at random**

Algorithm 3: RANSAC



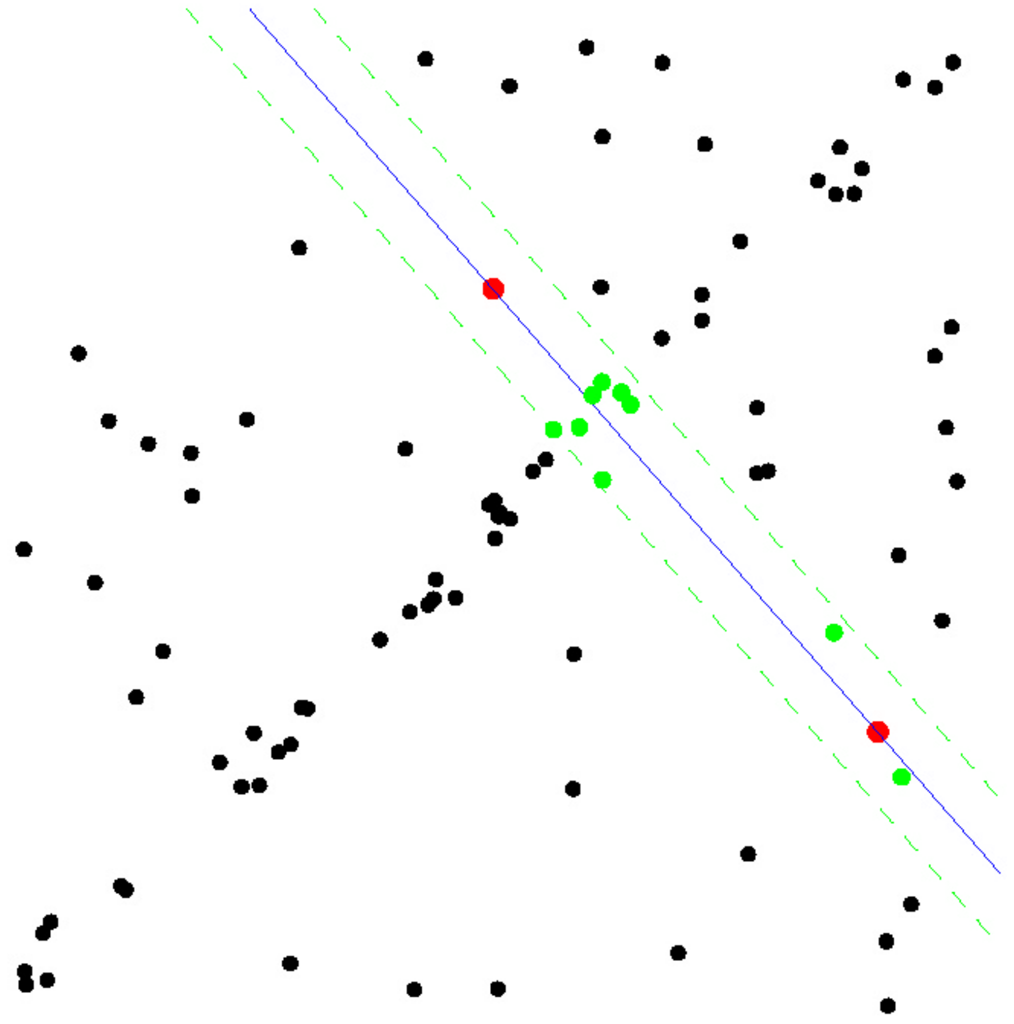
- Select sample of 2 points at random
- **Calculate model parameters that fit the data in the sample**

RANSAC



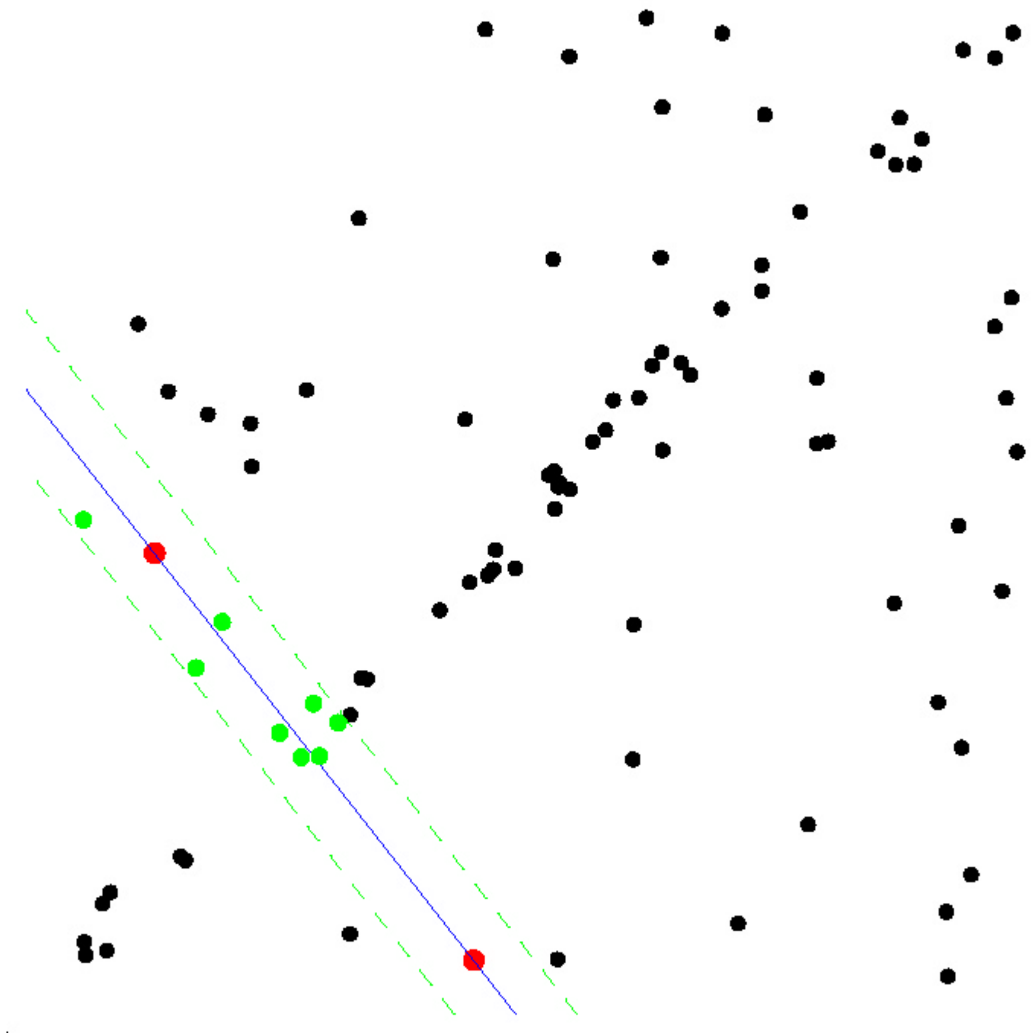
- Select sample of 2 points at random
- Calculate model parameters that fit the data in the sample
- **Calculate error function for each data point**

Algorithm 3: RANSAC



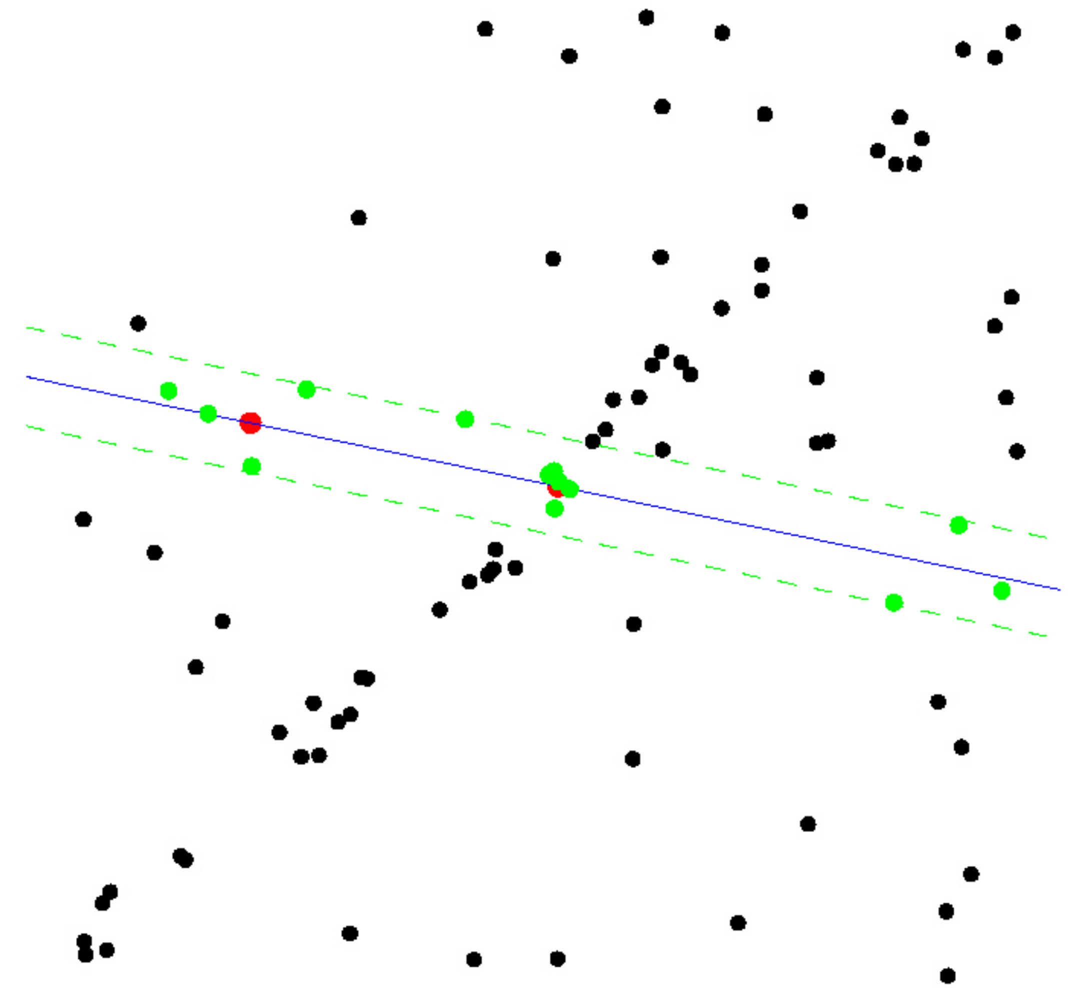
- Select sample of 2 points at random
- Calculate model parameters that fit the data in the sample
- Calculate error function for each data point
- **Select data that support current hypothesis**

Algorithm 3: RANSAC



- Select sample of 2 points at random
- Calculate model parameters that fit the data in the sample
- Calculate error function for each data point
- Select data that support current hypothesis
- **Repeat sampling**

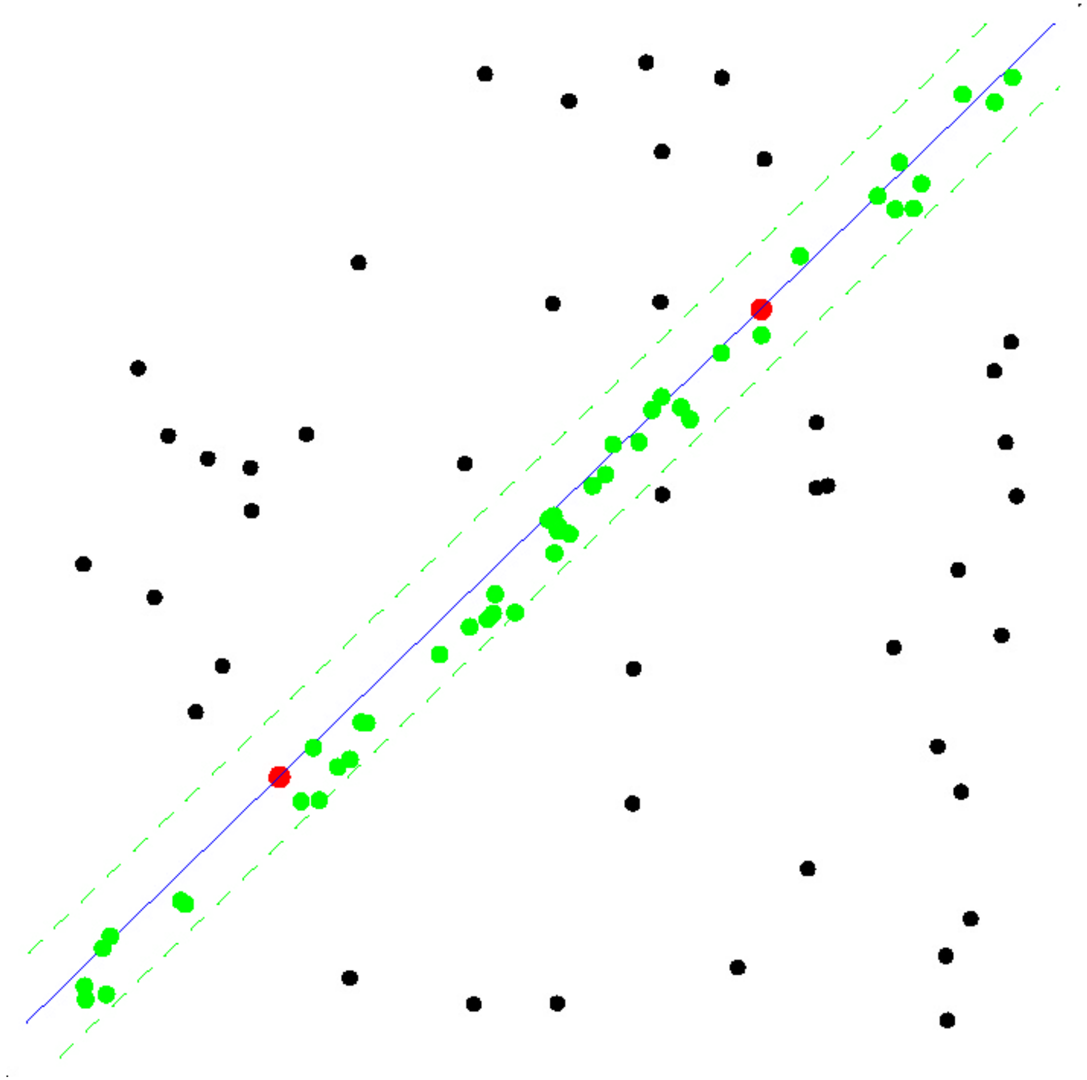
Algorithm 3: RANSAC



- Select sample of 2 points at random
- Calculate model parameters that fit the data in the sample
- Calculate error function for each data point
- Select data that support current hypothesis
- **Repeat sampling**

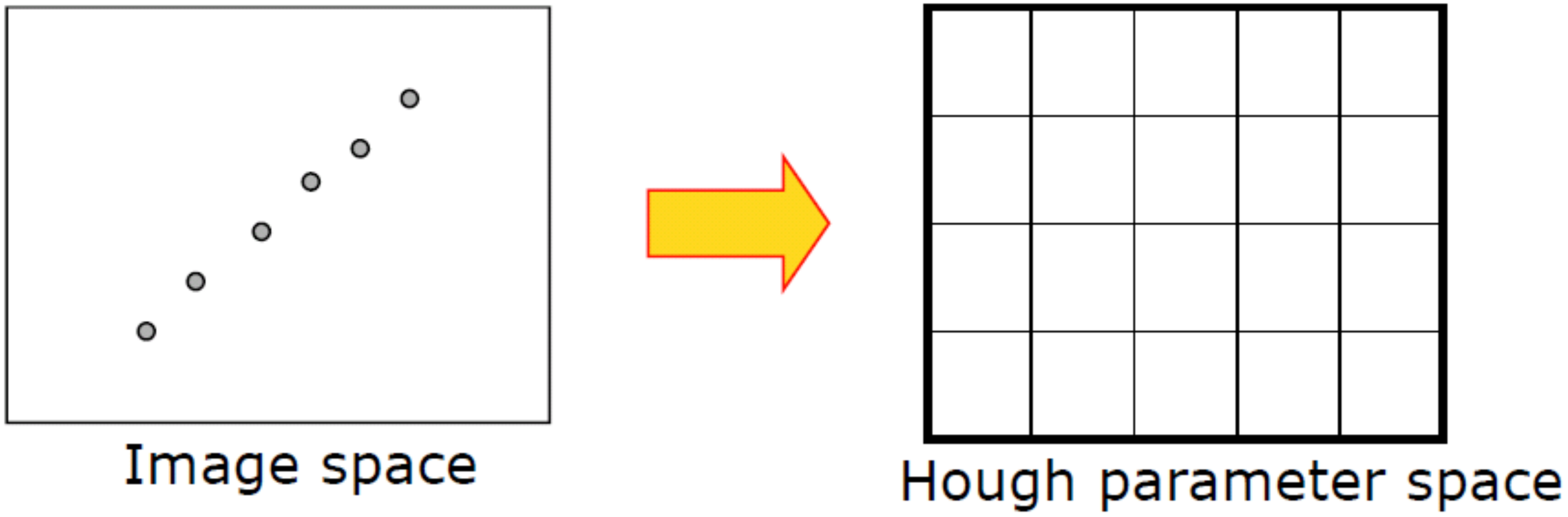
Algorithm 3: RANSAC

ALL-INLIER SAMPLE

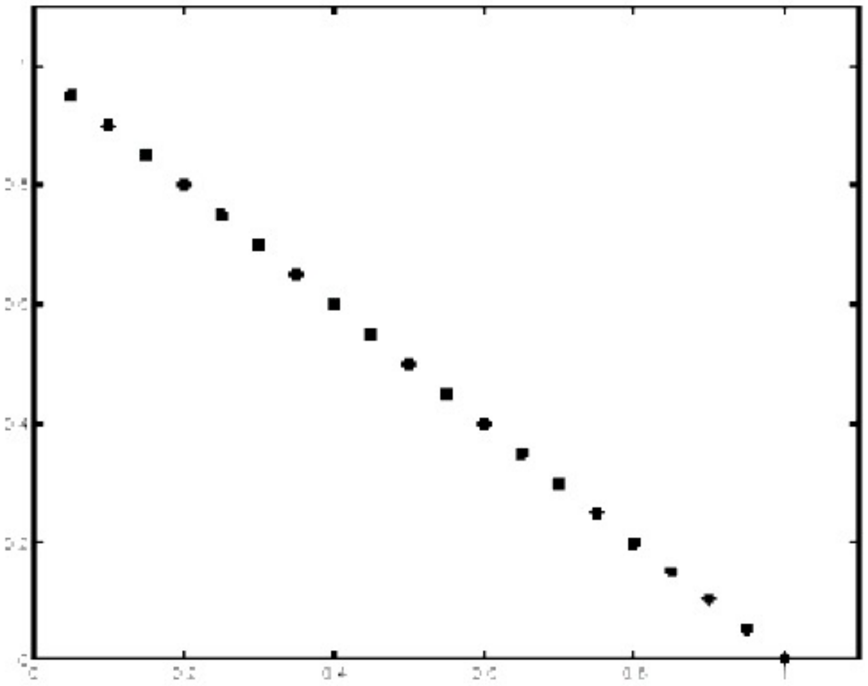


Algorithm 4: Hough-Transform

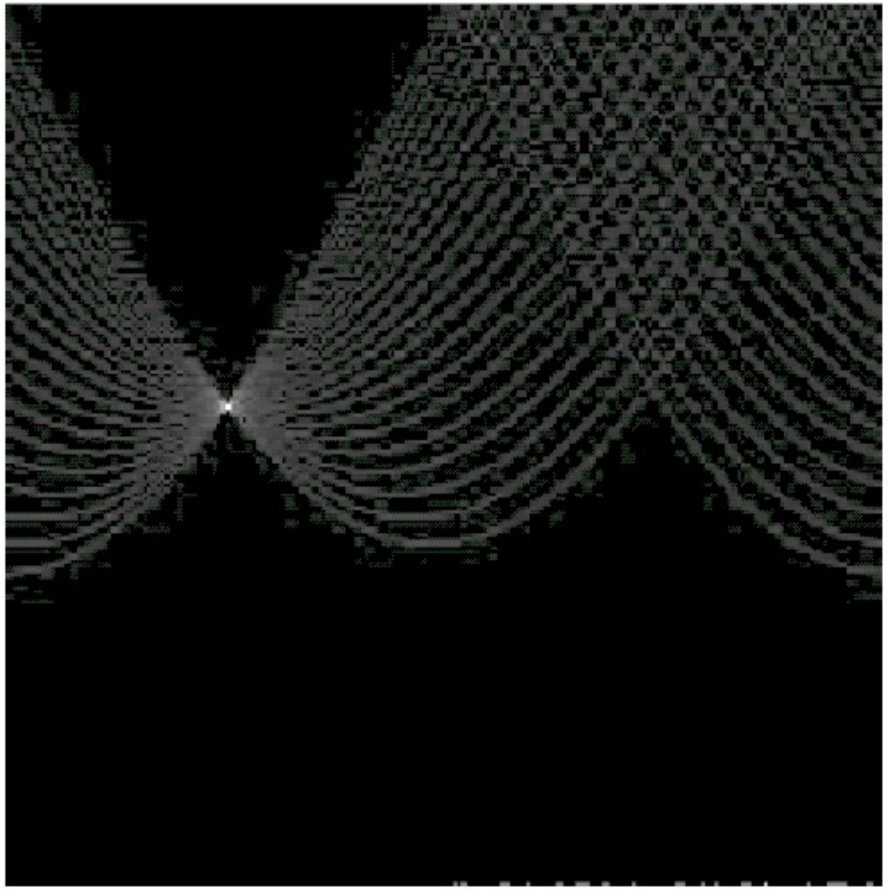
- Hough Transform uses a voting scheme



Algorithm 4: Hough-Transform



features

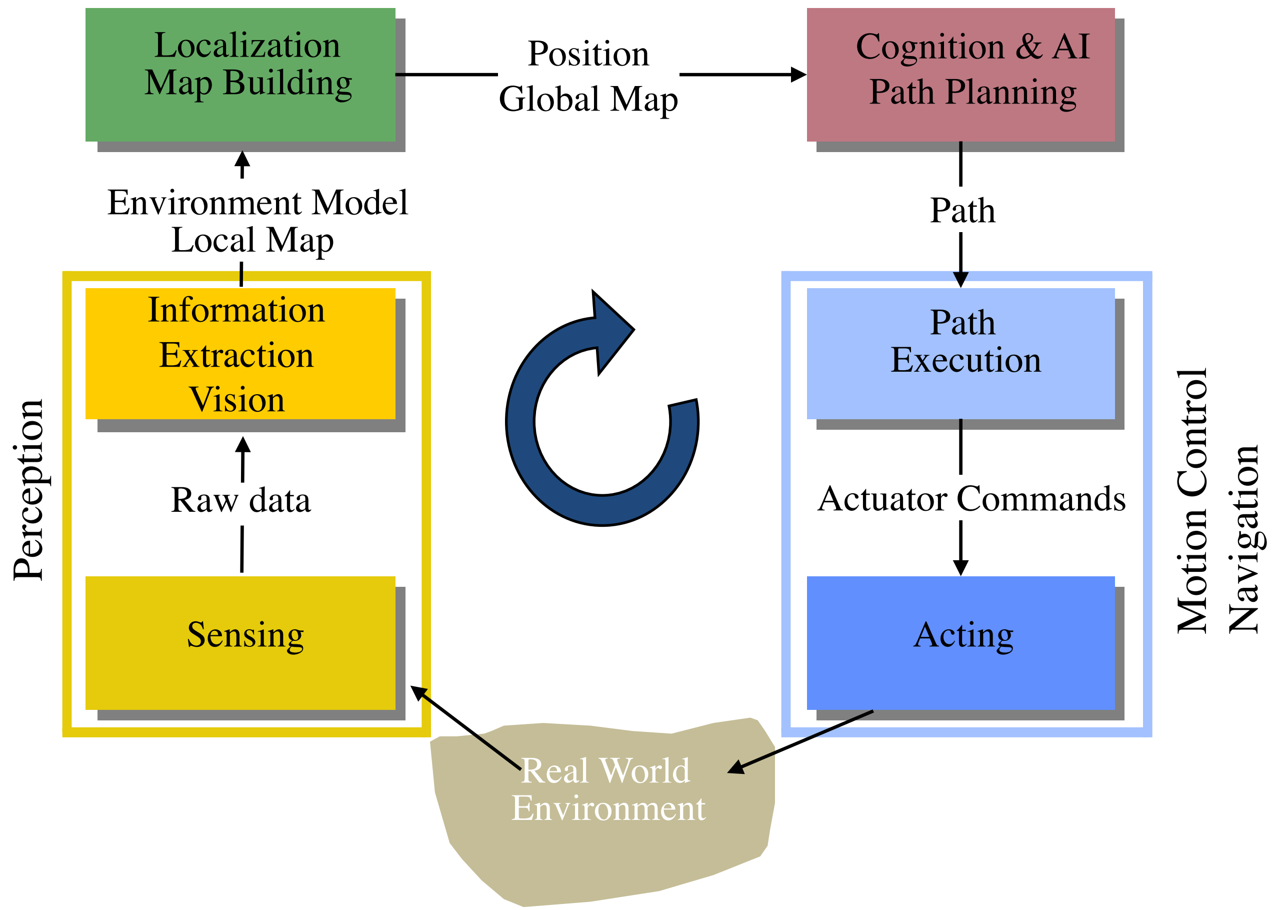


votes

- Autonomous mobile robots move around in the environment. Therefore **ALL** of them:
  - They need to know **where** they **are**.
  - They need to know **where** their **goal** is.
  - <u>They need to know **how** to get there.</u>

- Different levels:
  - Control:
    - How much power to the motors to move in that direction, reach desired speed
  - Navigation:
    - Avoid obstacles
    - Classify the terrain in front of you
    - Follow a path
  - Planning:
    - Long distance path planning
    - What is the way, optimize for certain parameters

General Control Scheme for Mobile Robot Systems



MAPS

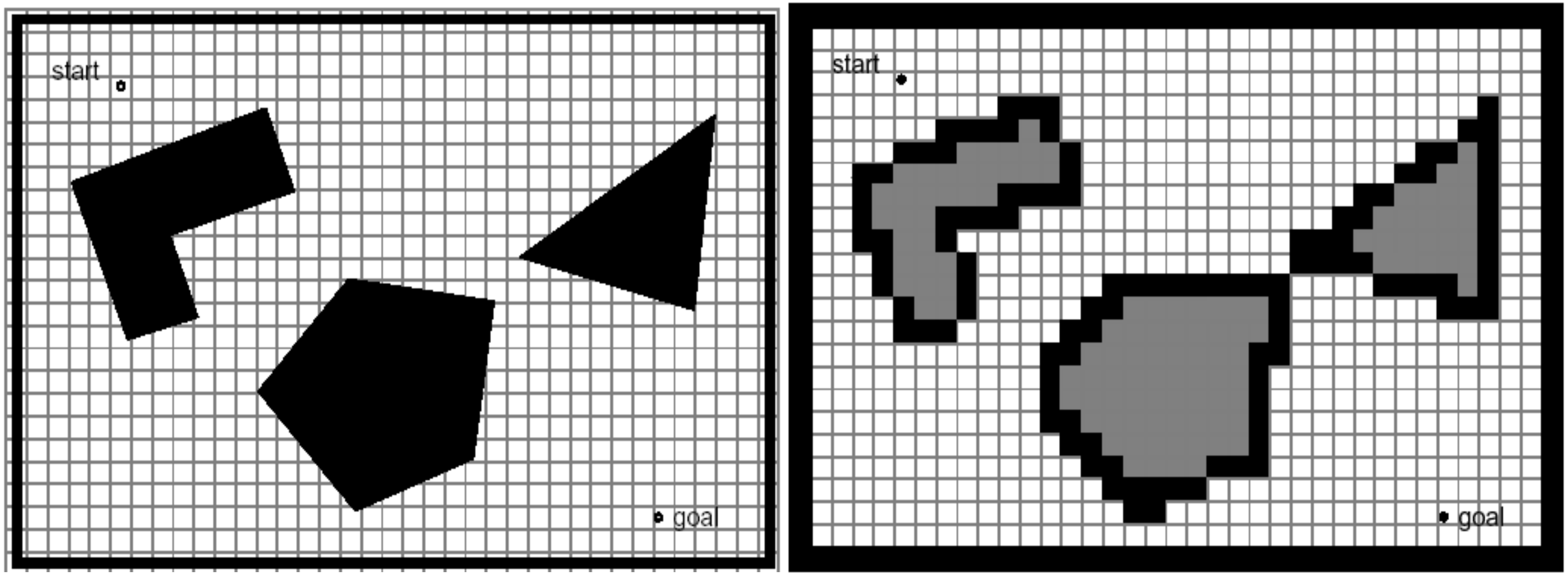
Representation of the Environment

- Environment Representation
  - Continuous Metric $\rightarrow$ x, y, θ
  - Discrete Metric $\rightarrow$ metric grid
  - Discrete Topological $\rightarrow$ topological grid

- Environment Modeling
  - Raw sensor data, e.g. laser range data, grayscale images
    - large volume of data, low distinctiveness on the level of individual values
    - makes use of all acquired information
  - Low level features, e.g. line other geometric features
    - medium volume of data, average distinctiveness
    - filters out the useful information, still ambiguities
  - High level features, e.g. doors, a car, the Eiffel tower
    - low volume of data, high distinctiveness
    - filters out the useful information, few/ no ambiguities

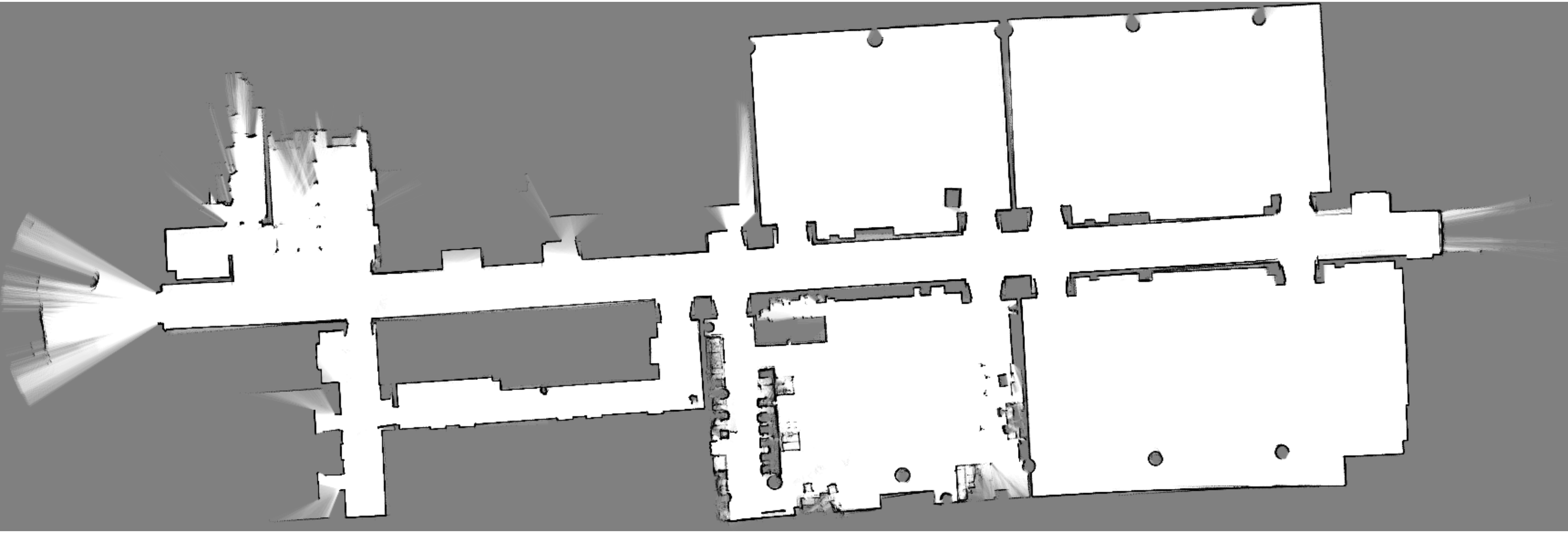
Map Representation: Approximate cell decomposition

- Fixed cell decomposition => 2D grid map
  - Cells: probability of being occupied =>
    - 0 free; 0.5 (or 128) unknown; 1 or (255) occupied



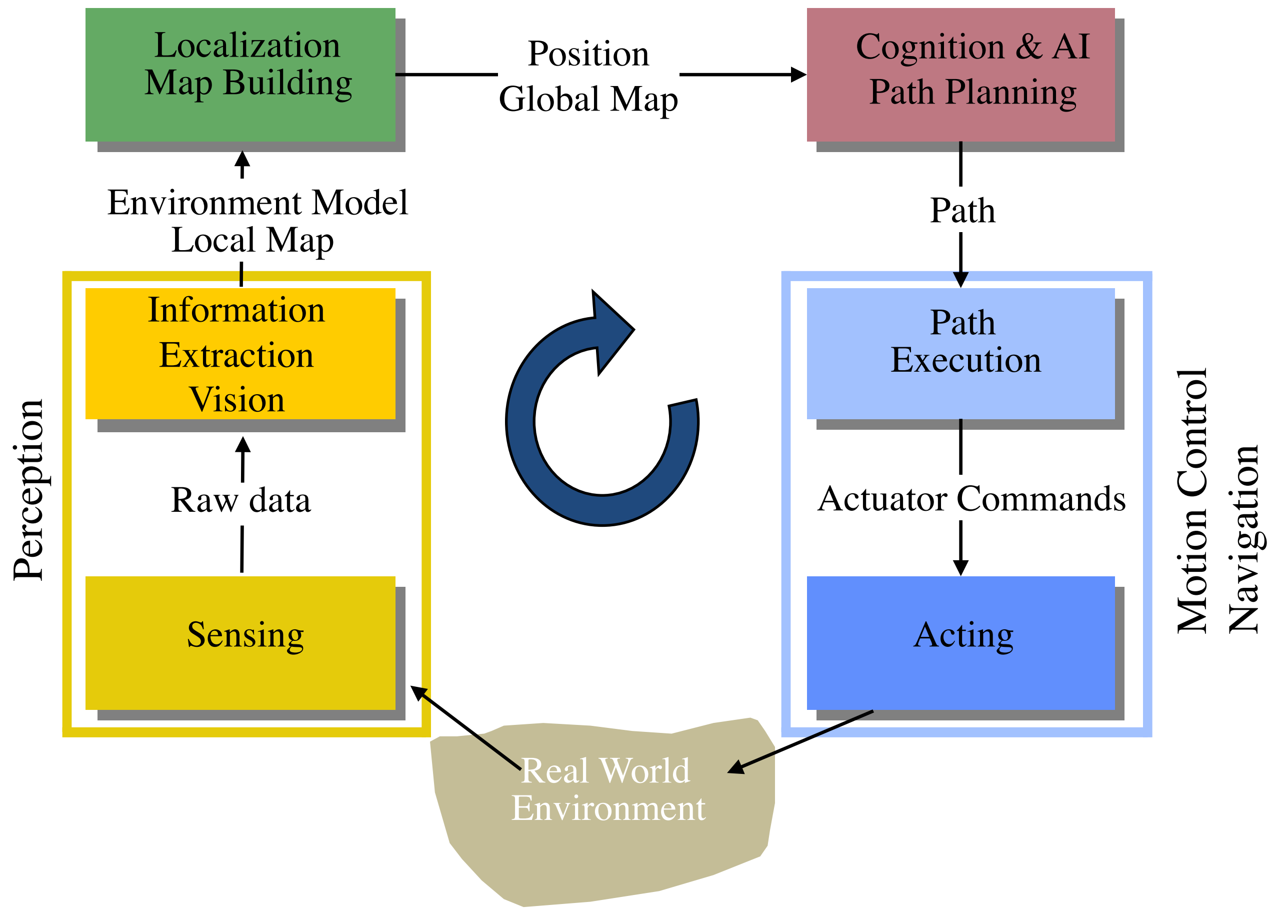
Map Representation: Occupancy grid

- Fixed cell decomposition: occupancy grid example: STAR Center



PLANNING

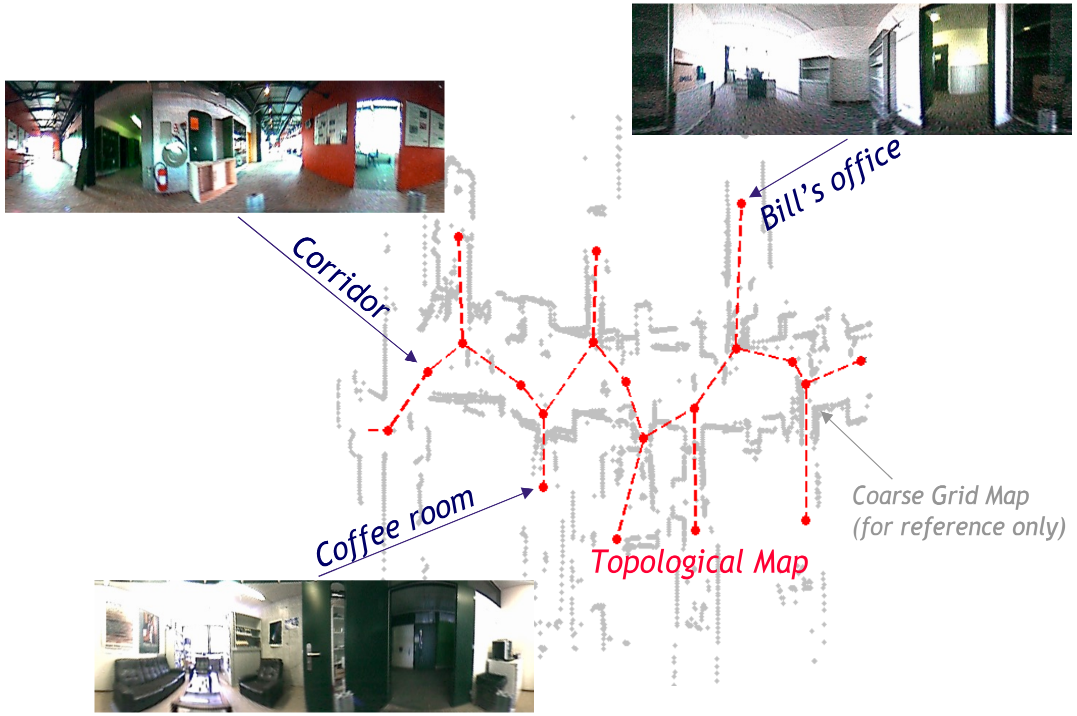
General Control Scheme for Mobile Robot Systems



The Planning Problem

- The problem: find a path in the work space (physical space) from the initial position to the goal position avoiding all collisions with the obstacles

- Assumption: there exists a good enough map of the environment for navigation.

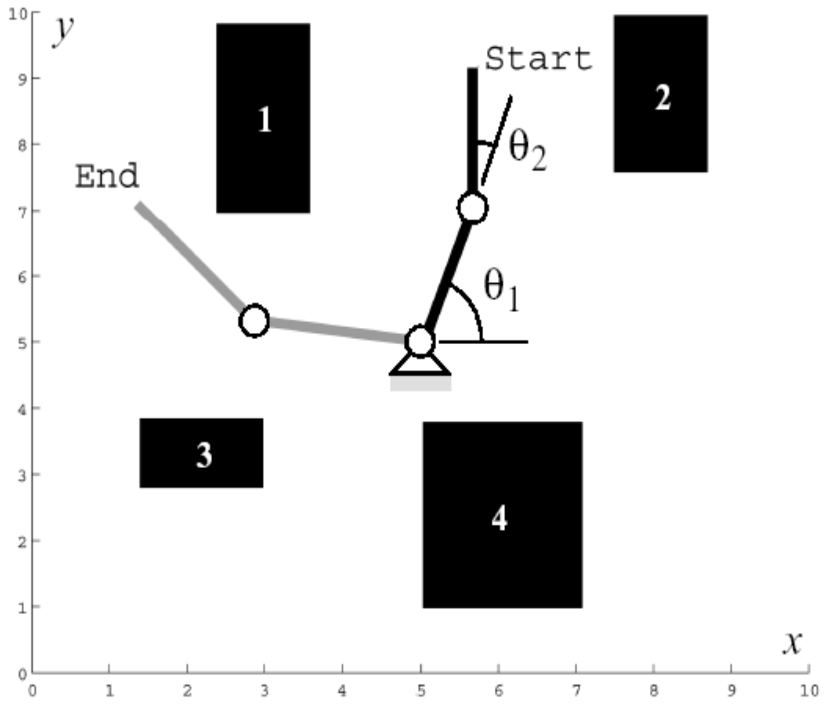


The Planning Problem

- We can generally distinguish between
  - (global) path planning and
  - (local) obstacle avoidance.
- First step:
  - Transformation of the map into a representation useful for planning
  - This step is planner-dependent
- Second step:
  - Plan a path on the transformed map
- Third step:
  - Send motion commands to controller
  - This step is planner-dependent (e.g. Model based feed forward, path following)

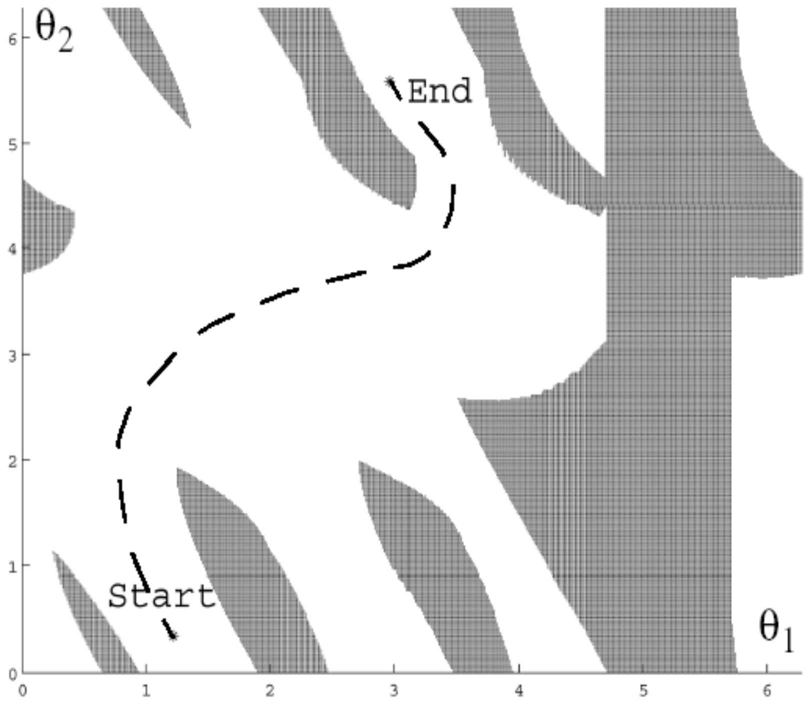
Work Space (Map) → Configuration Space

- State or configuration q can be described with k values q_i



a)

Work Space



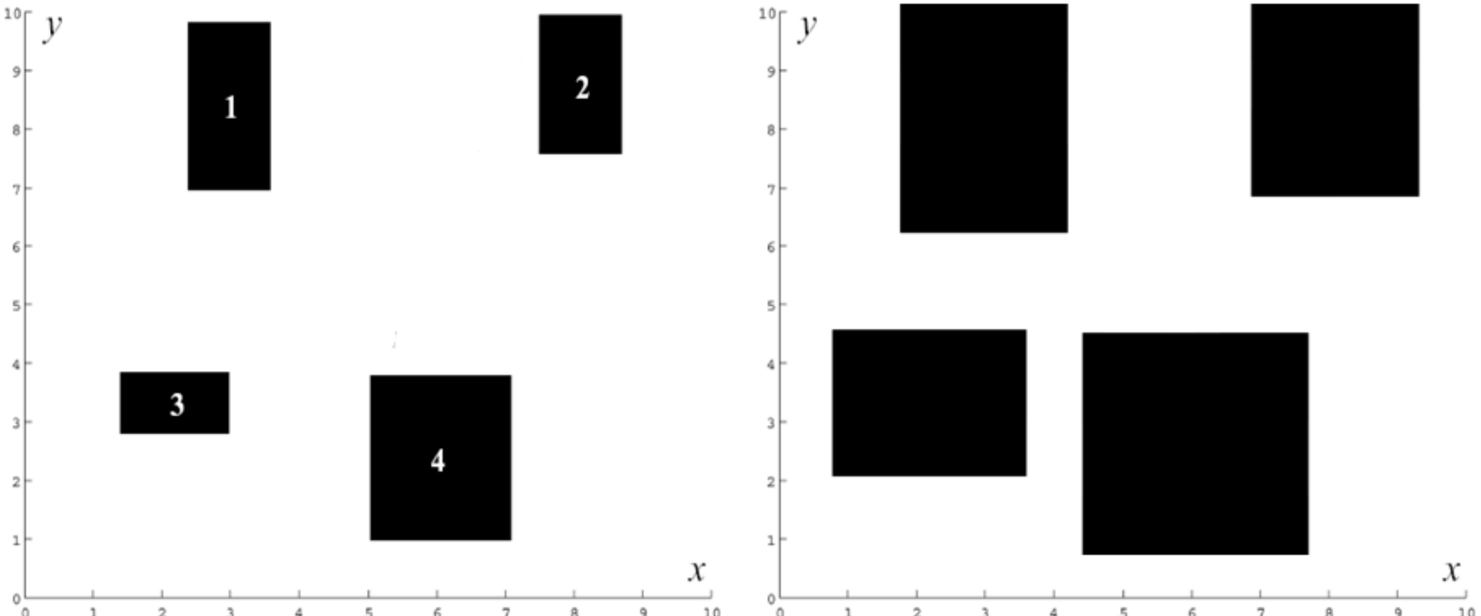
b)

Configuration Space:
the dimension of this space is equal to the Degrees of Freedom (DoF) of the robot

- What is the configuration space of a mobile robot?

Configuration Space for a Mobile Robot

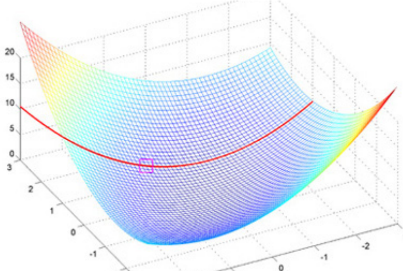
- Mobile robots operating on a flat ground (2D) have 3 DoF: (x, y, θ)
- Differential Drive: only two motors => only 2 degrees of freedom directly controlled (forward/ backward + turn) => non-holonomic
- Simplification: assume robot is holonomic and it is a point => configuration space is reduced to 2D (x,y)
- => inflate obstacle by size of the robot radius to avoid crashes => obstacle growing



Path Planning: Overview of Algorithms

1. Optimal Control

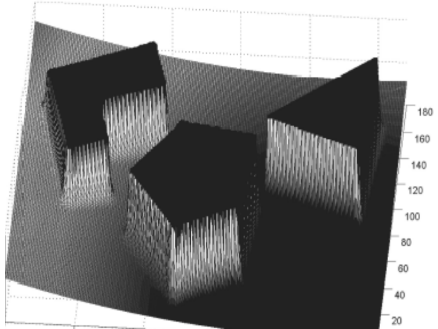
- Solves truly optimal solution
- Becomes intractable for even moderately complex as well as nonconvex problems



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

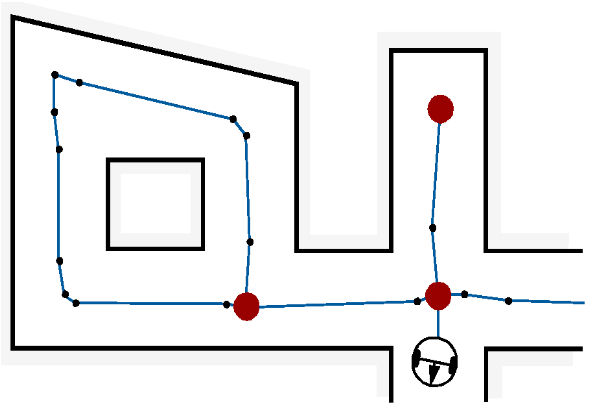
2. Potential Field

- 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

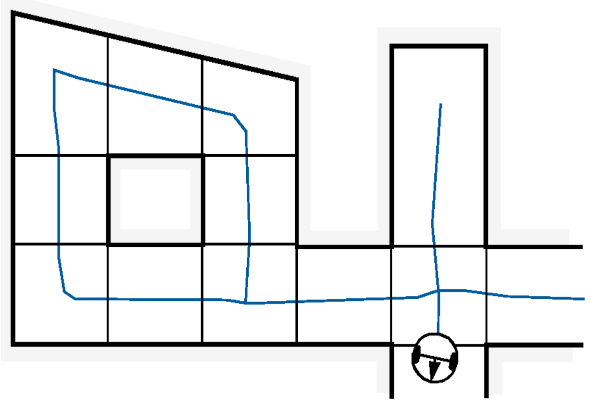


3. Graph Search

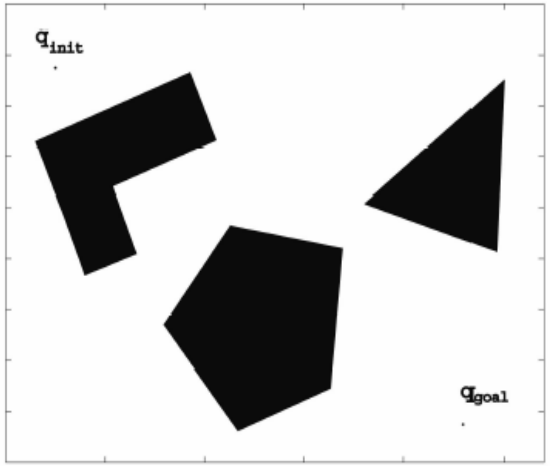
- Identify a set edges between nodes within the free space



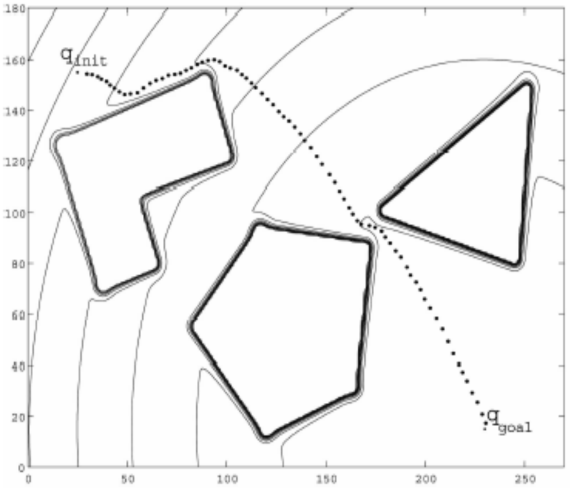
- Where to put the nodes?

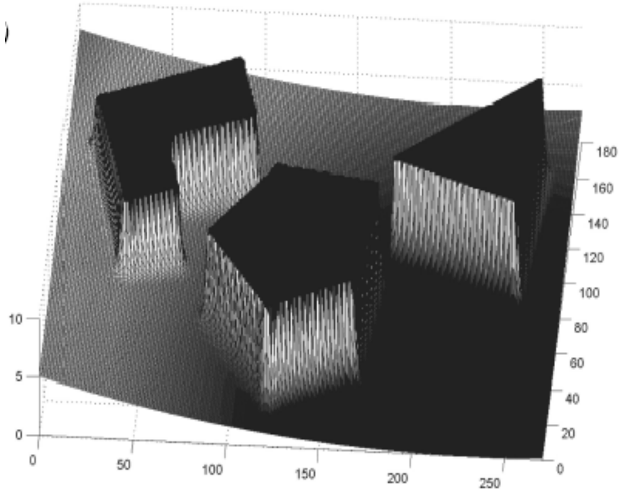


Potential Field Path Planning Strategies



- Robot is treated as a *point under the influence* of an artificial potential field.
- Operates in the continuum
  - Generated robot movement is similar to a ball rolling down the hill
  - Goal generates attractive force
  - Obstacle are repulsive forces





Potential Field Path Planning: Potential Field Generation

- Generation of potential field function *U(q)*
  - attracting (goal) and repulsing (obstacle) fields
  - summing up the fields
  - functions must be differentiable
- Generate artificial force field *F(q)*

$$F(q) = -\nabla U(q) = -\nabla U_{att}(q) - \nabla U_{rep}(q) = \begin{bmatrix} \frac{\partial U}{\partial x} \\ \frac{\partial U}{\partial y} \end{bmatrix}$$

- Set robot speed (v_x, v_y) proportional to the force *F*(*q*) generated by the field
  - the force field drives the robot to the goal
  - if robot is assumed to be a point mass
  - Method produces both a plan *and* the corresponding control

Potential Field Path Planning: Attractive Potential Field

- Parabolic function representing the Euclidean distance $\rho_{goal} = \left\| q - q_{goal} \right\|$ to the goal

$$
\begin{aligned}
U_{att}(q) &= \frac{1}{2} k_{att} \cdot \rho_{goal}^{2}(q) \\
&= \frac{1}{2} k_{att} \cdot (q - q_{goal})^{2}
\end{aligned}
$$

- Attracting force converges linearly towards 0 (goal)

$$
\begin{aligned}
F_{att}(q) &= -\nabla U_{att}(q) \\
&= k_{att} \cdot (q - q_{goal})
\end{aligned}
$$

Potential Field Path Planning: Repulsing Potential Field

- Should generate a barrier around all the obstacle
  - strong if close to the obstacle
  - not influence if far from the obstacle

$$U_{rep}(q) = \begin{cases} \frac{1}{2} k_{rep} \left(\frac{1}{\rho(q)} - \frac{1}{\rho_0} \right)^2 & \text{if } \rho(q) \leq \rho_0 \\ 0 & \text{if } \rho(q) \geq \rho_0 \end{cases}$$

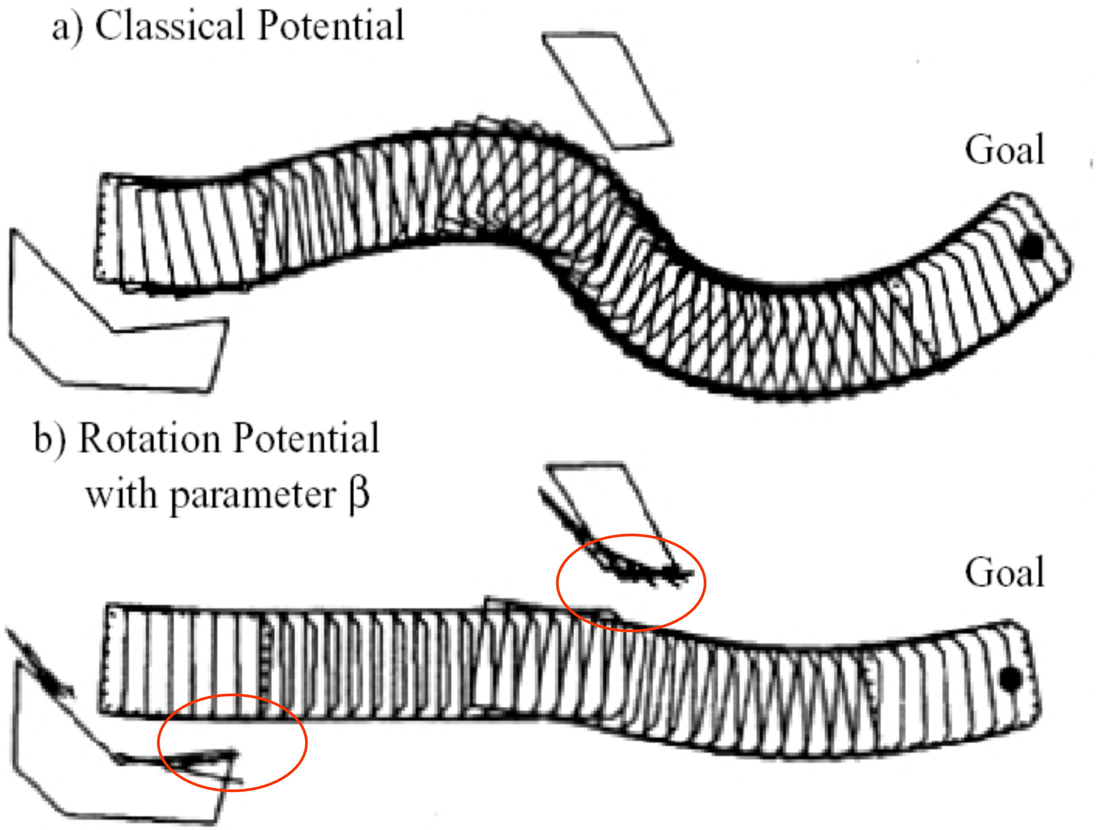
  - $\rho(q)$: minimum distance to the object
  - Field is positive or zero and *tends to infinity* as q gets closer to the object

Potential Field Path Planning:

- Notes:
  - Local minima problem exists
  - problem is getting more complex if the robot is not considered as a point mass
  - If objects are non-convex there exists situations where several minimal distances exist $\rightarrow$ can result in oscillations

Potential Field Path Planning: Extended Potential Field Method

- Additionally a *rotation potential field* and a *task potential field* is introduced
- Rotation potential field
  - force is also a function of robots orientation relative to the obstacles. This is done using a gain factor that reduces the repulsive force when obstacles are parallel to robot's direction of travel
- Task potential field
  - Filters out the obstacles that should not influence the robots movements, i.e. only the obstacles in the sector in front of the robot are considered

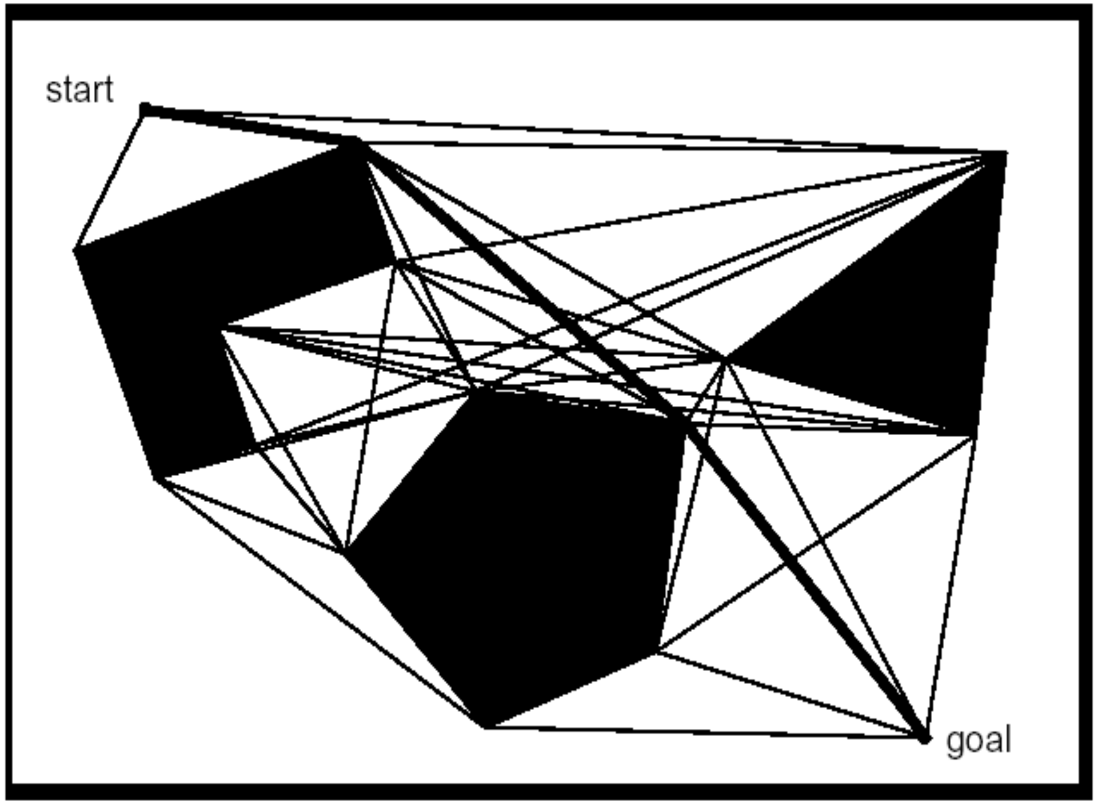


Khatib and Chatila

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



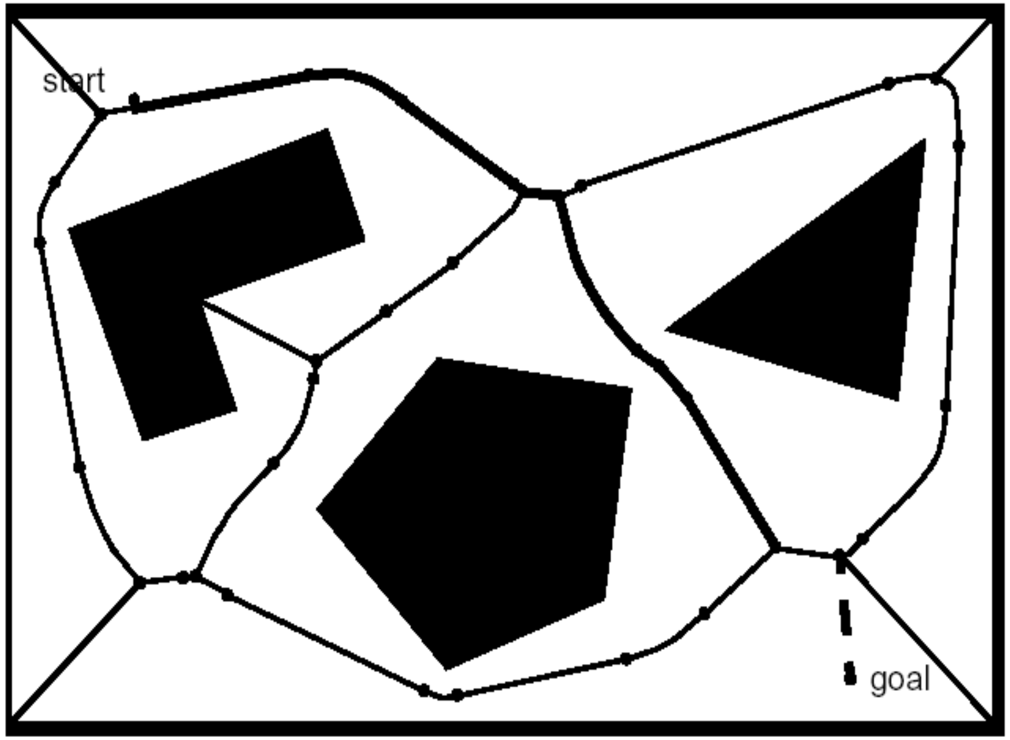
- Particularly suitable for polygon-like obstacles
- Shortest path length
- Grow obstacles to avoid collisions

Graph Construction: Visibility Graph

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

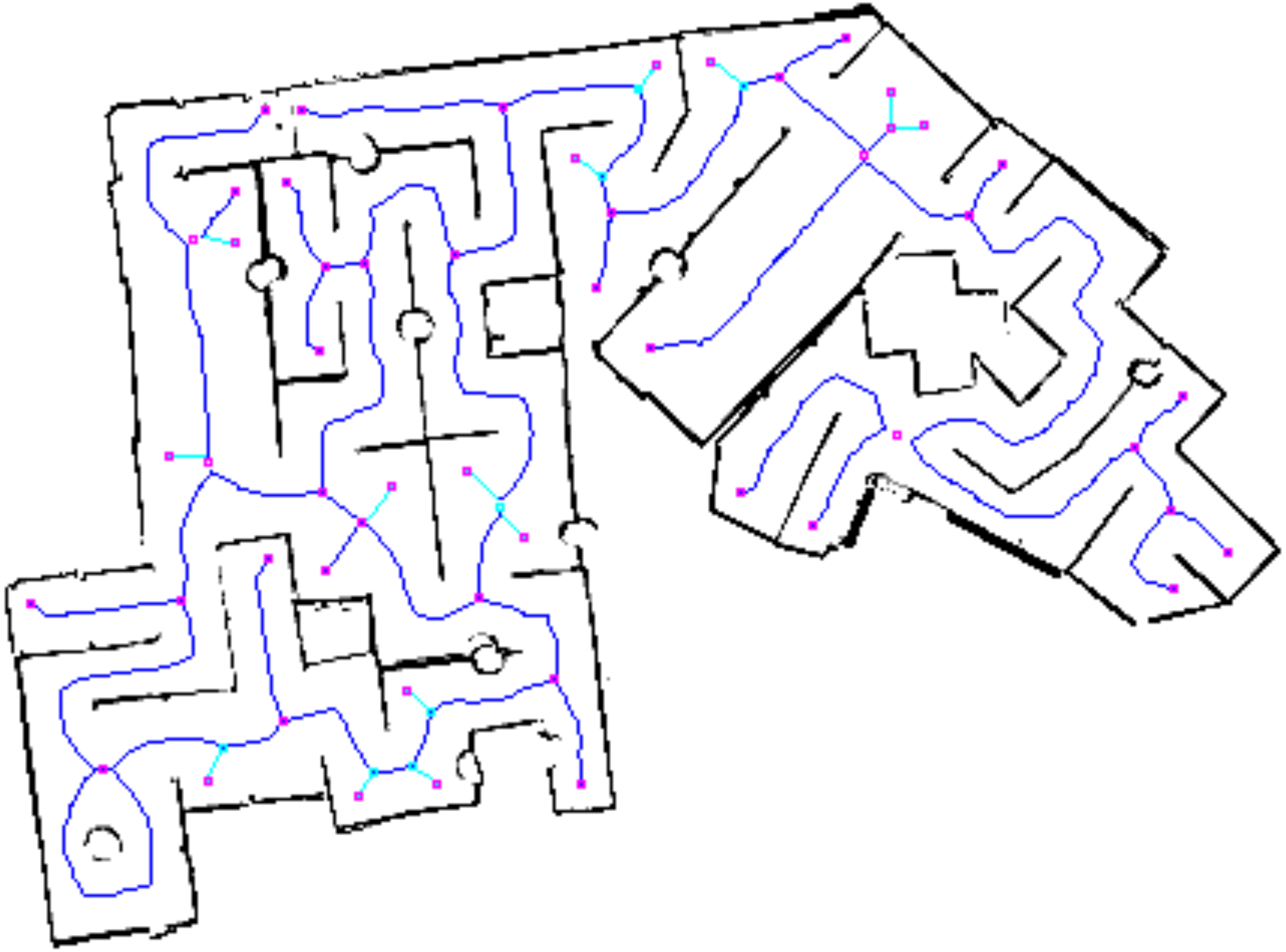
- Cons
  - The solution path found by the visibility graph tend to take the robot as close as possible to the obstacles: the common solution is to grow obstacles by more than robot's radius
  - Number of edges and nodes increases with the number of polygons
  - Thus it can be inefficient in densely populated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

Graph Construction: Voronoi Diagram



- Tends to maximize the distance between robot and obstacles

Topology Graph

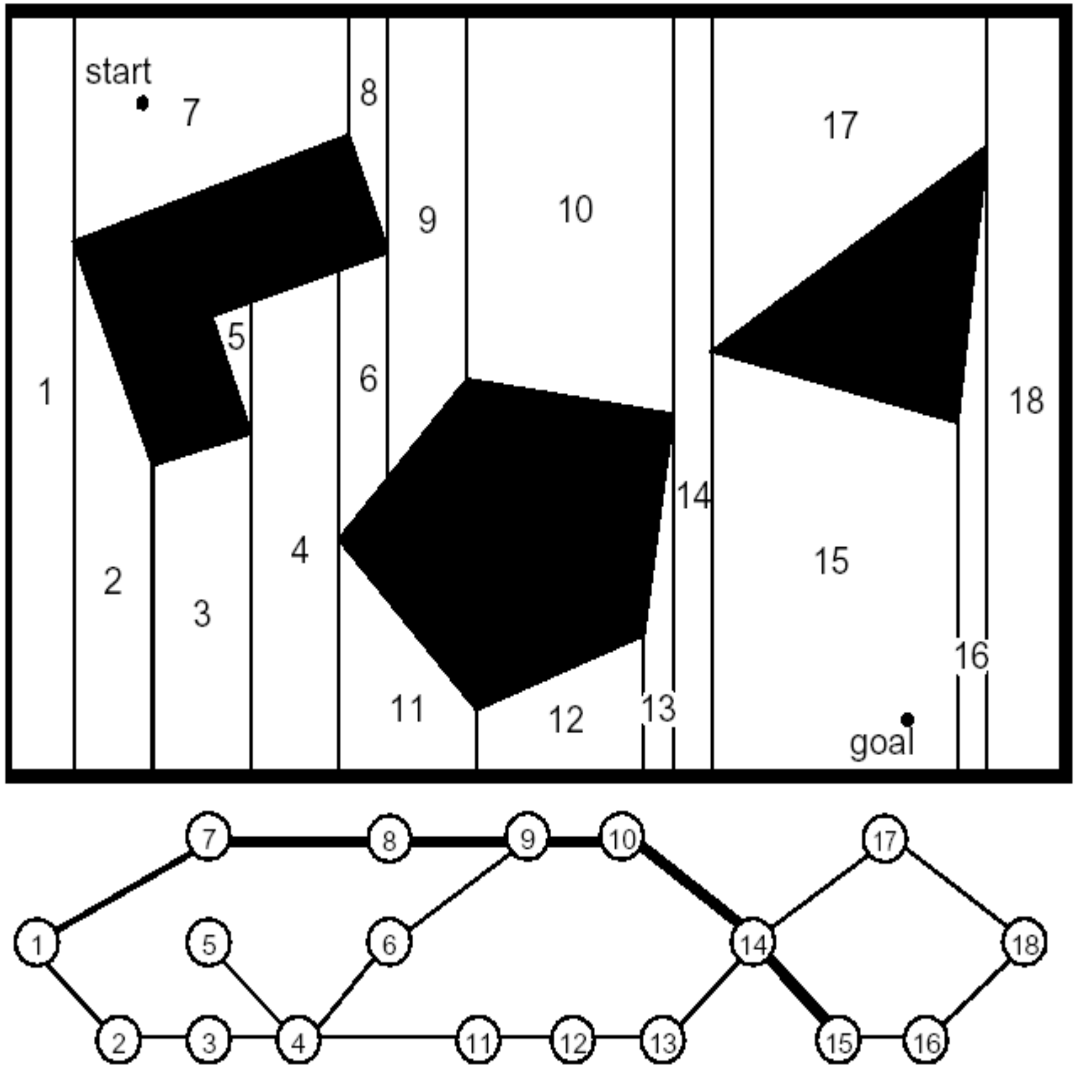


Graph Construction: Voronoi Diagram

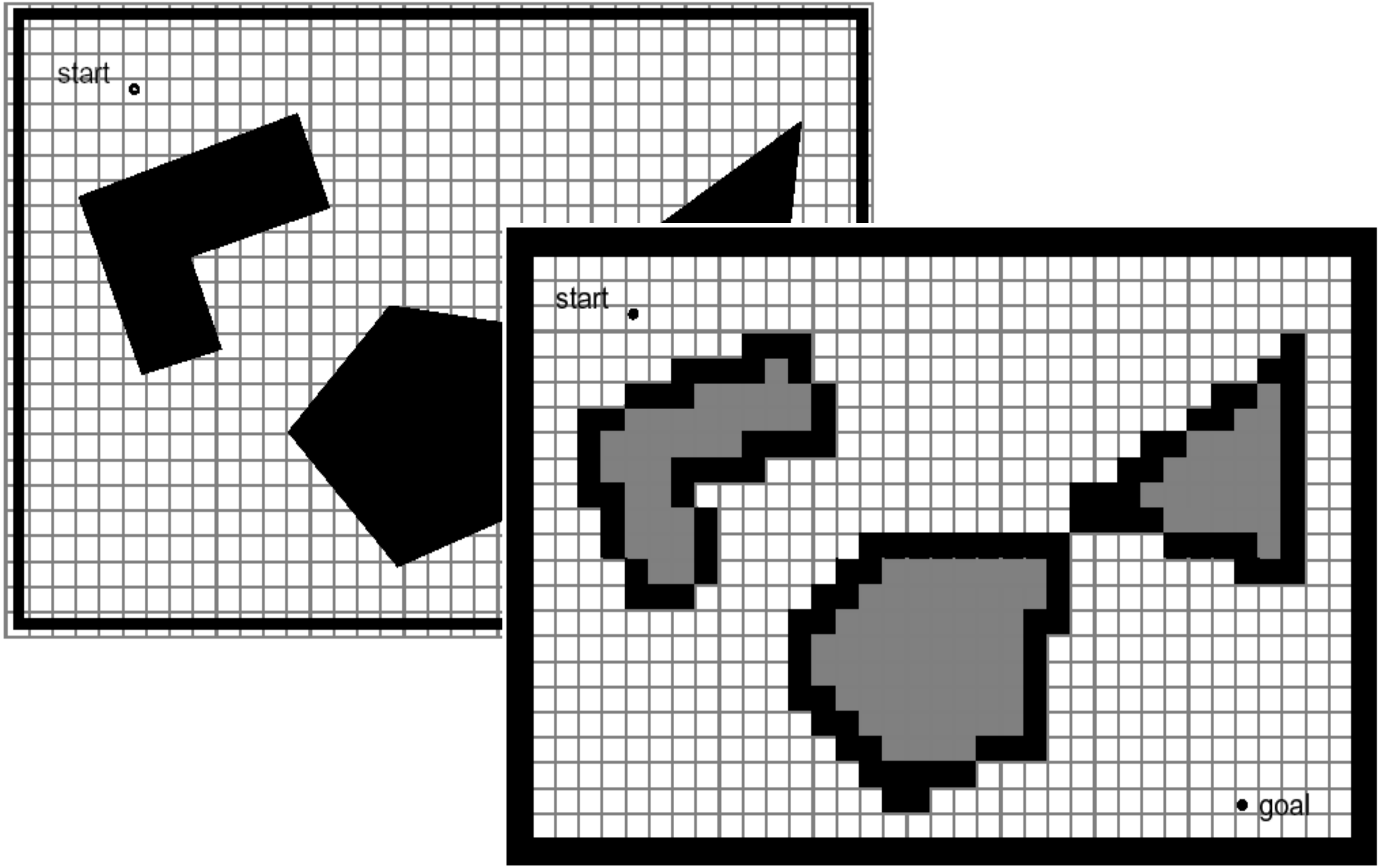
- Pros
  - Using range sensors like laser or sonar, a robot can navigate along the Voronoi diagram using simple control rules

- 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

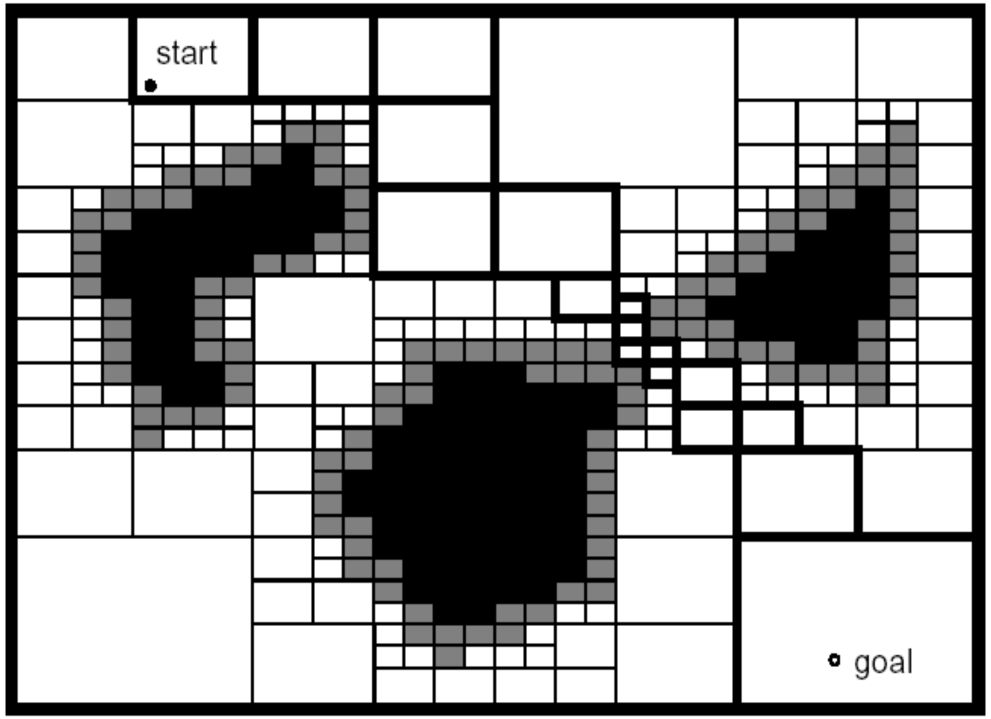
Graph Construction: Exact Cell Decomposition (2/4)



Graph Construction: Approximate Cell Decomposition (3/4)



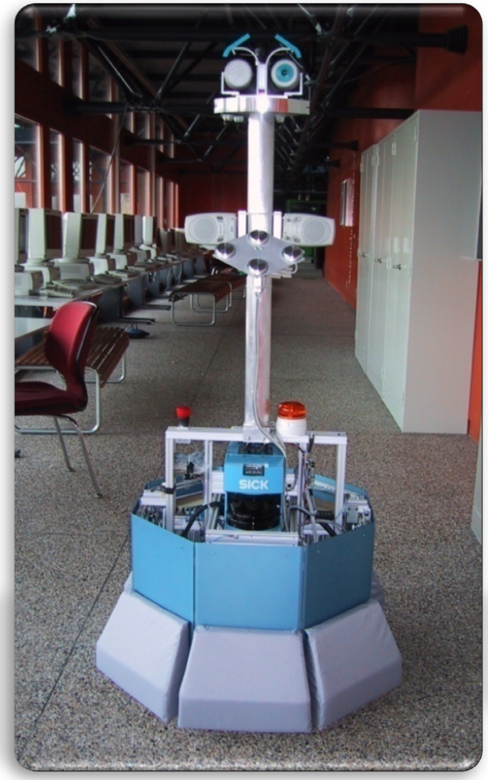
Graph Construction: Adaptive Cell Decomposition (4/4)



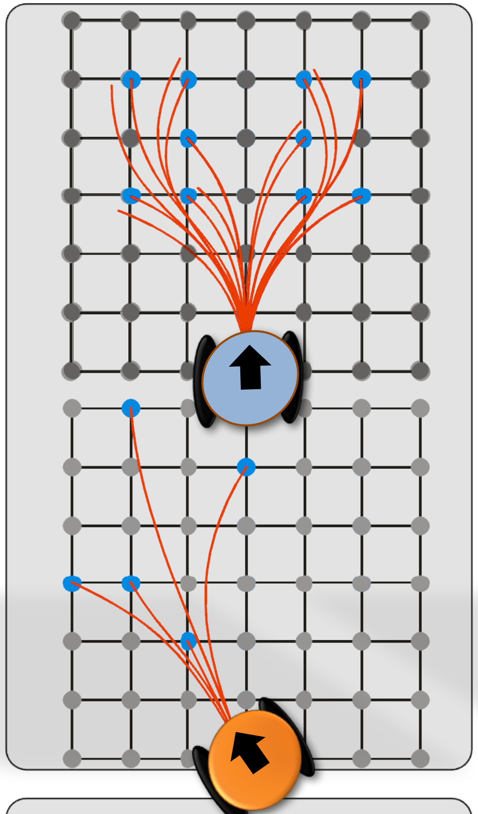
- Close relationship with map representation (Quadtree)!

Graph Construction: State Lattice Design (1/2)

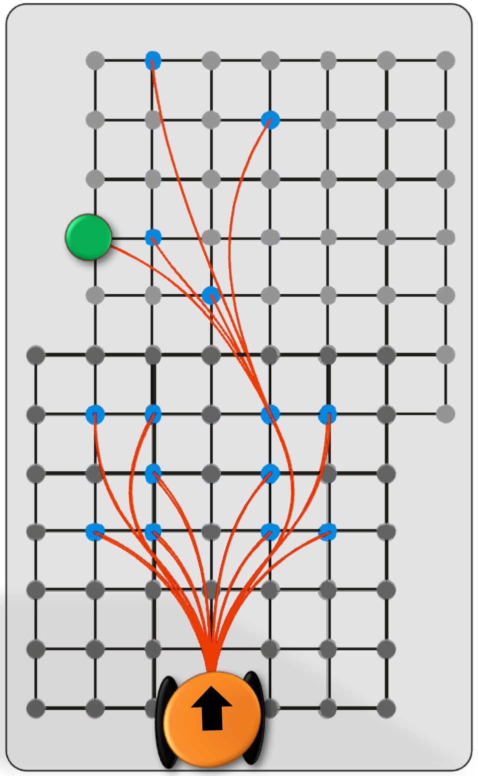
- Enforces edge feasibility



Offline:
Motion Model



Offline:
Lattice Gen.

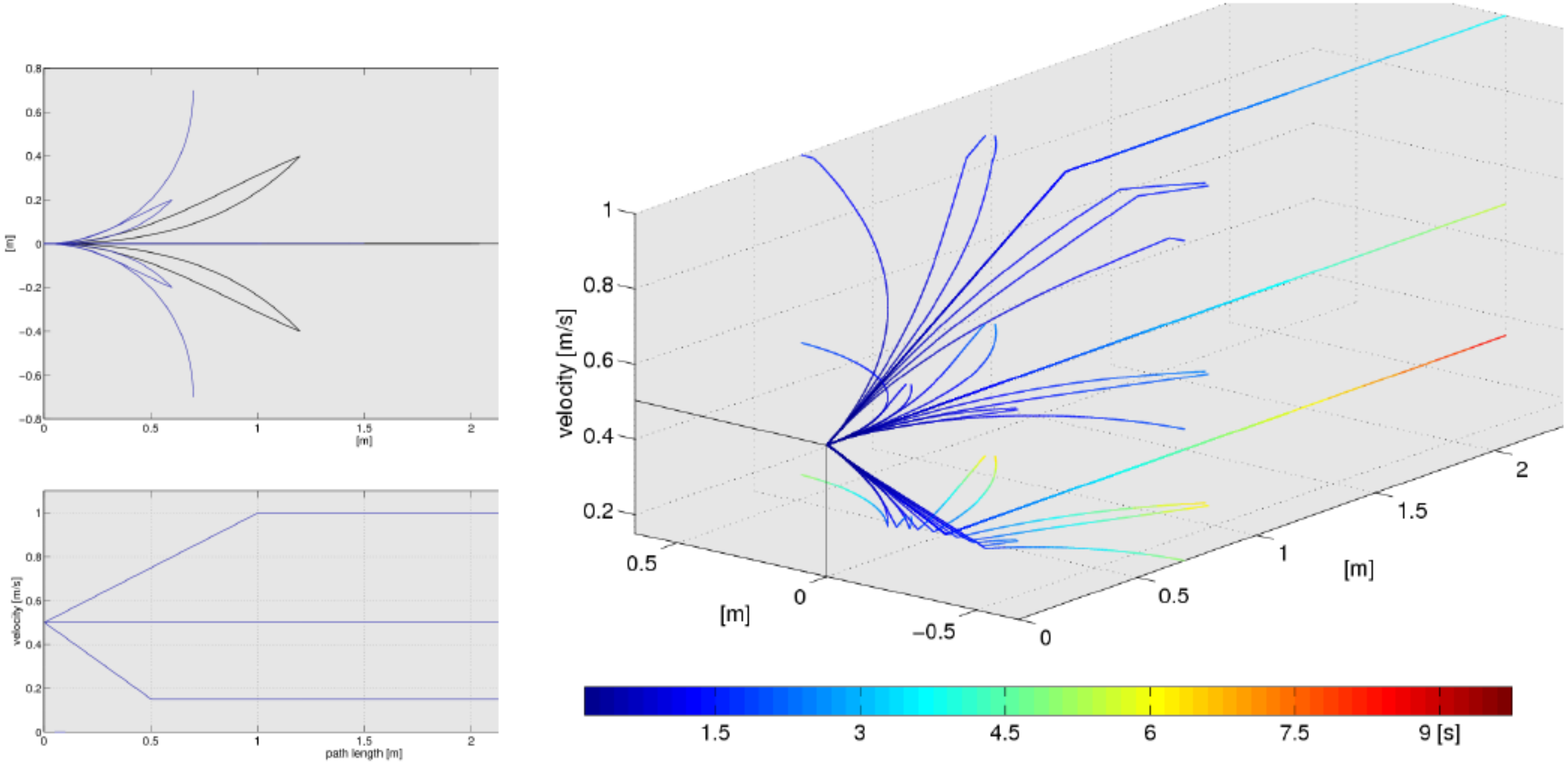


Online:
Incremental Graph Constr.

Graph Construction: State Lattice Design (2/2)

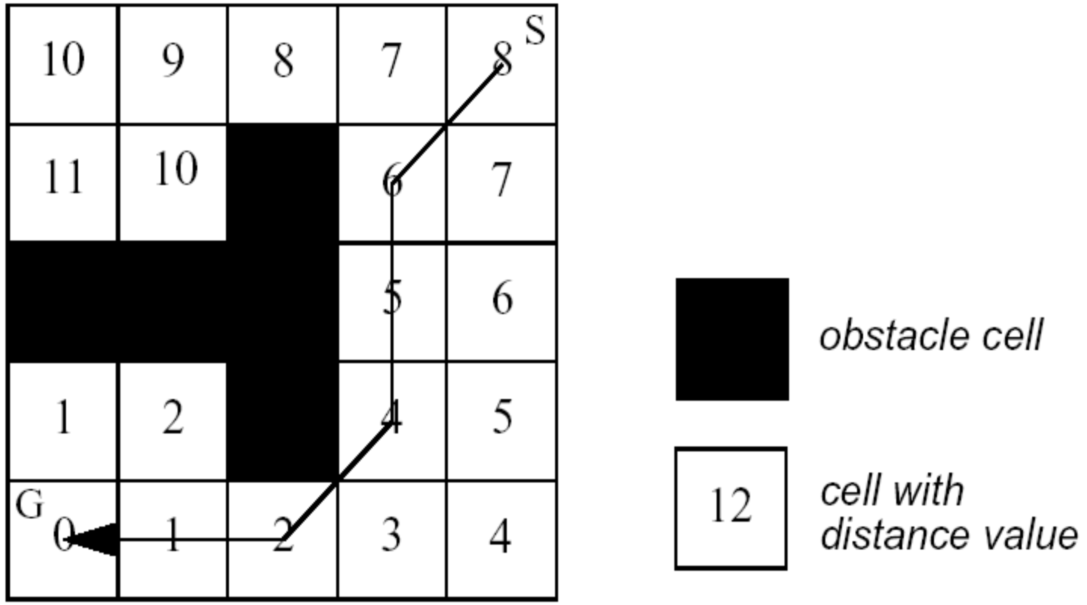
Martin Rufli

- 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

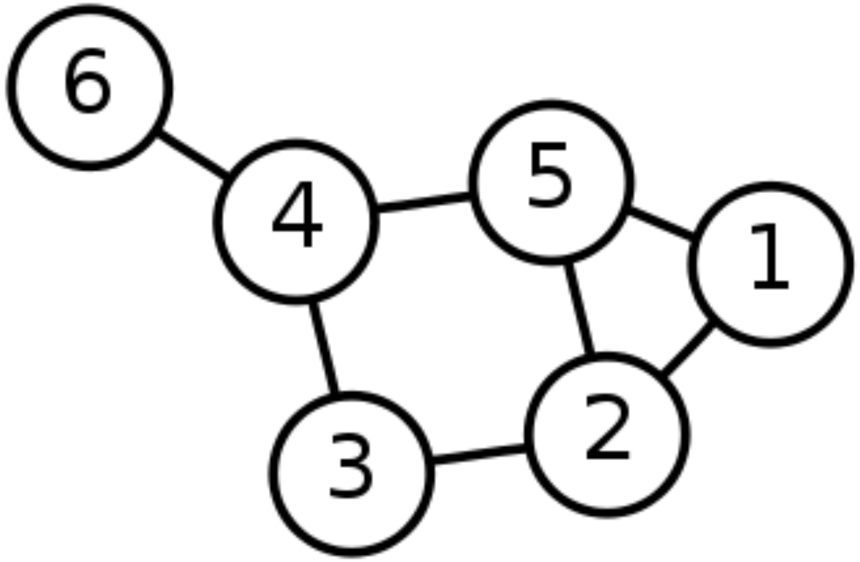
"Computer Science is no more about computers than astronomy is about telescopes."

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 material from http://www.cs.utexas.edu/~tandy/barrera.ppt

Diklstra's Algorithm

- Assign all vertices infinite distance to goal
- Assign 0 to distance from start
- Add all vertices to the queue

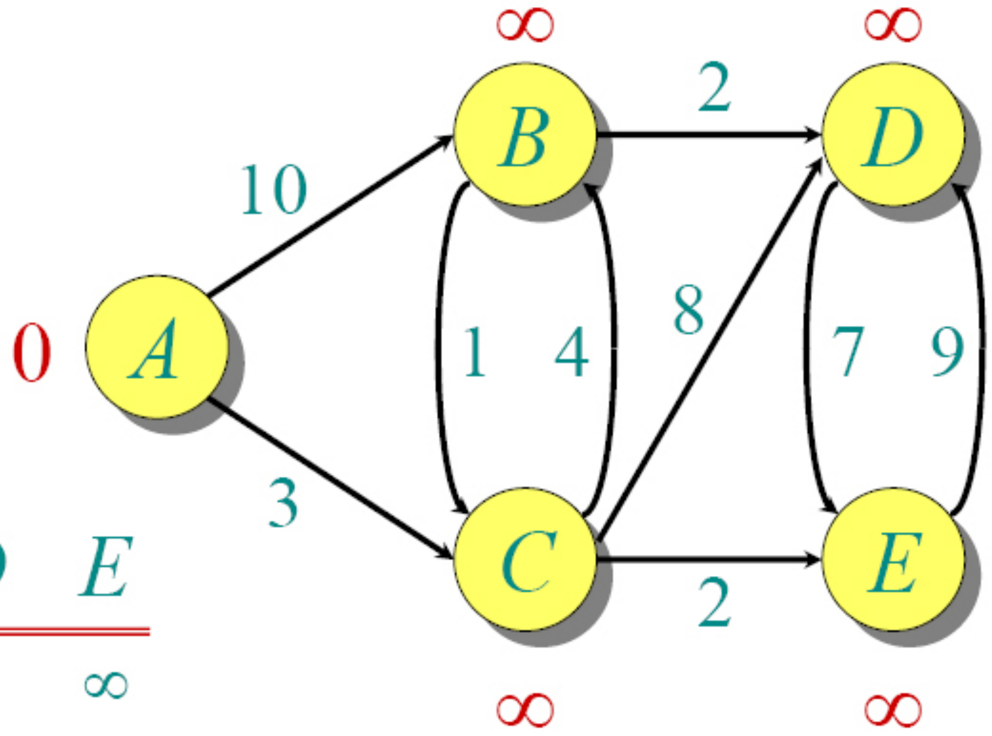
- While the queue is not empty:
  - Select vertex with smallest distance and remove it from the queue
  - Visit all neighbor vertices of that vertex,
  - calculate their distance and
  - update their (the neighbors) distance if the new distance is smaller

Diklstra's Algorithm - Pseudocode

dist[s] ← o	(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

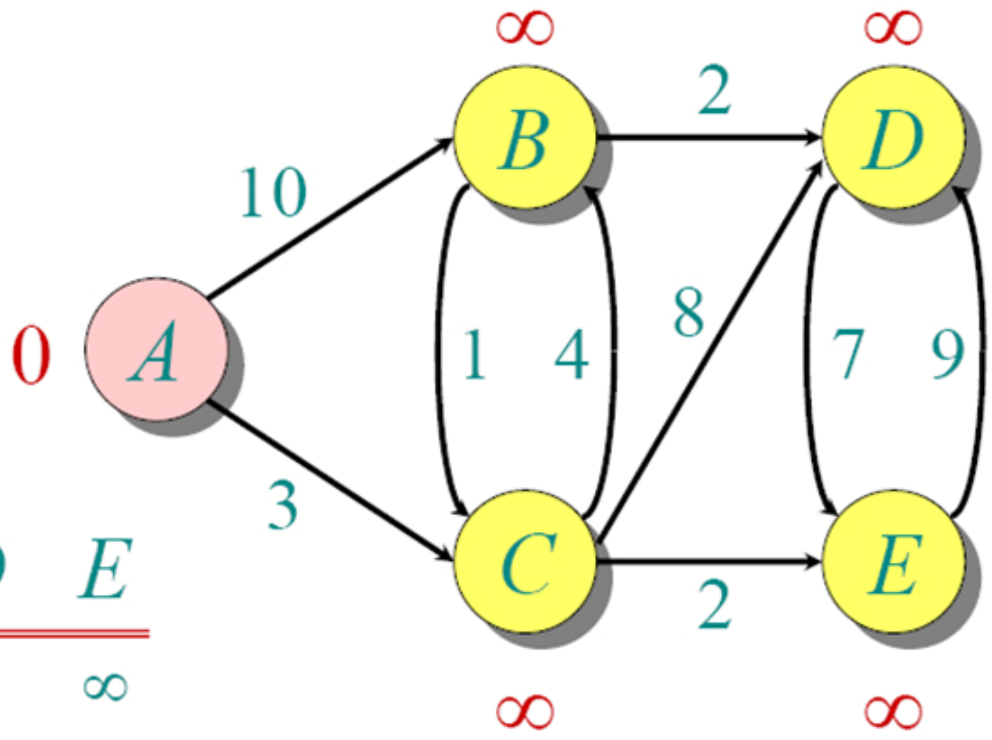
Initialize:



Q:	A	B	C	D	E
	0	∞	∞	∞	∞

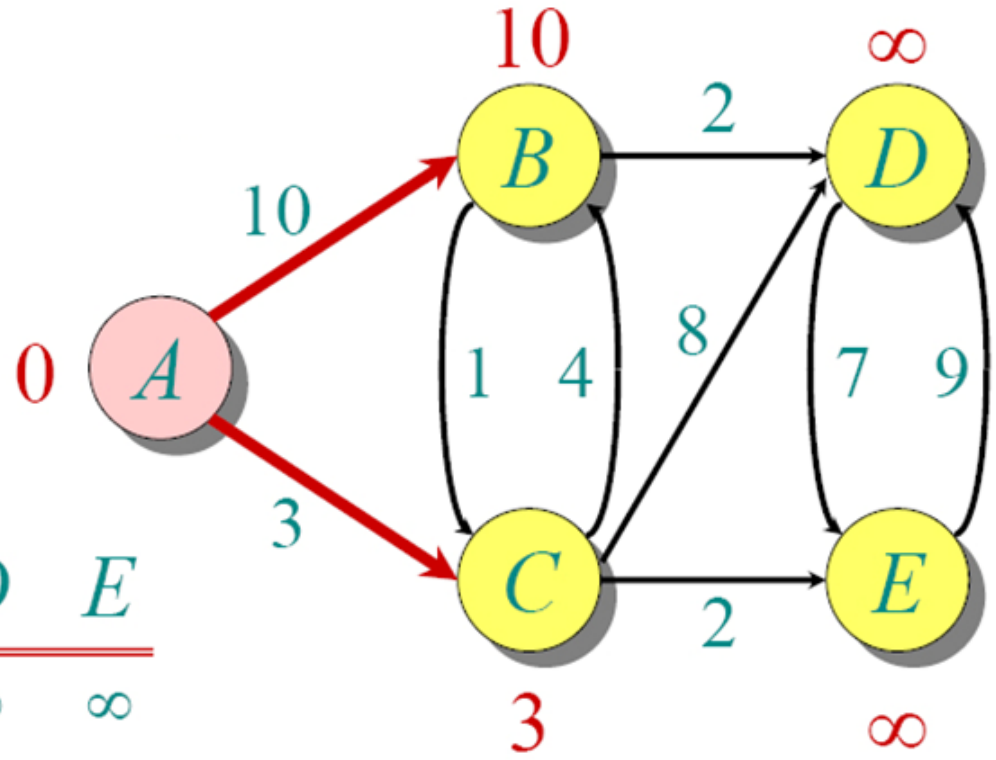
S: {}

Dijkstra Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞

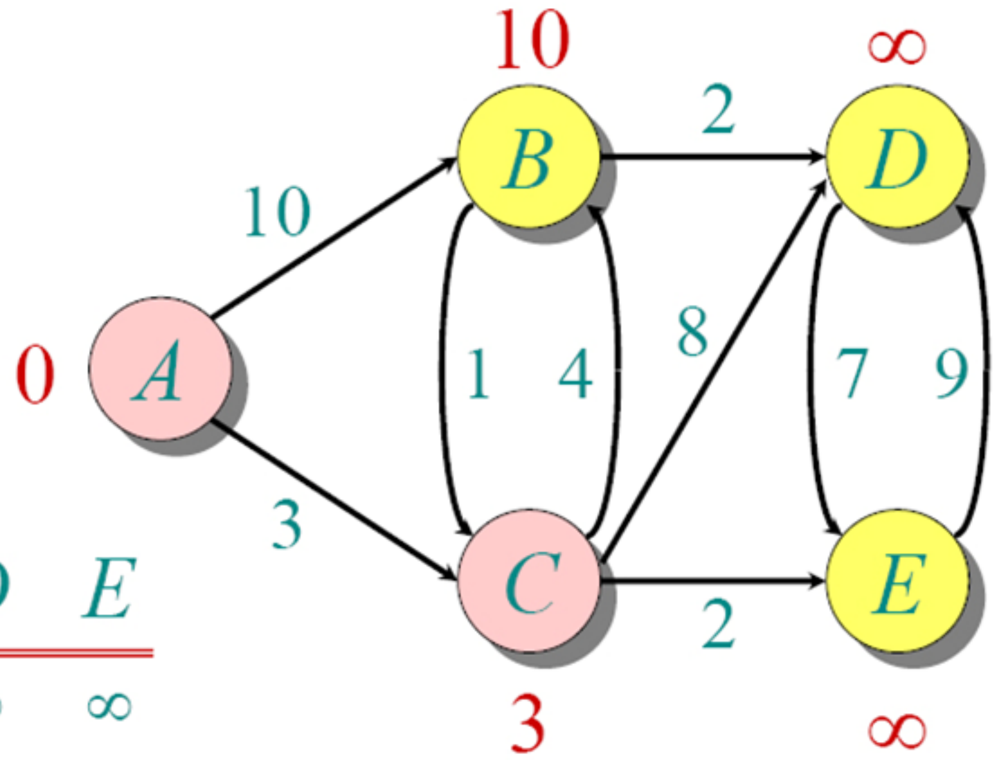
Dijkstra Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A }

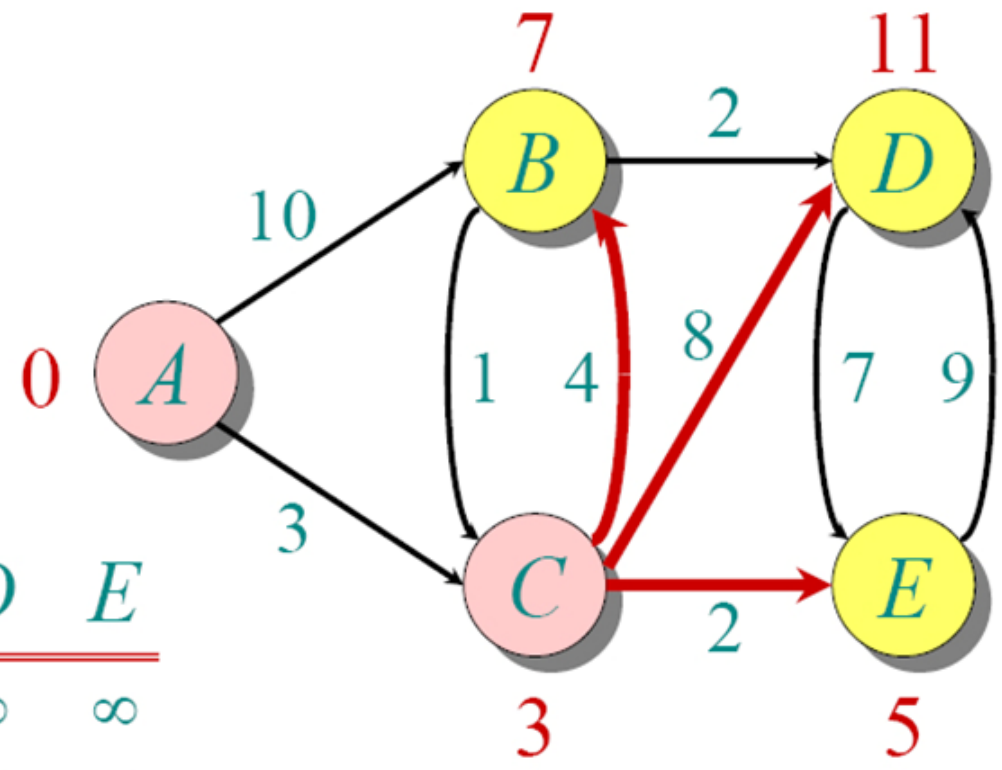
Dijkstra Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: $\{ A, C \}$

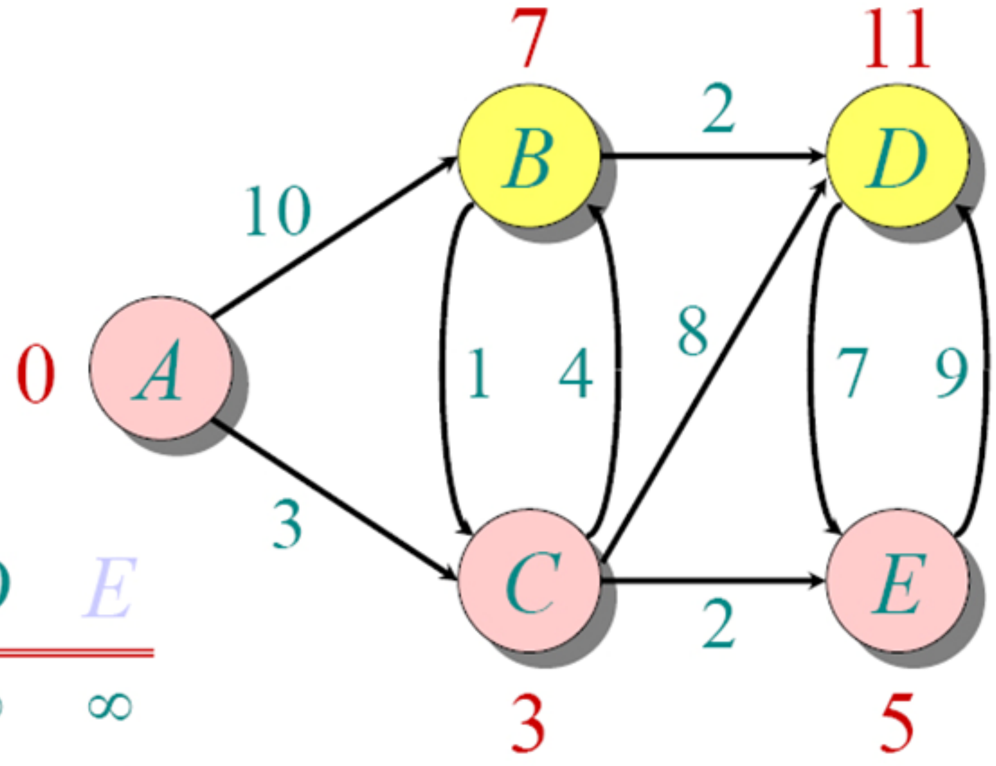
Dijkstra Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { A, C }

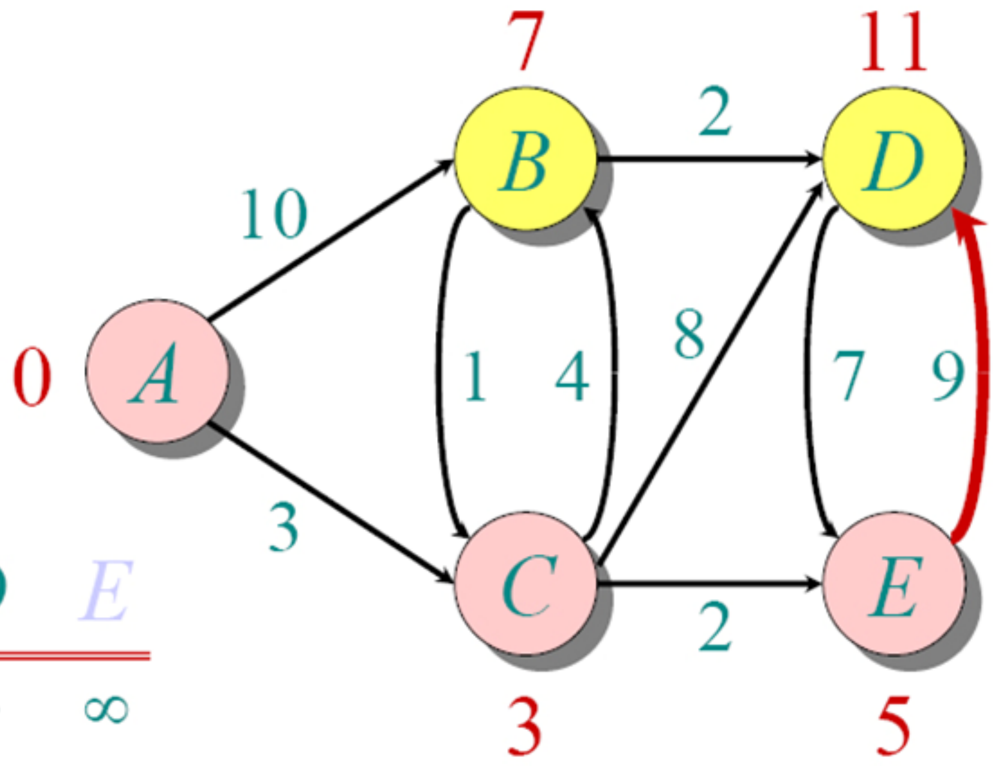
Dijkstra Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { A, C, E }

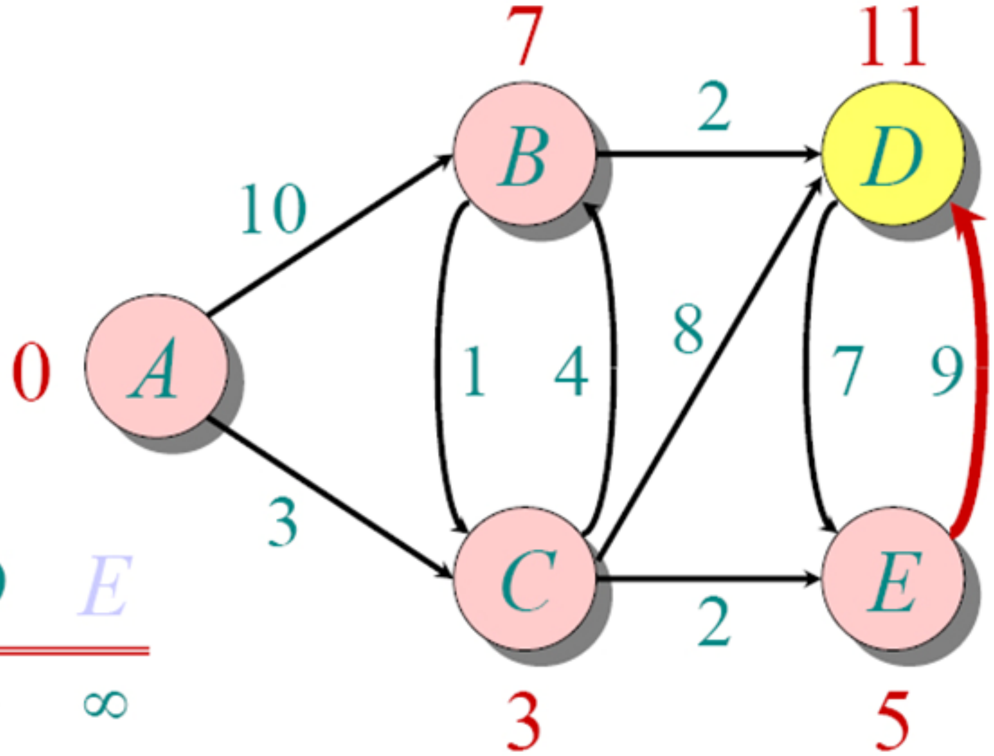
Dijkstra Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { A, C, E }

Dijkstra Example

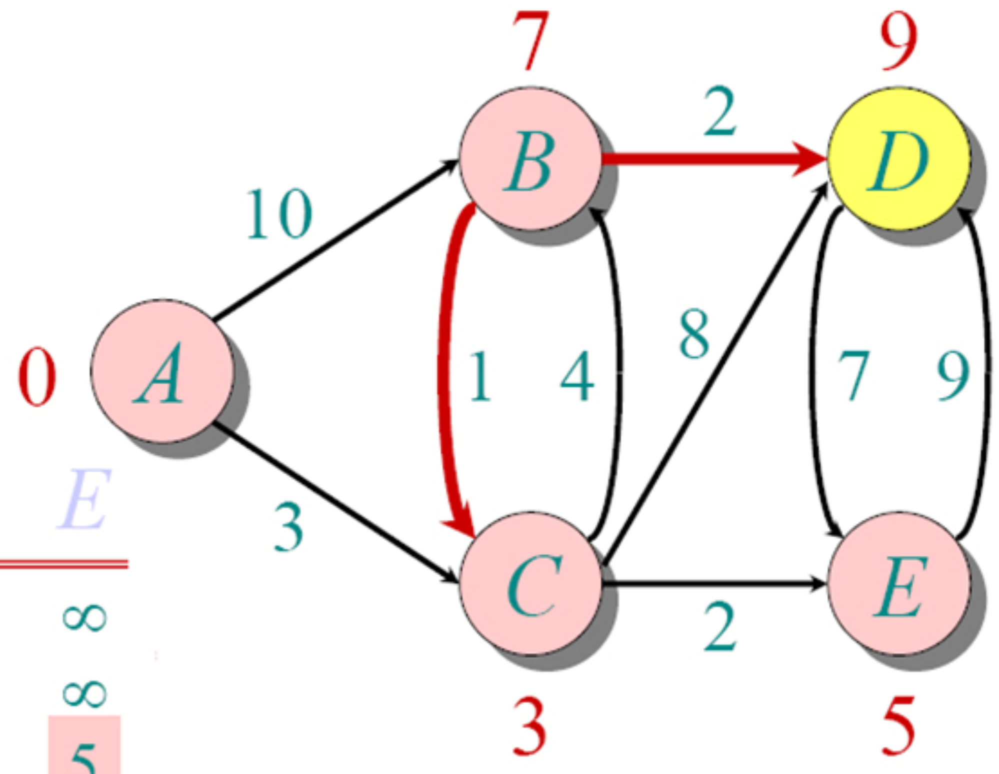


Q:	A	B	C	D	E
	$\boxed{0}$	∞	∞	∞	∞
		10	$\boxed{3}$	∞	∞
		7		11	$\boxed{5}$
		$\boxed{7}$		11	

S: { A, C, E, B }

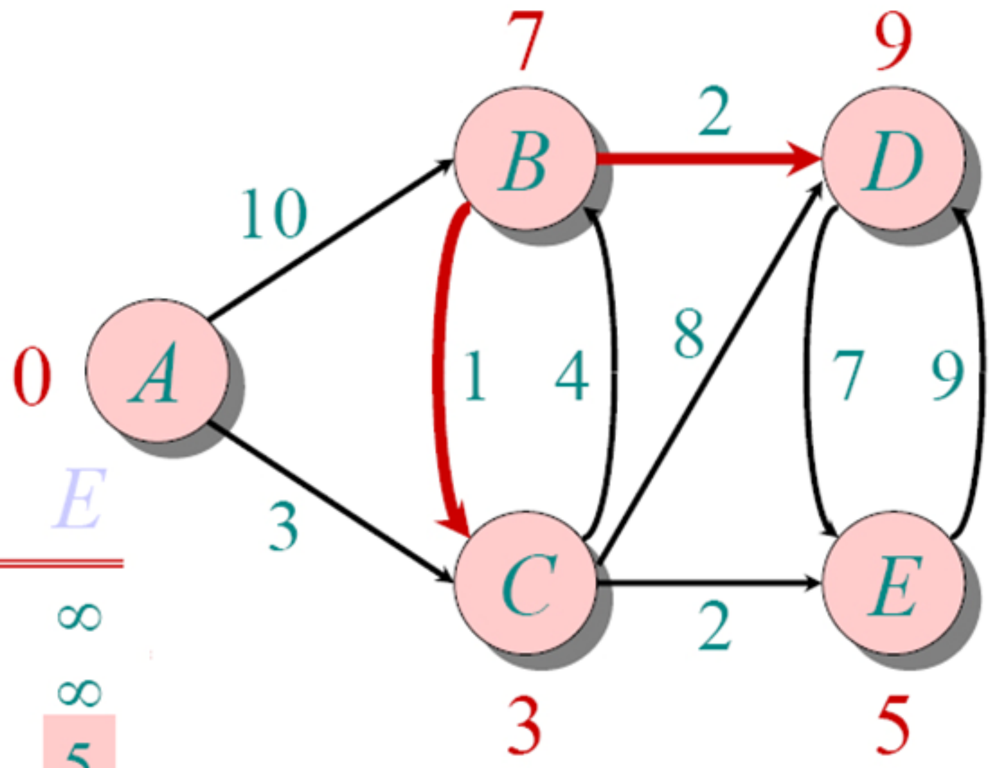
Dijkstra Example

Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	



S: { A, C, E, B }

Dijkstra Example

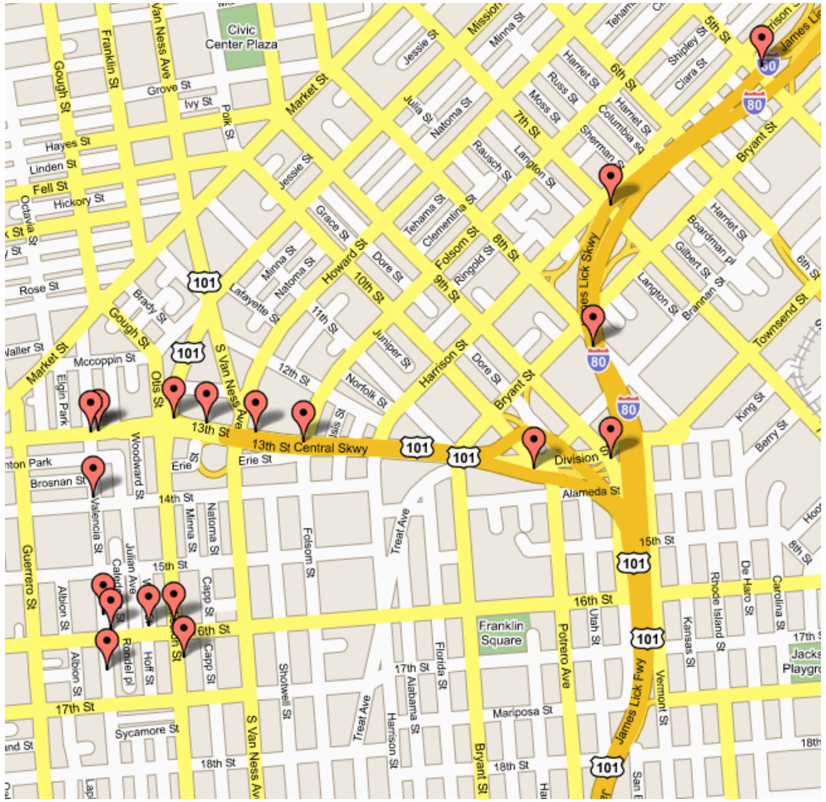


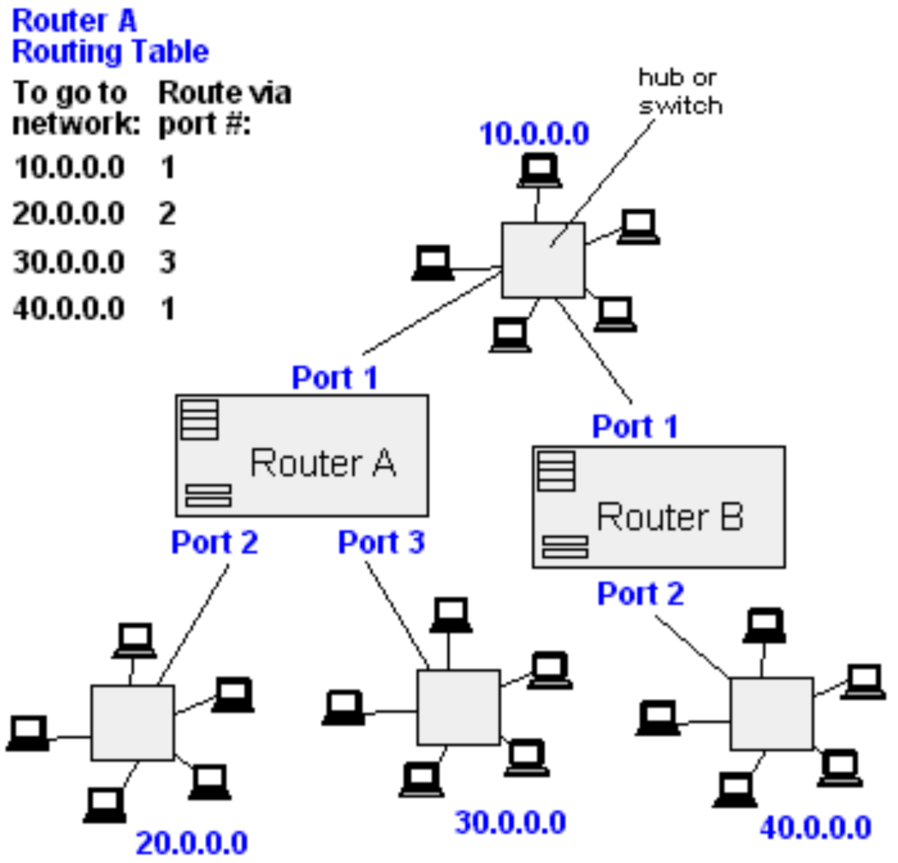
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

S: { A, C, E, B, D }

APPLICATIONS OF DIJKSTRA'S ALGORITHM

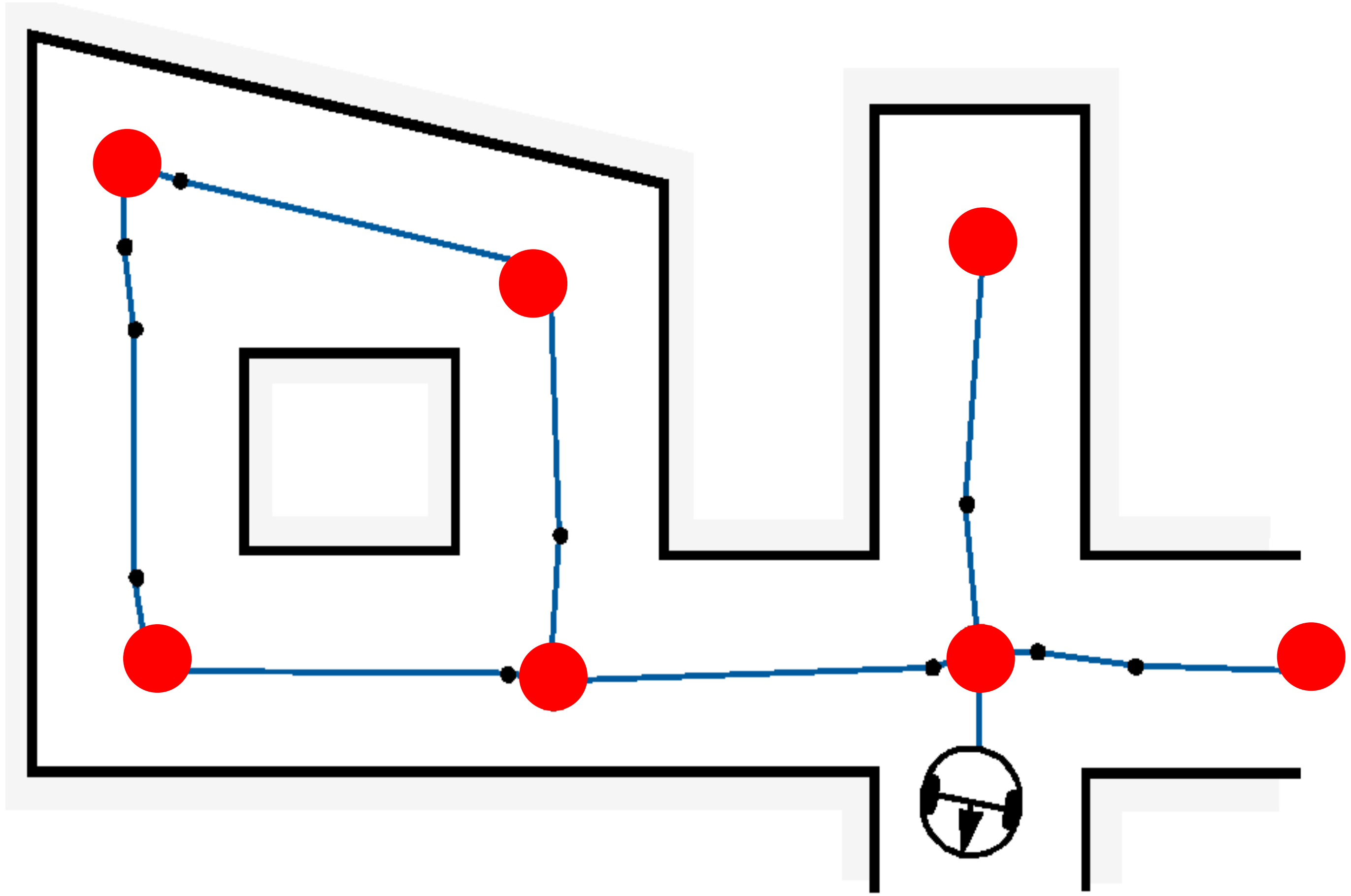
- Navigation Systems
- Internet Routing





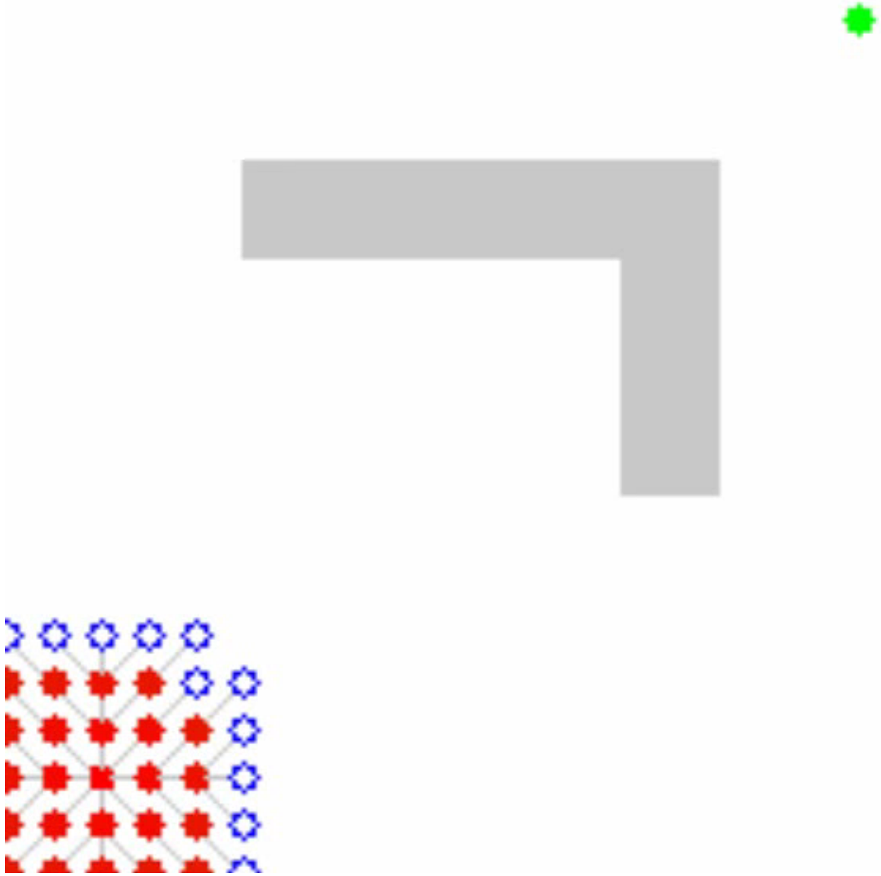
Dijkstra’s Algorithm for Path Planning: Topological Maps

- 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



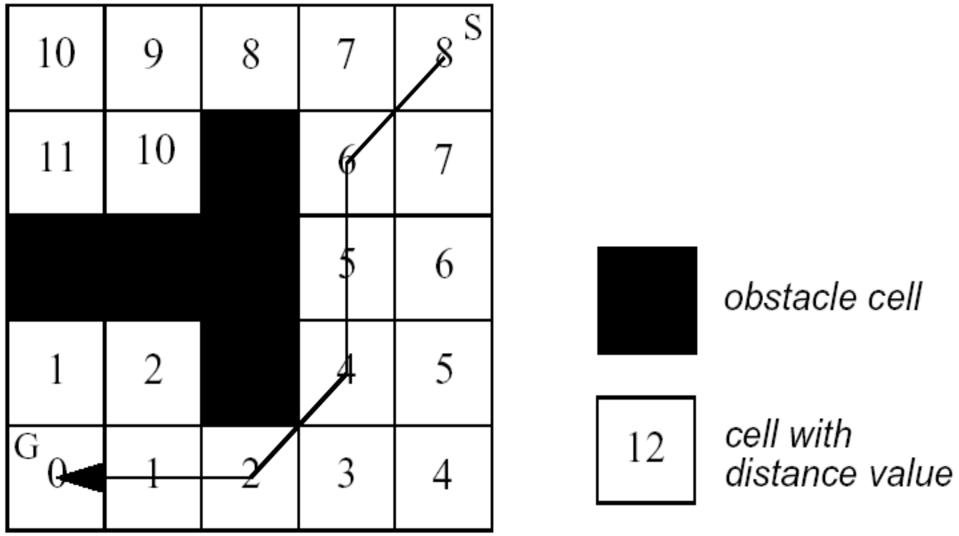
Dijkstra's Algorithm for Path Planning: Grid Maps

- 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

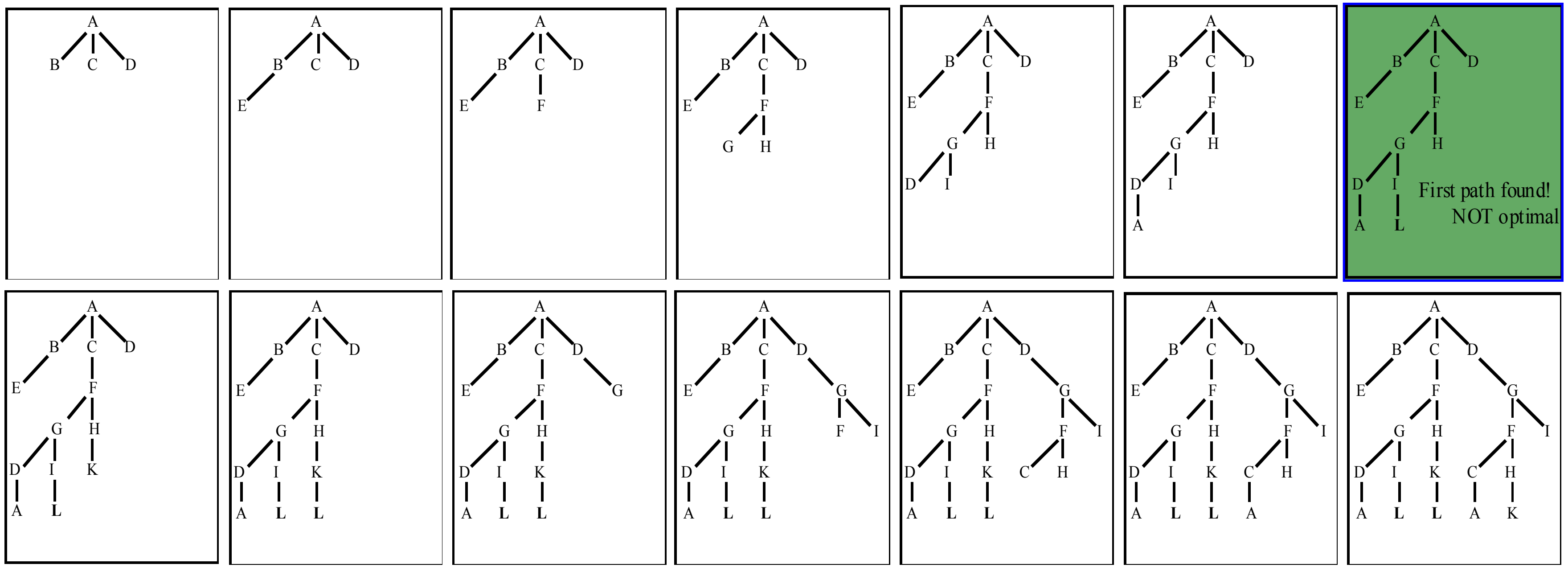


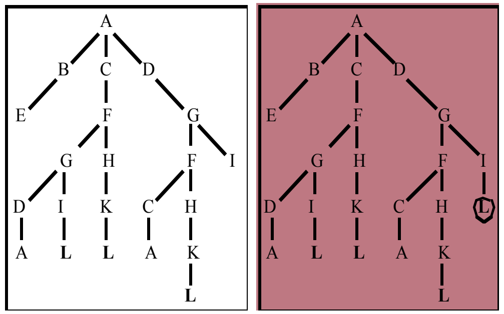
Graph Search Strategies: Breath-First Search

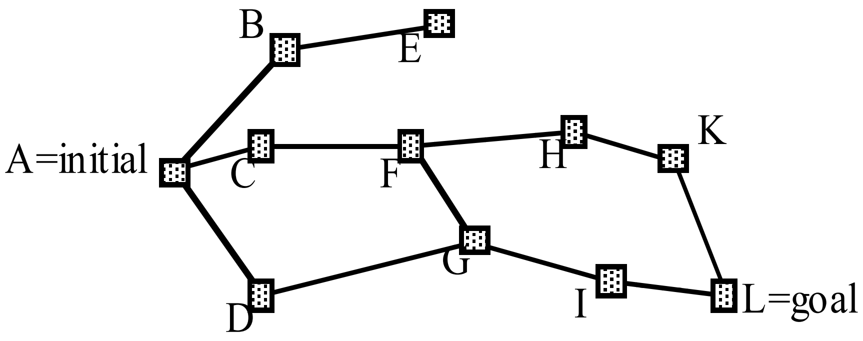
- Corresponds to a wavefront expansion on a 2D grid
- Breath-First: Dijkstra's search where all edges have weight 1



Graph Search Strategies: Depth-First Search

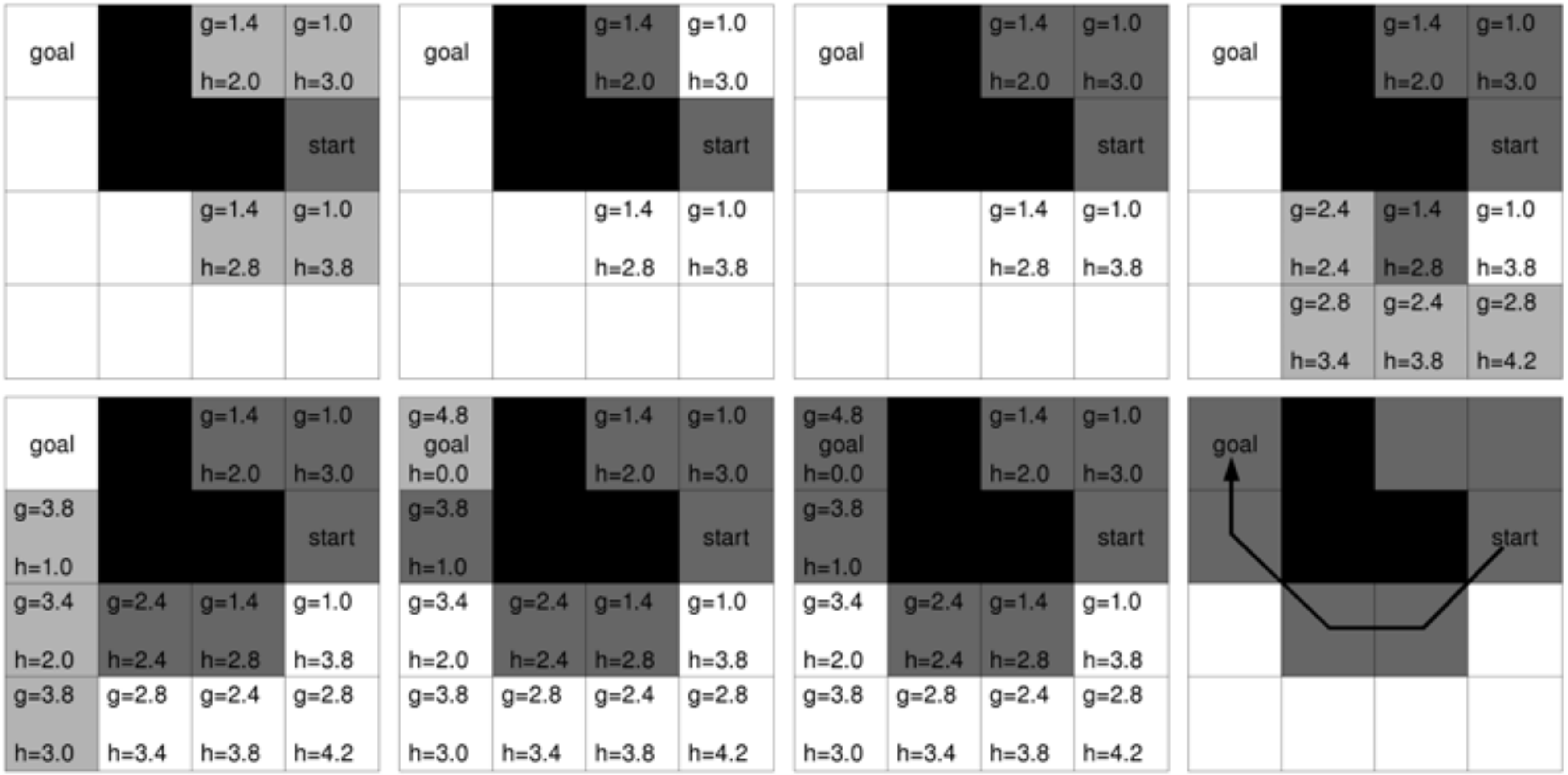






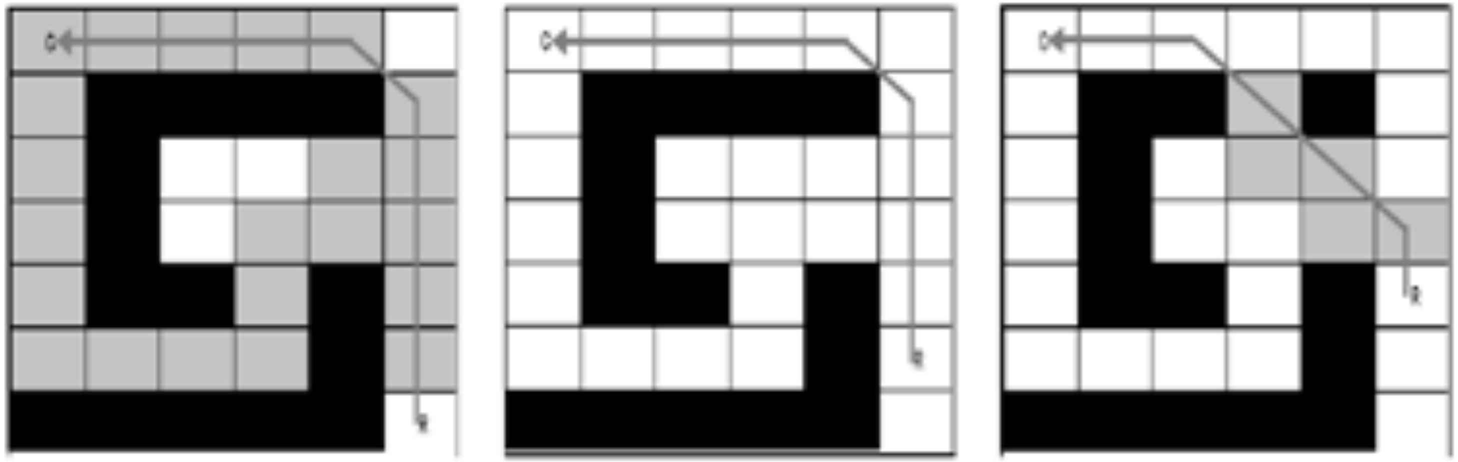
Graph Search Strategies: A* Search

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



Graph Search Strategies: D* Search

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

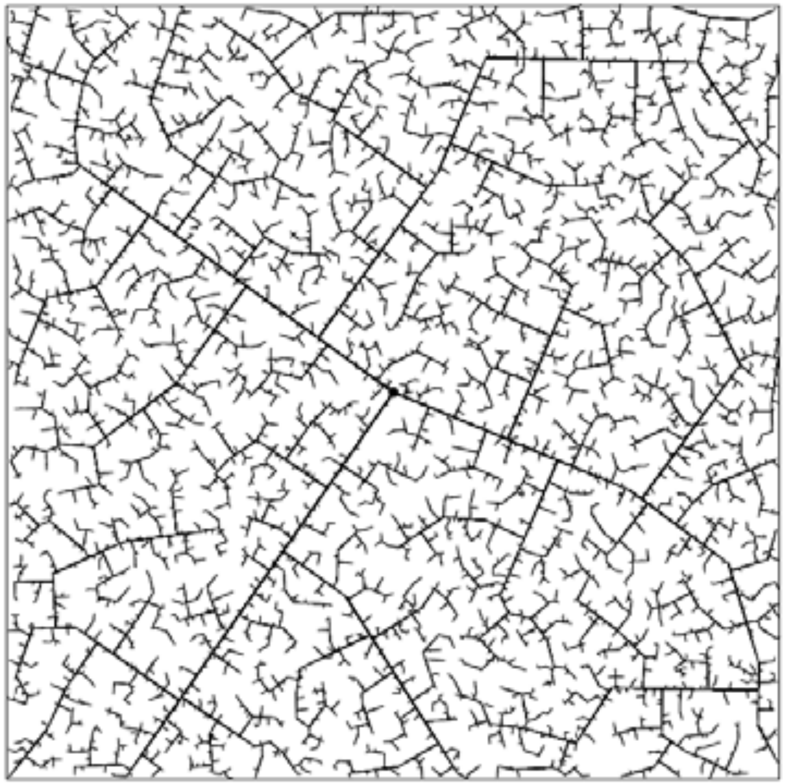


$\epsilon = 1.0$ $\epsilon = 1.0$ $\epsilon = 1.0$

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

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