Robot Autonomy

Lecture 6: Motion Planning I

Oliver Kroemer

Piano Mover Problem

- Given a workspace W with obstacles $O \subset W$ and a c-space C with mapping function $A(q) \subset W$ where $q \in C$ then:

$$C_{obs} = \{q \in C | A(q) \cap O \neq \emptyset\}$$

$$C_{free} = C \setminus C_{obs}$$

- Want the robot to find a continuous path τ in C_{free} from initial configuration q_i to goal configuration q_g

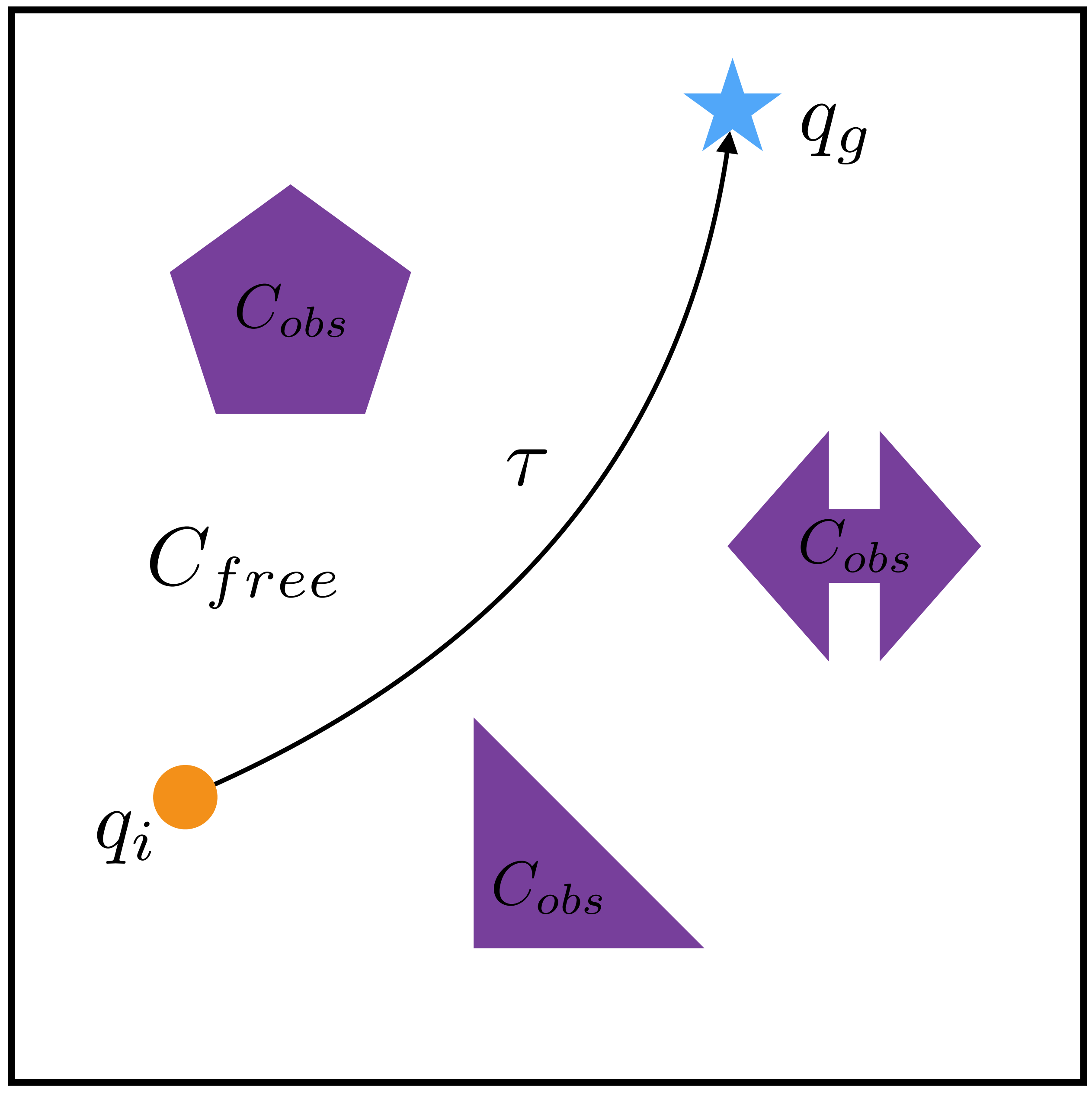
$$\tau : [0, 1] \rightarrow C_{free}$$

$$\tau(0) = q_i$$

$$\tau(1) = q_g$$

- Assume that robot can move in any direction in c-space

Piano Mover Problem

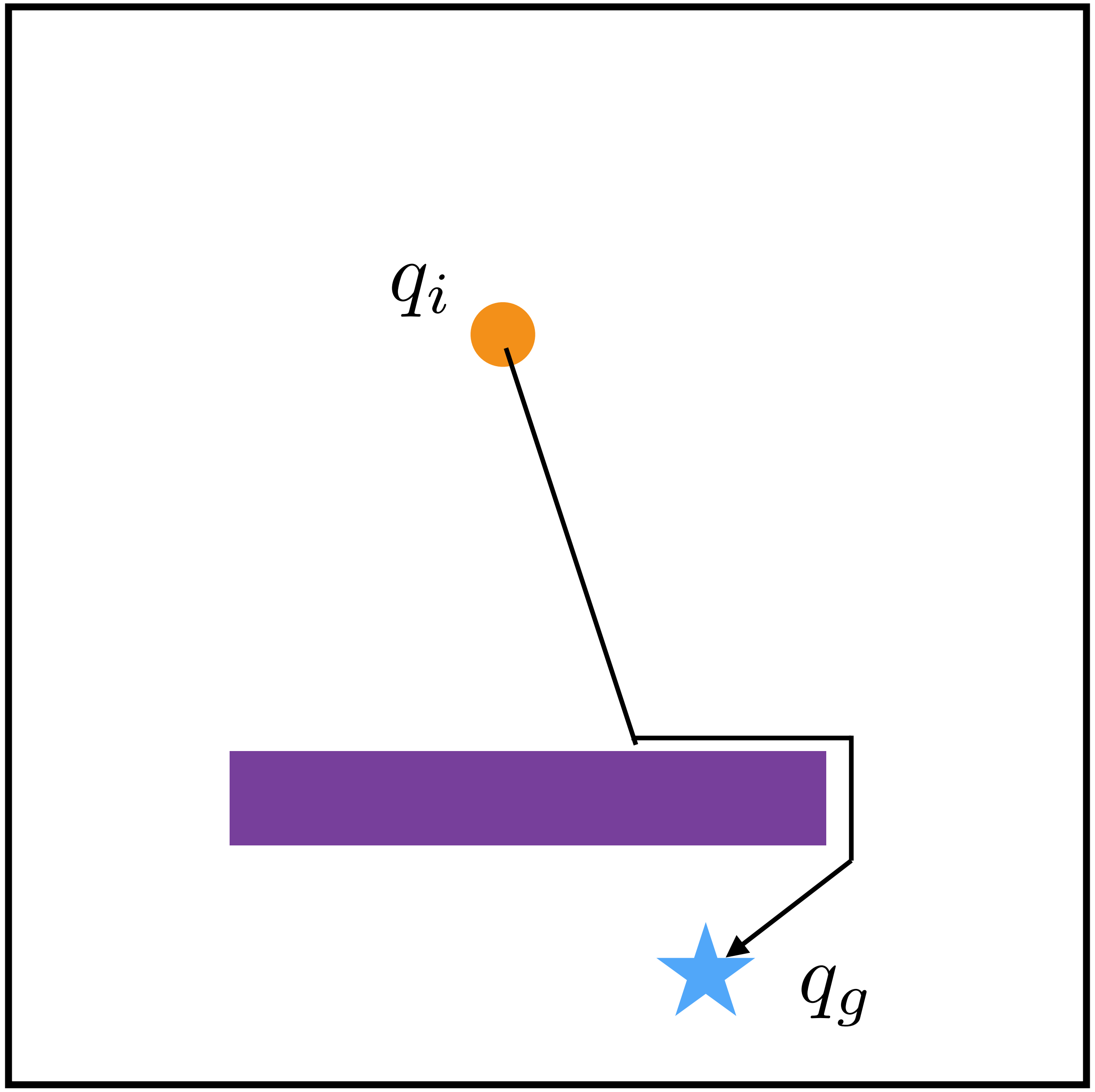


Reactive Approach

- Reactive approaches: do not plan ahead, just react
- Robot can create the path as it goes along
  - Move towards the goal
  - Avoid collisions along the way

- Simple and relies mainly on local sensing and knowledge
  - Bug algorithms
  - Potential fields

Bug Algorithms

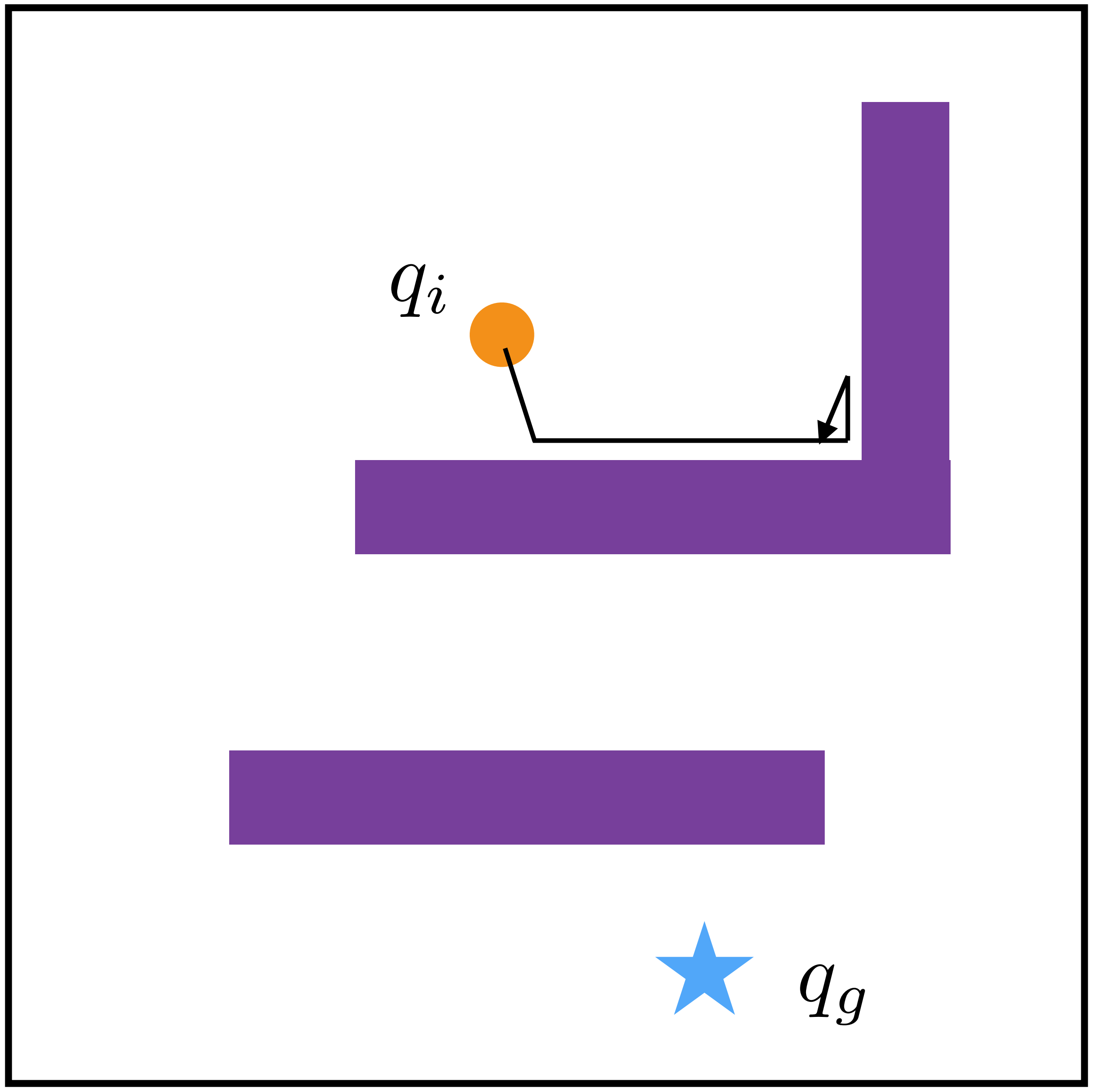


2D c-space

Behaviour:
- Head towards goal
- Follow wall until you can move towards goal again

Bug Algorithms

Behaviour:
- Head towards goal
- Follow wall until you can move towards goal again

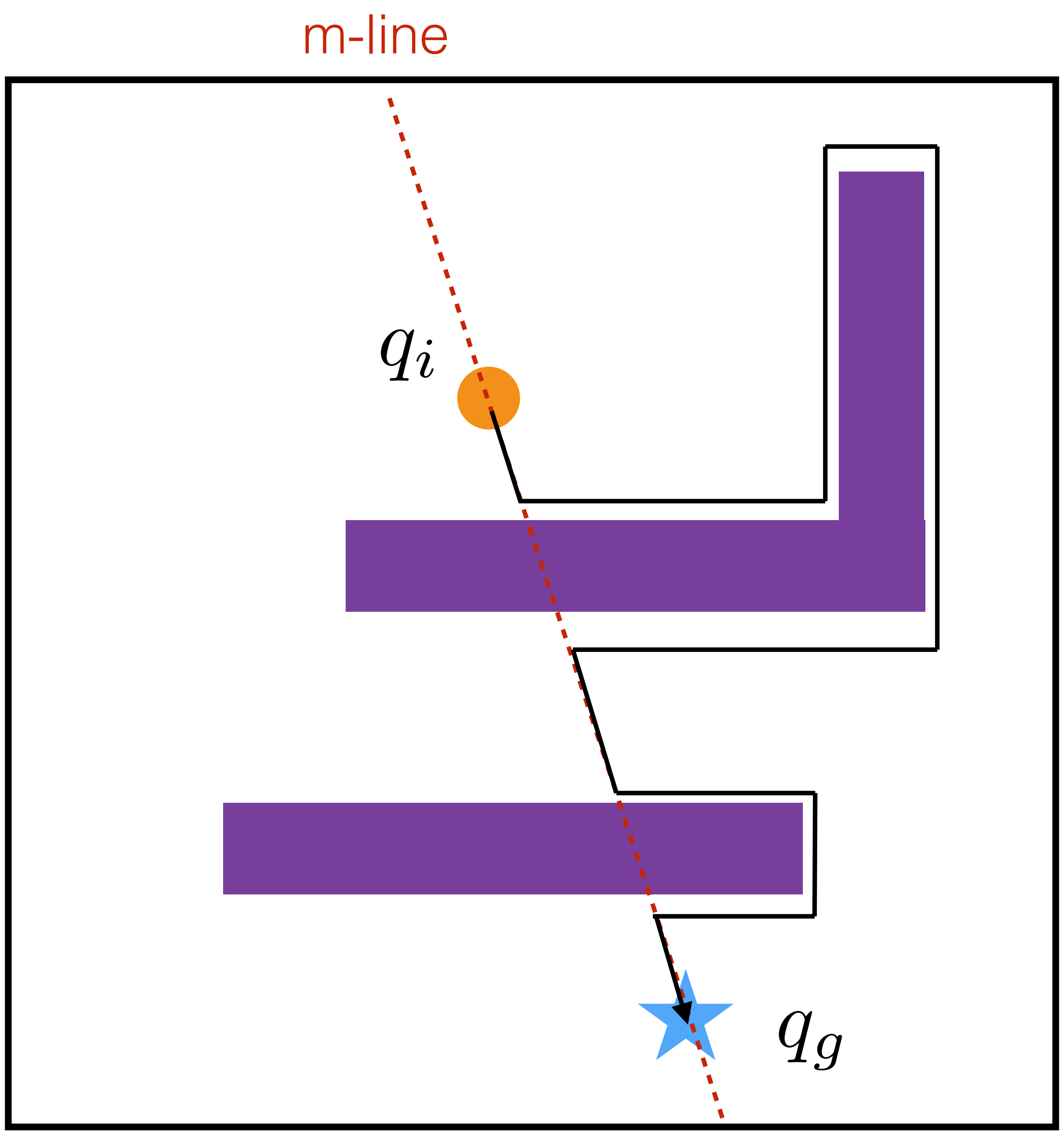


2D c-space

Bug Algorithms

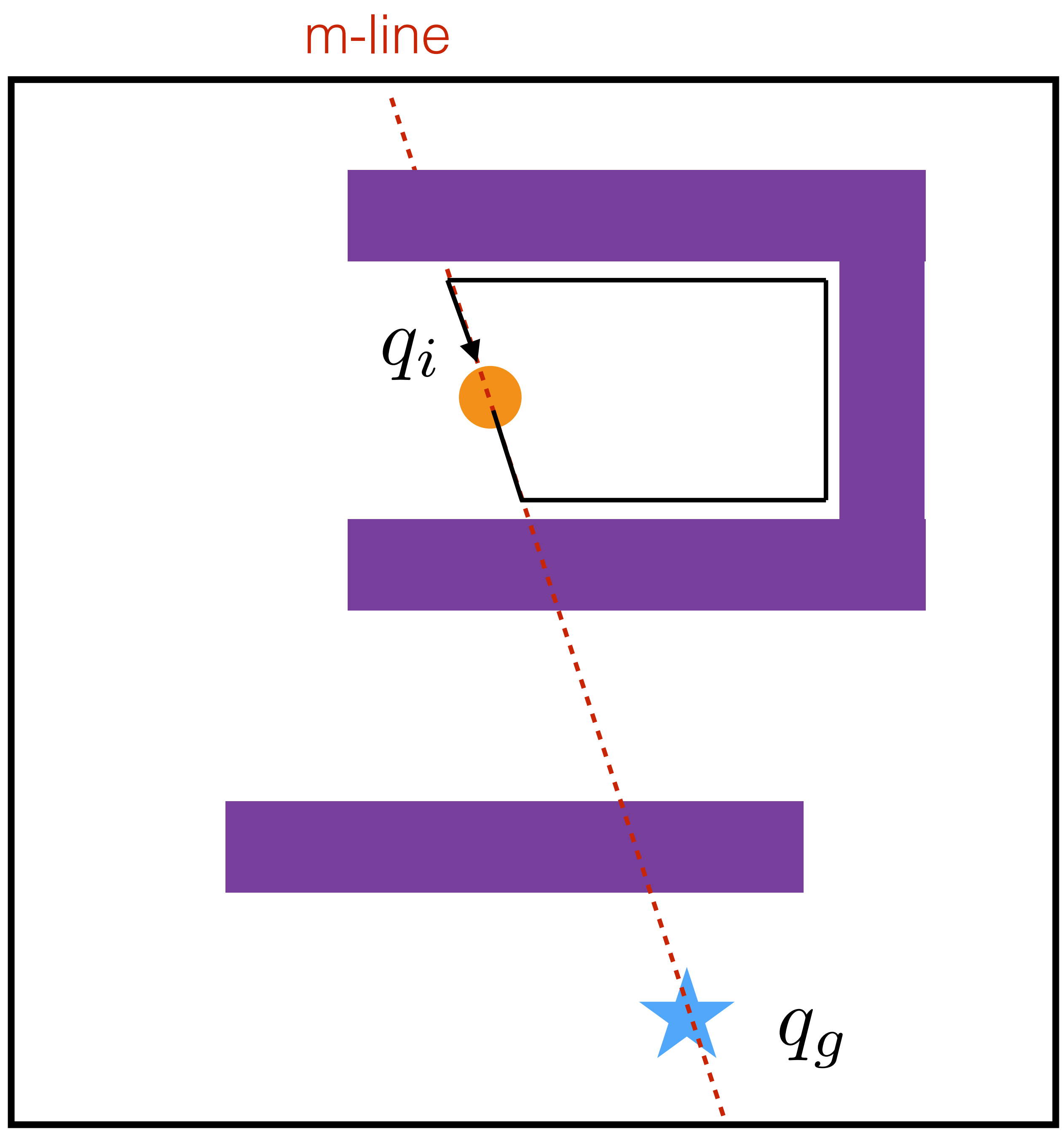
<u>Behaviour</u>:
- Head towards goal
- Follow wall until you **reach m-line** and can move towards goal again

2D c-space



Bug Algorithms

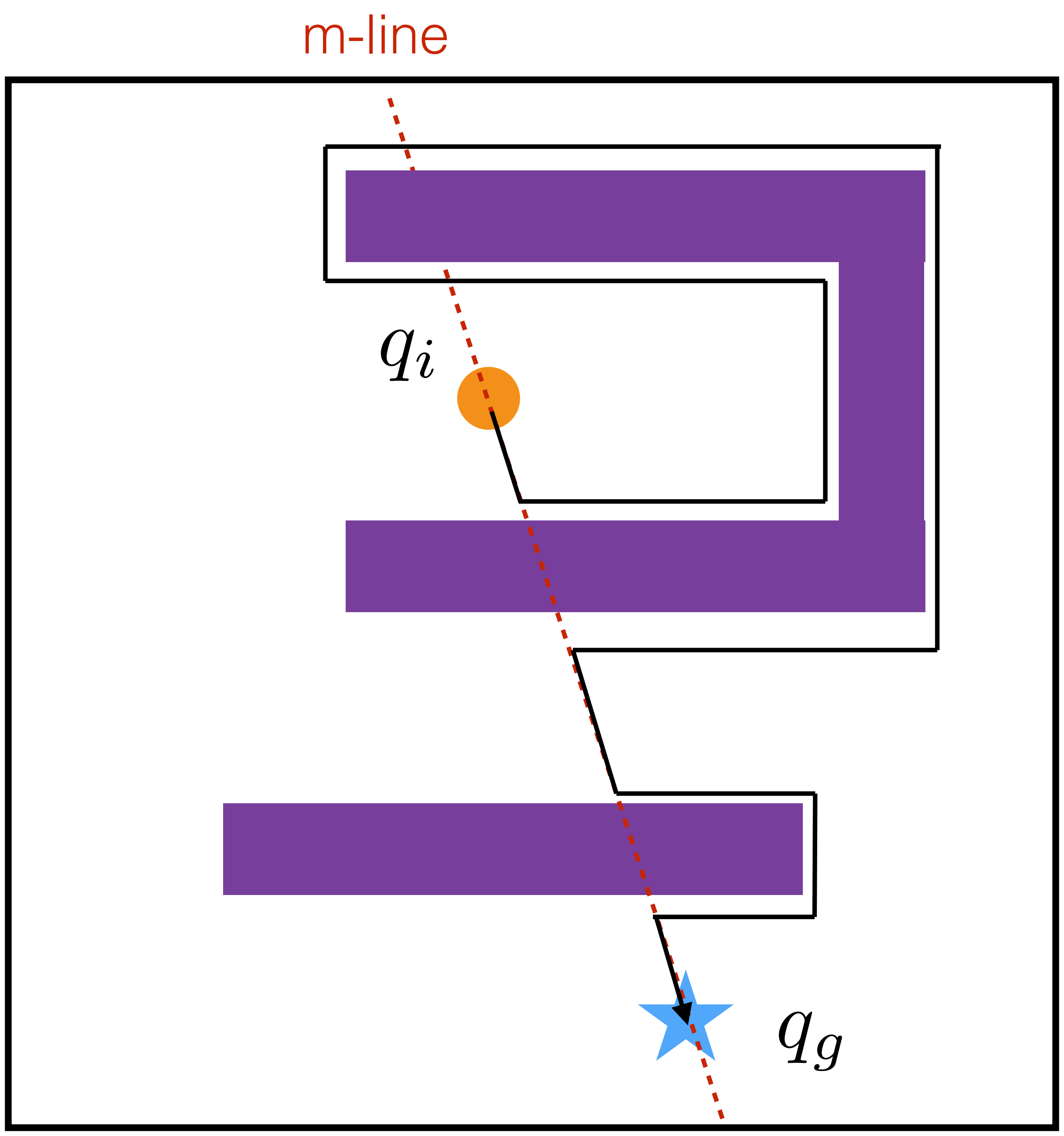
Behaviour:
- Head towards goal
- Follow wall until you **reach m-line** and can move towards goal again



2D c-space

Bug Algorithms

Behaviour:
- Head towards goal
- Follow wall until you reach m-line **closer to goal** and can move towards goal again



2D c-space

Bug Algorithms

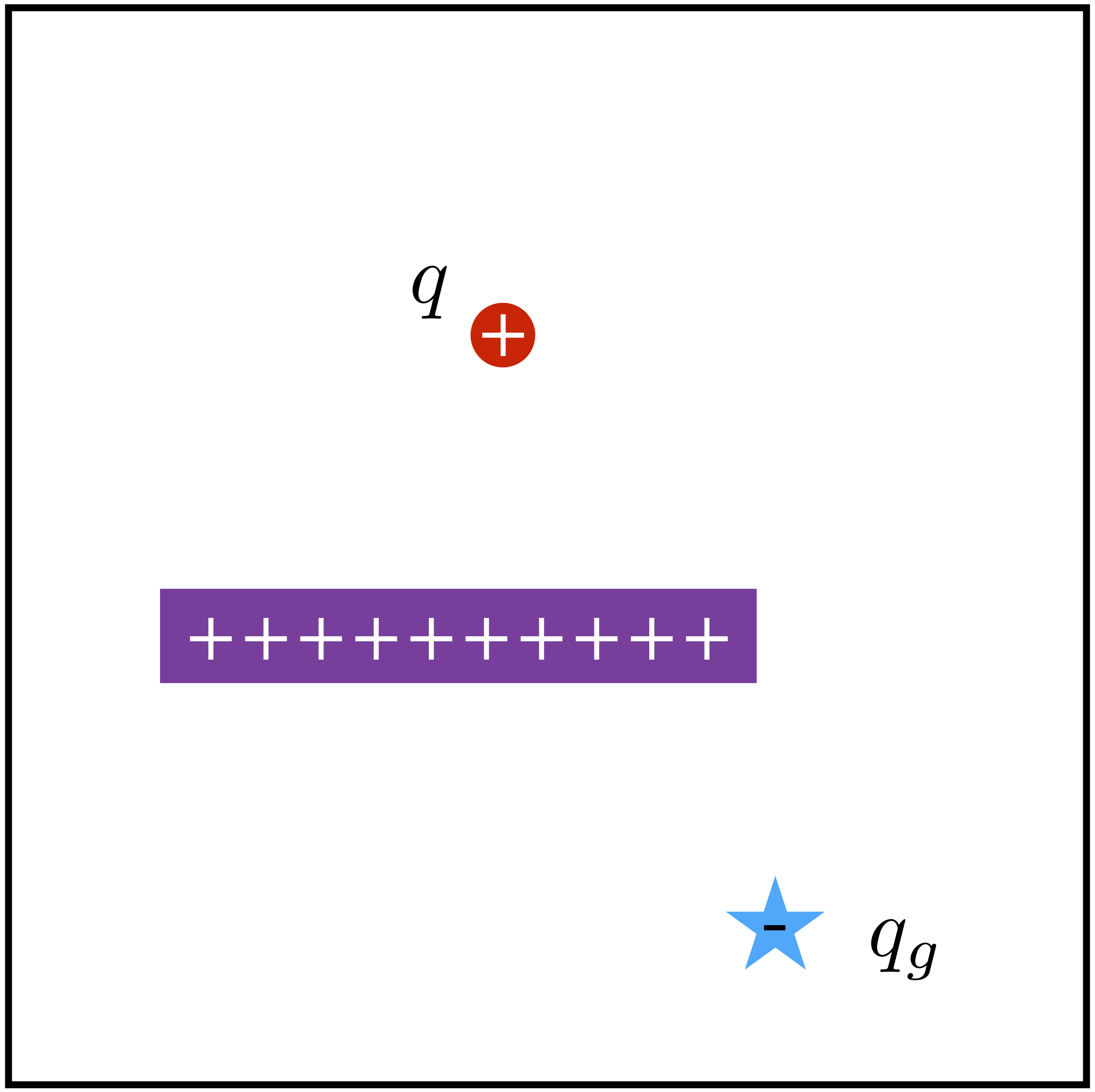
- Variety of other Bugs (TangentBug, VisBug, …)
  - Some use additional sensing, e.g. rangefinders

- Simple and robust behaviour
- Rely primarily on local information and sensing

- Applicable to 2D spaces (some bugs work in 3D)
  - Not suitable for robot arms
- Global information can often provide better paths

Potential Fields

Treat robot, goal, and obstacles as having electric charge

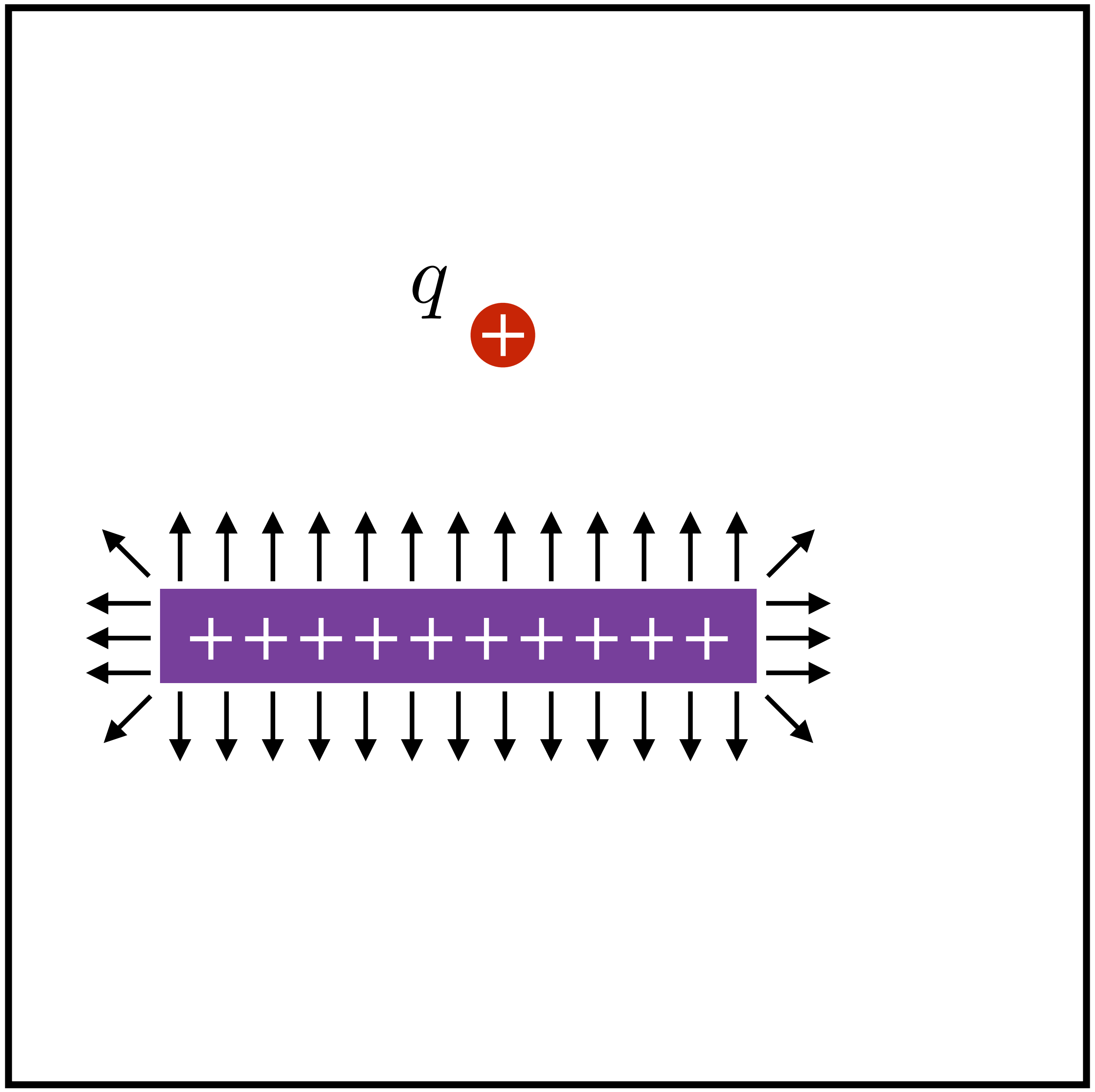


Same Charge:
Repulsion
(local)

Opposite Charge:
Attraction
(global)

Potential Fields

Treat robot, goal, and obstacles as having electric charge

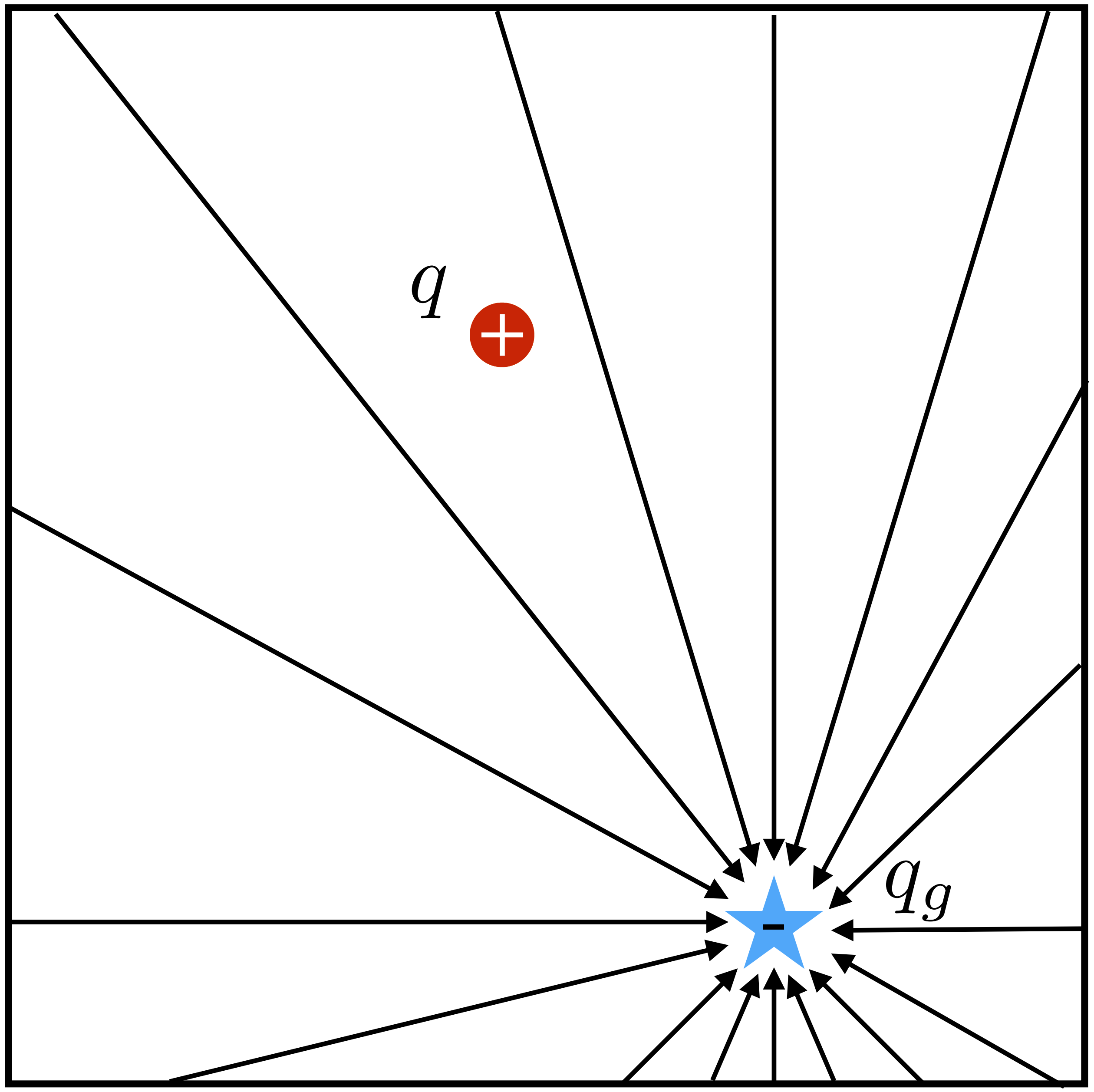


Same Charge: Repulsion (local)

Opposite Charge: Attraction (global)

Potential Fields

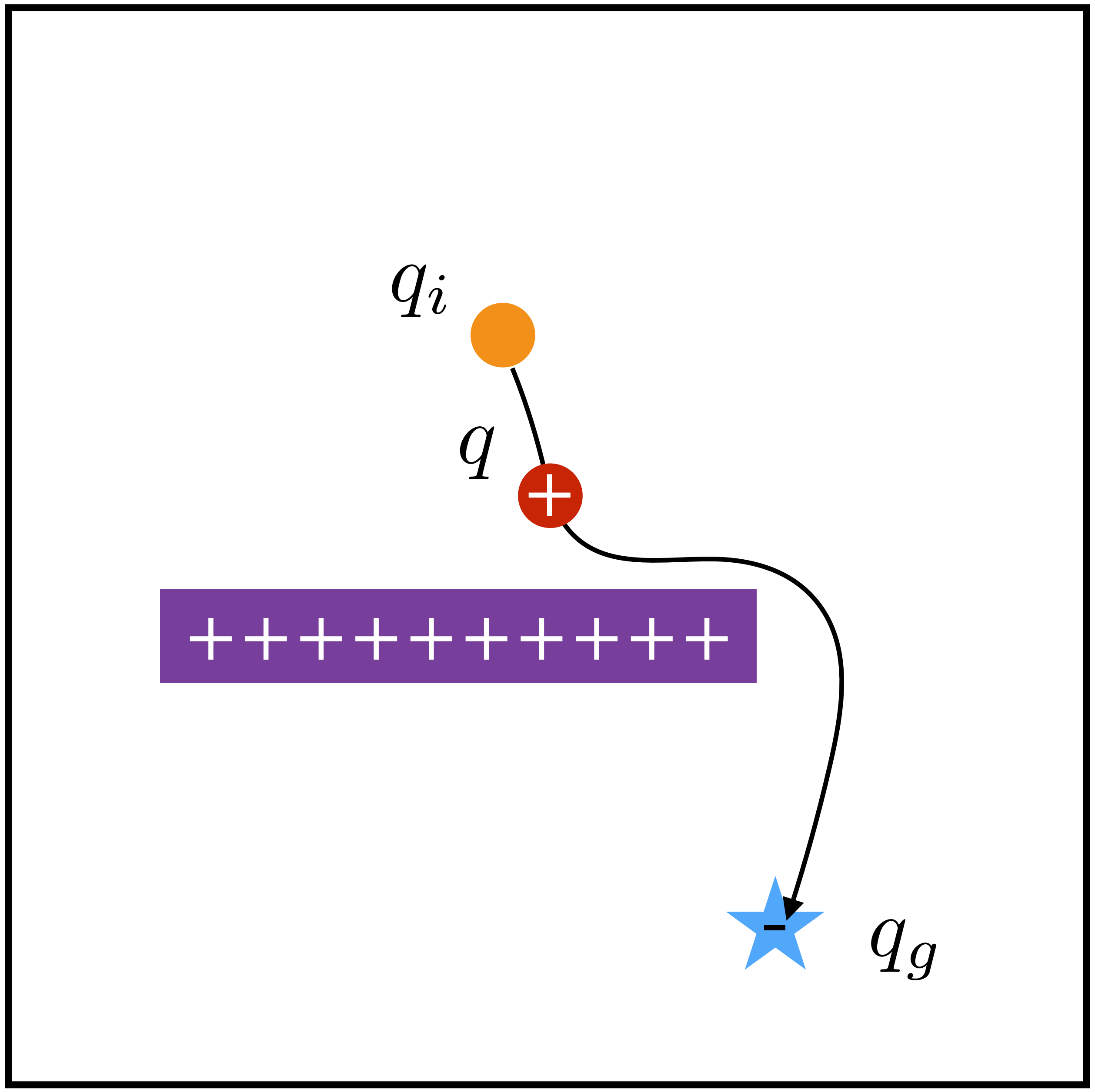
Treat robot, goal, and obstacles as having electric charge



Opposite Charge: Attraction (global)

Potential Fields

Treat robot, goal, and obstacles as having electric charge



Same Charge:
Repulsion

Opposite Charge:
Attraction

Proposed originally (Khatib) for real-time collision avoidance

Potential Fields

- Charged objects create a potential field
  Potential function:

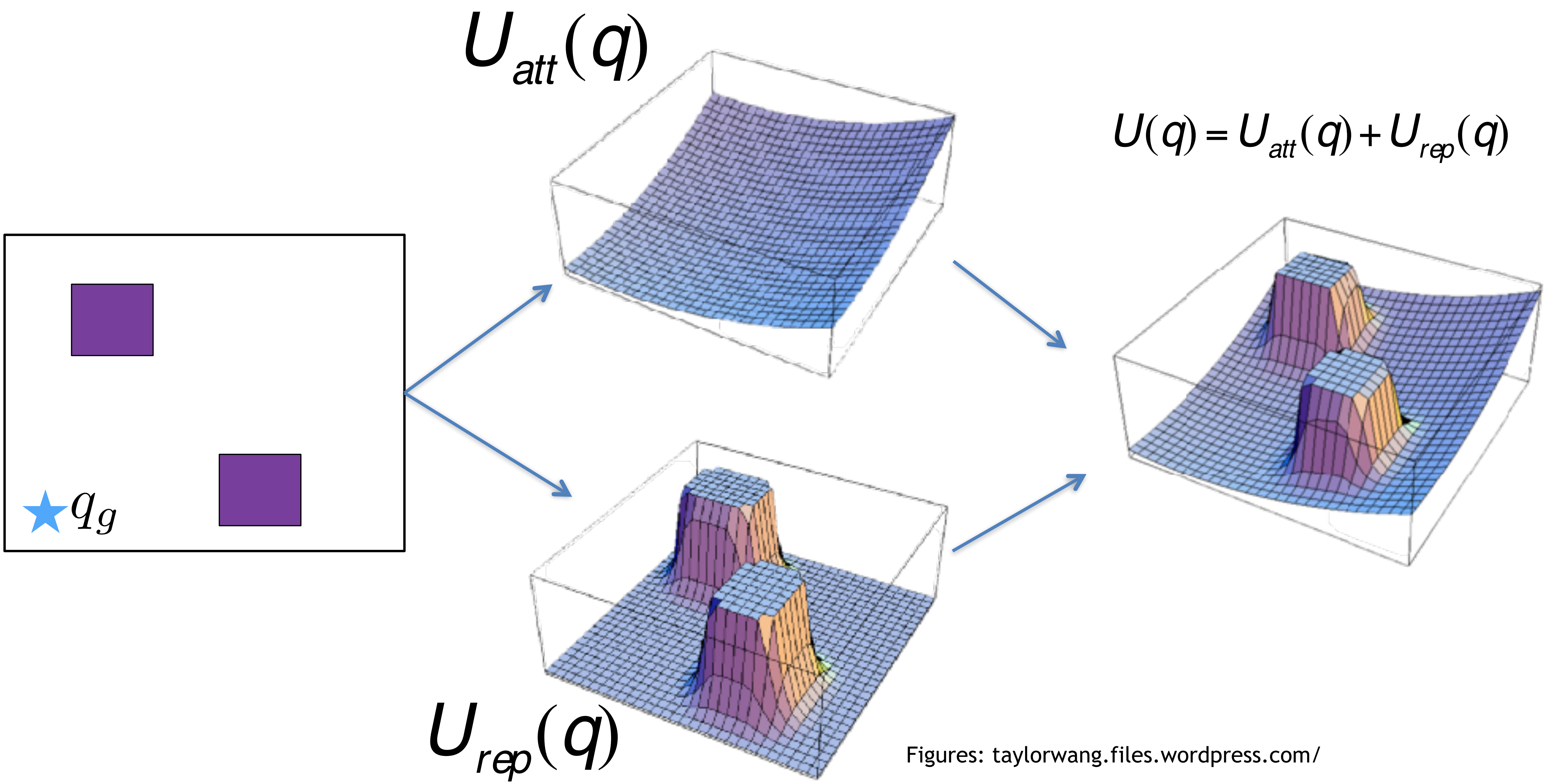
$$U(q) \in \mathbb{R}$$

- Robot moves towards lower energetic configuration
  Follow negative gradient of potential:

$$-\nabla U(q)$$

$$q_{t+1} = q_t - \alpha \nabla U(q_t)$$

Potential Fields

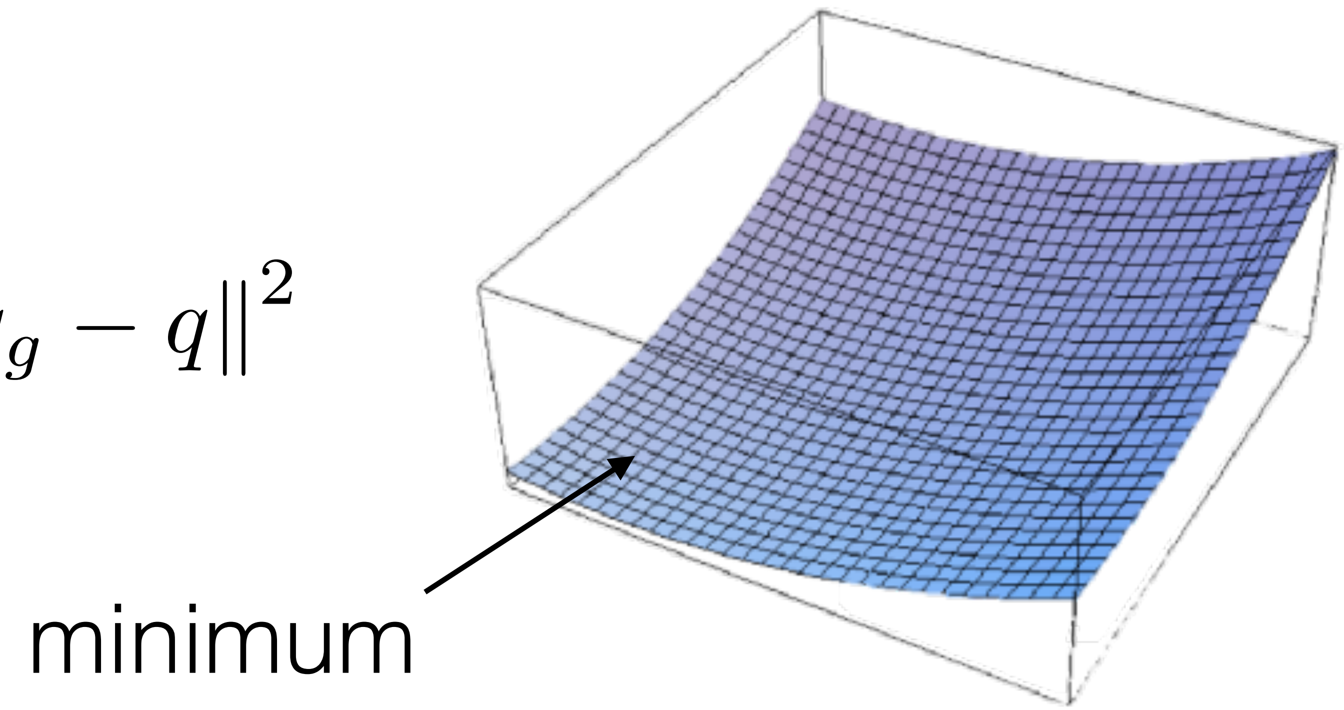


Figures: taylorwang.files.wordpress.com/

Attractive Field

- Define quadratic field for attraction to goal

$$U_{att}(q) = \frac{1}{2} k_a \left\| q_g - q \right\|^2$$



- Gradient always points to goal

$$-\nabla U_{att}(q) = k_a (q_g - q)$$

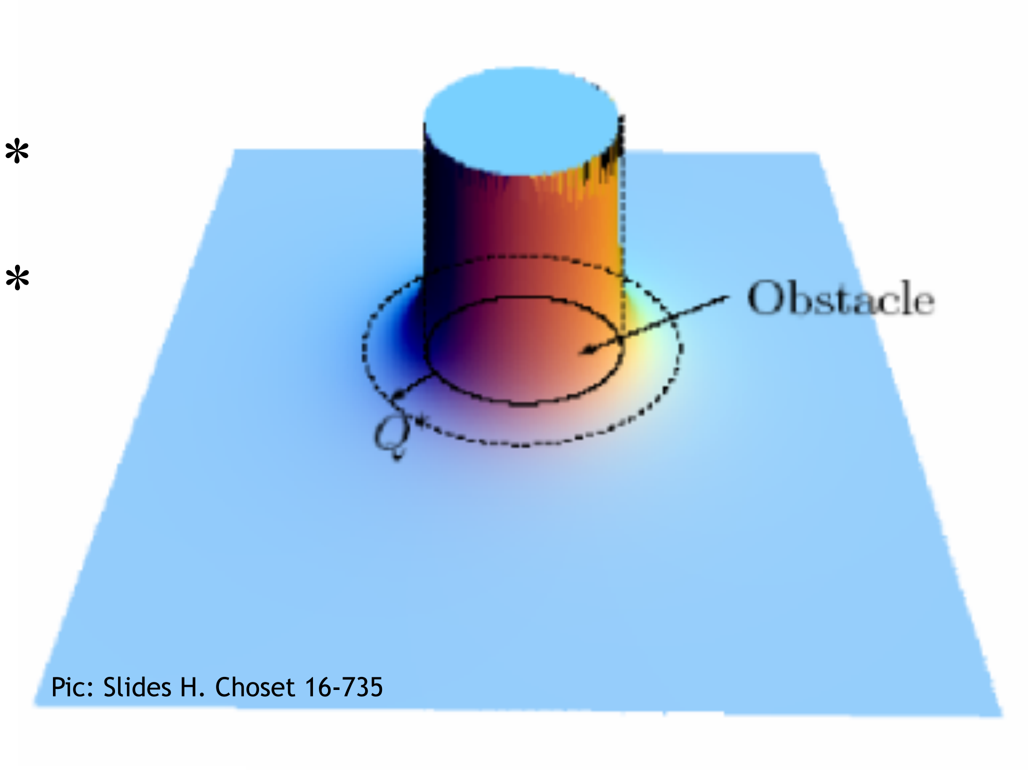
Repulsive Field

- Define repulsive field based on min distance to obstacles

$$U_{rep}(q) = \begin{cases} \frac{1}{2}\left(\frac{1}{p(q)} - \frac{1}{Q^*}\right)^2 & \text{if } p(q) \leq Q^* \\ 0 & \text{if } p(q) > Q^* \end{cases}$$

Minimum distance to object to q

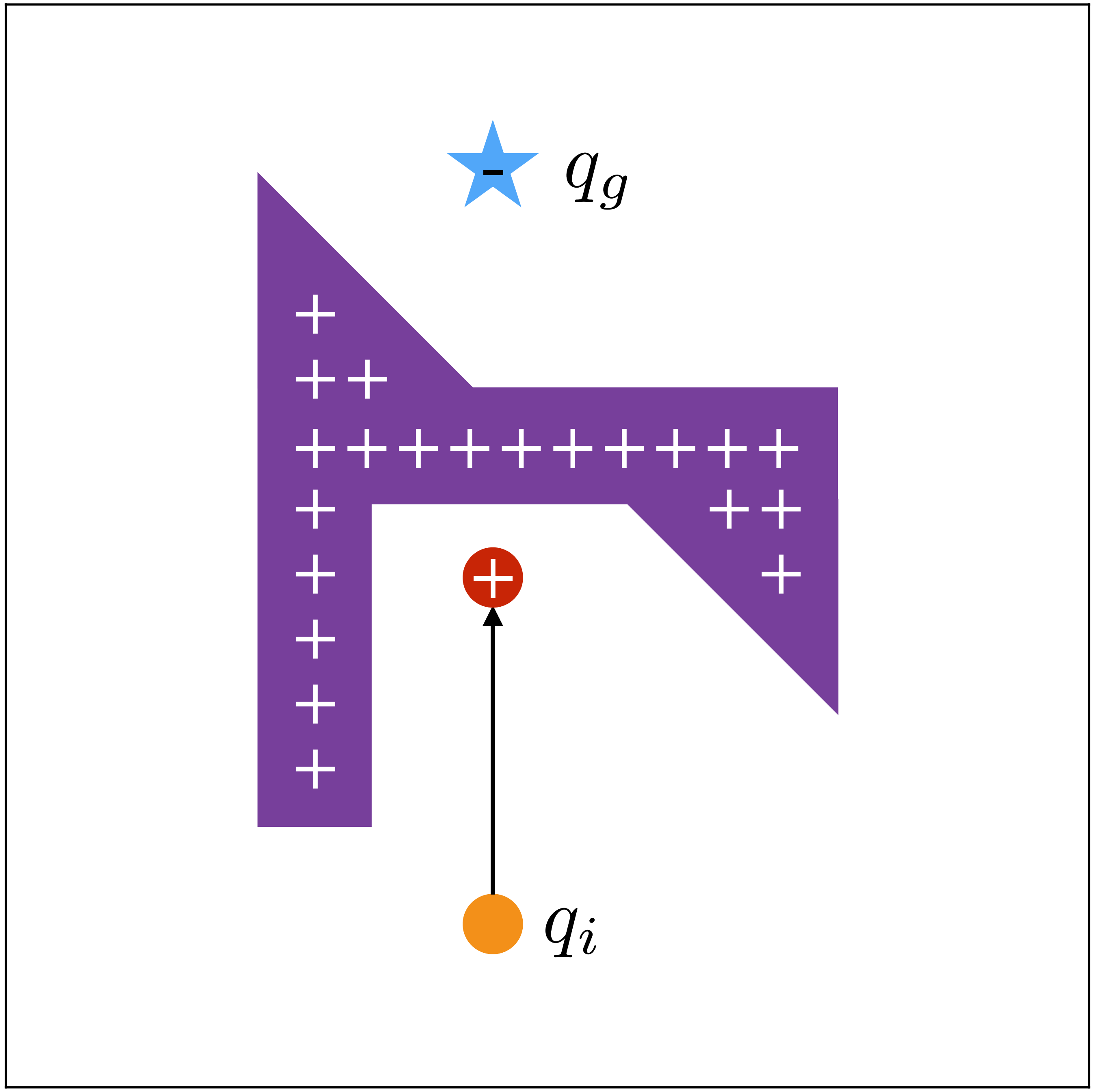
Distance of influence



- Potential becomes infinite at obstacle boundary
- Potential becomes zero beyond a distance of Q*
- Obstacles can create oscillations

Local Minima

Obstacles can lead to getting stuck in additional local minima



Local Optimum

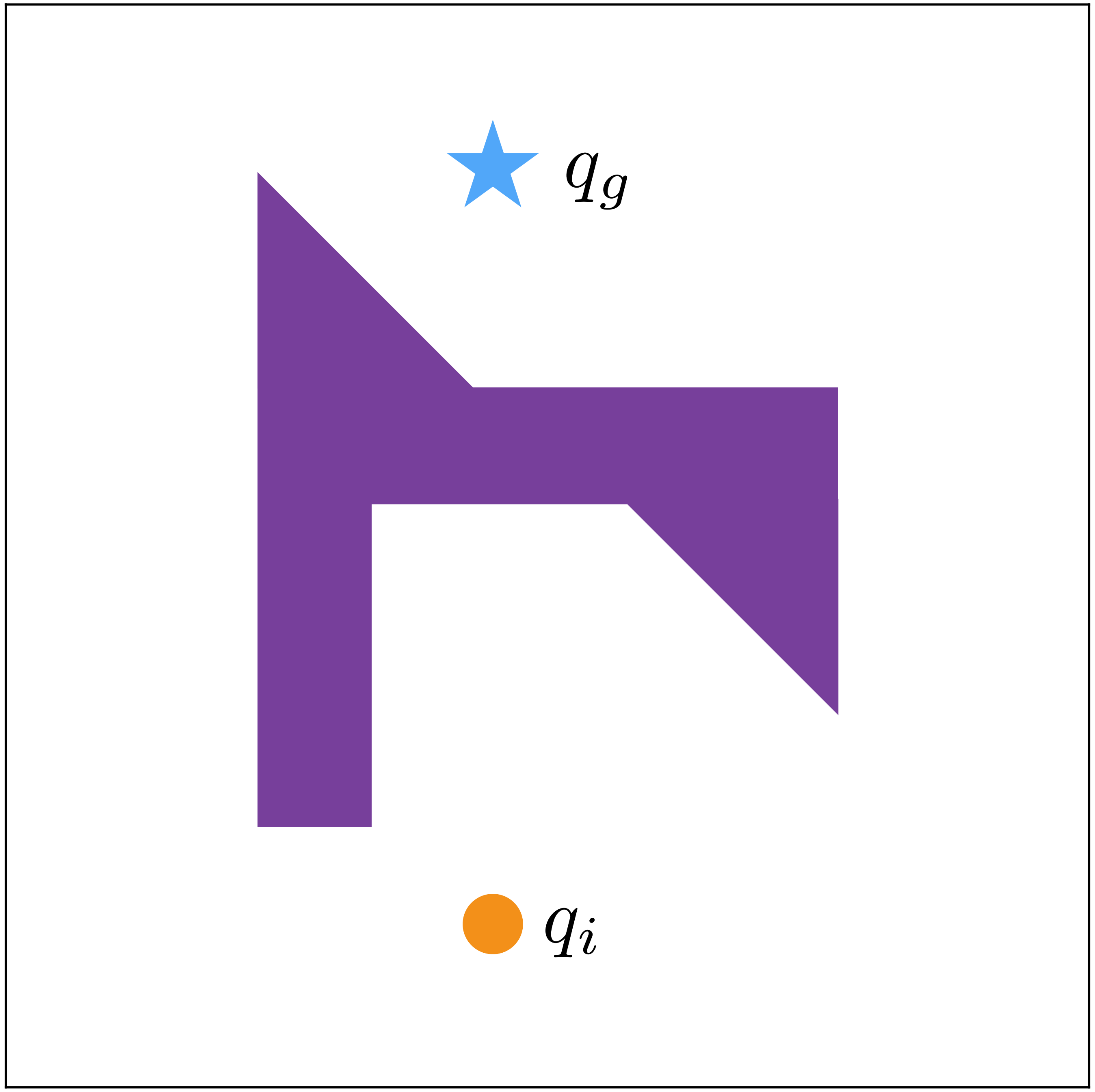
Distance to Goal

- Ideally $U(q)$ would capture the shortest distance to goal
  - Minimum distance from q to q_g while avoiding obstacles
  - Follow gradient to find shortest path to goal
  - Similar principle to dynamic programming

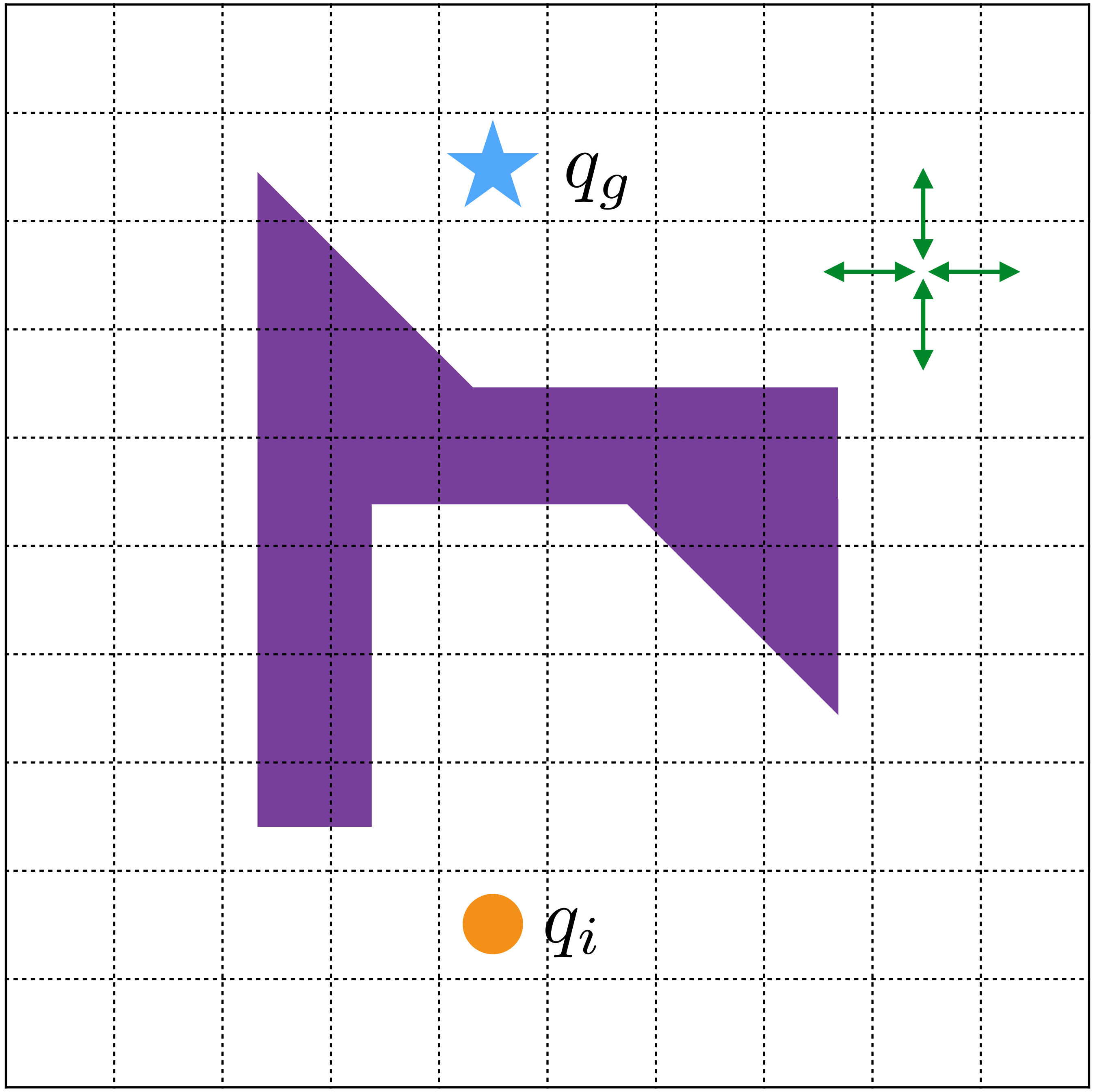
- Computing min continuous distance is generally intractable

- Can approximate the shortest distance by discretization

Wavefront Path Planning



Wavefront Path Planning



Wavefront Path Planning

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

Wavefront Path Planning

Shortest Discretized Path

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

Always move to the adjacent cell with the lowest value

Wavefront Path Planning

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

Shortest Discretized Path

Always move to the adjacent cell with the lowest value

Wavefront Path Planning

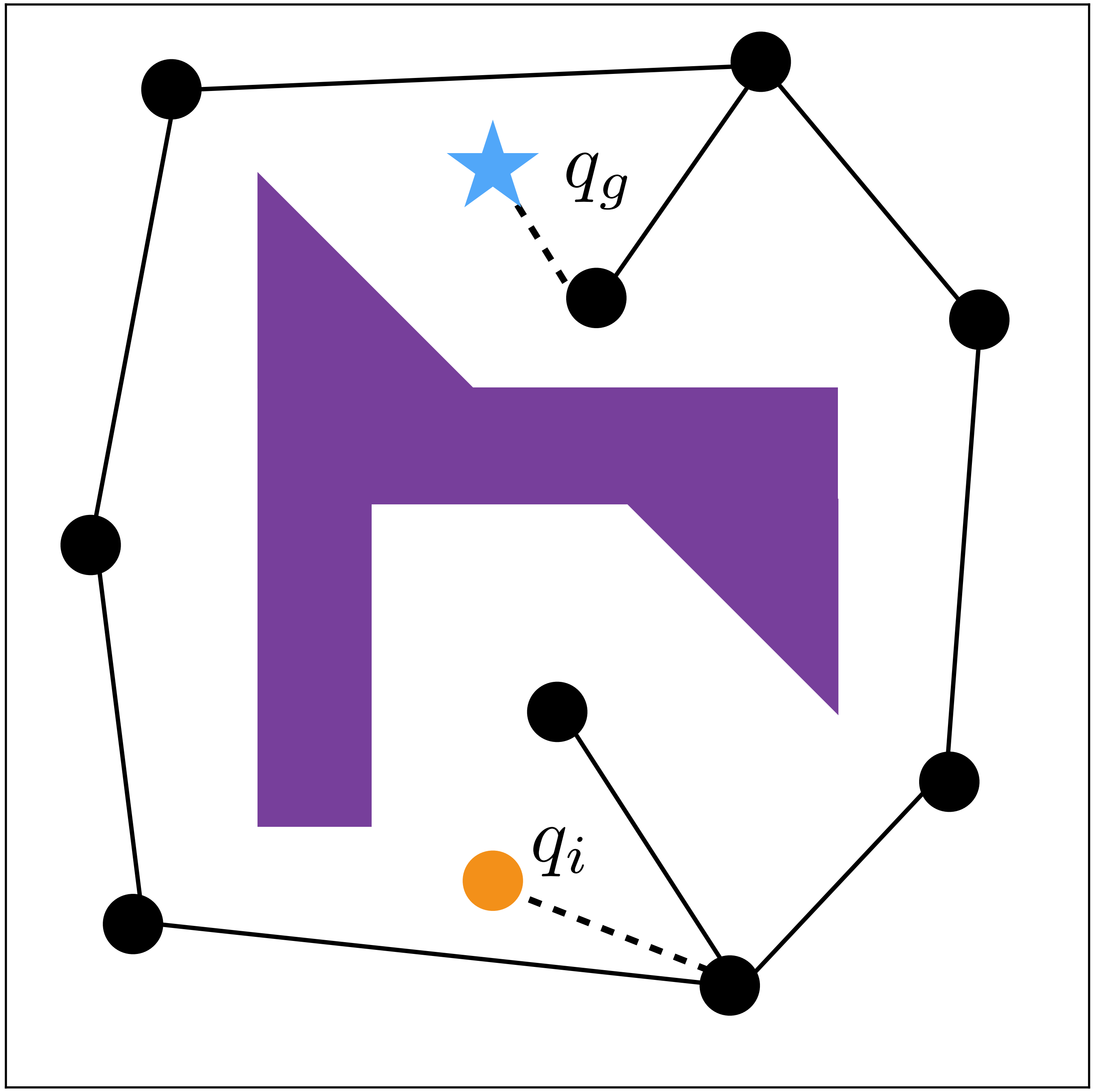
- Discretization makes the path planning tractable

- Grid-based discretisation performs poorly
  - Computationally expensive
  - Does not scale to high-dimensional spaces
  - Approximate obstacles may remove potential paths

- How can we “discretize” the space more efficiently?

Topological Graphs

Capture free space and connectivity using a graph structure



The graph consists of vertices and edges in free space

Topological Graphs

- Use graph to turn continuous problem into discrete one
- Define a topological graph $G(V, E)$ as:
  - Vertices

$$v_i \in C_{free}$$

  - Edges

$$e_{ij} \equiv \tau : [0, 1] \to C_{free}$$

- Define swath of graph as all reachable configurations in graph

$$S = \bigcup_{e \in E} e([0, 1]) \qquad S \subset C_{free}$$

Road Maps

- Road map is a topological graph s.t. for all $q_i \in C_{free}$ and $q_g \in C_{free}$
  - **Accessibility**:
    There is a path from q_i to some $q' \in S$
  - **Departability**:
    There is a path from q_g to $q'' \in S$
  - **Connectivity Preservation**:
    If there is a path in C_{free} from q_i to q_g

    then there is a path in S from q' to q''

Combinatorial vs Sample-based Planning

- Combinatorial Planning
  - Create graph to capture connectivity of C_{free}
  - Explicitly represent C_{obs}
- Sample-based Planning
  - Sample vertices in c-space to create graph
  - Use collision checking to detect obstacles

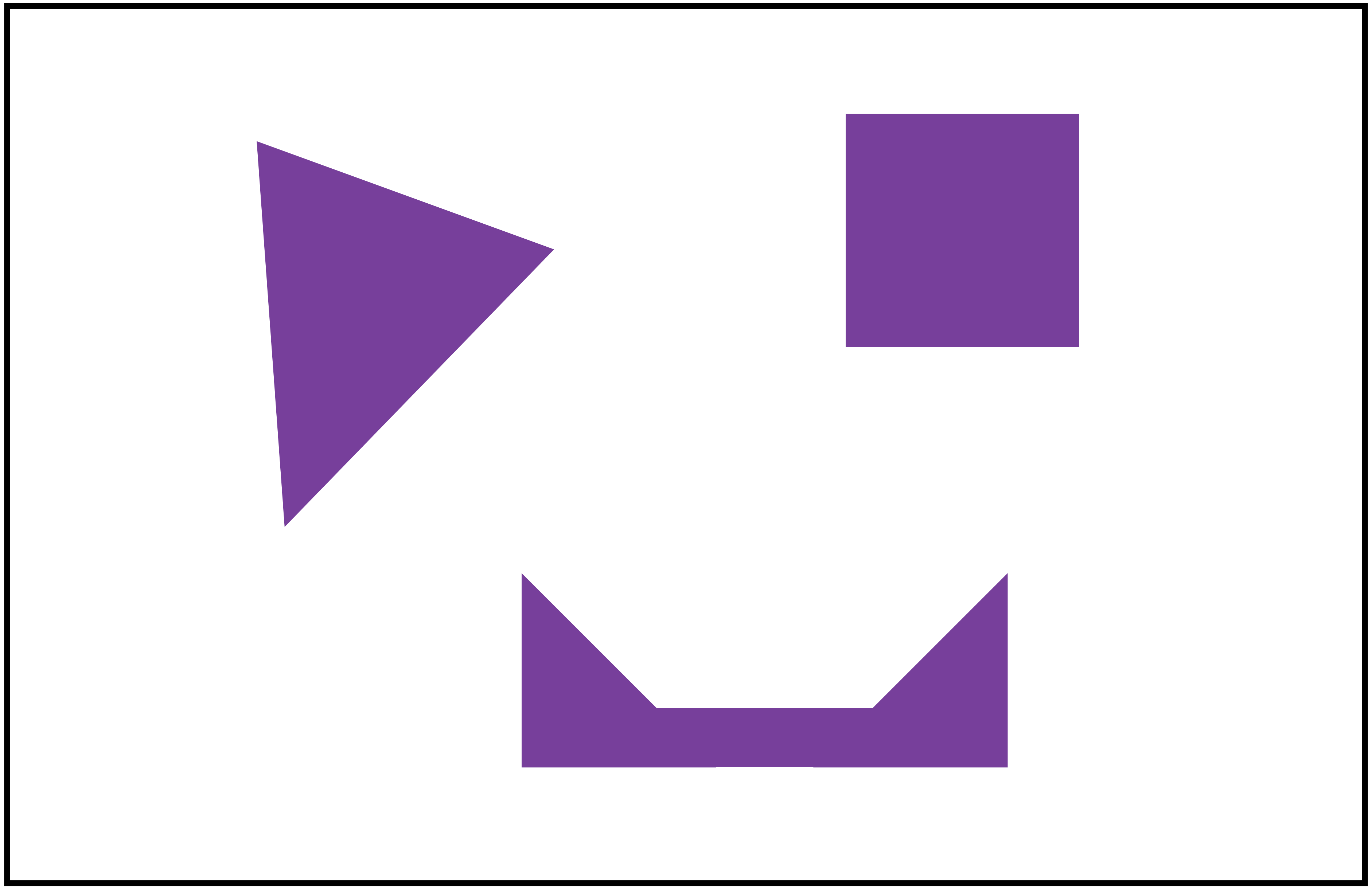
Completeness

- Complete:
  Planner is complete if 1) it always returns a solution if one exists and 2) otherwise returns a failure in bounded time

- Probabilistic Complete:
  Planner is probabilistic complete if the probability of a solution existing tends to zero as the number of sampled points increases and no solution is found. No time bound.

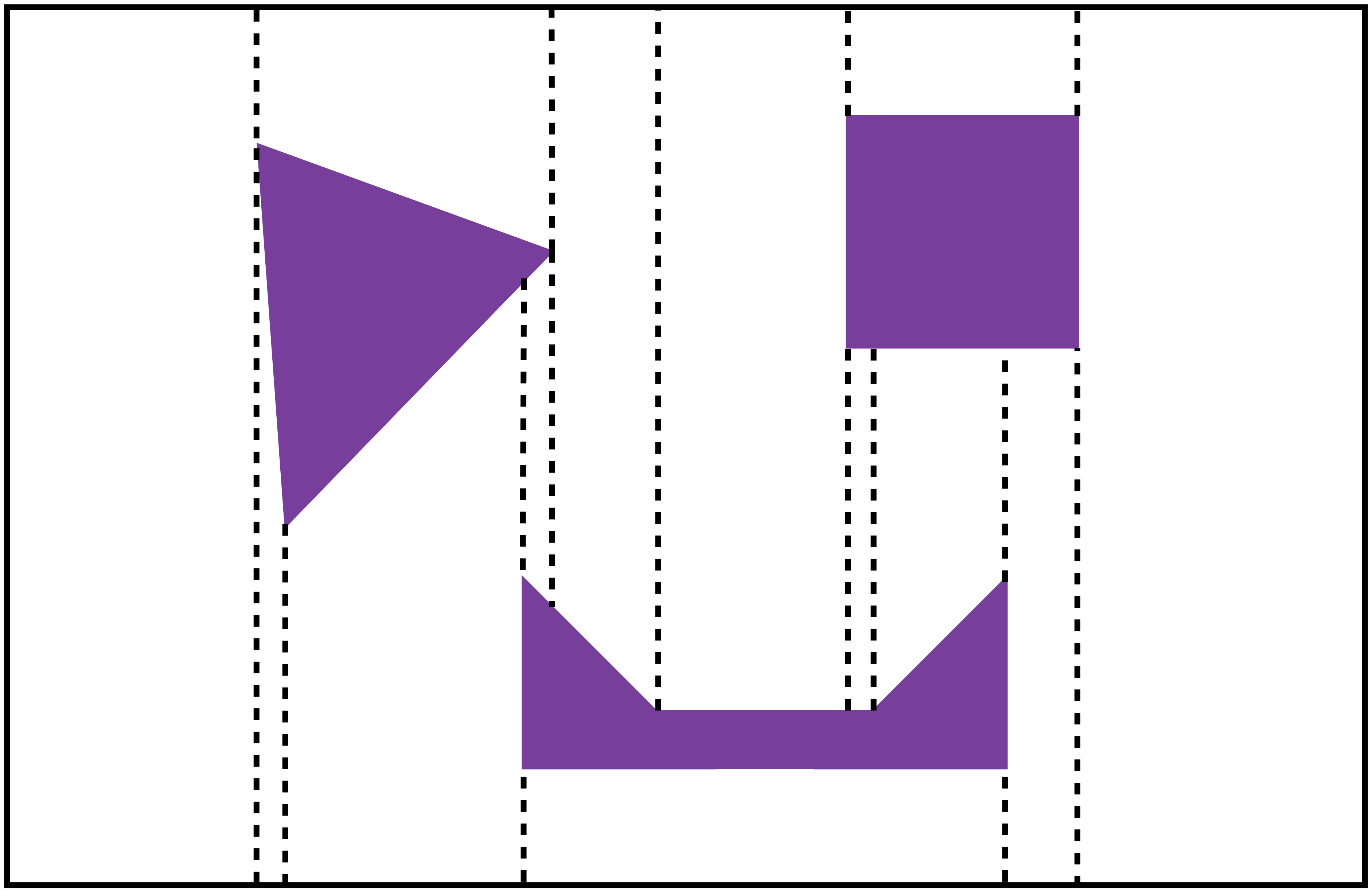
- Focus on feasibility rather than optimality

Example C-Space



Vertical Decomposition

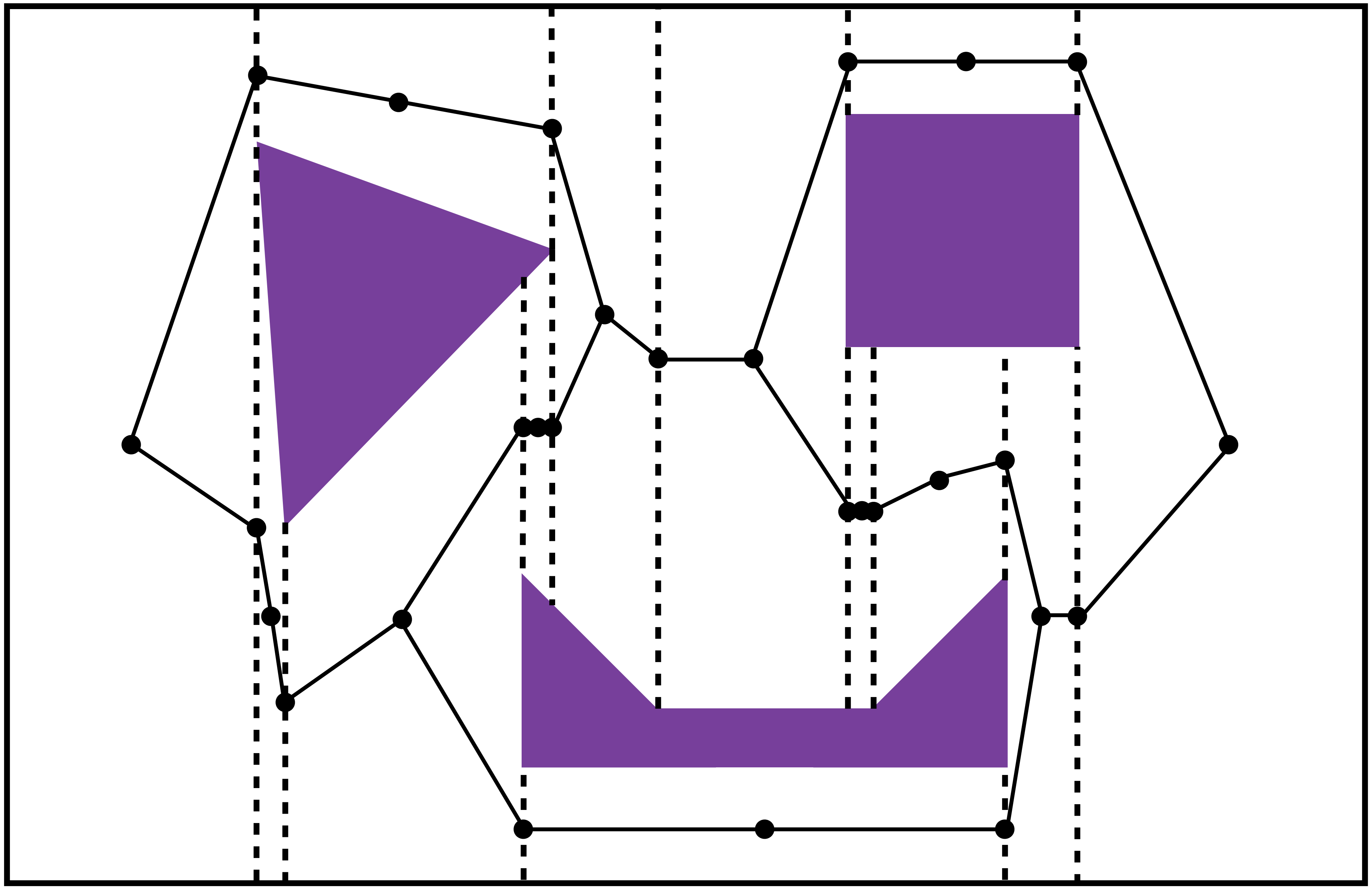
- Divide c-space using vertical cuts at obstacle corners



- Divides space into rectangles and trapezoids (convex)

Vertical Decomposition

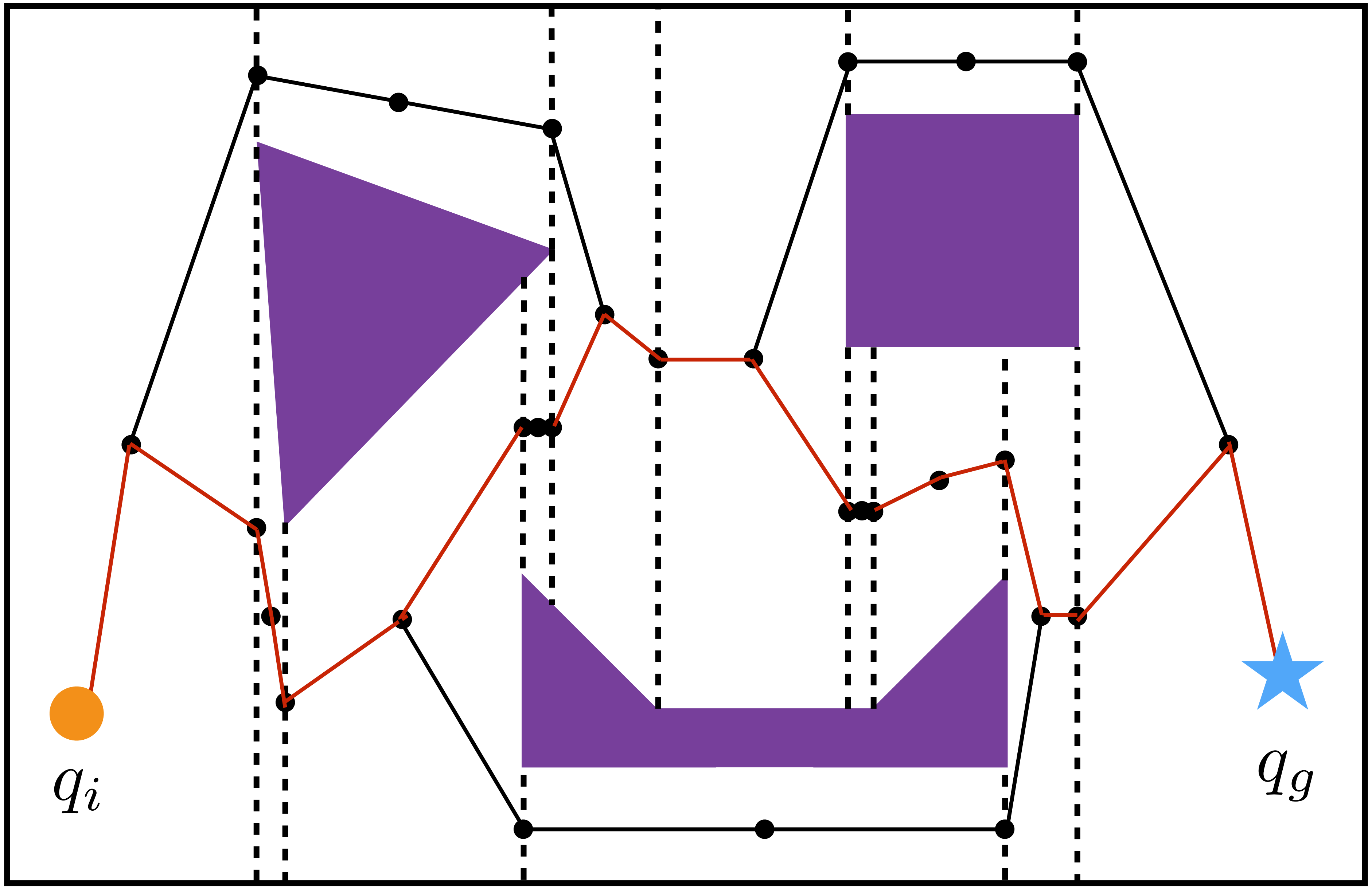
- Connect adjacent cells to create roadmap (complete)



- Convex cells ensure accessibility and departability

Vertical Decomposition

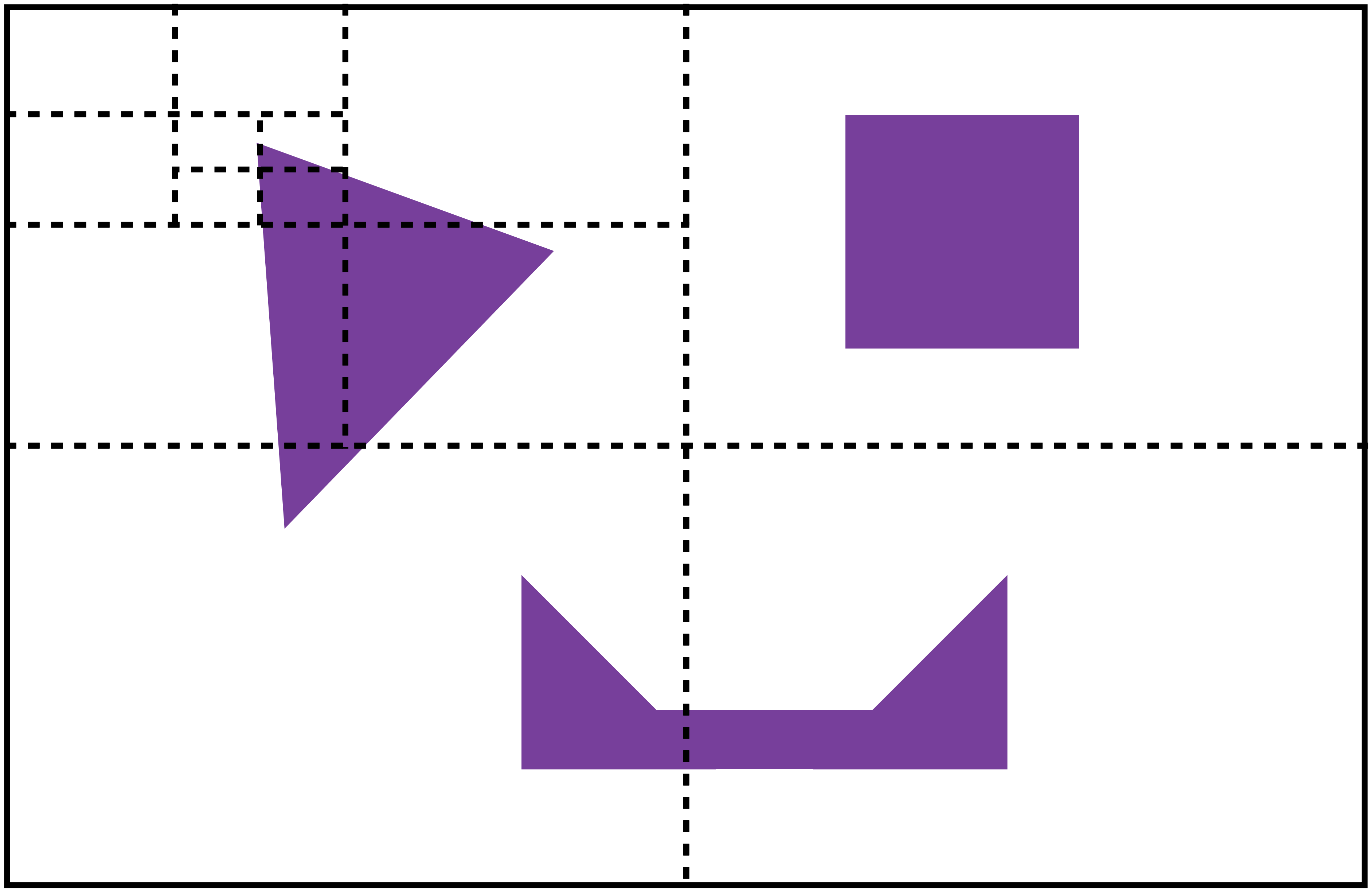
- Connect adjacent cells to create roadmap (complete)



- Convex cells ensure accessibility and departability

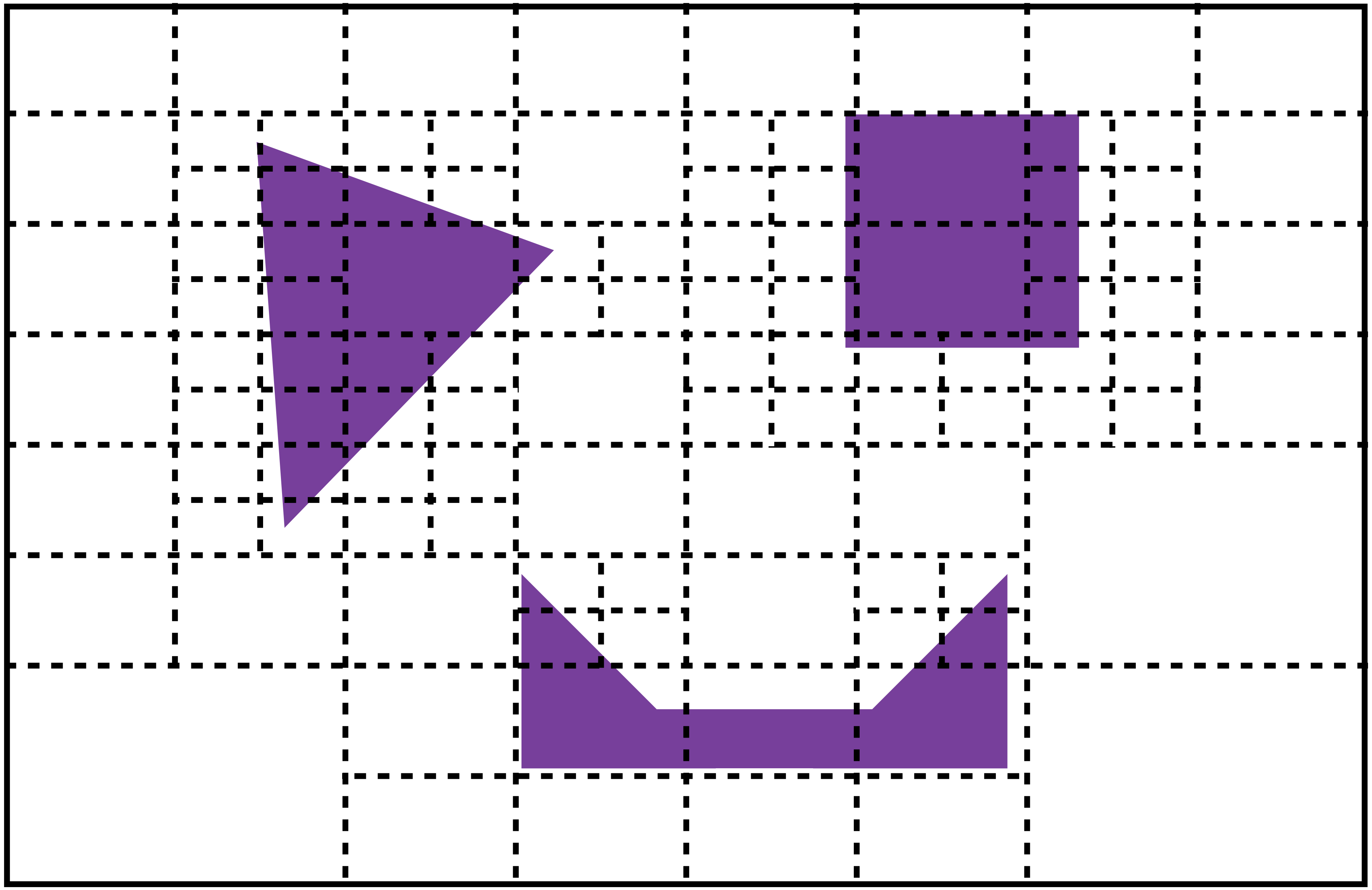
Approximate Decomposition

- Divide mixed cells (obs+free) into smaller cells



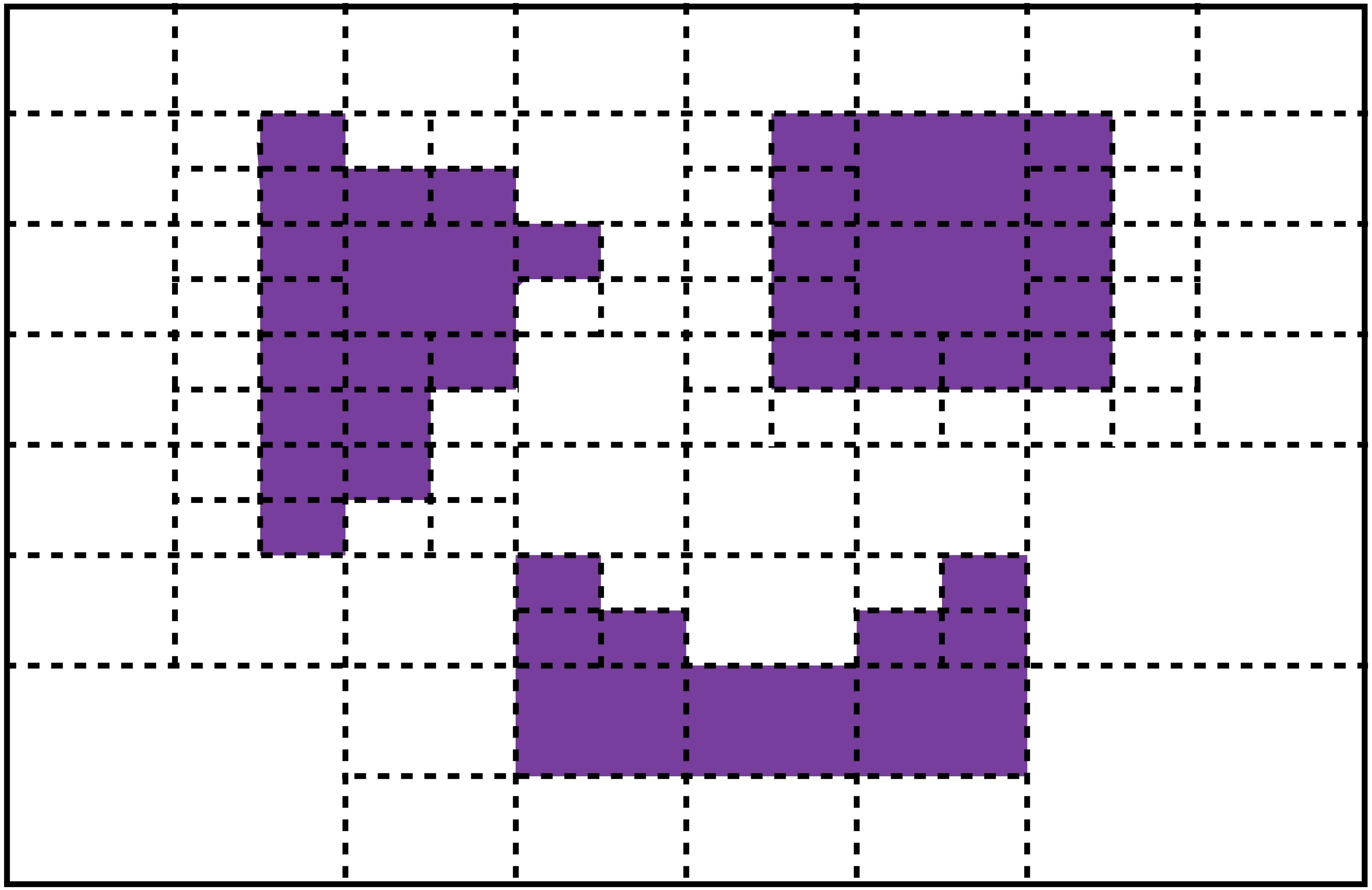
- Continue until not mixed, or min resolution is reached

Approximate Decomposition



Approximate Decomposition

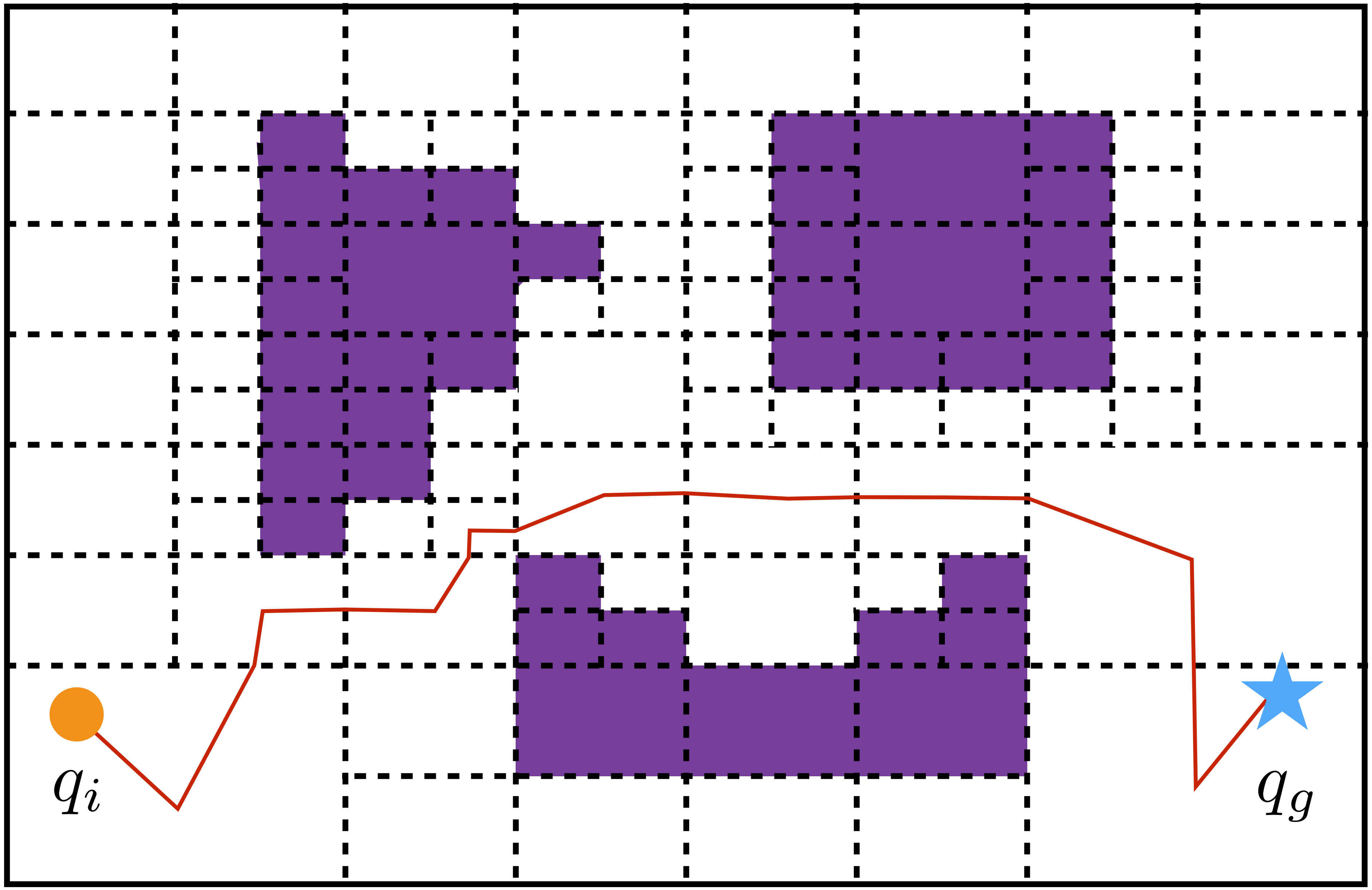
- Create graph again based on cell adjacency (not shown)



- Min resolution restricts completeness of the approach

Approximate Decomposition

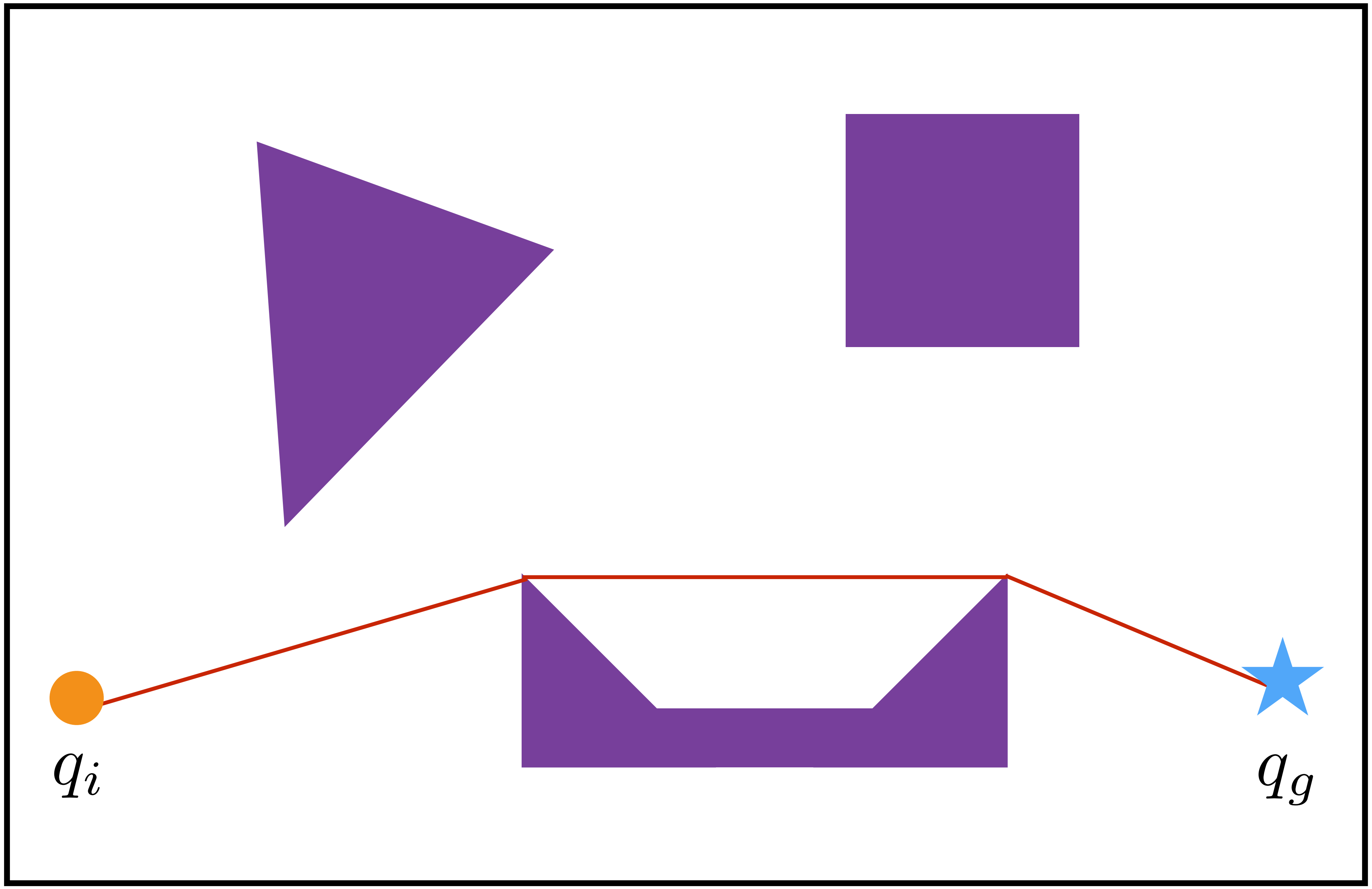
- Create graph again based on cell adjacency (not shown)



- Min resolution restricts completeness of the approach

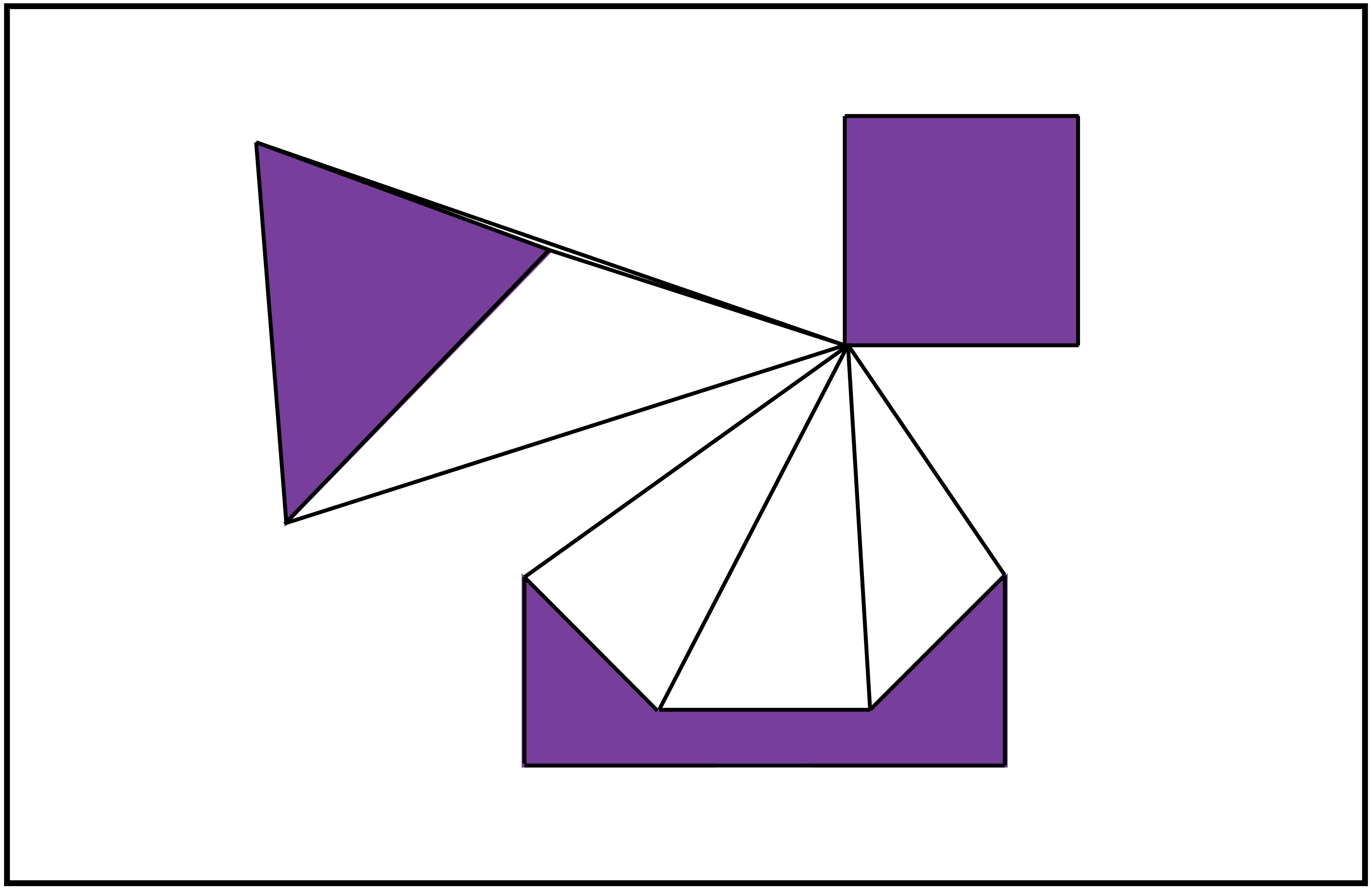
Visibility Graph

- Shortest path between start and goal hits the corners



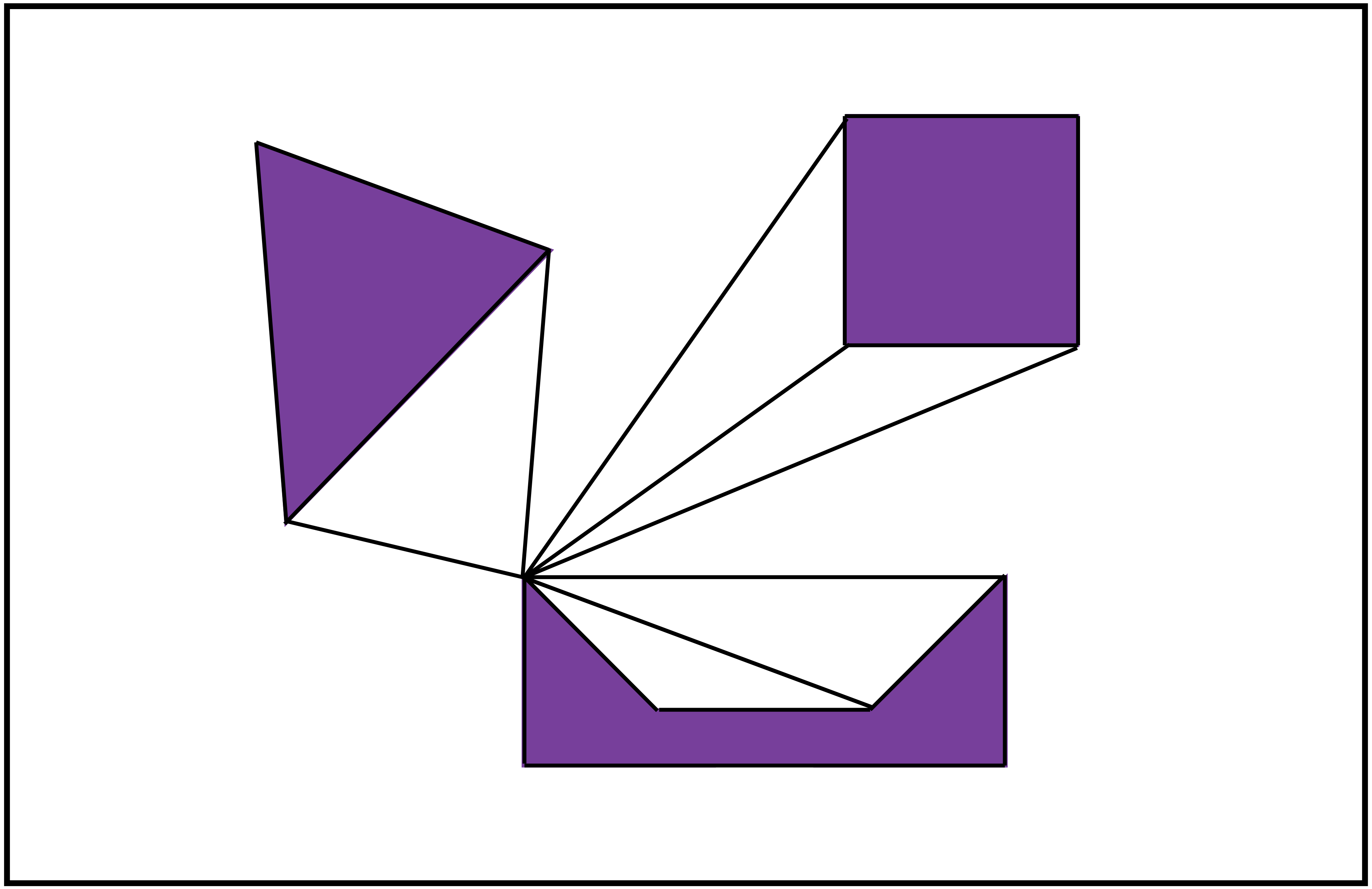
Visibility Graph

- Create graph by adding edges between visible corners



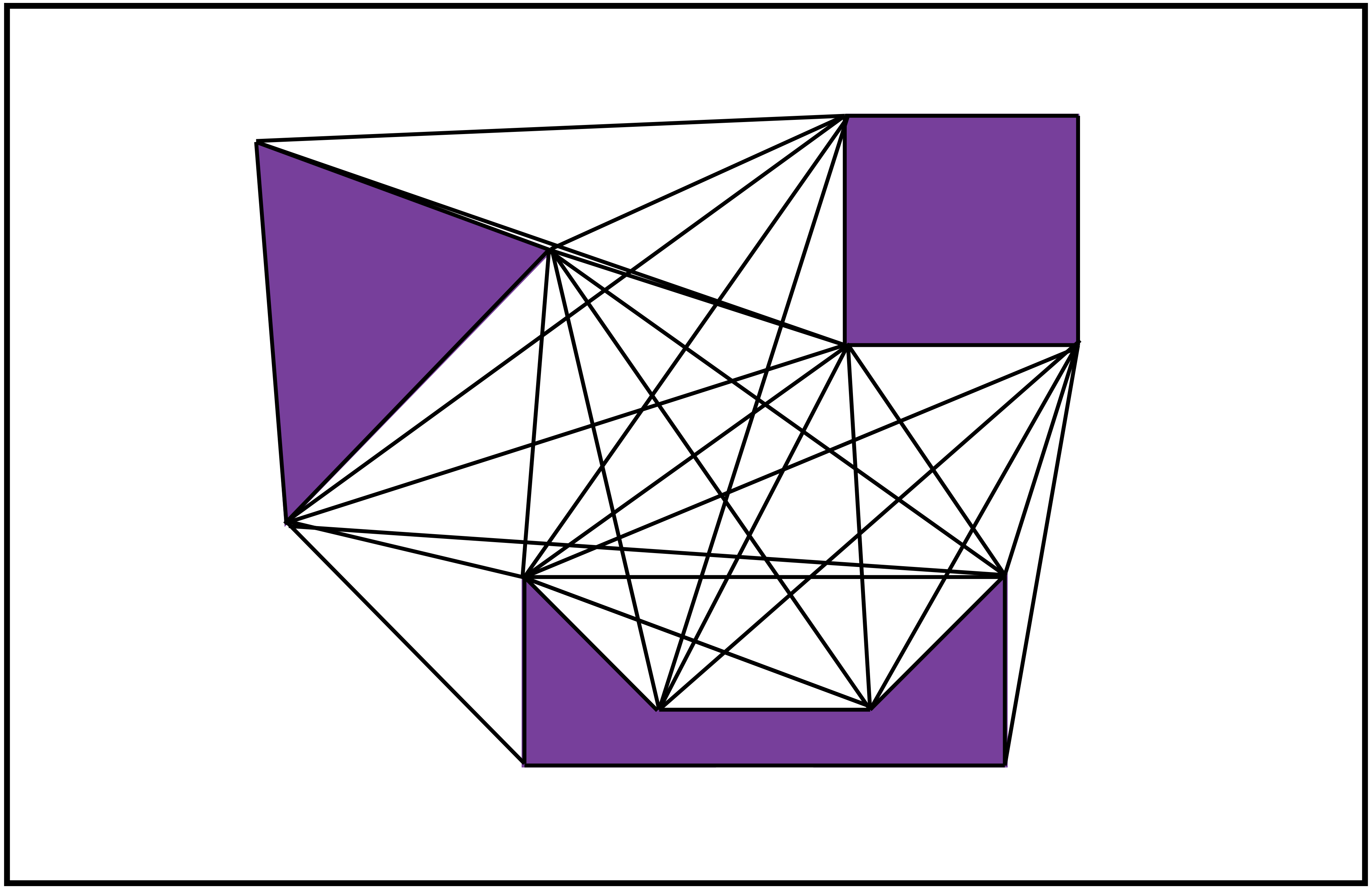
Visibility Graph

- Create graph by adding edges between visible corners



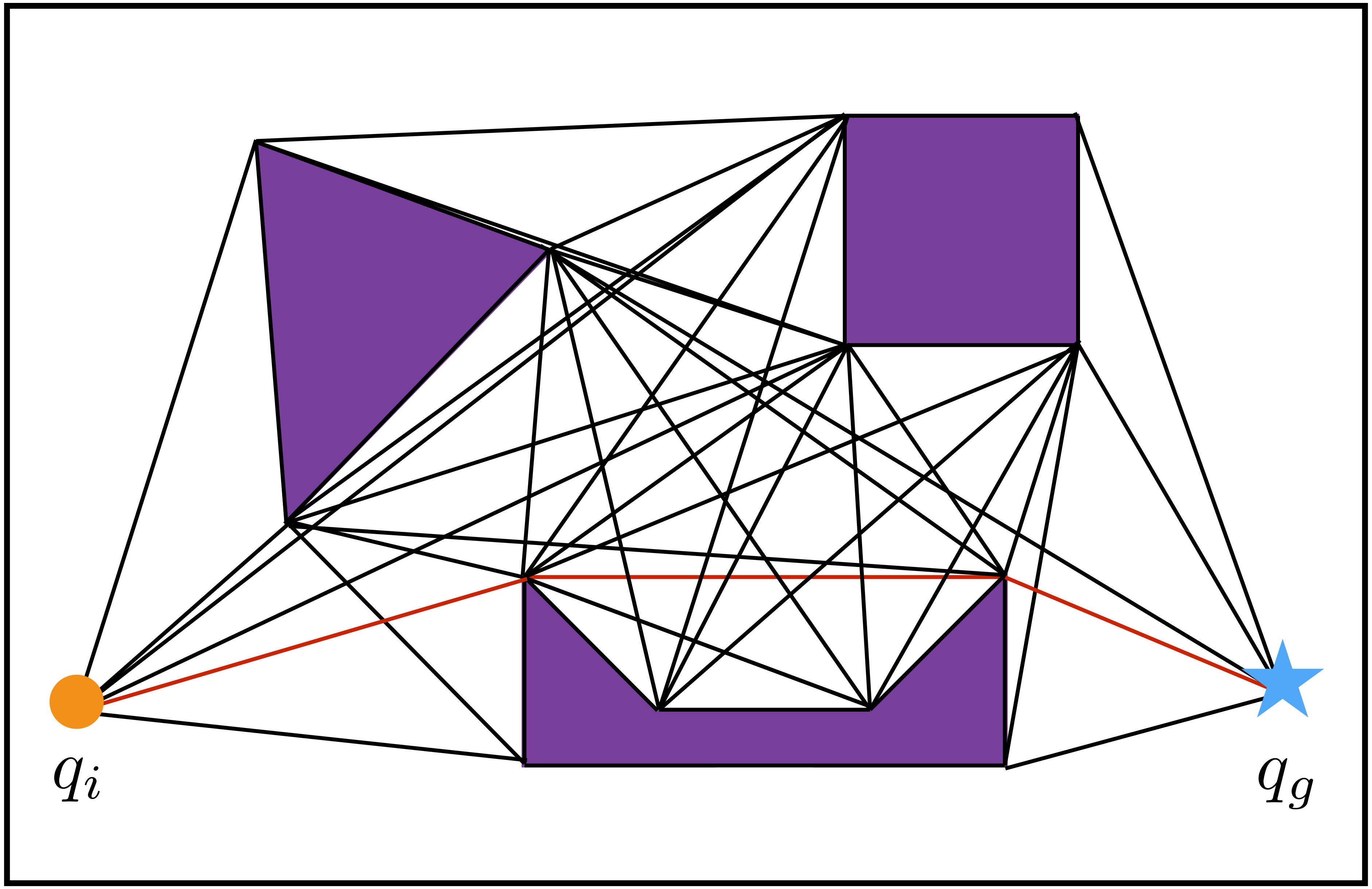
Visibility Graph

- Create graph by adding edges between visible corners



Visibility Graph

- Connect start and goal to visible corners



- This approach is complete and connects C_{free}

Combinatorial Planning

- Combinatorial methods for creating graphs:
  - Vertical decomposition
  - Approximate decomposition
  - Visibility graph

- Most of these methods are complete and capture C_{free}
- Intractable in higher dimensional c-spaces
- Require explicit C_{obs} representation
- Next time: probabilistic roadmaps

Questions?