Intro to Robotics

Lecture 11

Potential Field -- Basic Idea

- Model physics of robot
- Attract to goal
- Repulse from obstacles

Basic Idea

- Originally was described in terms of potentials
  - Potential energy is energy at a position (or configuration)
  - integral of force $$U(q) = -\int_{q_0}^{q} F(q)\,dq + U(q_0)$$
  - Force is derivative of potential energy

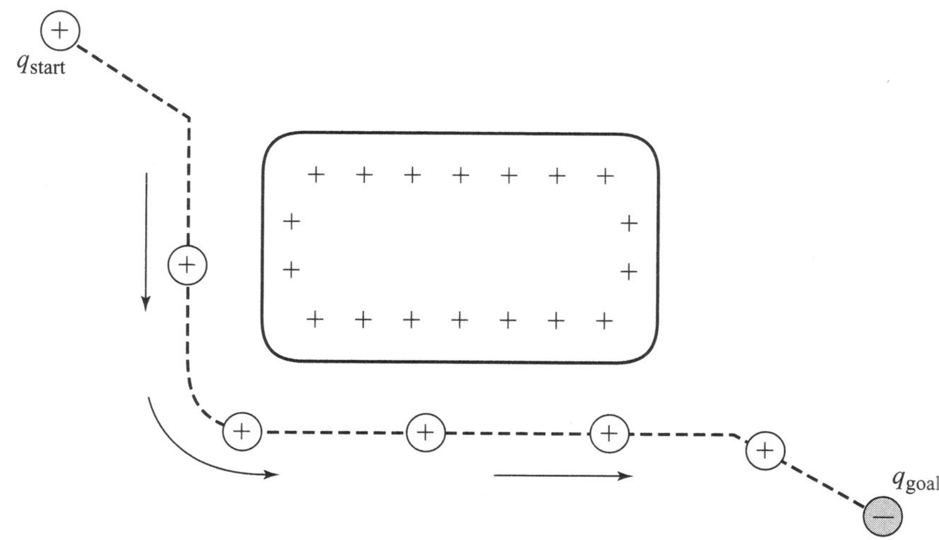
$$\frac{dU}{dq} = -F(q)$$

    - Gradient in higher dimensions

$$grad(U(q)) = \nabla U(q) = (\frac{\partial U}{\partial q_1}, \ldots, \frac{\partial U}{\partial q_n})$$

Potential Field Path Planning

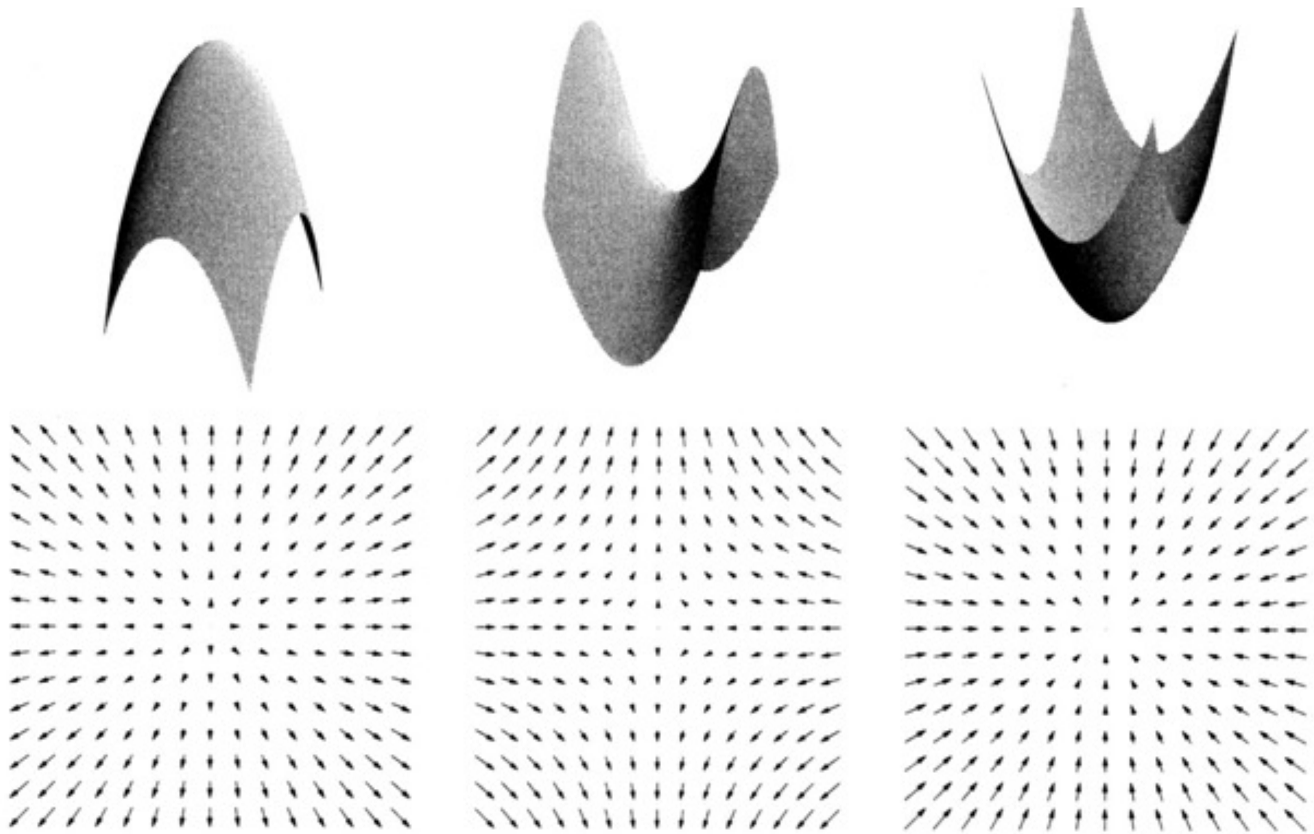
- Potential function guides the robot as if it were a particle moving in a gradient field.
- Analogy: robot is positively charged particle, moving towards negative charge goal
- Obstacles have “repulsive” positive charge



- Potential functions can be viewed as a landscape
- Robot moves from high-value to low-value

  Using a “downhill” path (i.e negative of the gradient).
- This is known as gradient descent –follow a functional surface until you reach its minimum
  - Really, an extremum

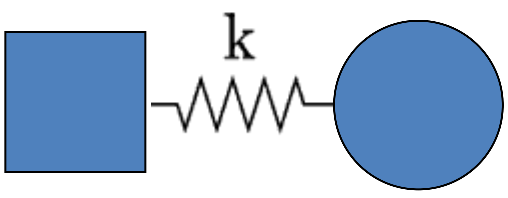
Potential Field



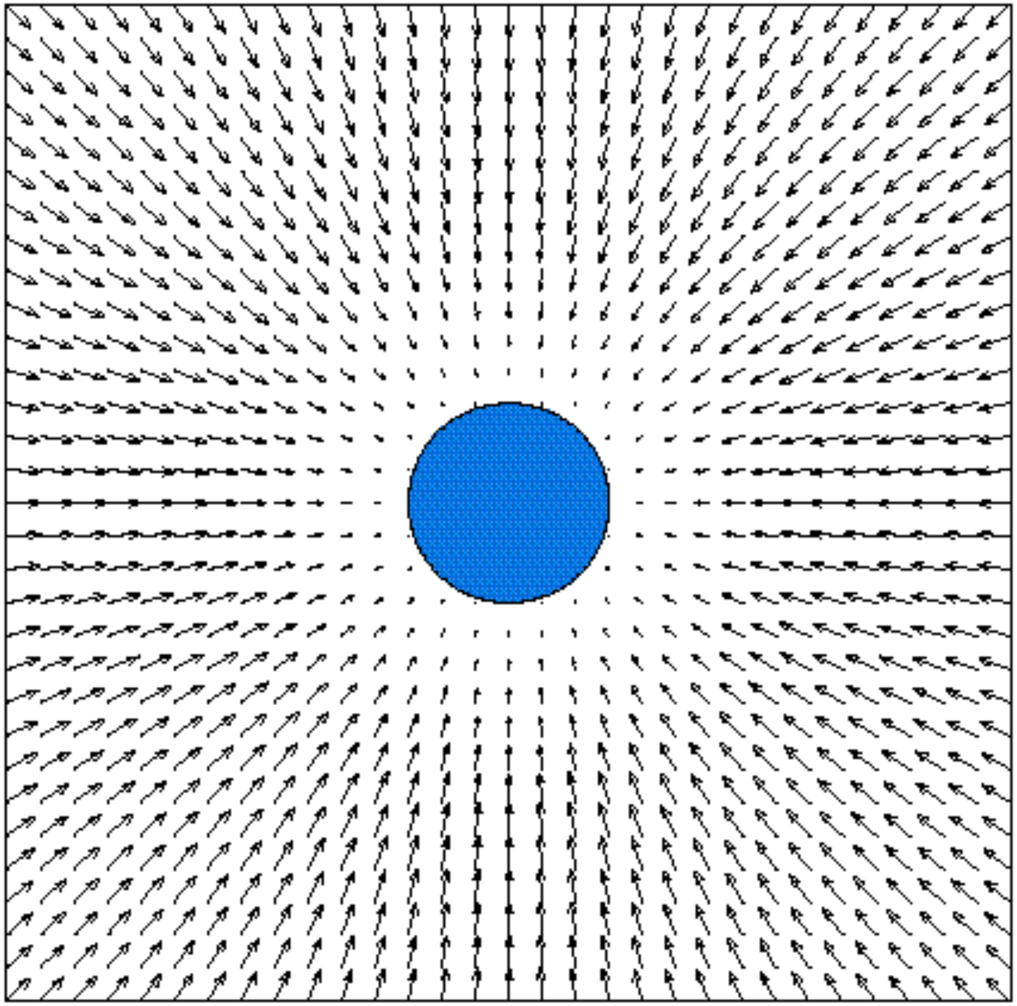
(Maximum) (Saddle) (Minimum)

Linear Force

- Force is linear with distance
  - Like the spring force



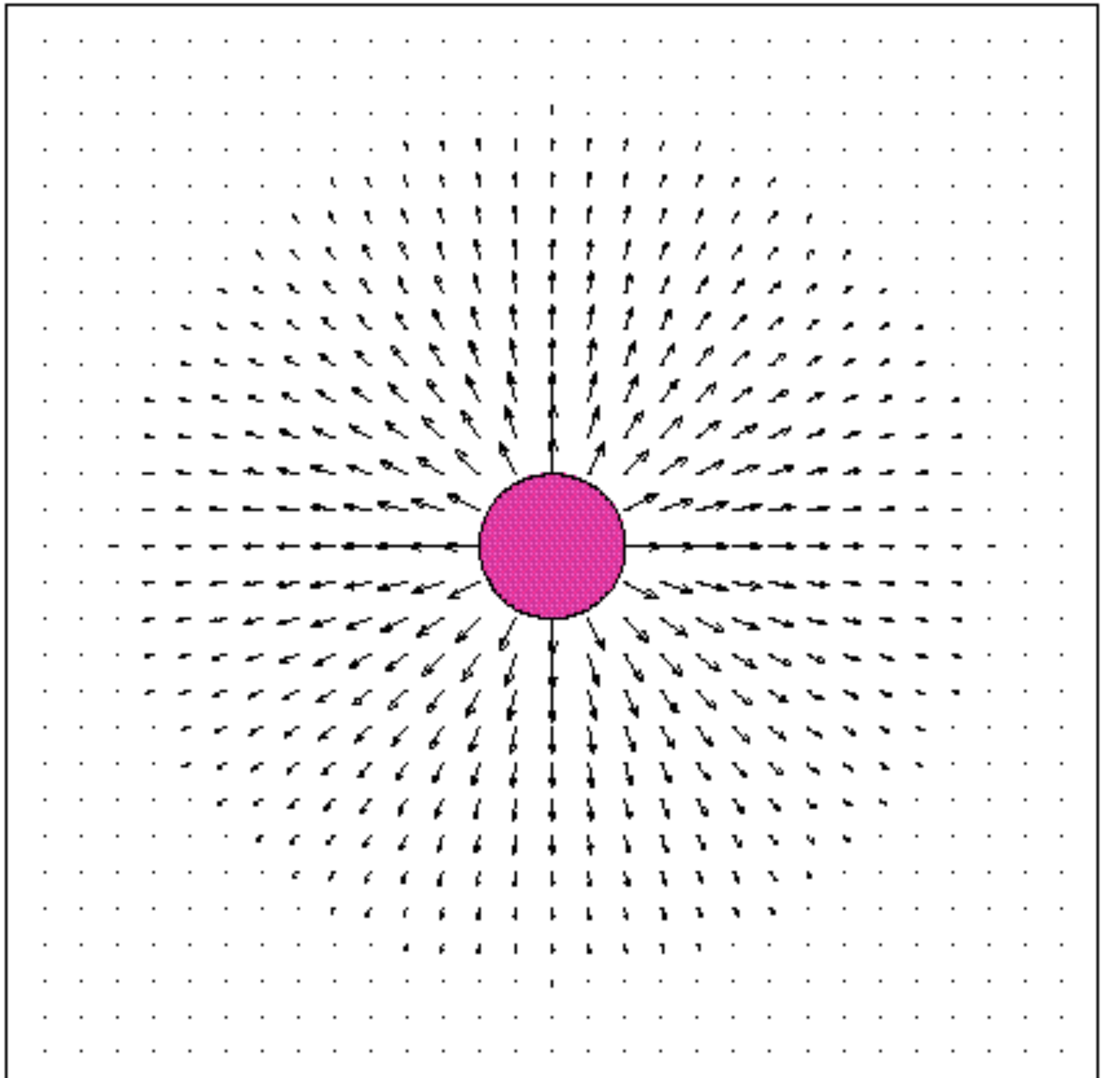
Attractive Potential Field



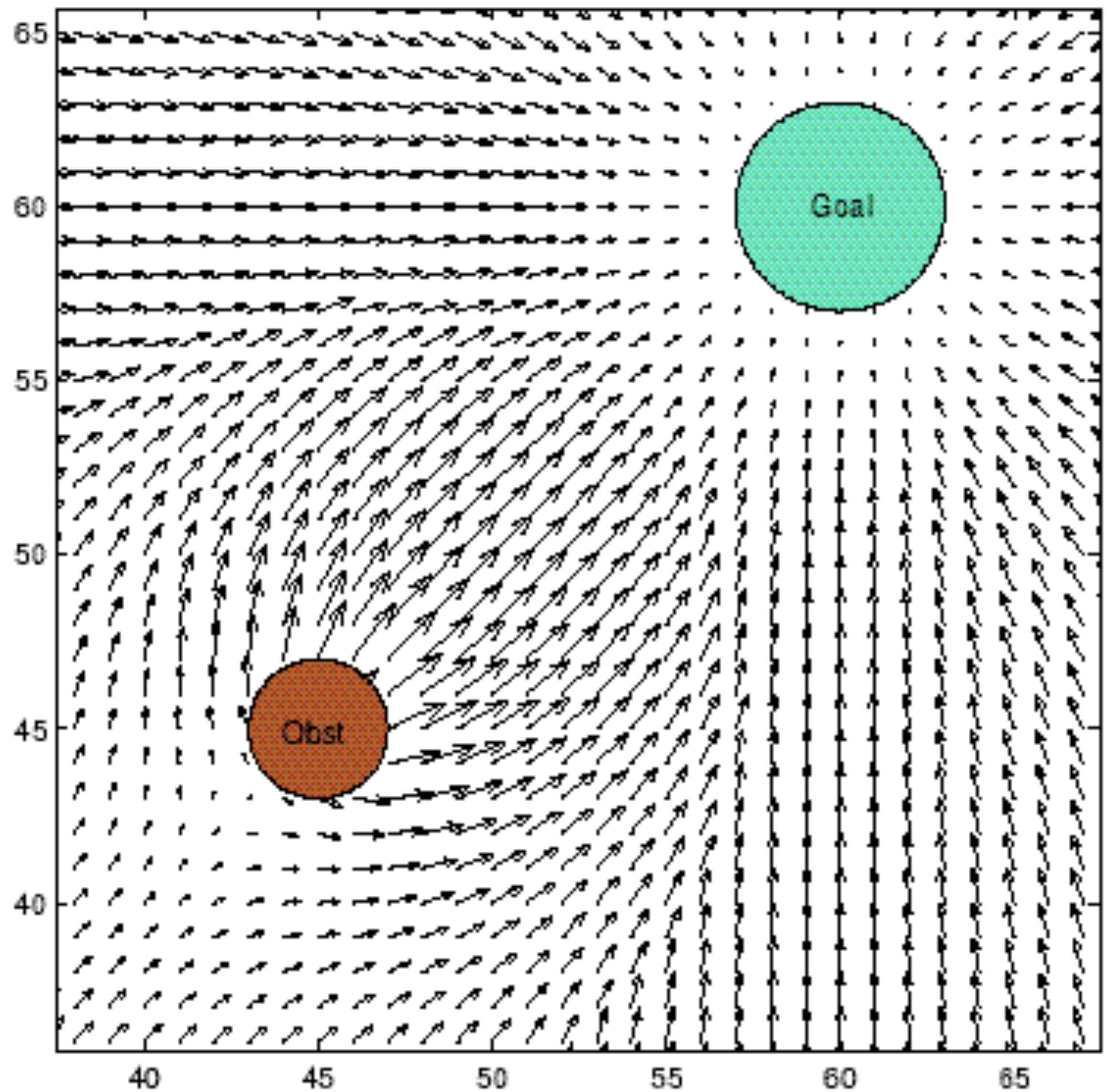
Repulse from Obstacles

- Use inverse quadratic
  - 1/dist^2
  - What is that force law like?

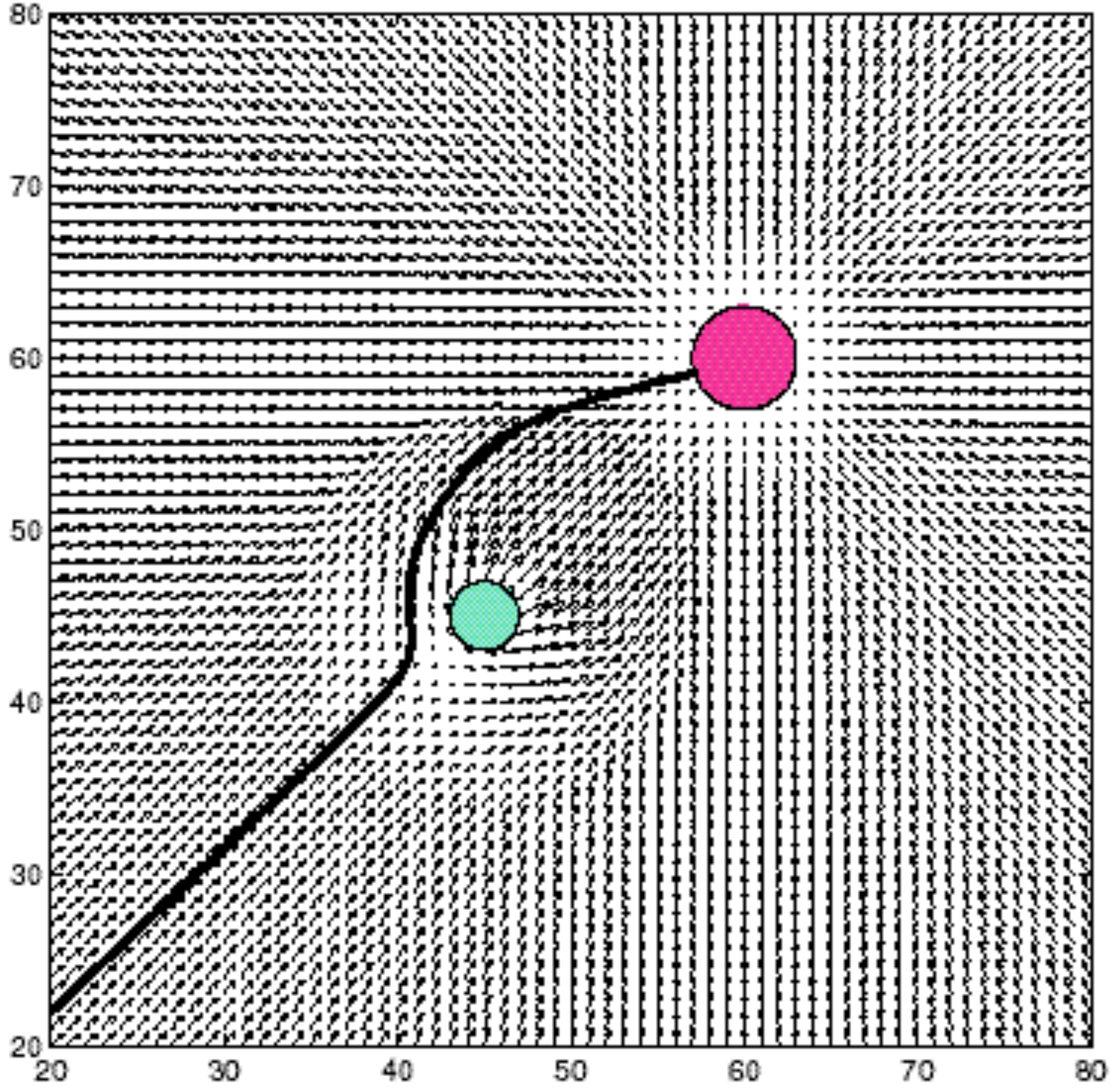
Repulsive Potential Field



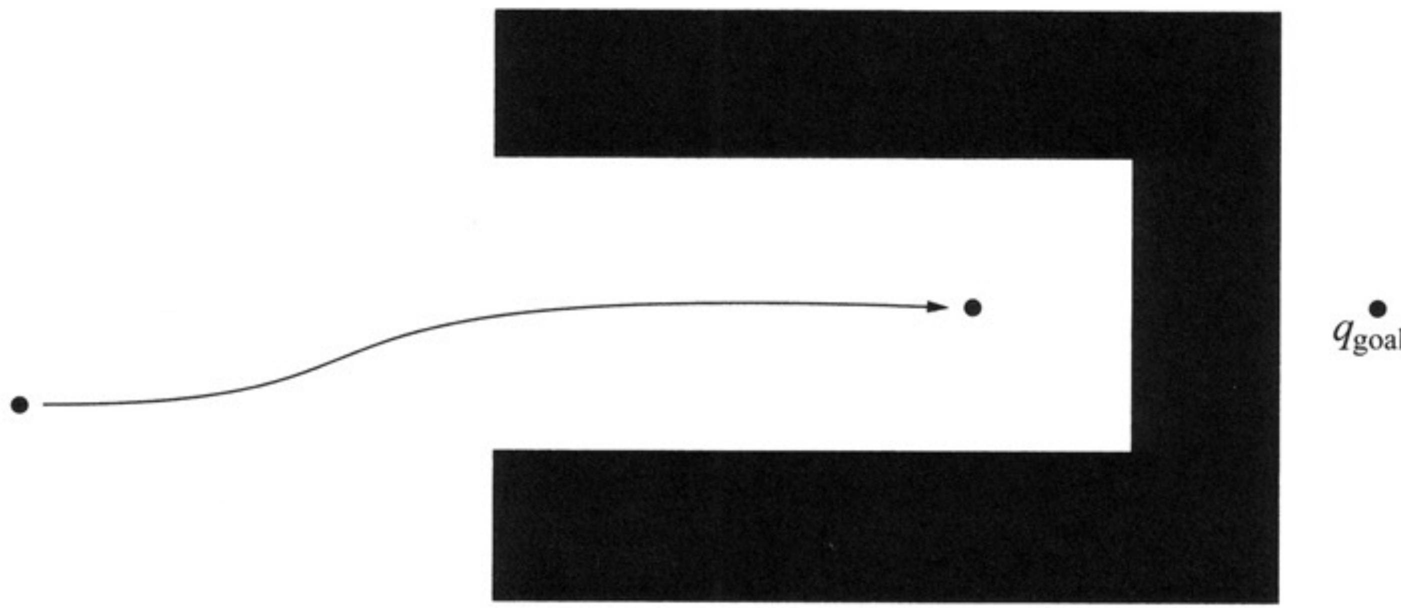
Vector Sum of Two Fields



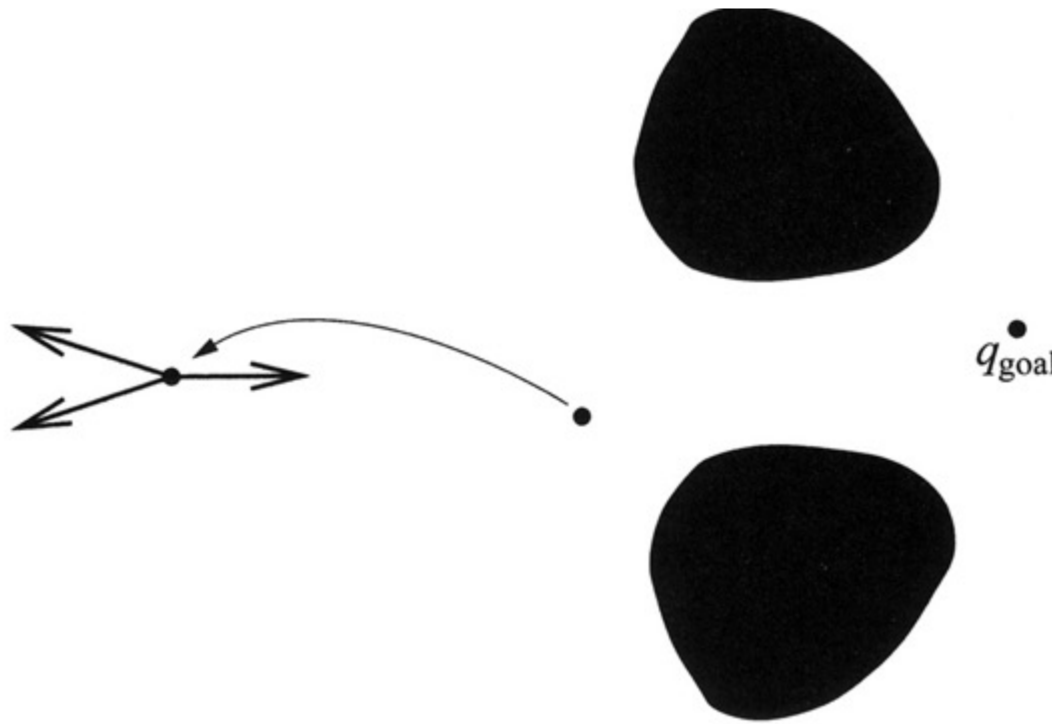
Resulting Robot Trajectory



Main Problem



Local minimum inside the concavity.



Local minimum without concave obstacles

Some solutions to local minima

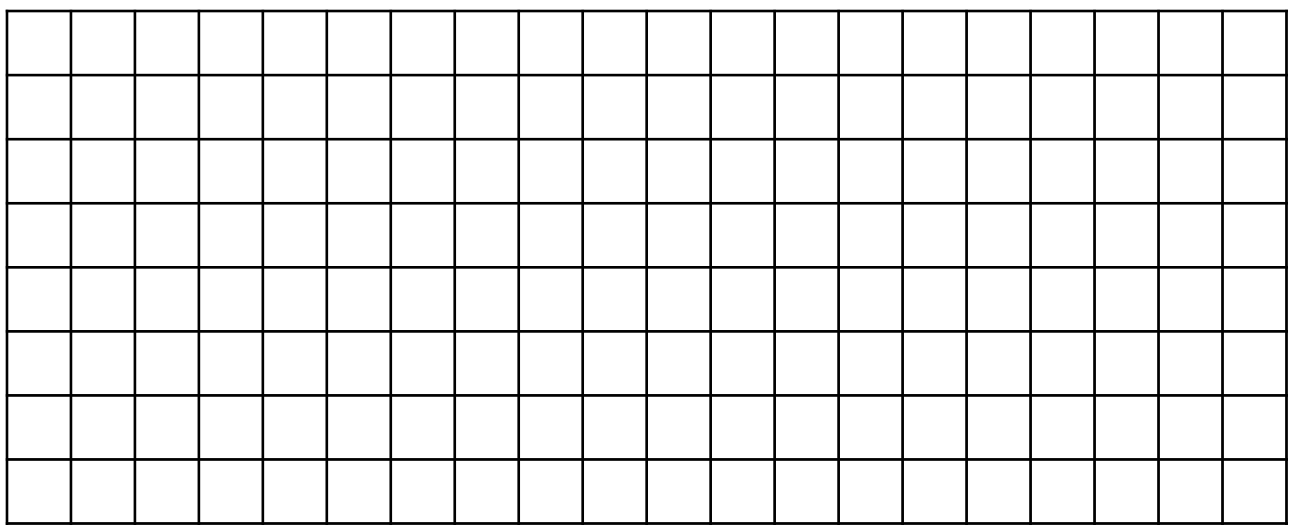
- Build graph from local minima
  - Search graph
- Random perturbation to escape
  - Make sure you don’t push into obstacle
- Change parameters to get unstuck
  - Might not work
- Build potential field with only one minimum
  - Navigation function

Wavefront planner

- Use BFS on a grid
- Label cells with values
  - Start with zero
  - Expand from goal
  - Add +1 to neighbors of current wavefront
  - Use gradient descent to search from start to goal

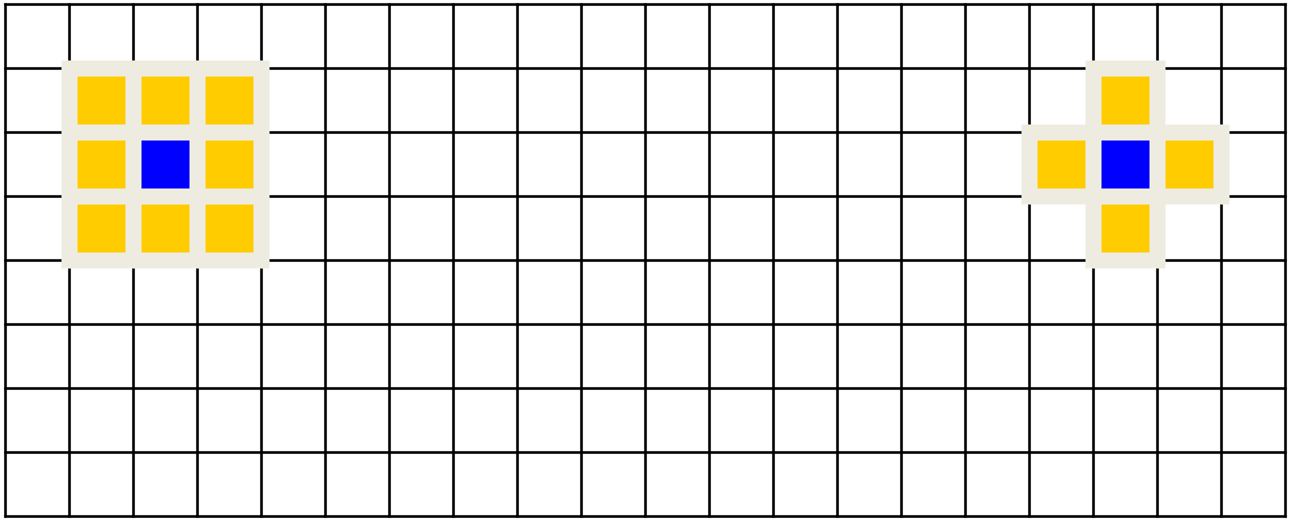
Representations: A Grid

- Distance is reduced to discrete steps
  - For simplicity, we'll assume distance is uniform
- Direction is now limited from one adjacent cell to another



Representations: Connectivity

- 8-Point Connectivity
  - *(chessboard metric)*
- 4-Point Connectivity
  - *(Manhattan metric)*



The Wavefront Planner: Setup

7	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0	0
3	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**2**
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The Wavefront in Action (Part 1)

- Starting with the goal, set all adjacent cells with "0" to the current cell + 1
  - 4-Point Connectivity or 8-Point Connectivity?
  - Your Choice. We'll use 8-Point Connectivity in our example

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
7	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0	0
3	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	3
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	**2**

The Wavefront in Action (Part 2)

- Now repeat with the modified cells
  - This will be repeated until goal is reached
- 0's will only remain when regions are unreachable

7	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0	0
3	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4
1	0	0	0	0	0	0	0	0	0	0	0	0	0	4	3	3
0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	3	**2**
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The Wavefront in Action (Part 3)

- Repeat again...

7	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0	0
3	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	5	5	5	5
2	0	0	0	0	0	0	0	0	0	0	0	0	5	4	4	4
1	0	0	0	0	0	0	0	0	0	0	0	0	5	4	3	3
0	0	0	0	0	0	0	0	0	0	0	0	0	5	4	3	**2**
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The Wavefront in Action (Part 4)

- And again...

7	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	6	6	6	6
3	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	5	5	5	5
2	0	0	0	0	0	0	0	0	0	0	0	6	5	4	4	4
1	0	0	0	0	0	0	0	0	0	0	0	6	5	4	3	3
0	0	0	0	0	0	0	0	0	0	0	0	6	5	4	3	**2**
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The Wavefront in Action (Part 5)

- And again until...

7	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	7	7	7	7	7
4	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	6	6	6	6
3	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	5	5	5	5
2	0	0	0	0	0	0	0	0	0	0	7	6	5	4	4	4
1	0	0	0	0	0	0	0	0	0	0	7	6	5	4	3	3
0	0	0	0	0	0	0	0	0	0	0	7	6	5	4	3	**2**
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The Wavefront in Action (Done)

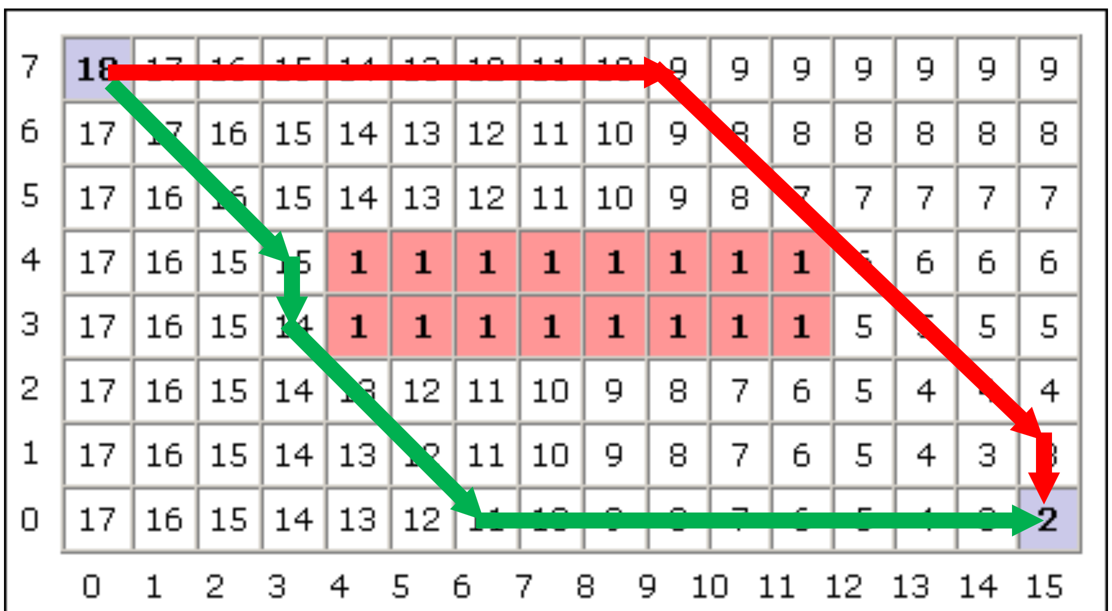
- You’re done

7	**18**	17	16	15	14	13	12	11	10	9	9	9	9	9	9	9
6	17	17	16	15	14	13	12	11	10	9	8	8	8	8	8	8
5	17	16	16	15	14	13	12	11	10	9	8	7	7	7	7	7
4	17	16	15	15	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	6	6	6	6
3	17	16	15	14	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	5	5	5	5
2	17	16	15	14	13	12	11	10	9	8	7	6	5	4	4	4
1	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	3
0	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	**2**
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The Wavefront

- To find the shortest path simply always move toward a cell with a lower number
  - The numbers generated by the Wavefront planner are roughly proportional to their distance from the goal

Two possible shortest paths shown



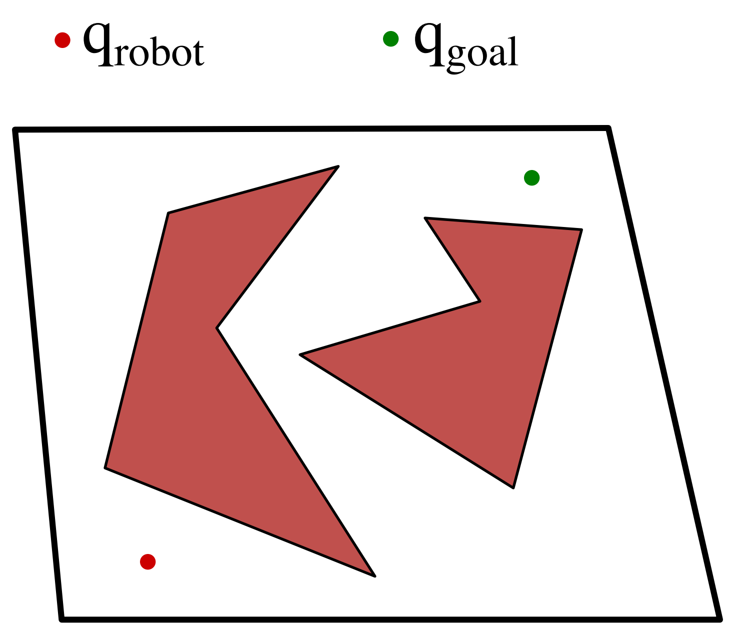
Motion Planning Methods

Input

- geometric descriptions of a robot and its environment (obstacles)
- initial and goal configurations

Output

- a path from start to finish (or the recognition that none exists)



Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces (1996)

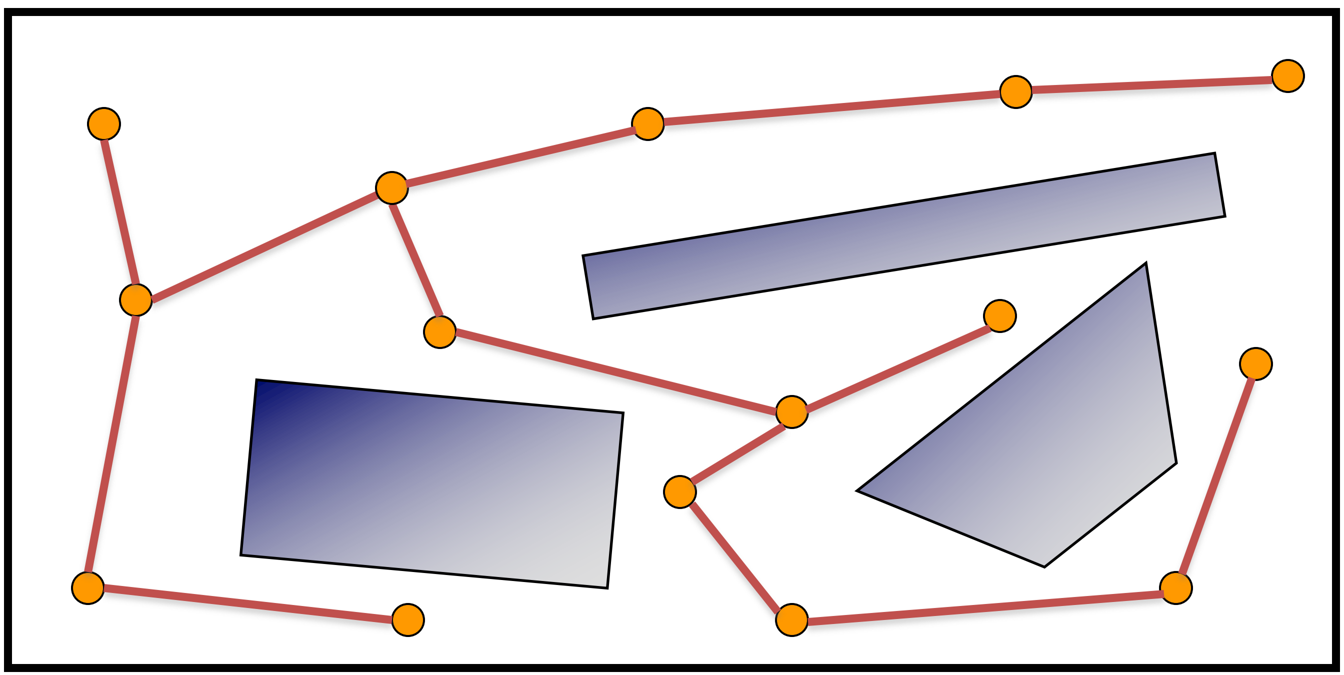
L. Kavraki, P. Švestka,
J.-C. Latombe, M. Overmars

Free-Space and C-Space Obstacle

- How do we know whether a configuration is in the free space?
  - Computing an explicit representation of the free-space is very hard in practice.
- Solution: Compute the position of the robot at that configuration in the workspace. Explicitly check for collisions with any obstacle at that position:
  - If colliding, the configuration is within C-space obstacle
  - Otherwise, it is in the free space
- Performing collision checks is relative simple

PRM -- Roadmap

Not the entire free configuration space, but rather a roadmap through it

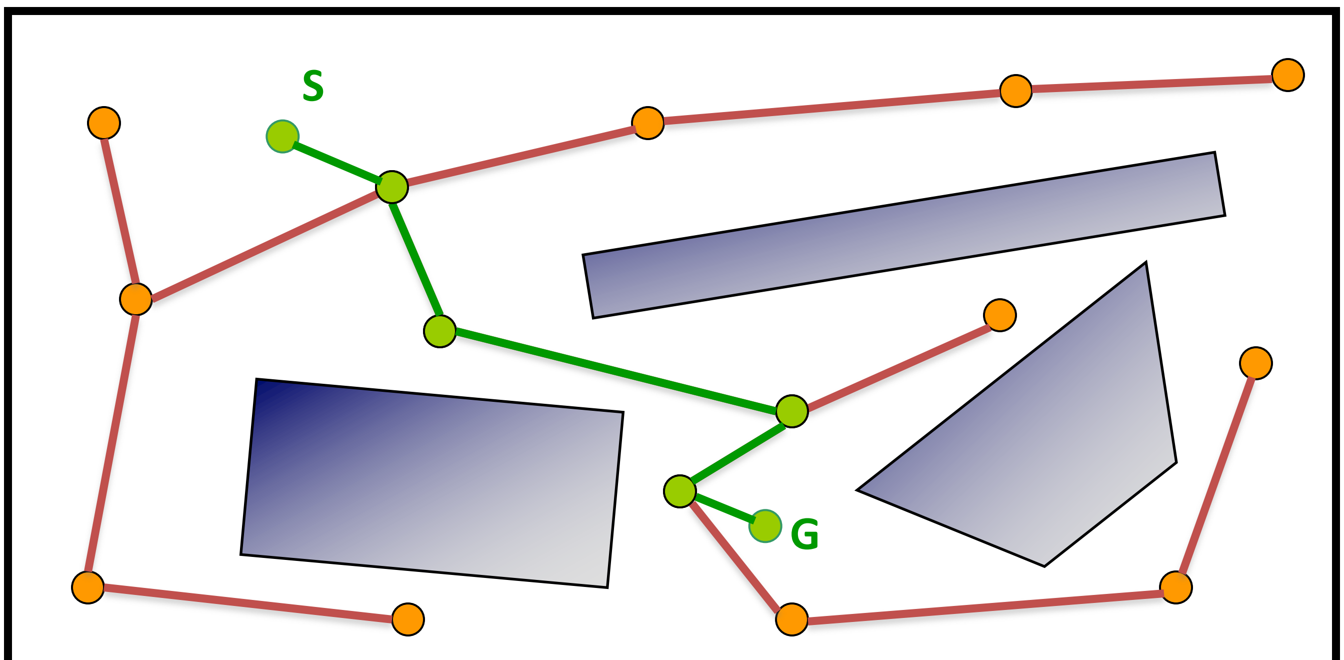


Probabilistic roadmaps

Build roadmap

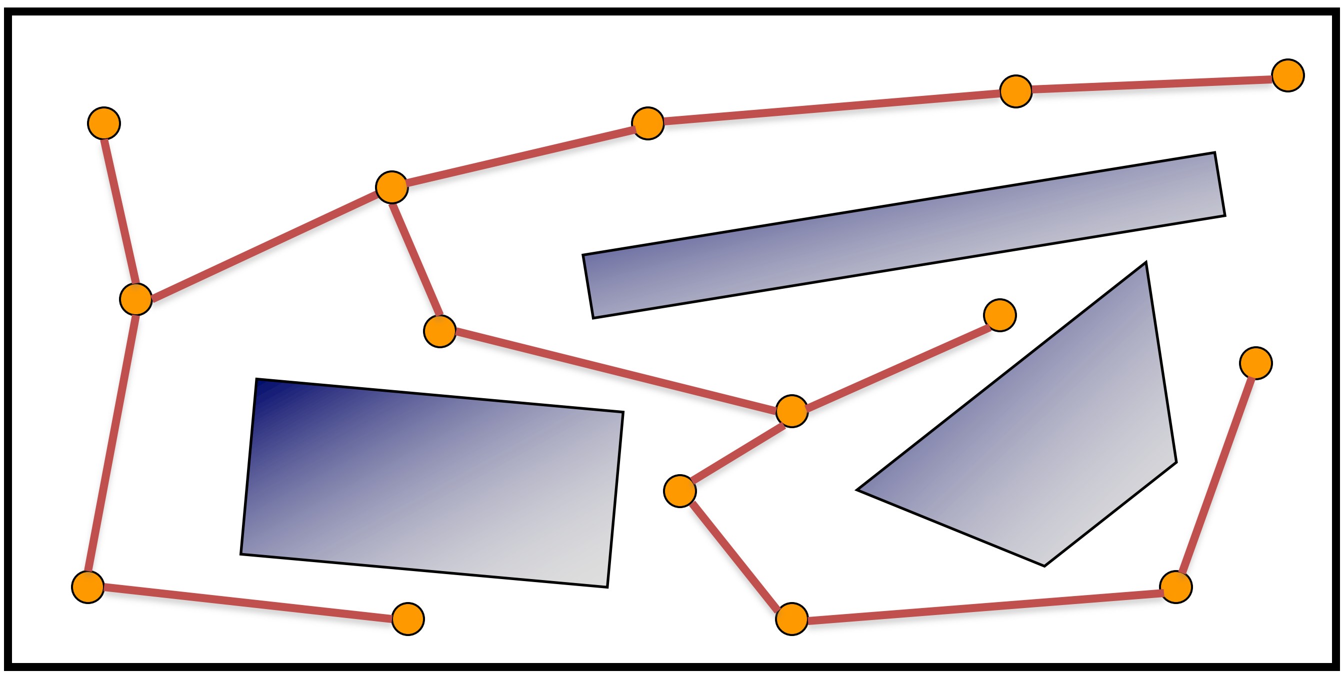
Find nearest notes from start and goal

Search a path in the roadmap



Roadmap

- Sample the C-space
- Connect nearby notes with edges
- Sample along the edges to test if it is collision-free



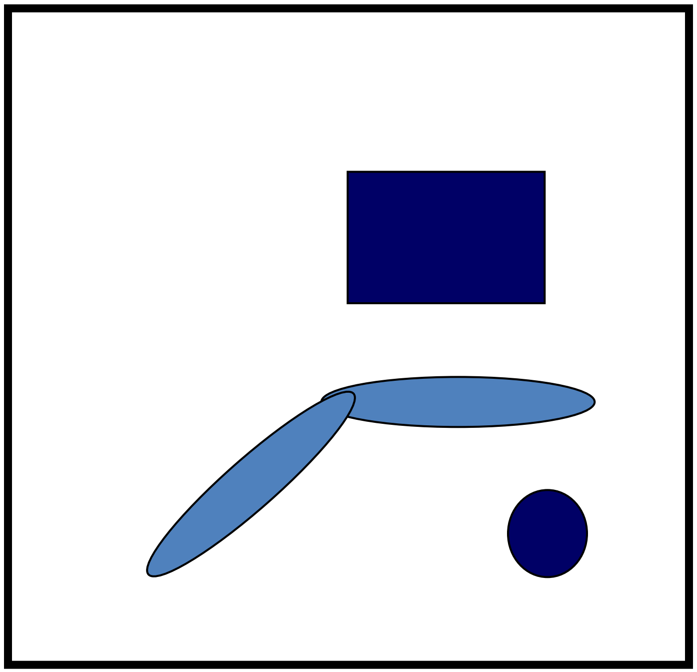
Two geometric primitives in configuration space

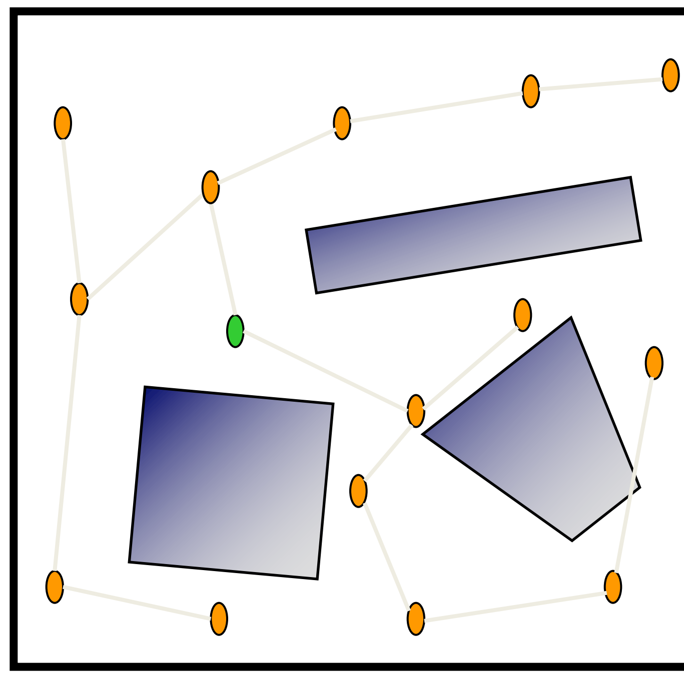
- **CLEAR**(q)

  Is configuration q collision free or not?

- **LINK**(q, q')

  Is the path between q and q' collision-free?





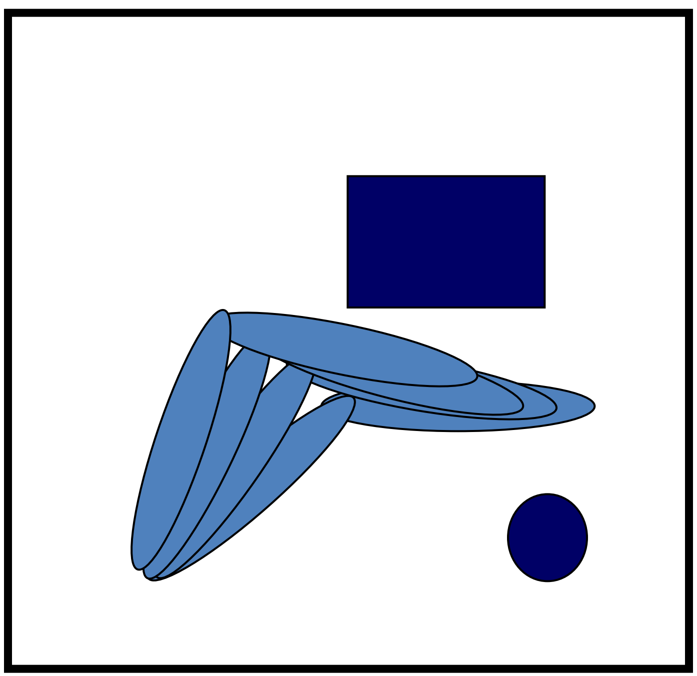
Two geometric primitives in configuration space

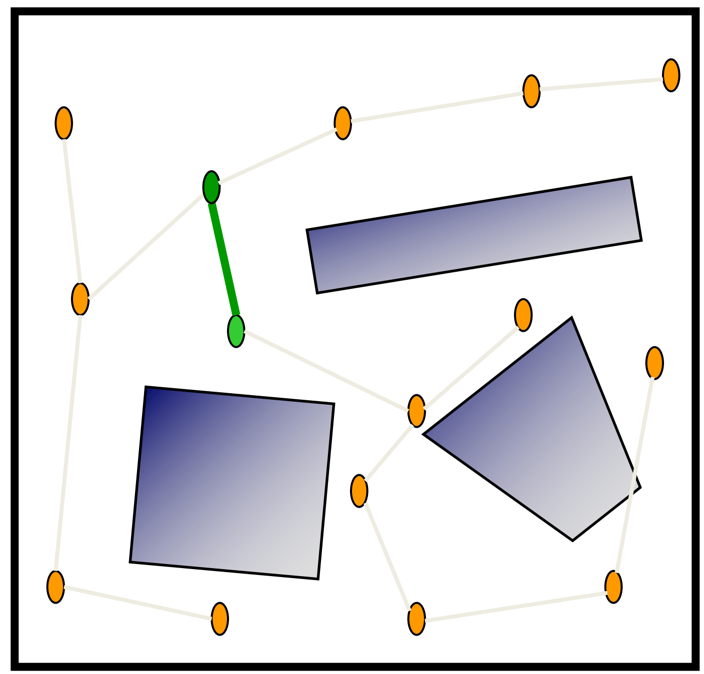
- **CLEAR**(q)

  Is configuration q collision free or not?

- **LINK**(q, q')

  Is the path between q and q' collision-free?





PRM algorithm overview

- Roadmap is an undirected acyclic graph ***R = (N, E)***
- Nodes ***N*** are robot configurations in free C-space, called **milestones**
- Edges ***E*** represent local paths between configurations

PRM algorithm overview

- Learning Phase
  - Construction step: randomly generate nodes and edges
  - Expansion step: improve graph connectivity in “difficult” regions
- Query Phase

PRM algorithm overview

- Learning Phase
  - Construction step: randomly generate nodes and edges
  - Expansion step: improve graph connectivity in “difficult” regions
- Query Phase

Learning: construction overview

1. ***R* = (*N*, *E*)** begins empty

2. A random free configuration ***c*** is generated and added to ***N***

3a. Candidate neighbors to ***c*** are partitioned from ***N***

3b. Edges are created between these neighbors and ***c***, such that acyclicity is preserved

4. Repeat 2-3 until "done"

Learning: construction overview

1. $R = (N, E)$ begins empty

2. A random free configuration c is generated and added to N

3a. Candidate neighbors to c are partitioned from N

3b. Edges are created between these neighbors and c, such that acyclicity is preserved

4. Repeat 2-3 until “done”

Generating random configurations

2. A random free configuration ***c*** is generated and added to ***N***

- Random sampling over uniform probability distribution of values for each DOF

- New configuration is checked for collision (intersect obstacles, intersect self)

Learning: construction overview

1. ***R* = (*N*, *E*)** begins empty

2. A random free configuration ***c*** is generated and added to ***N***

3a. Candidate neighbors to ***c*** are partitioned from ***N***

3b. Edges are created between these neighbors and ***c***, such that acyclicity is preserved

4. Repeat 2-3 until “done”

Neighbor set

3a. Candidate neighbors to *c* are partitioned from *N*

- Only want to consider ***k*** neighbors some maximum distance ***maxdist*** away from ***c***, according to a distance function ***D***:

$$N_c = \{ c' \in N \mid D(c, c') \leq maxdist \}$$

$$D(c, n) = \max_{x \in robot} \| x(n) - x(c) \|$$

max Euclidean distance between a point on the robot at the two configurations

Learning: construction overview

1. ***R* = (*N*, *E*)** begins empty

2. A random free configuration ***c*** is generated and added to ***N***

3a. Candidate neighbors to ***c*** are partitioned from ***N***

3b. Edges are created between these neighbors and ***c***, such that acyclicity is preserved

4. Repeat 2-3 until "done"

Creating edges

3b. Edges are created between these neighbors and ***c***, such that acyclicity is preserved

- neighbors are considered in order of increasing distance from c
- nodes that are already in the same connected component of c at the time are ignored
- local planner is used to determine whether a local path exists

General local planner

1. Connect the two configurations in C-space with a straight line segment

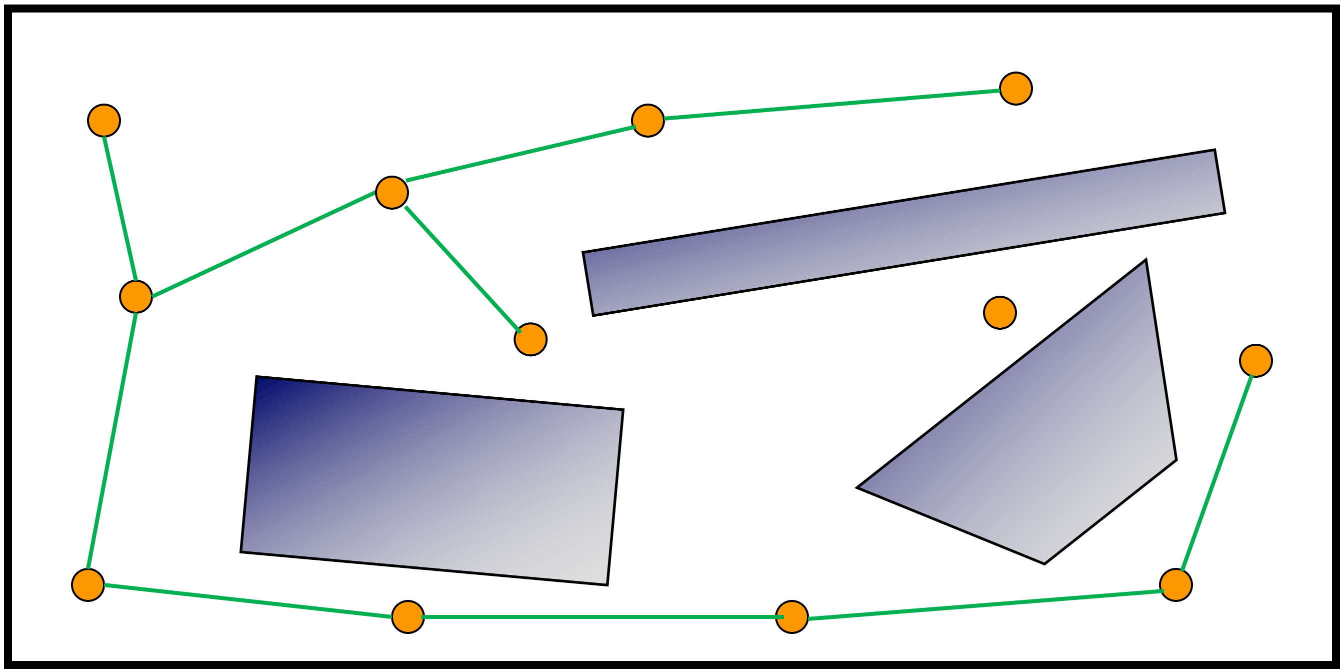
2. Check the joint limits

3. Discretize the line segment into a sequence of configurations $\boldsymbol{c_1}, \ldots, \boldsymbol{c_m}$ such that for every ($\boldsymbol{c_i}$, $\boldsymbol{c_{i+1}}$), no point on the robot at $\boldsymbol{c_i}$ lies further than λ away from its position at $\boldsymbol{c_{i+1}}$

4. For each $\boldsymbol{c_i}$, grow robot by λ check for collisions

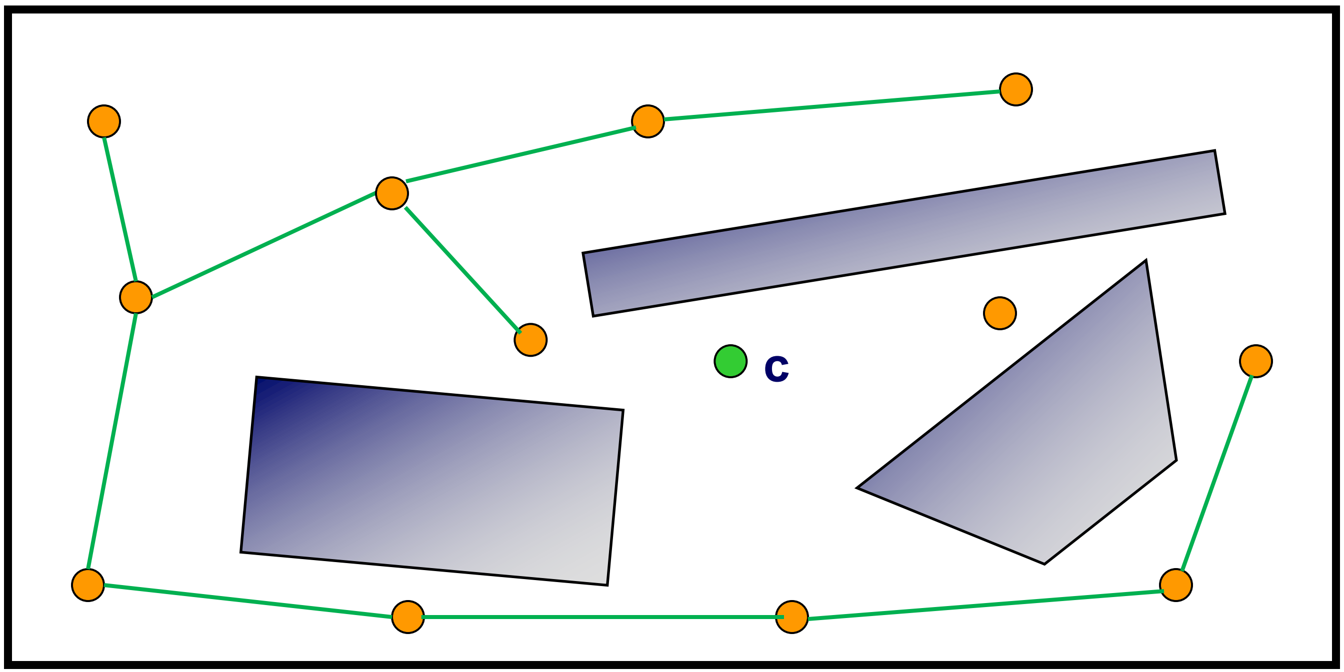
Construction step example

Graph after several iterations...



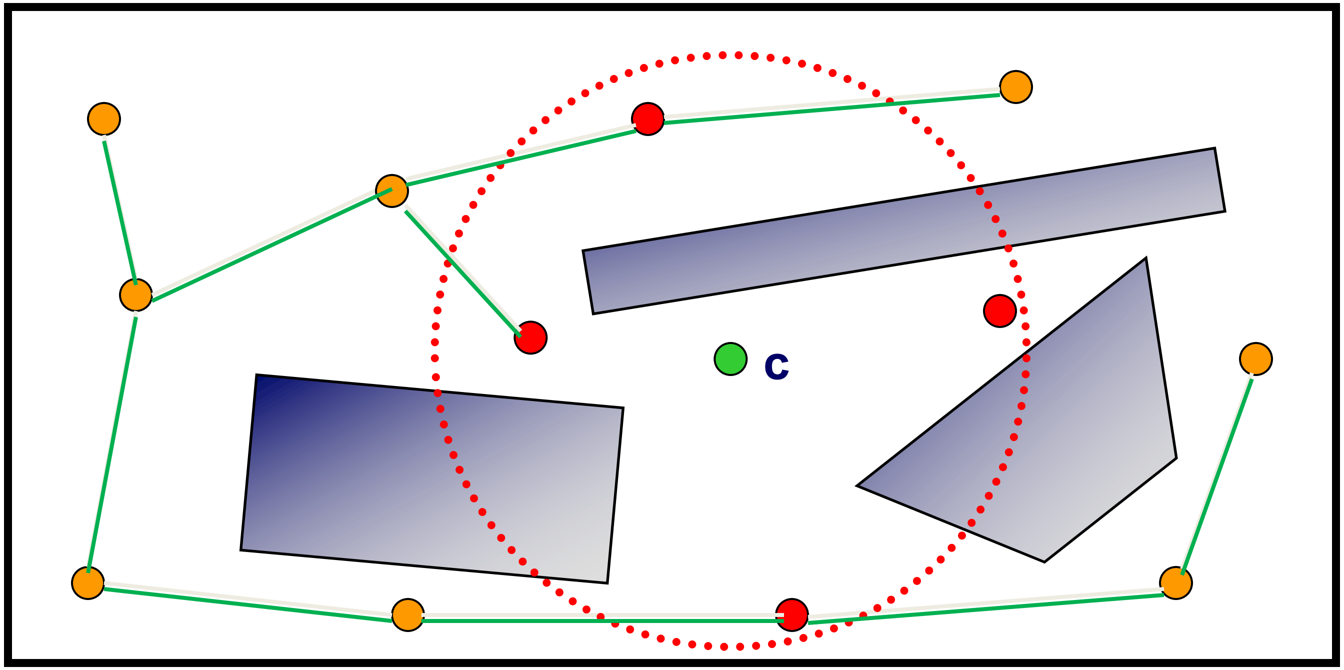
Construction step example

2. A random free configuration ***c*** is generated and added to ***N***



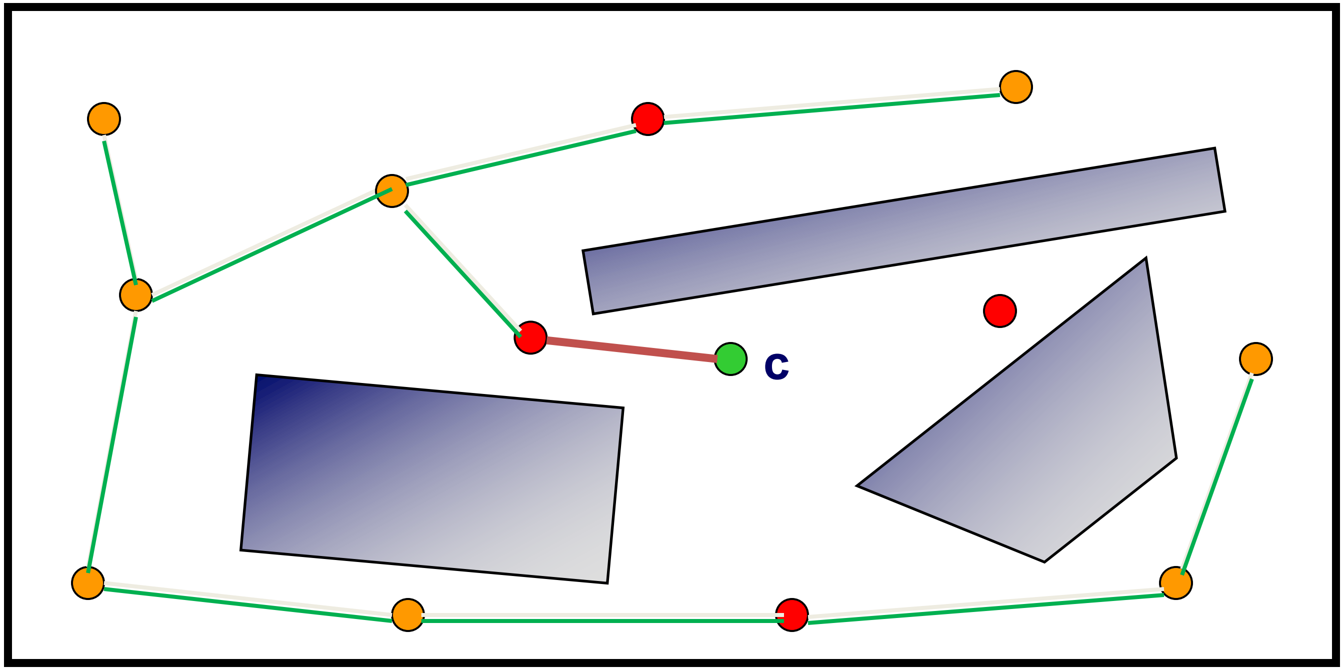
Construction step example

3a. Candidate neighbors to ***c*** are partitioned from ***N***



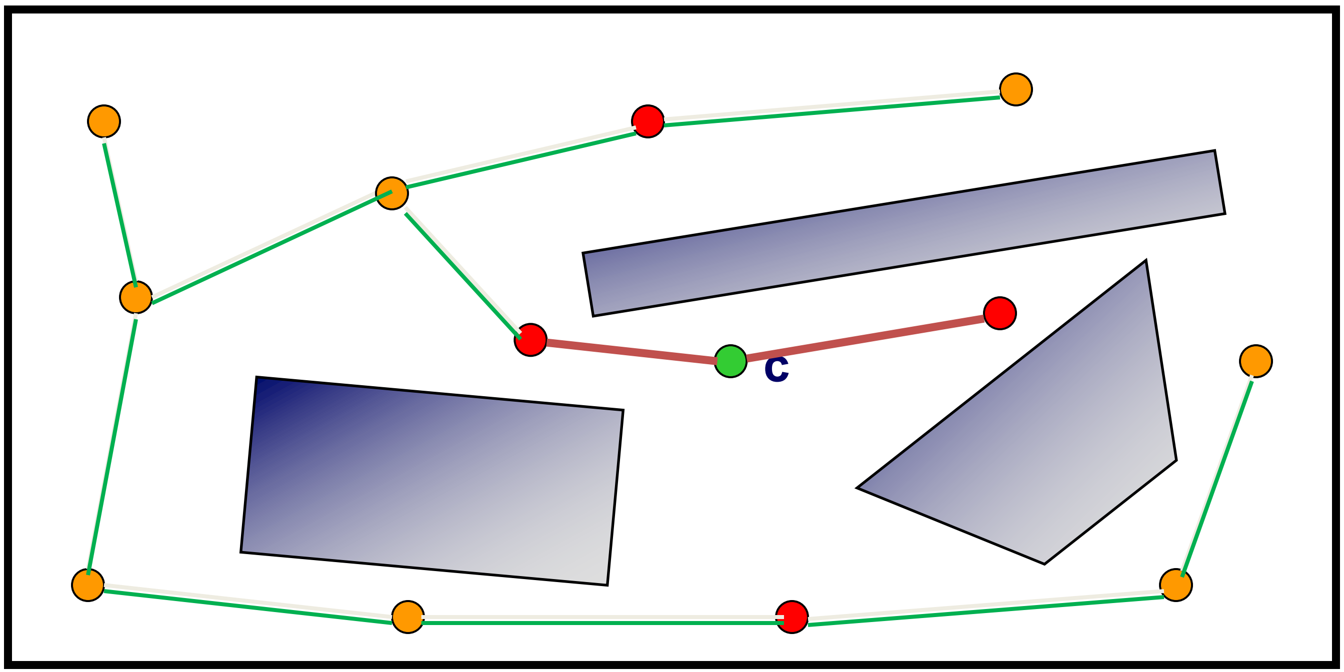
Construction step example

3b. Edges are created between these neighbors and ***c***, such that acyclicity is preserved



Construction step example

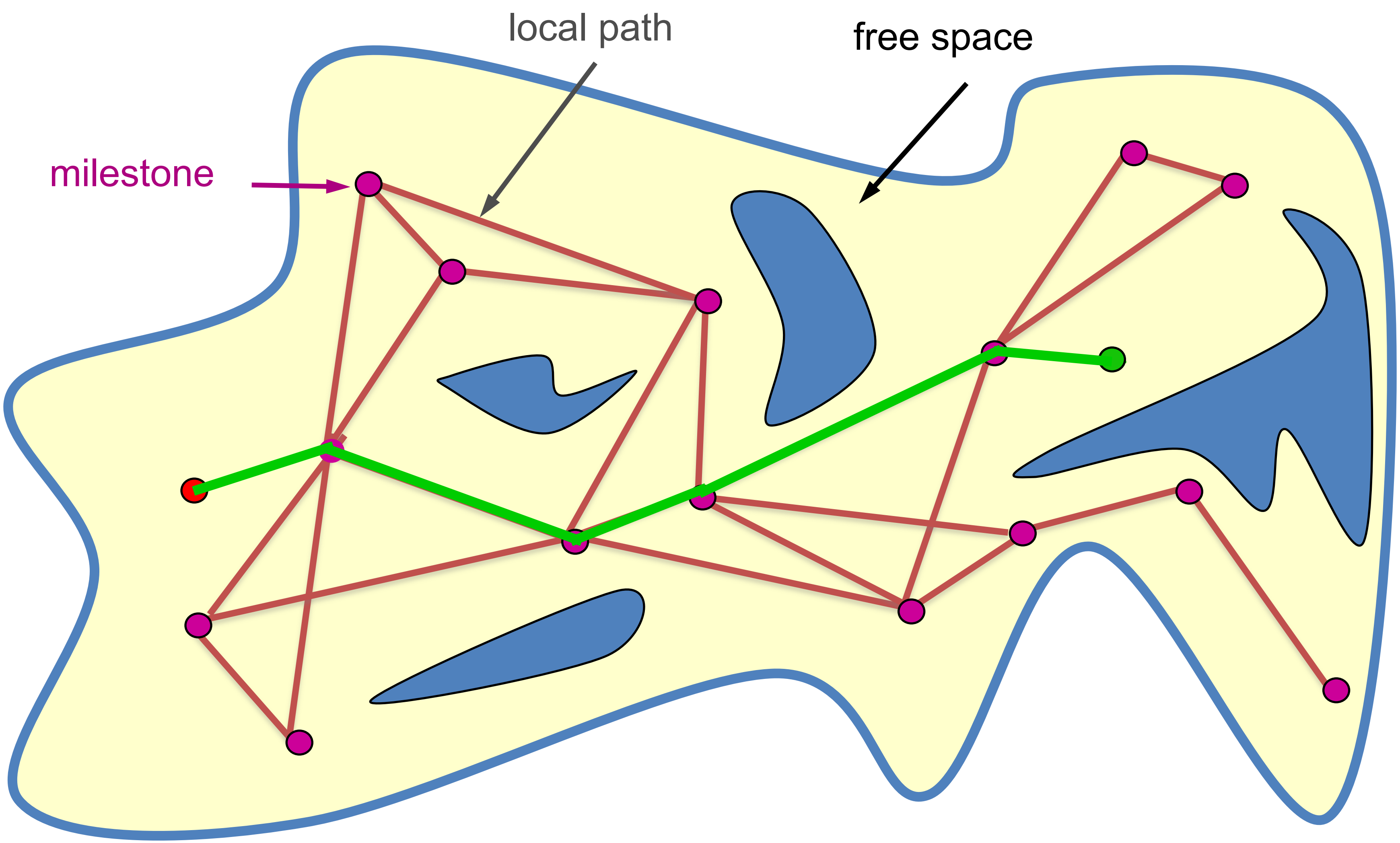
3b. Edges are created between these neighbors and ***c***, such that acyclicity is preserved



Another view

- Acyclicity not necessary
- From original paper
  - May make weird paths
- Allow connectivity up to a certain valence

Probabilistic Roadmap (PRM):



Simpler Outline

Input: geometry of the moving object & obstacles

Output: roadmap G = (V, E)

1: $V \leftarrow \emptyset$ and $E \leftarrow \emptyset$.

2: **repeat**

3: q ← sampled at random from C.

4: **if CLEAR**(q) **then**

5: Add q to V.

6: $N_q \leftarrow$ a set of nodes in V that are close to q.

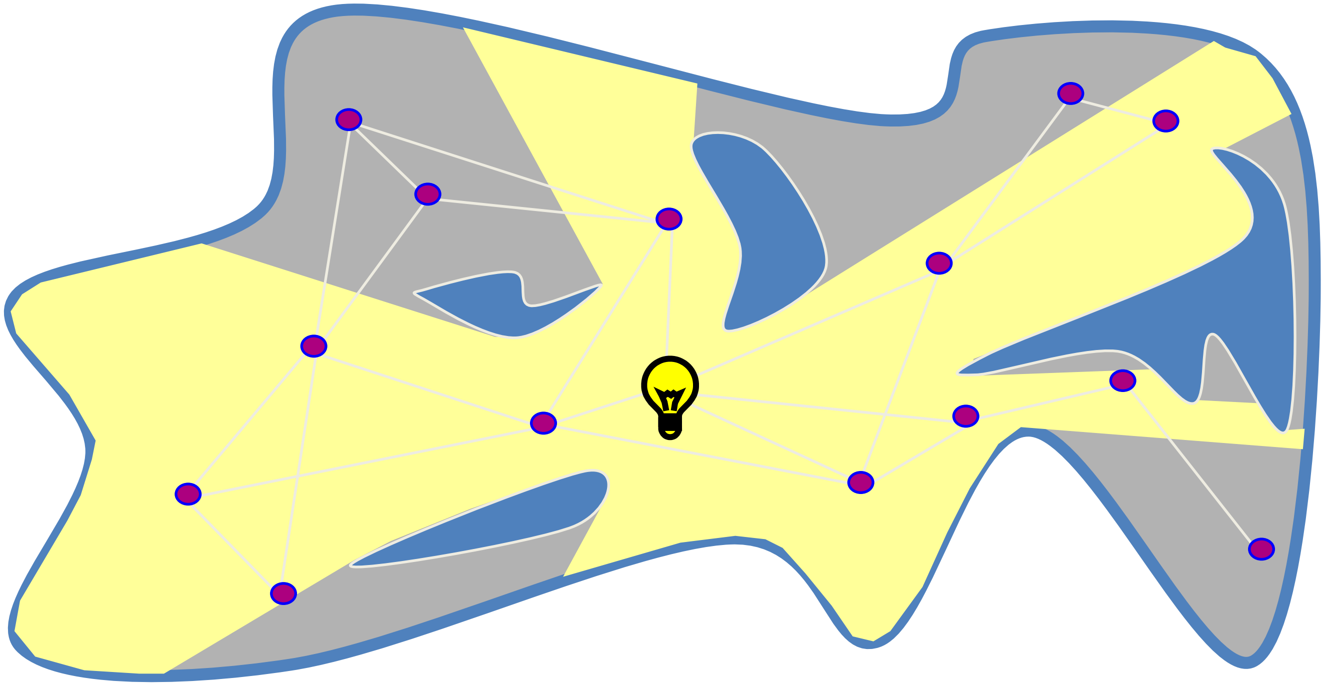
6: **for** each $q' \in N_q$, in order of increasing d(q,q')

7: **if LINK**(q',q) **then**

8: Add an edge between q and q' to E.

Why does it work? Intuition

- A small number of milestones **almost** "cover" the **entire** configuration space.



PRM algorithm overview

- Learning Phase
  - Construction step: randomly generate nodes and edges
  - Expansion step: improve graph connectivity in “difficult” regions
- Query Phase

Query phase

- Given start configuration ***s***, goal configuration ***g***, calculate paths ***Ps*** and ***Pg*** such that ***Ps*** and ***Pg*** connect ***s*** and ***g*** to nodes ***s'*** and ***g'*** that are themselves connected in the graph
- Return the path consisting of ***Ps*** concatenated with the path from ***s'*** to ***g'*** concatenated with the reverse of ***Pg***

Calculating P_s , P_g

- Consider nodes in graph in order of increasing distance from P_s, up to a ***maxdist*** away from ***s***
- Use local planner to find connection

Experimental results (briefly)

- Tested with up to 7-dof robot, both free- and fixed-base
- Given enough learning time, able to achieve 100% success, but not unreasonable results even with shorter learning periods
- Queries are fast (not more than a couple of seconds)

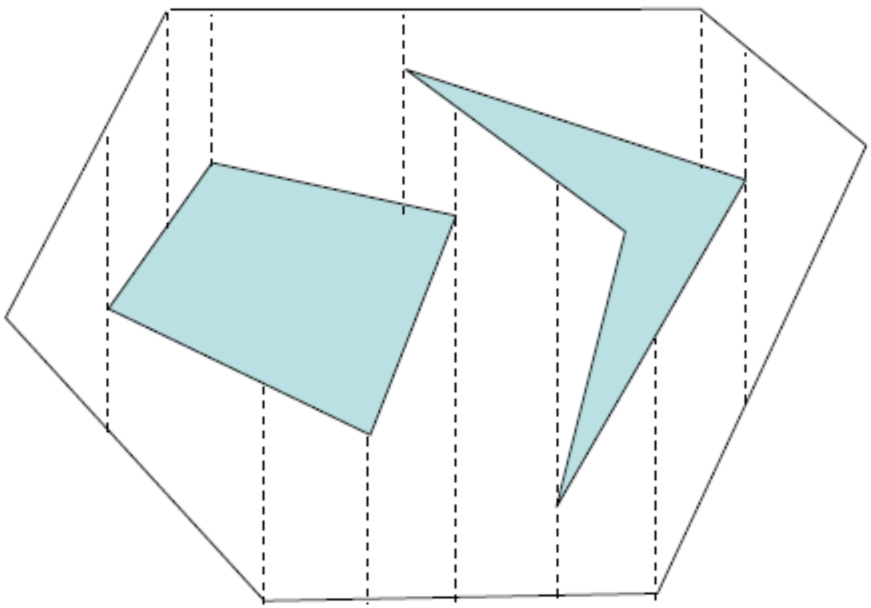
Conclusions

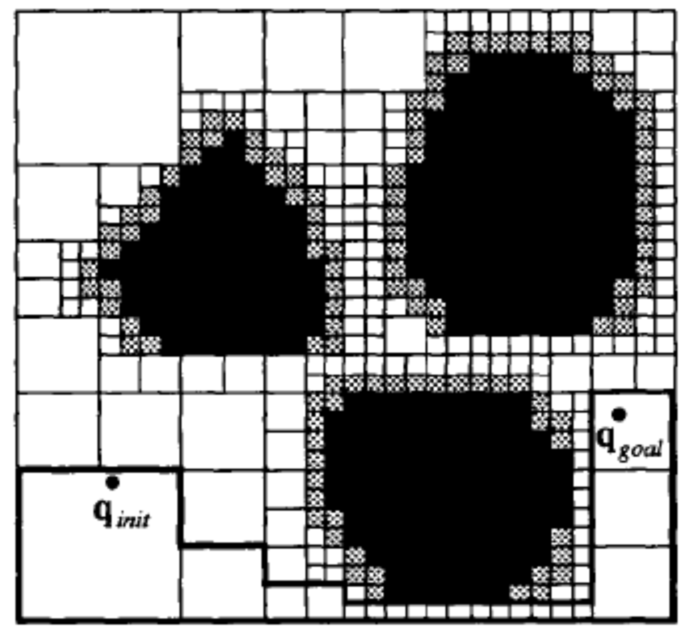
Pros

- Once learning is done, queries can be executed quickly
- Complexity reduction over full C-space representation
- Adaptive: can incrementally build on roadmap
- Probabilistically complete, which is usually good enough

Cell Decompositions

- Path Planning in two steps:
  - Planner determines cells that contain the start and goal
  - Planner searches for a path within adjacency graph





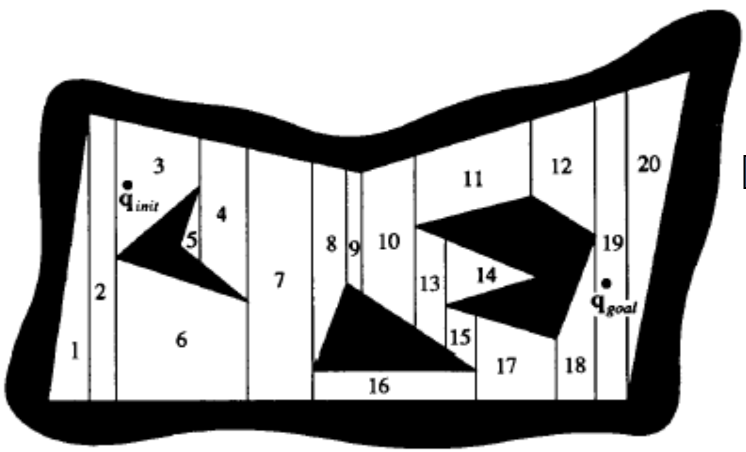
q_{init}
q_{goal}

Exact Cell Decomposition

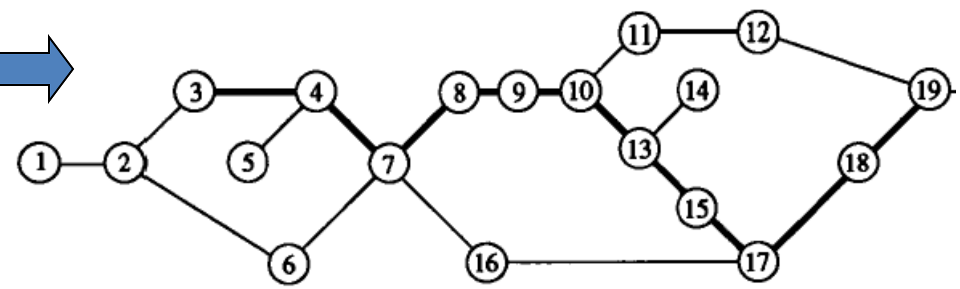
Trapezoidal Decomposition:

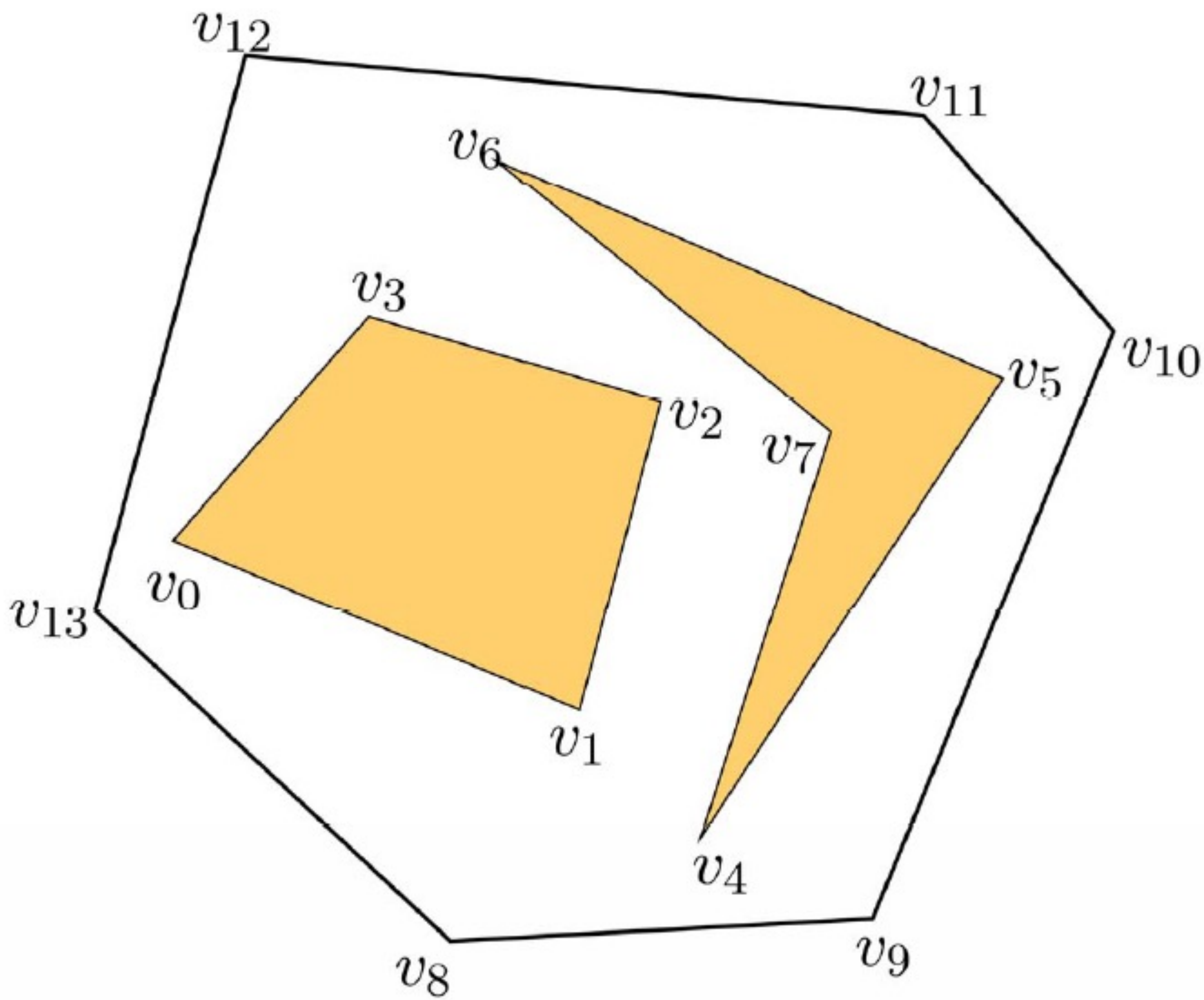
Decomposition of the free space into trapezoidal & triangular cells

Connectivity graph representing the adjacency relation between the cells

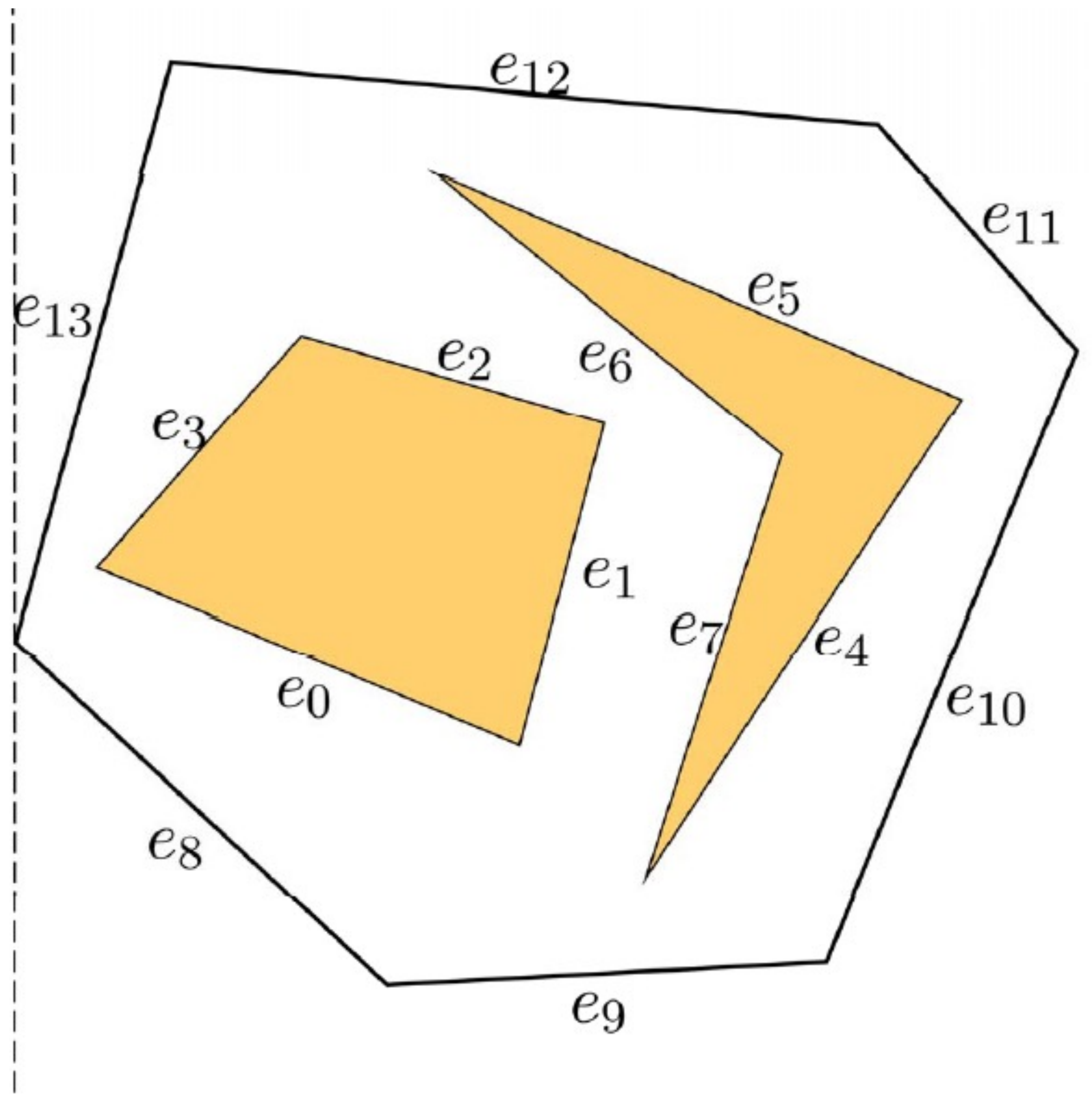


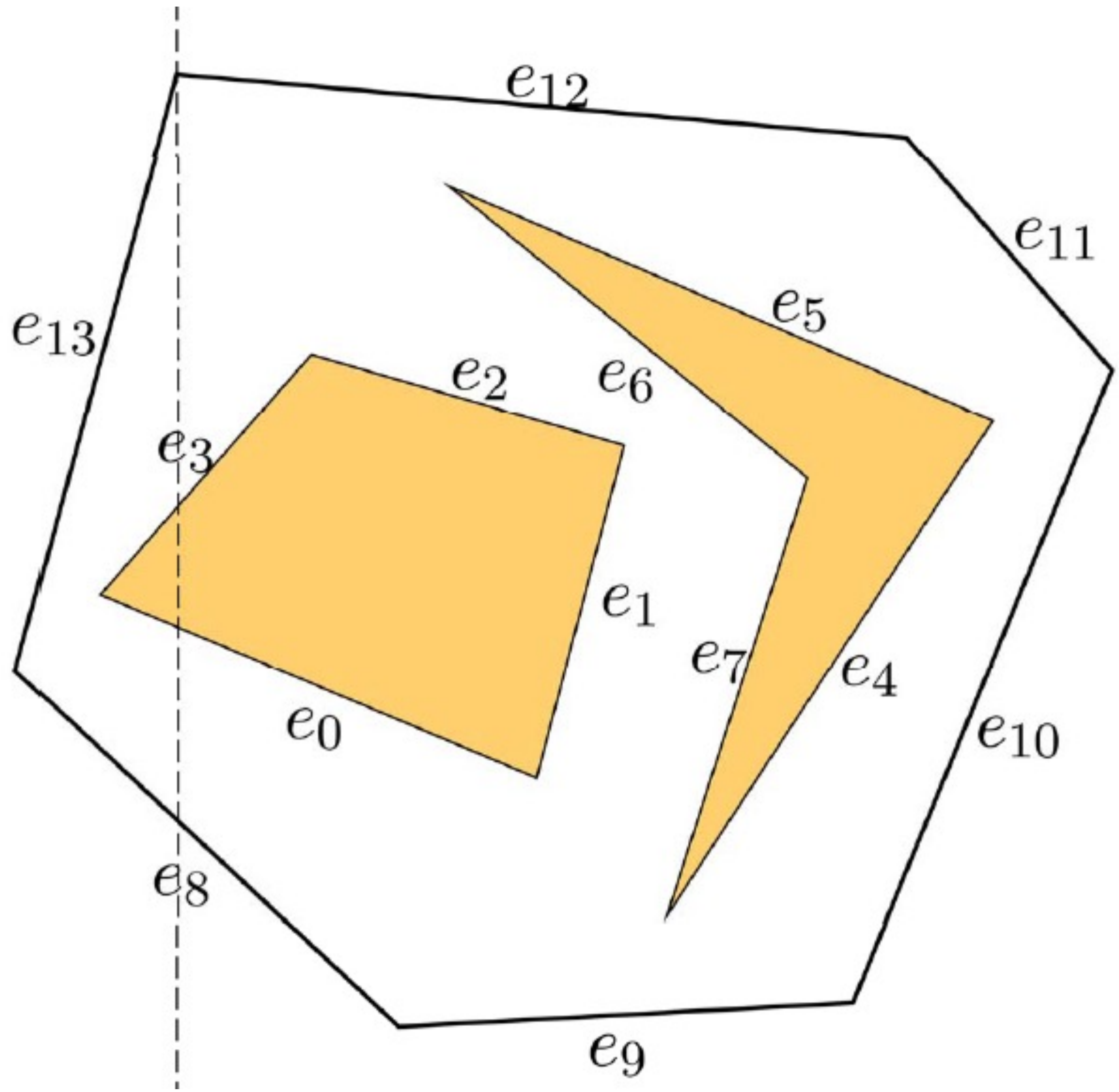
(Sweepline algorithm)

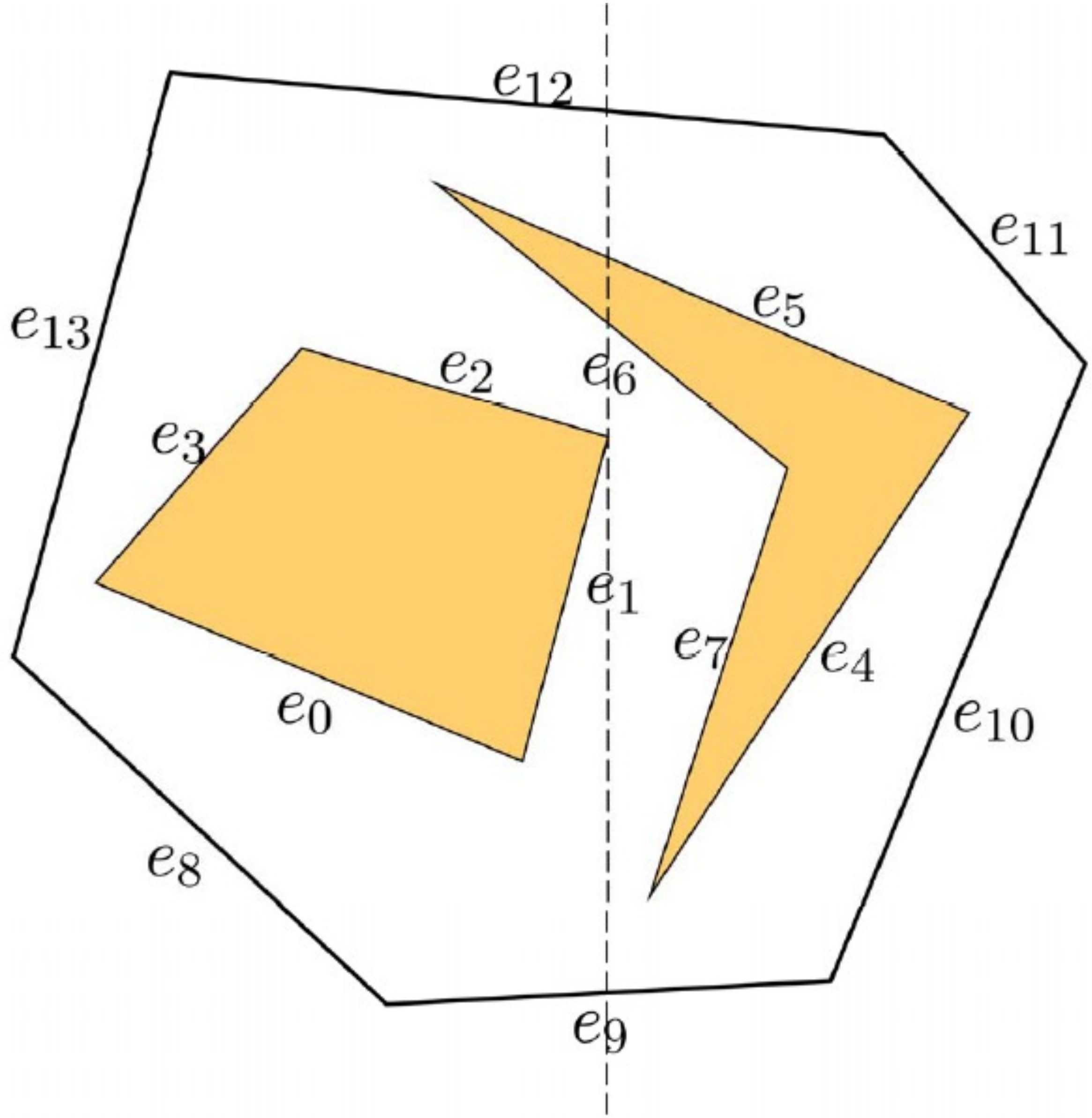




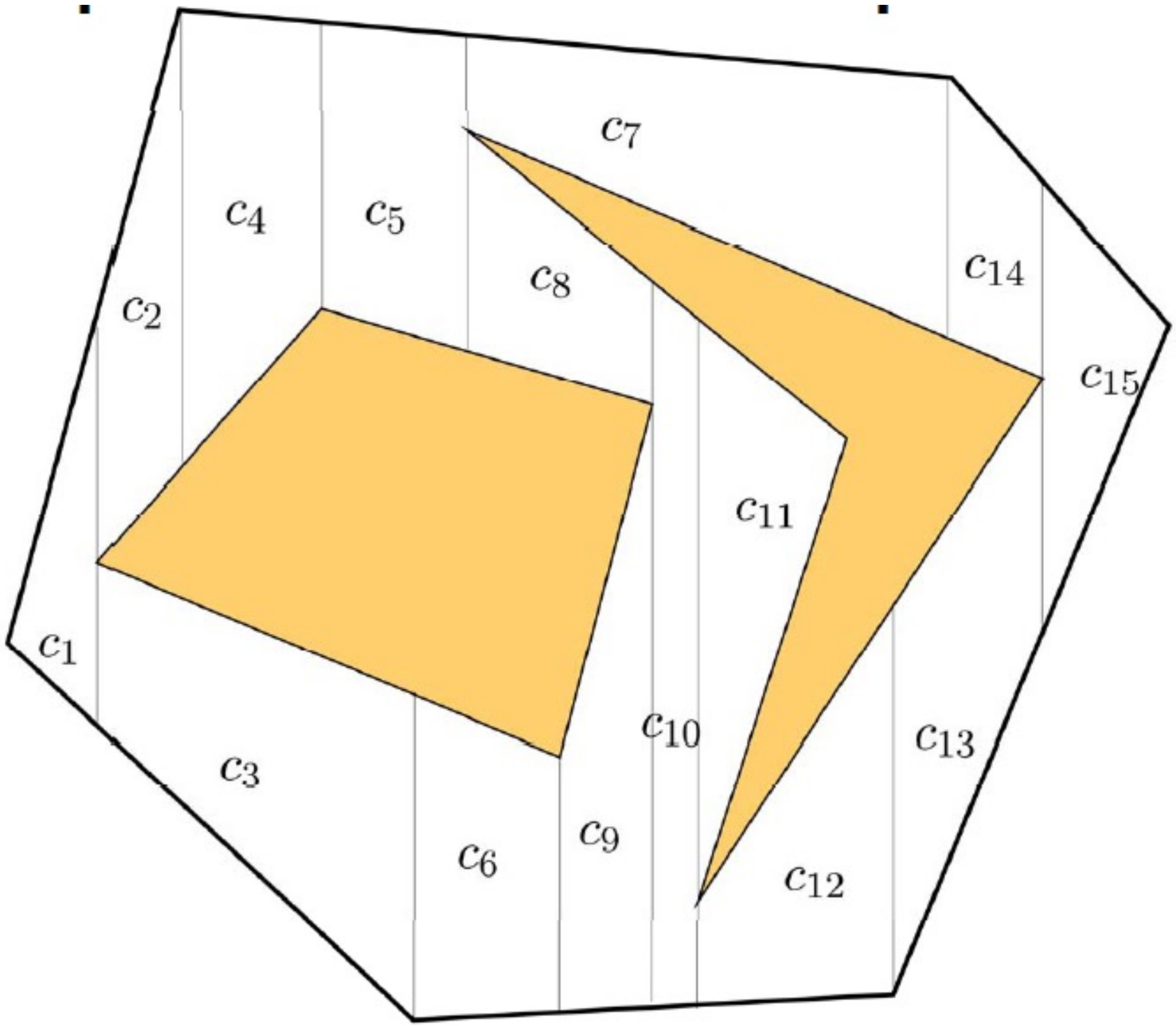
v_{12}
v_{11}
v_6
v_3
v_{10}
v_5
v_2
v_7
v_0
v_{13}
v_1
v_4
v_9
v_8


e_{12}
e_{11}
e_{13}
e_5
e_2
e_6
e_3
e_1
e_7
e_4
e_0
e_{10}
e_8
e_9

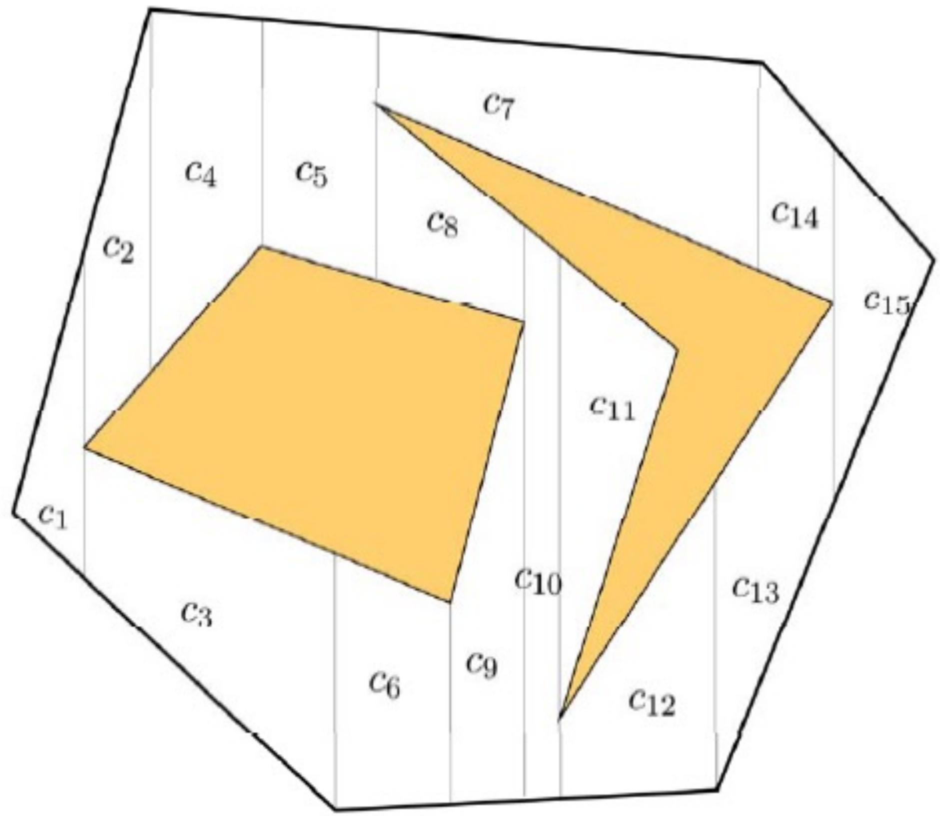

e_{12}
e_{11}
e_{13}
e_5
e_2
e_6
e_3
e_1
e_7
e_4
e_0
e_{10}
e_8
e_9

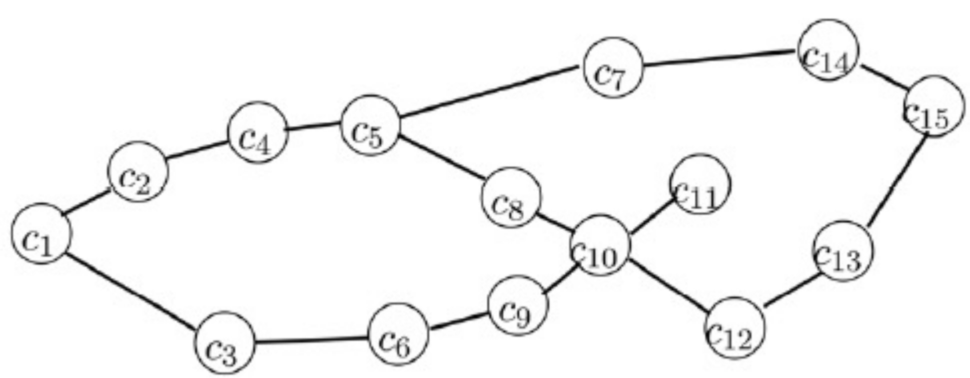


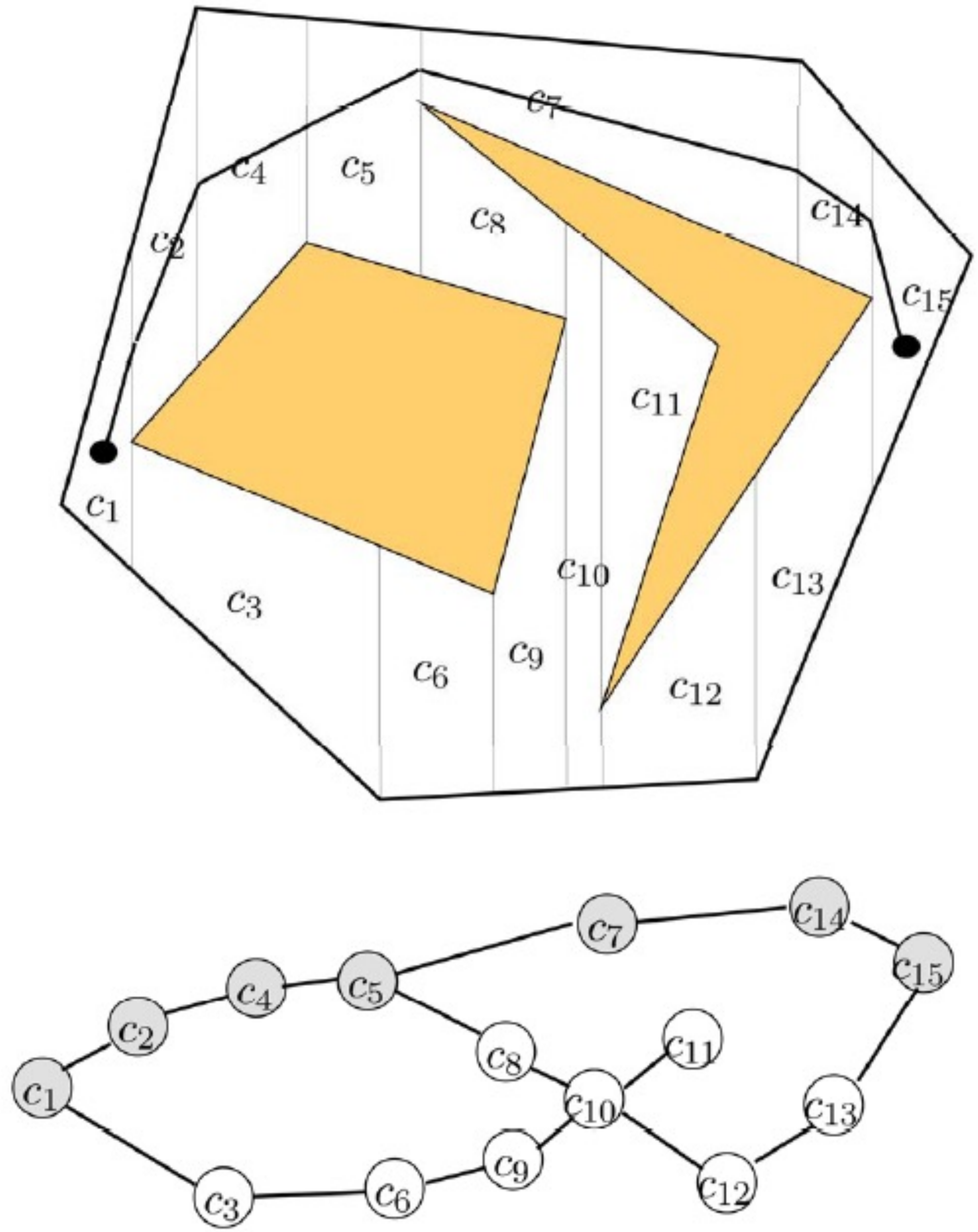
e_{12}
e_{11}
e_5
e_{13}
e_2
e_6
e_3
e_1
e_7
e_4
e_0
e_{10}
e_8
e_9



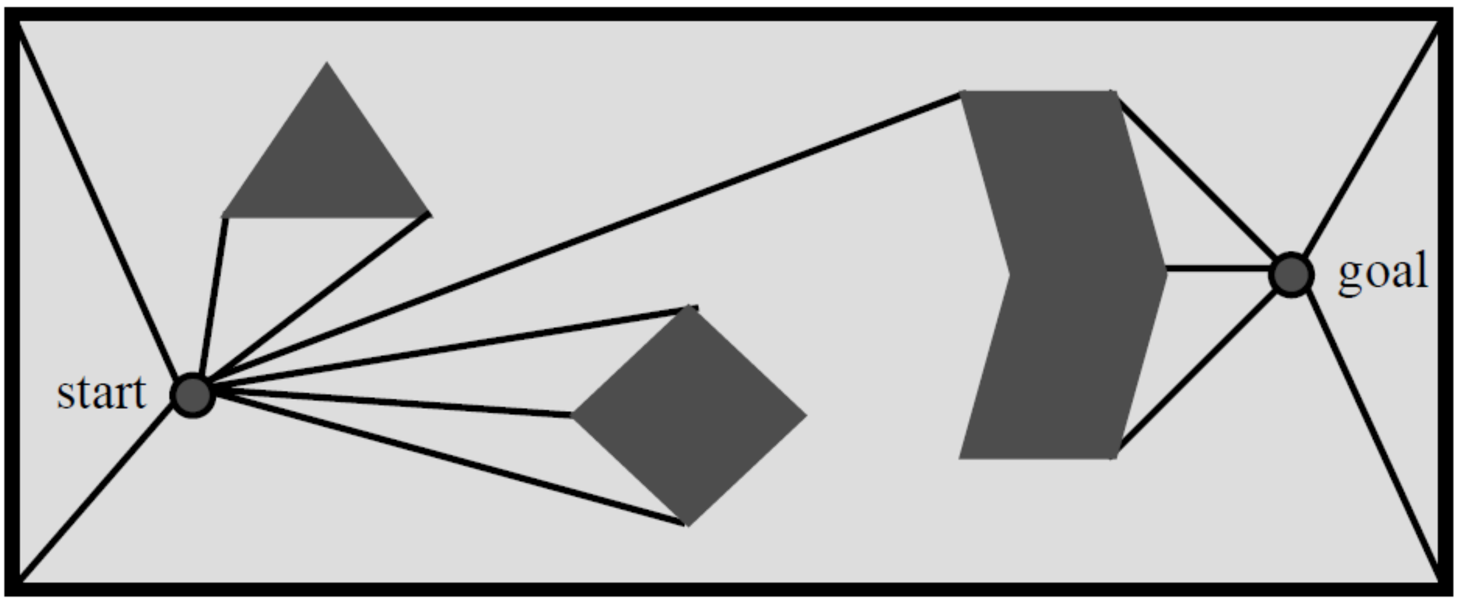
c_1
c_2
c_3
c_4
c_5
c_6
c_7
c_8
c_9
c_{10}
c_{11}
c_{12}
c_{13}
c_{14}
c_{15}

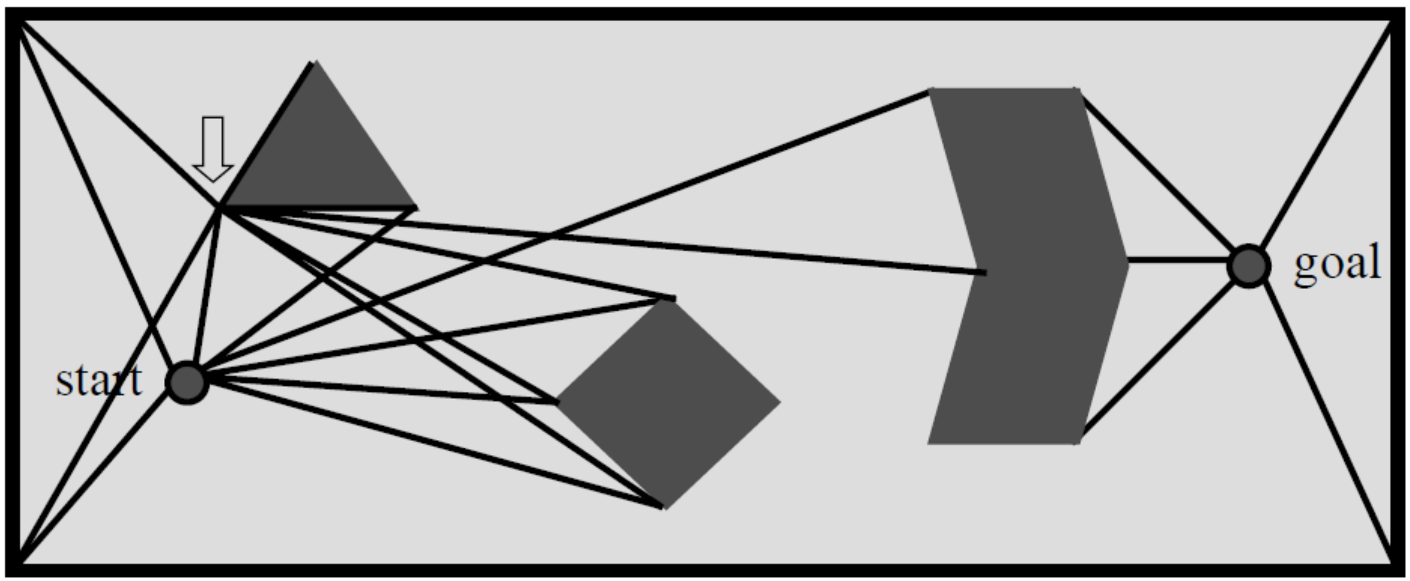

c_1
c_2
c_3
c_4
c_5
c_6
c_7
c_8
c_9
c_{10}
c_{11}
c_{12}
c_{13}
c_{14}
c_{15}

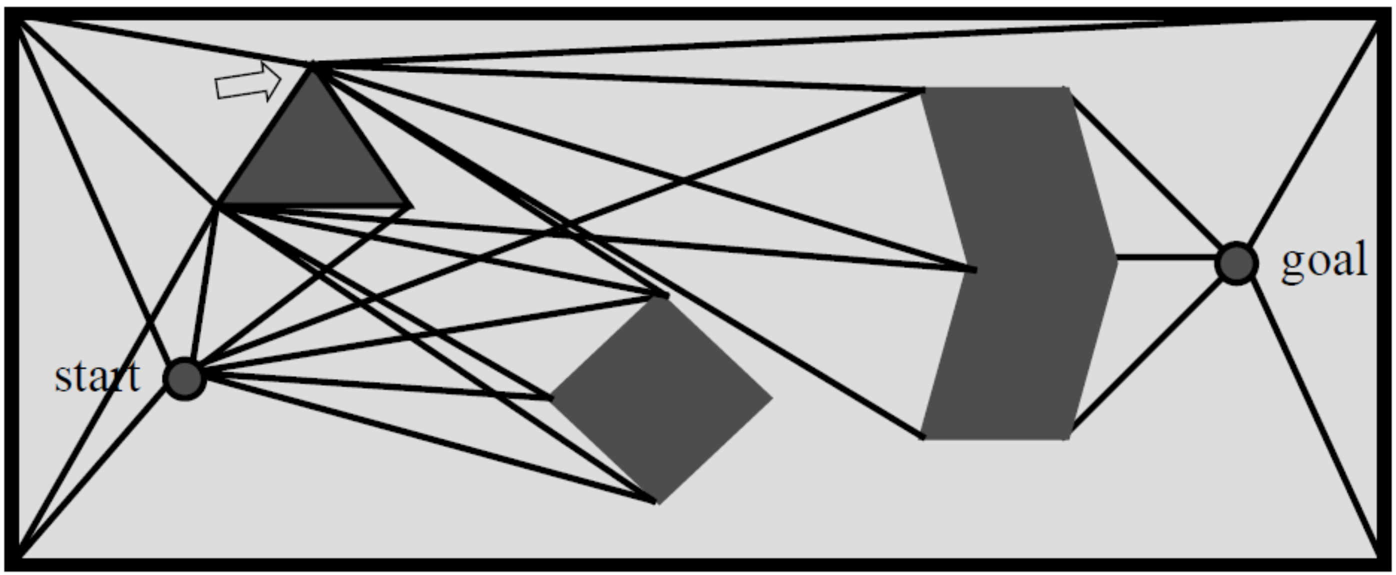

c_1
c_2
c_3
c_4
c_5
c_6
c_7
c_8
c_9
c_{10}
c_{11}
c_{12}
c_{13}
c_{14}
c_{15}

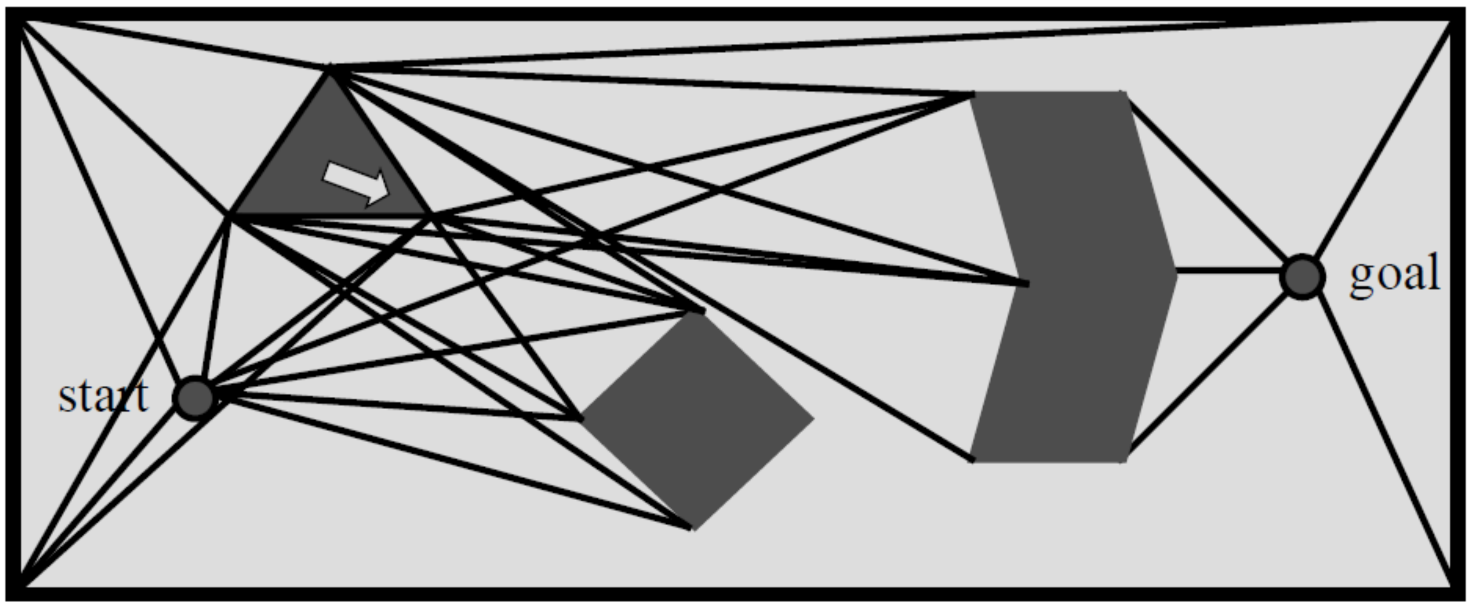

c1
c2
c3
c4
c5
c6
c7
c8
c9
c10
c11
c12
c13
c14
c15

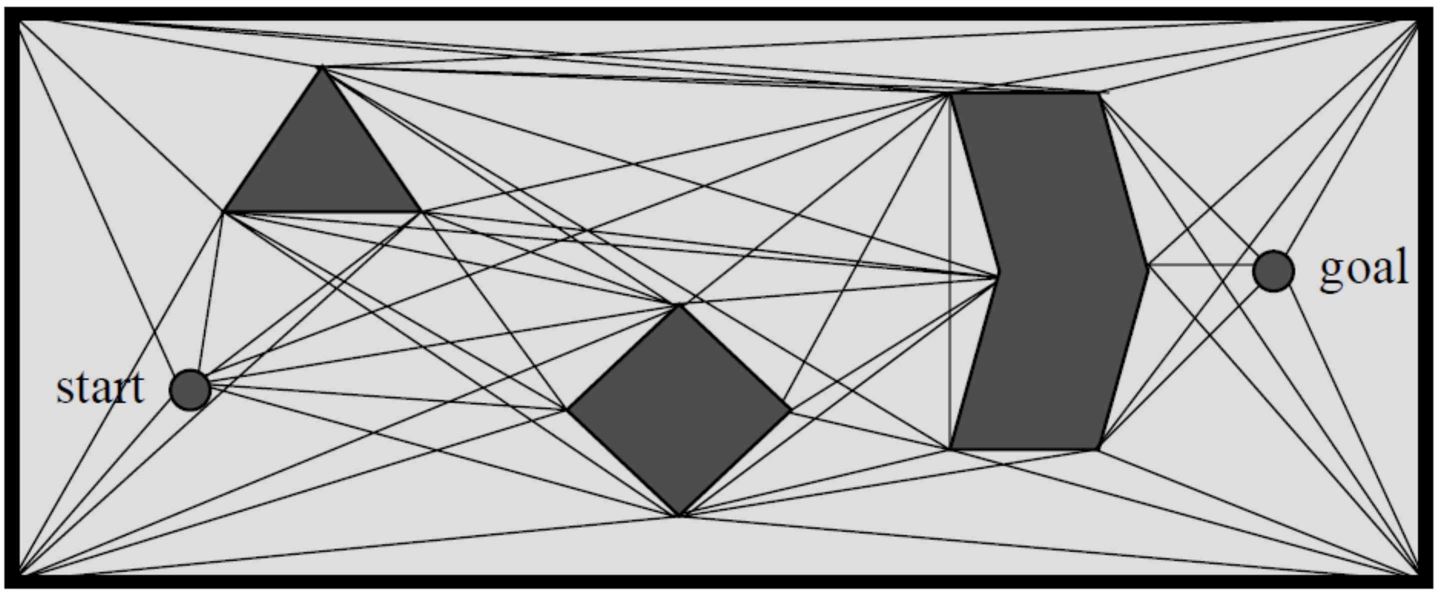
Visibility Graph Methods




start
goal


start
goal


start
goal


start
goal

End