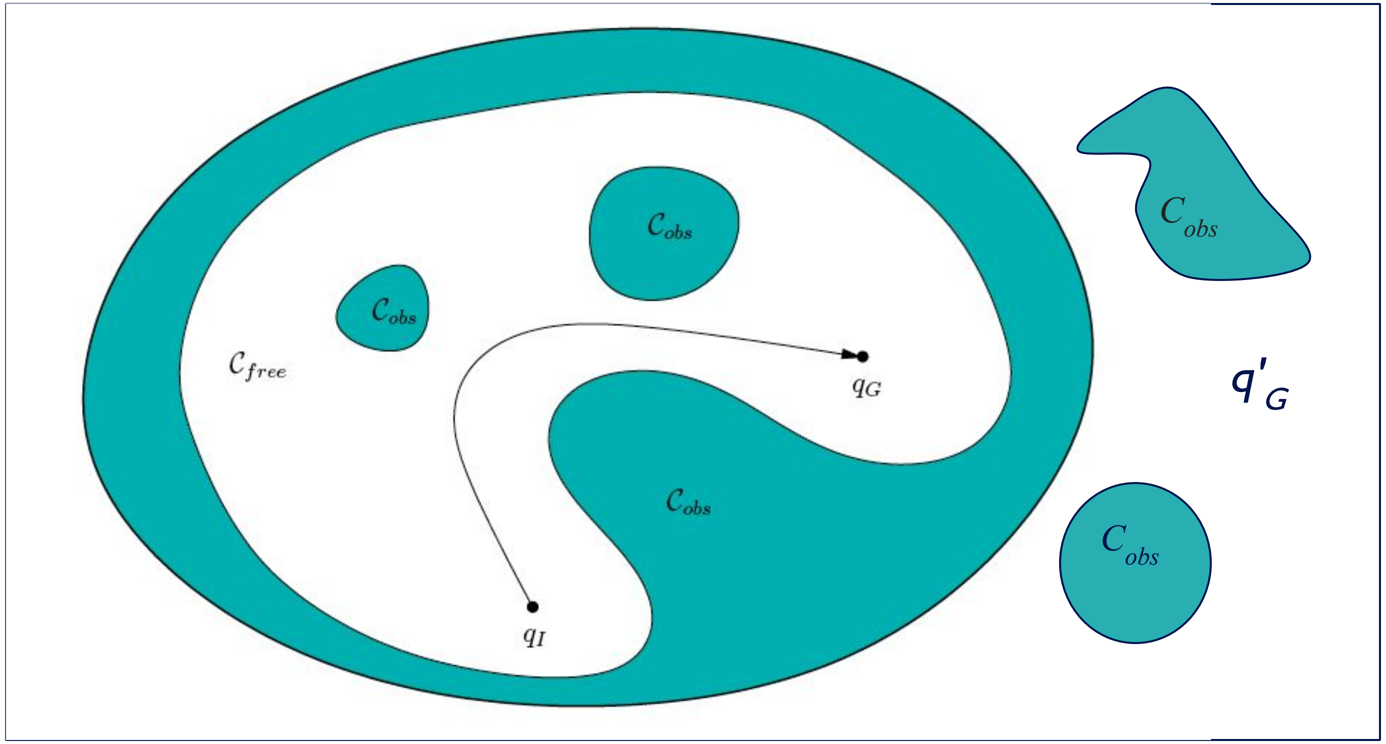
Graph Search Approaches to Planning

MEAM 520

Ariella Mansfield

The Basic Motion Planning Problem



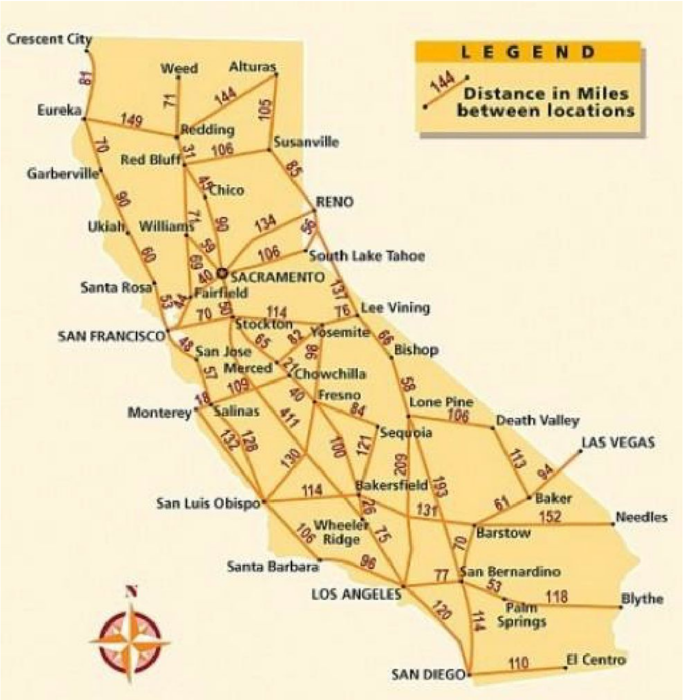
There exists a motion plan from q_I to q_G *iff* q_I and q_G belong to the same connected component of C_{free}

Adapted from Lavalle, 4.3.1

Motivating Examples

- Collision free trajectories through cluttered space

- Shortest path between points



Terminology and Notation

q **Configuration**
complete specification of the location of every point on the robot (via joint variables)

$\mathcal{Q}$ **Configuration Space**
set of all possible configurations considering only joint limits

$\mathcal{W}$ **Workspace**
Cartesian space in which robot moves

Terminology and Notation

$\mathcal{O}_i$ **Obstacles**

areas of the workspace that the robot should not occupy (physical objects or hazards)

Collision

when any part of the robot contacts an obstacle in the workspace

$\mathcal{A}(q)$ **Robot**

subset of the workspace occupied by the robot at configuration q

Terminology and Notation

$$\mathcal{O} = \cup \mathcal{O}_i$$

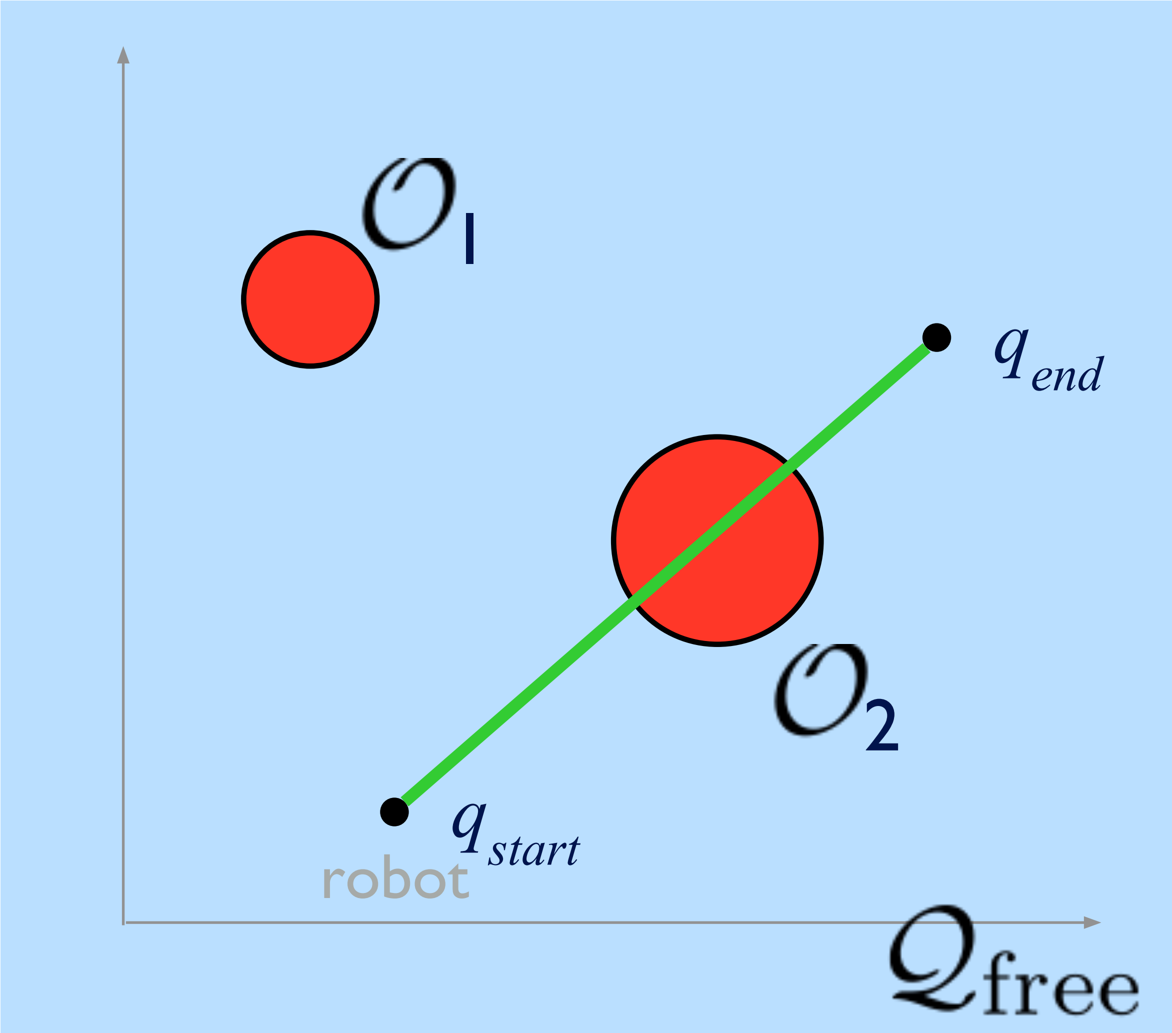
Configuration Space Obstacle
set of configurations for which the robot collides with an obstacle

$$\mathcal{QO} = \{q \in \mathcal{Q} \,|\, \mathcal{A}(q) \cap \mathcal{O} \neq \emptyset\}$$

Free Configuration Space
set of all collision-free configurations

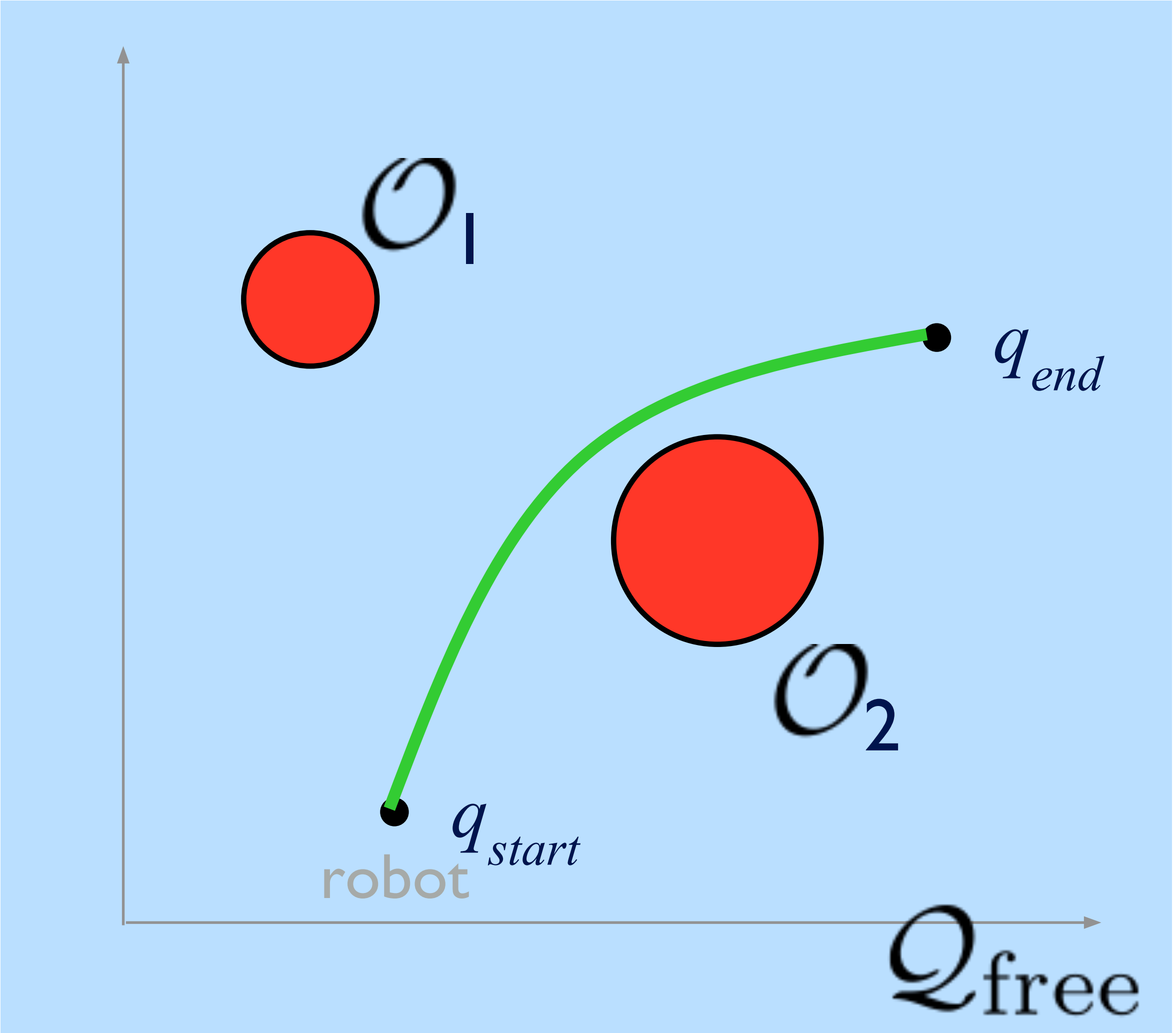
$$\mathcal{Q}_{\text{free}} = \mathcal{Q} \setminus \mathcal{QO}$$

Point Robot in 2D



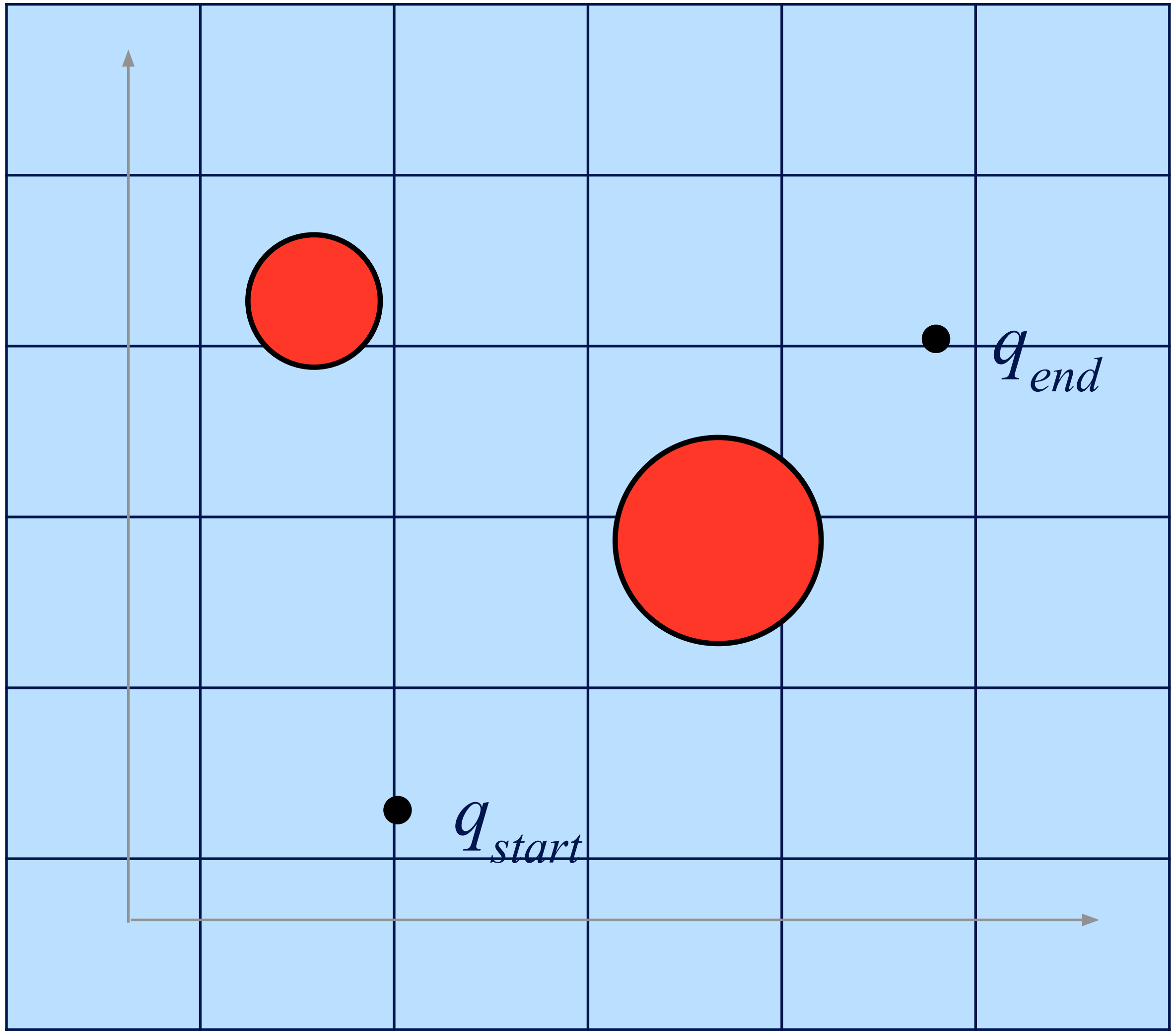
$$\mathcal{Q} = \mathcal{W} = \mathbb{R}^2$$

Point Robot in 2D



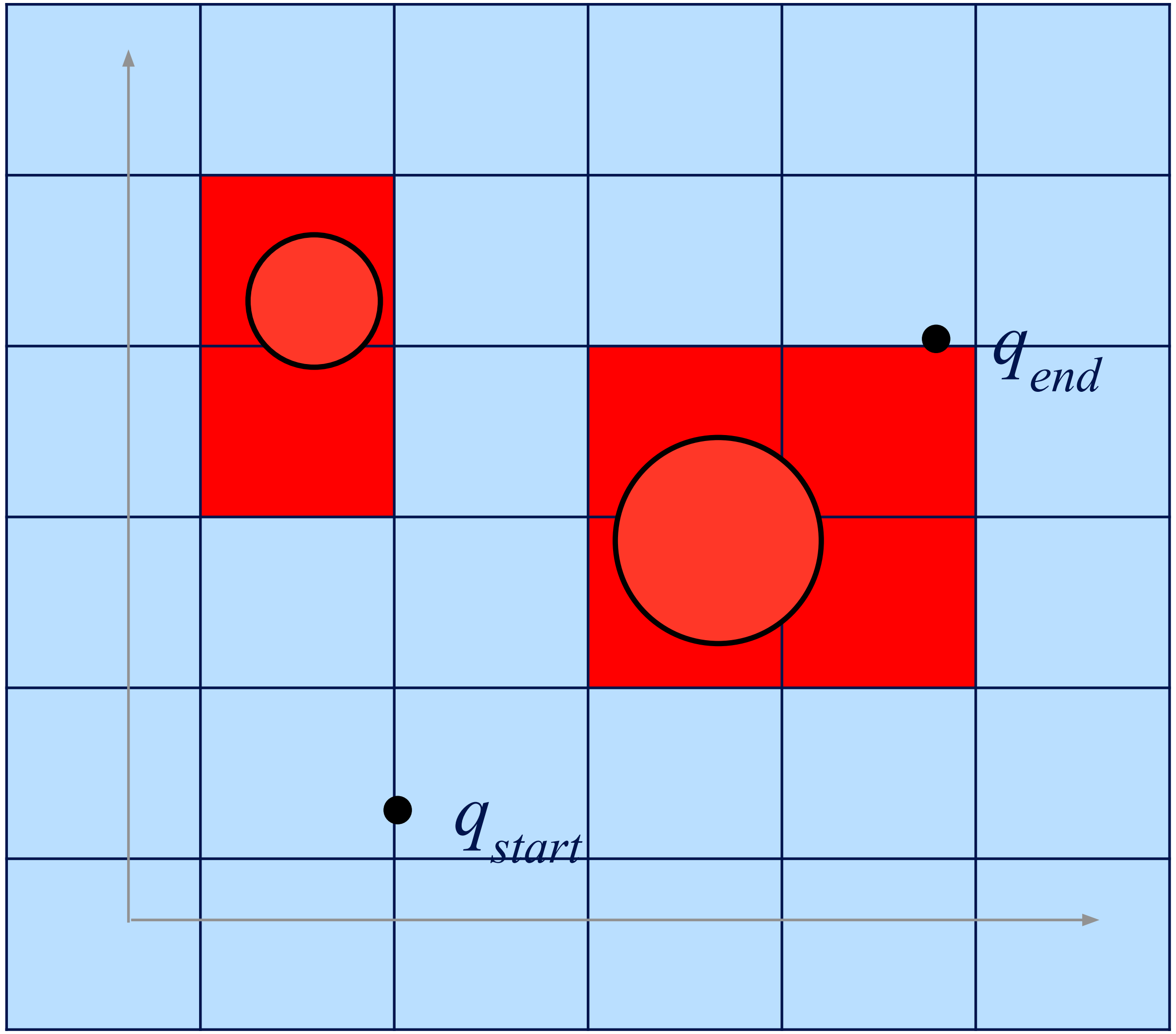
$$\mathcal{Q} = \mathcal{W} = \mathbb{R}^2$$

Discretize Space



n x n grid

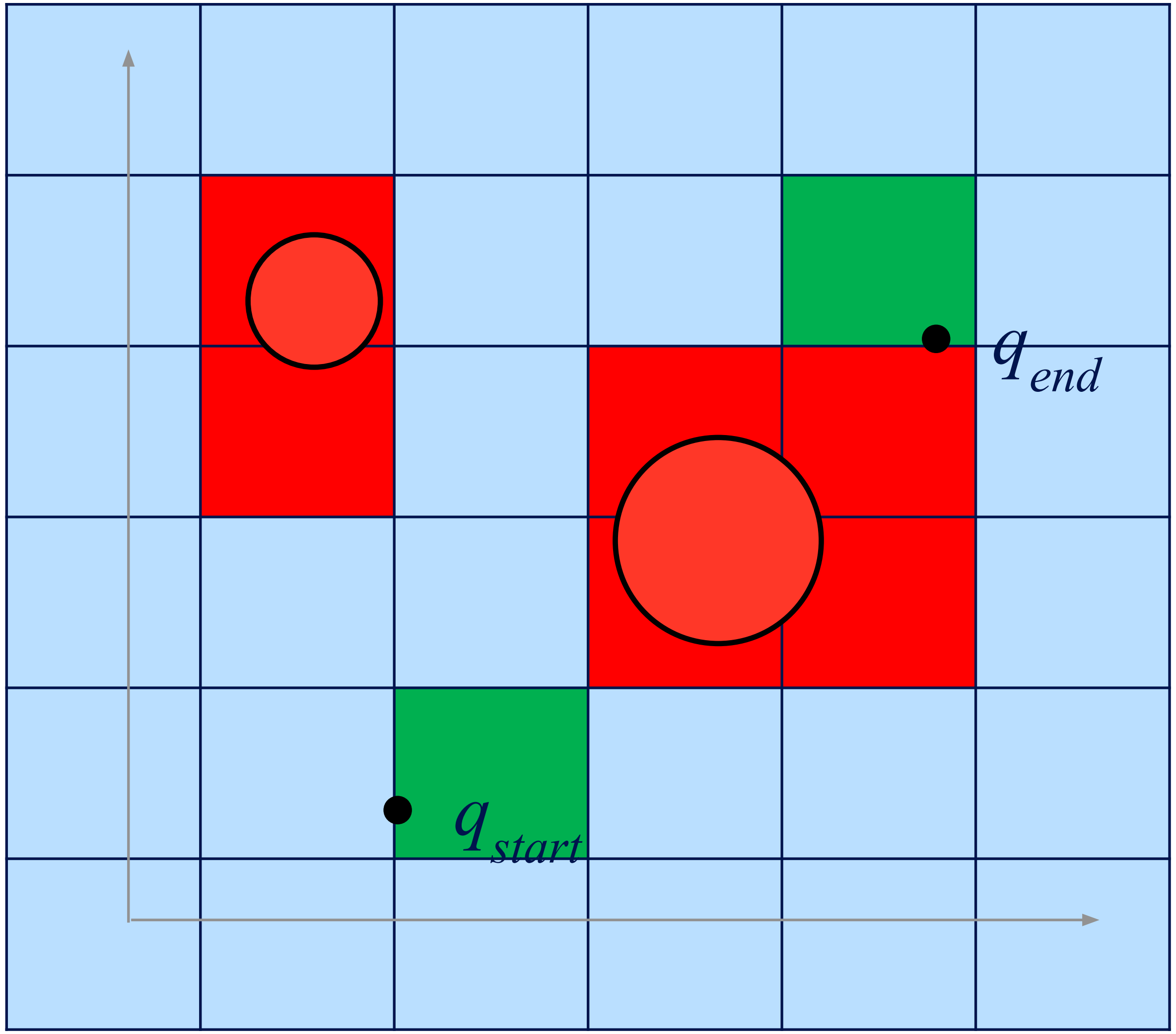
Discretize Space



n x n grid

Remove obstacles

Discretize Space



n x n grid

Remove obstacles

Find start and end cells

Wildfire

6	5	4	5	6	7
5		3	4	5	6
4		2			5
3	2	1			4
2	1	0	1	2	3
3	2	1	2	3	4

Pseudocode:

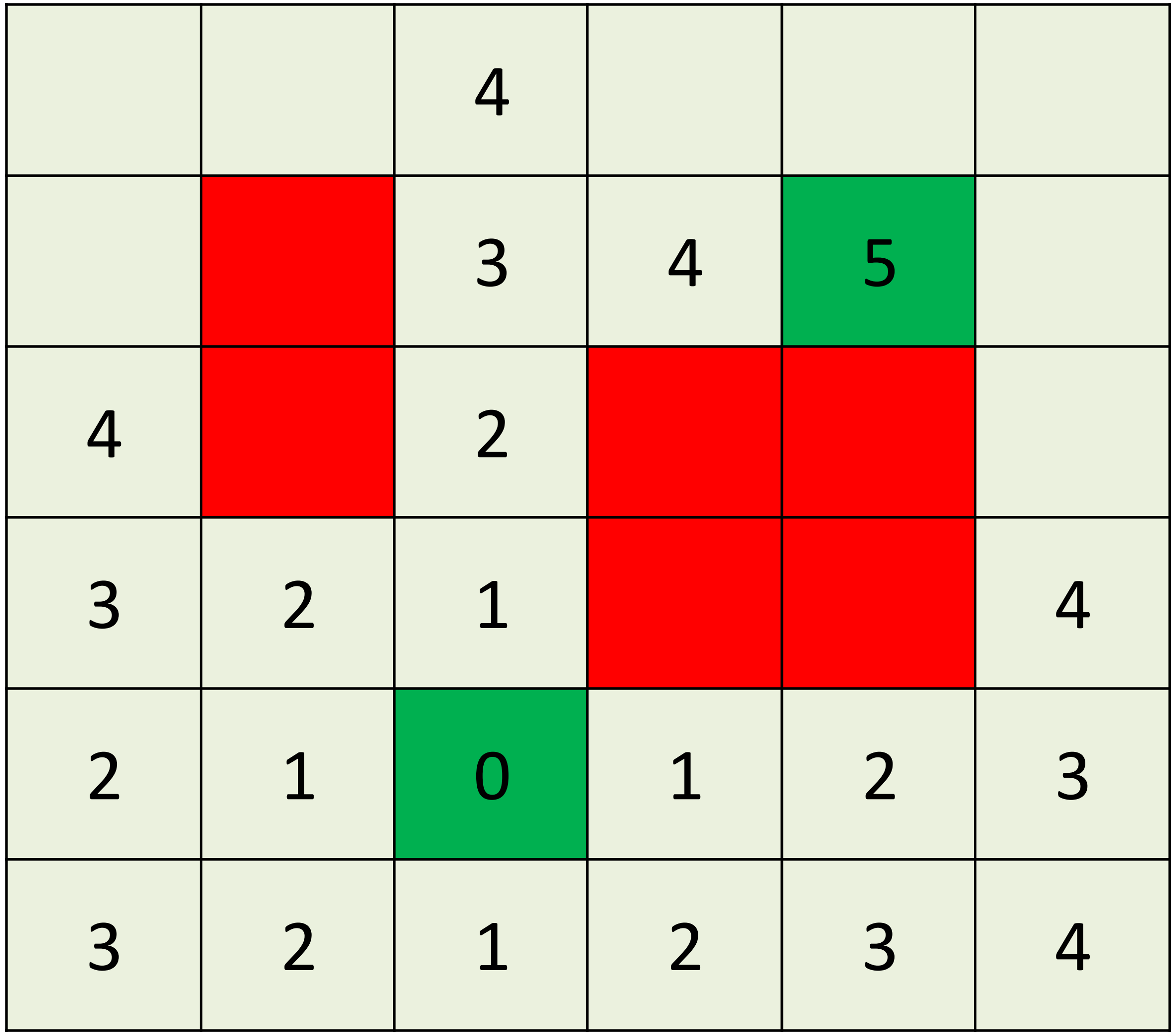
Start with i = 0 steps at q_{start}

While exist(empty cells)

- All neighbors have i+1 steps
- Ignore obstacle cells

Search all cells

Breadth First Search (BFS)



Pseudocode:

Start with i = 0 steps at q_{start}

Queue = neighbors of q_{start}

All neighbors have 1 step

While ~empty(*Queue*)

  q = next cell in *Queue*

  i = steps to q

  if a neighbor is q_{end}, STOP

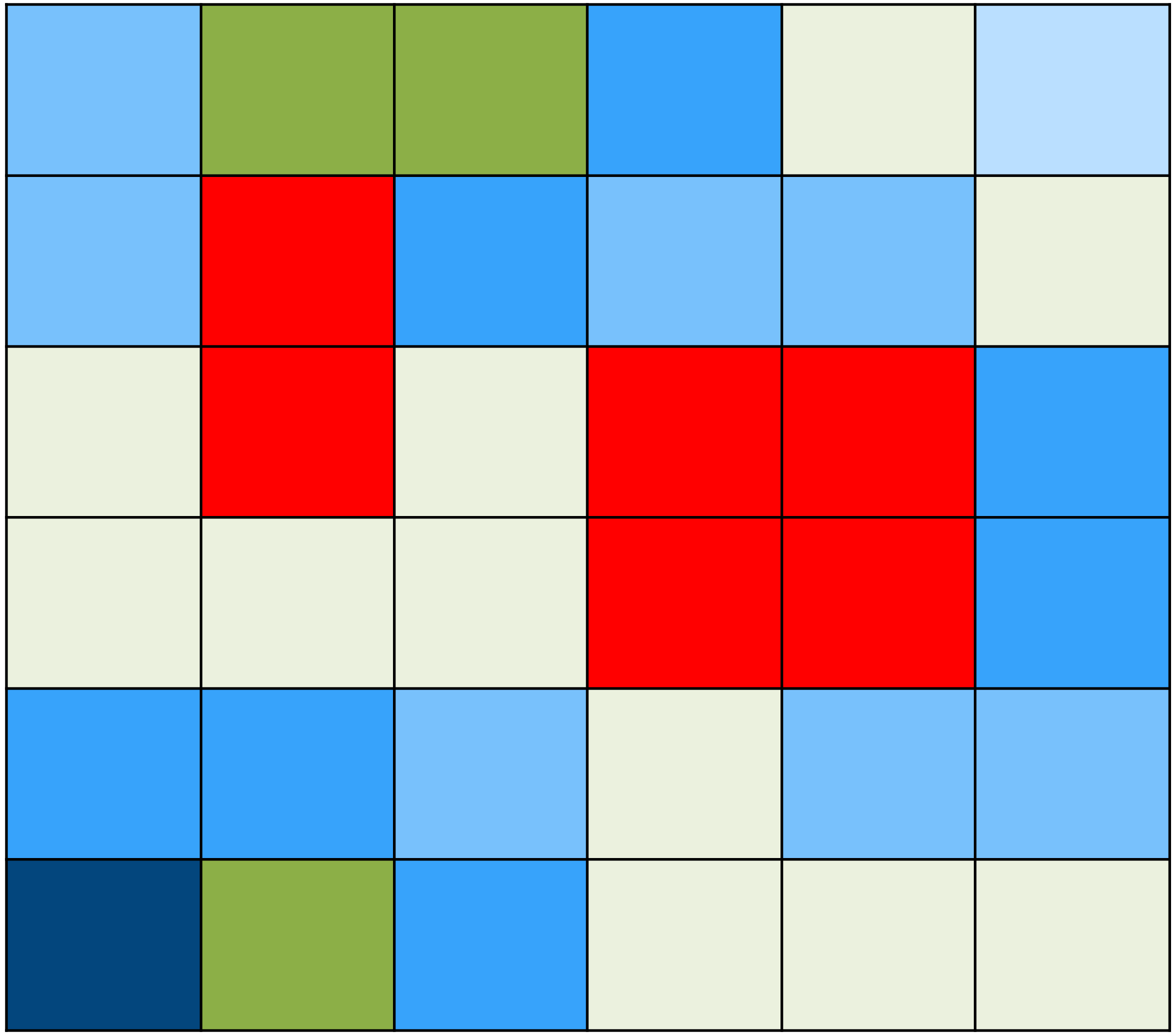
  Add all new neighbors to *Queue*

  All neighbors have i+1 steps

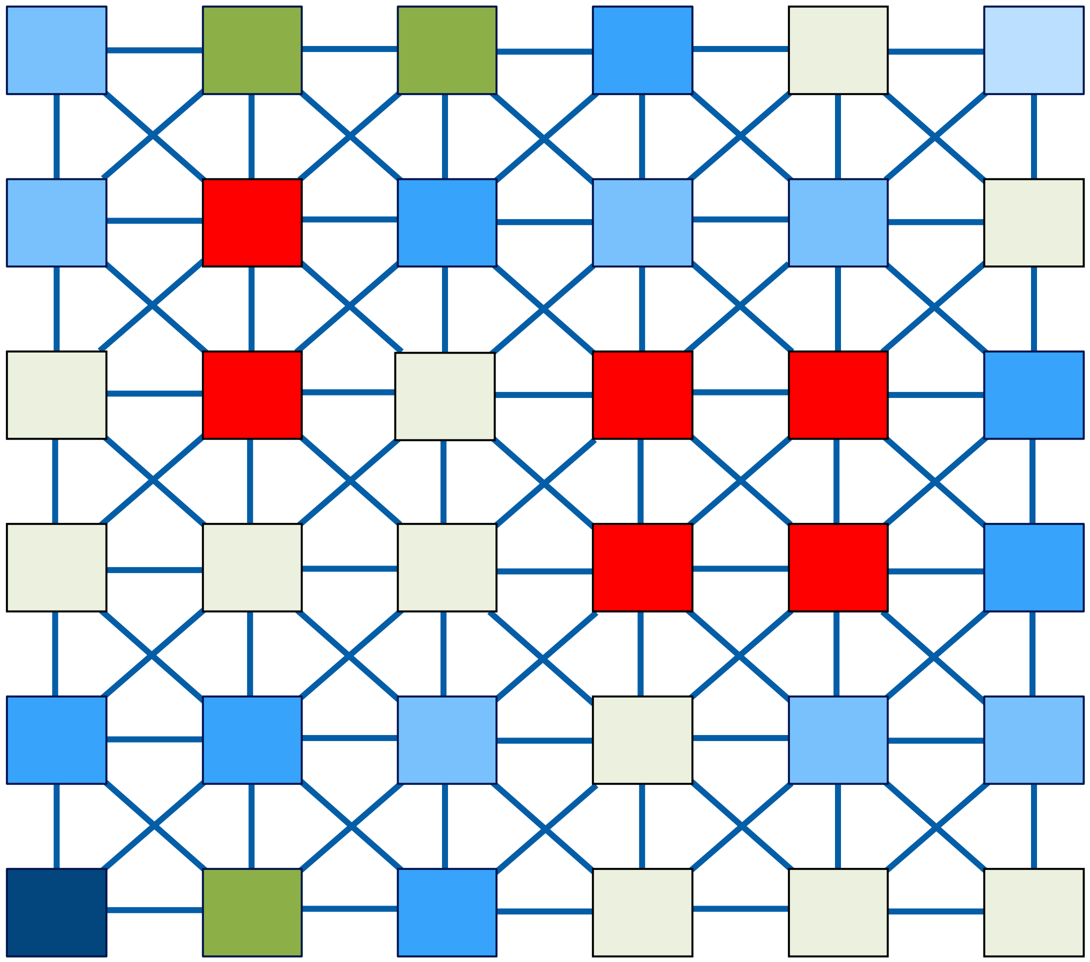
Potentially search all cells:

Computation is $O(|V|+|E|)$

Nonuniform costs



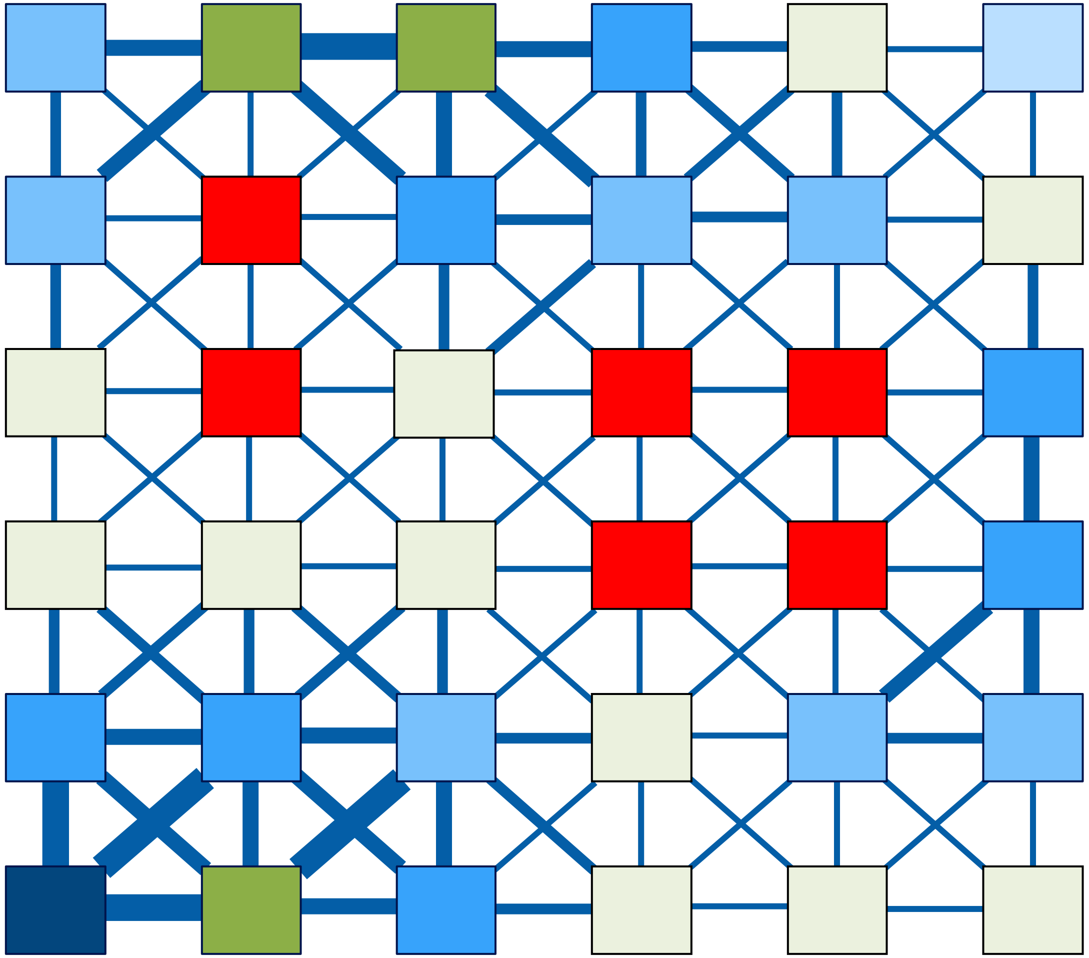
Graph Representation of the Configuration Space



Graph: vertices connected by edges

Penn Engineering

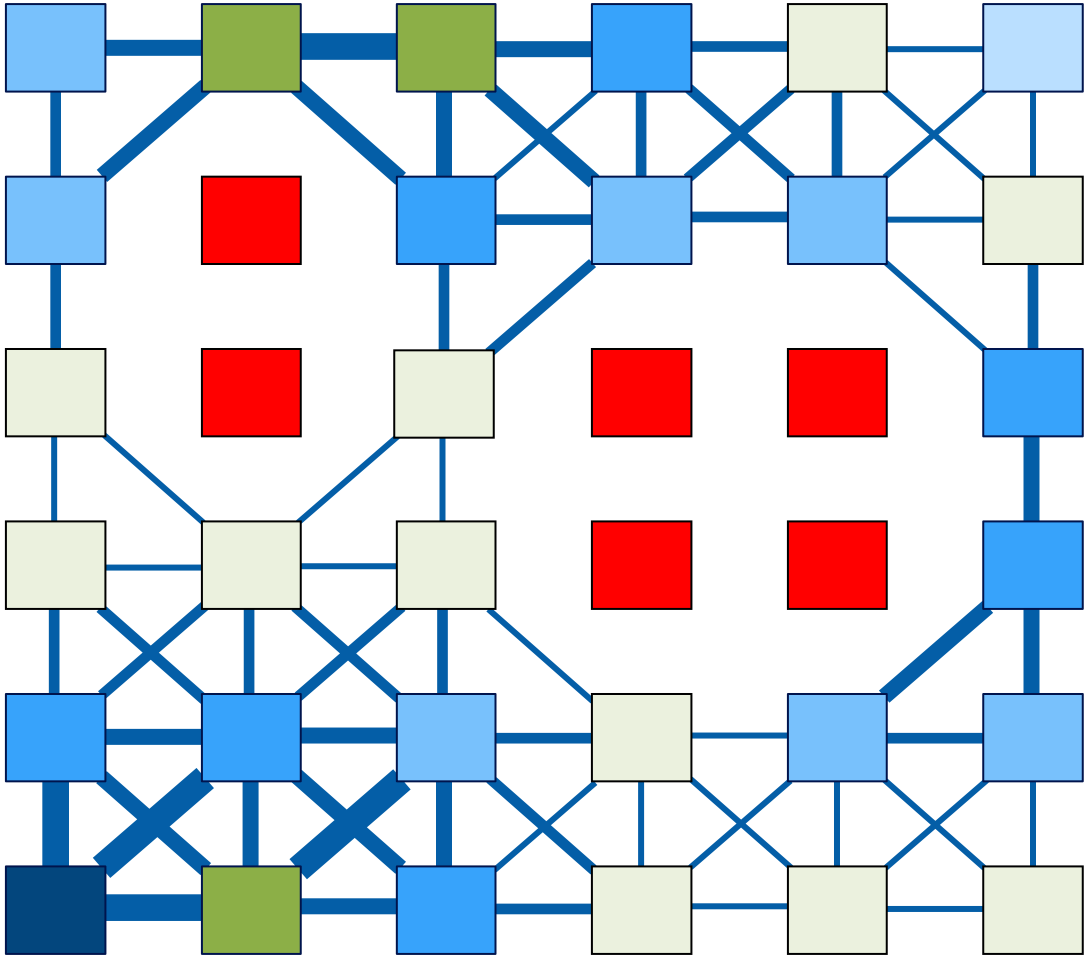
Graph Representation of the Configuration Space



Graph: vertices connected by edges

Assign costs

Graph Representation of the Configuration Space



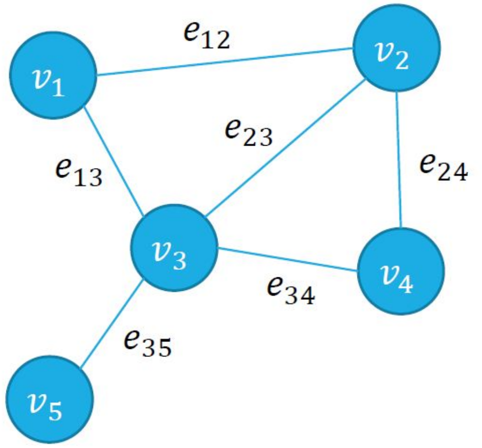
Graph: vertices connected by edges

Assign costs

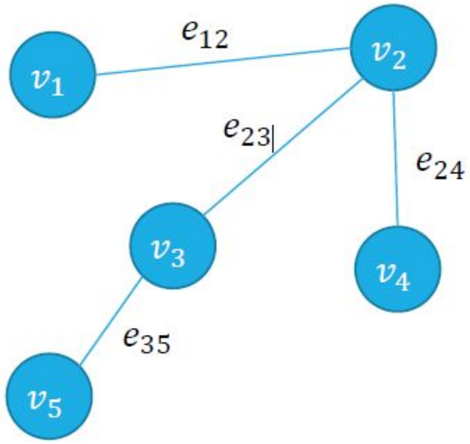
Remove edges to obstacles

Graph

- A **graph** is an ordered pair $G = (V, E)$, where V is a set of vertices or nodes and E is a set of edges
- An **edge** is a 2-tuple of vertices
  - Edges can be directed or undirected
  - Edges can be weighted (e.g. distance)
    - Edge e_{ij} has weight w_{ij}
- A **tree** is an undirected graph without cycles (only 1 path between 2 vertices)



Graph that is not a tree

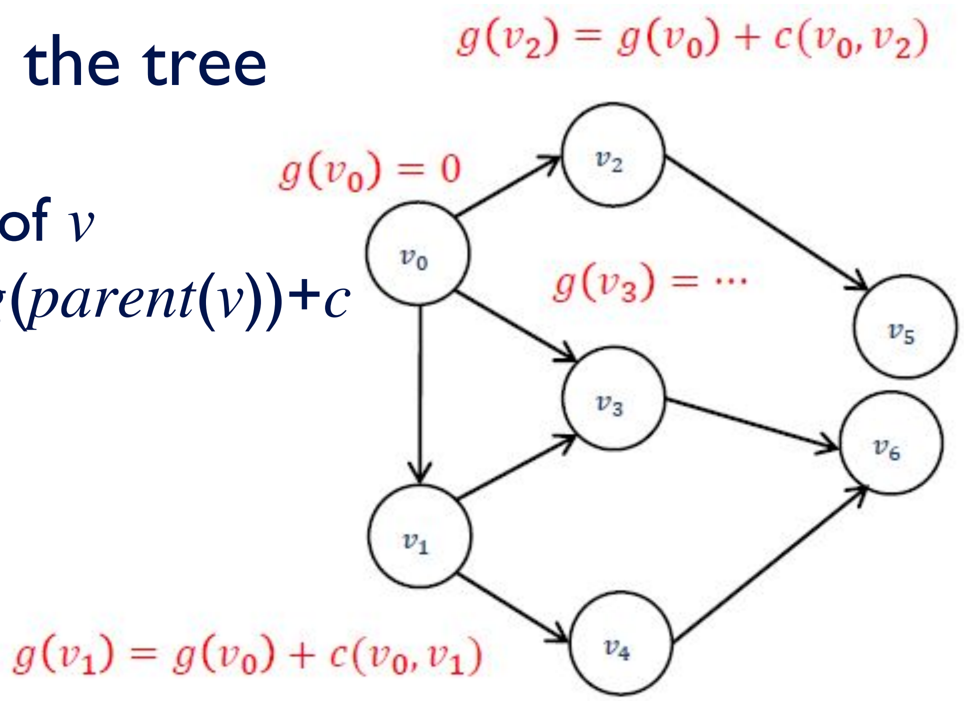


Graph that is a tree

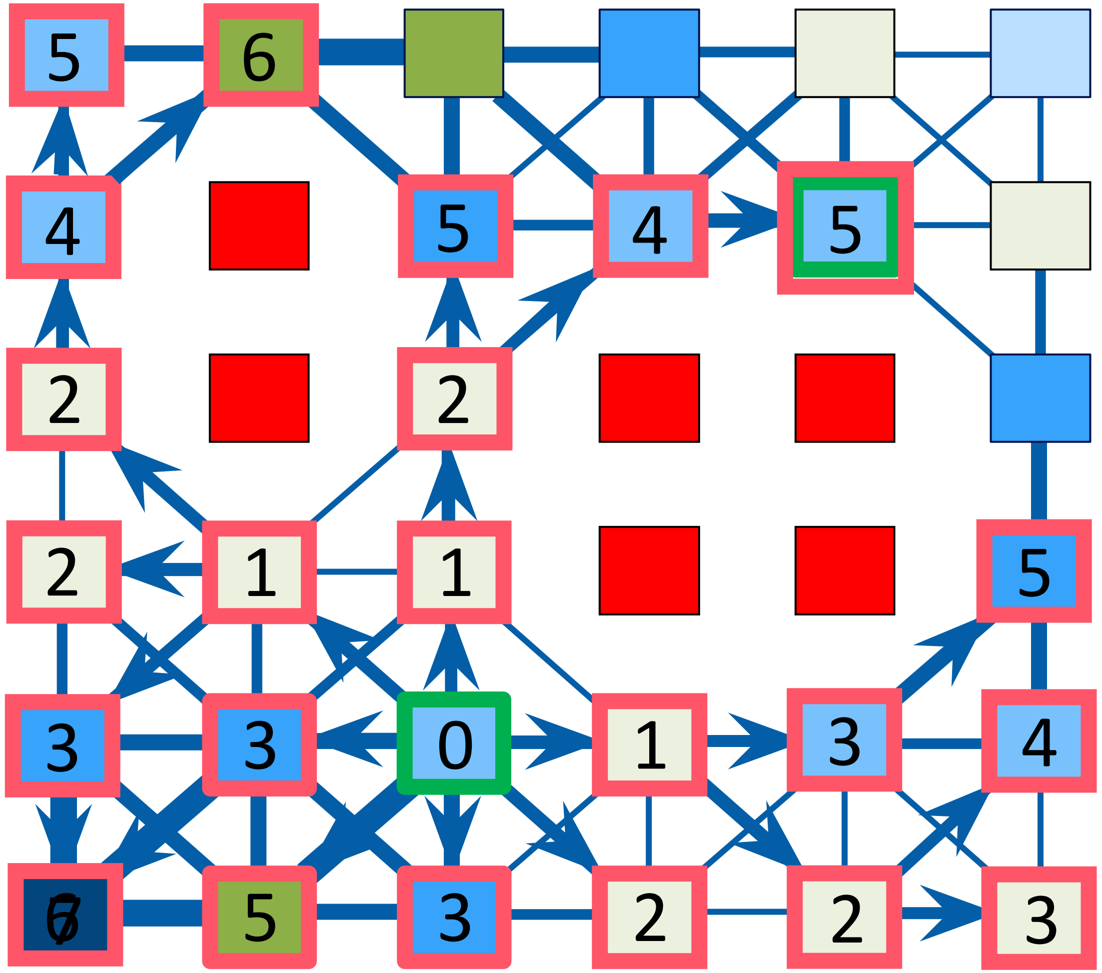
General Tree-Based Search

- General search strategy for finding a path from start to goal, and keeping track of it's length given edge costs $c(v_1, v_2)$.

1. Set the root of the tree as the start state and give it a value of 0
2. While there are unexpanded nodes in the tree
    1. Choose a leaf v to expand
    2. For each action, create a new child leaf of v
    3. Set the value of each child leaf as $g(v)=g(parent(v))+c(parent(v,v))$



Dijkstra's Algorithm



Pseudocode:

Start with i = 0 steps at q_{start}

Add neighbors of q_{start} to *boundary*

Update costs of neighbors

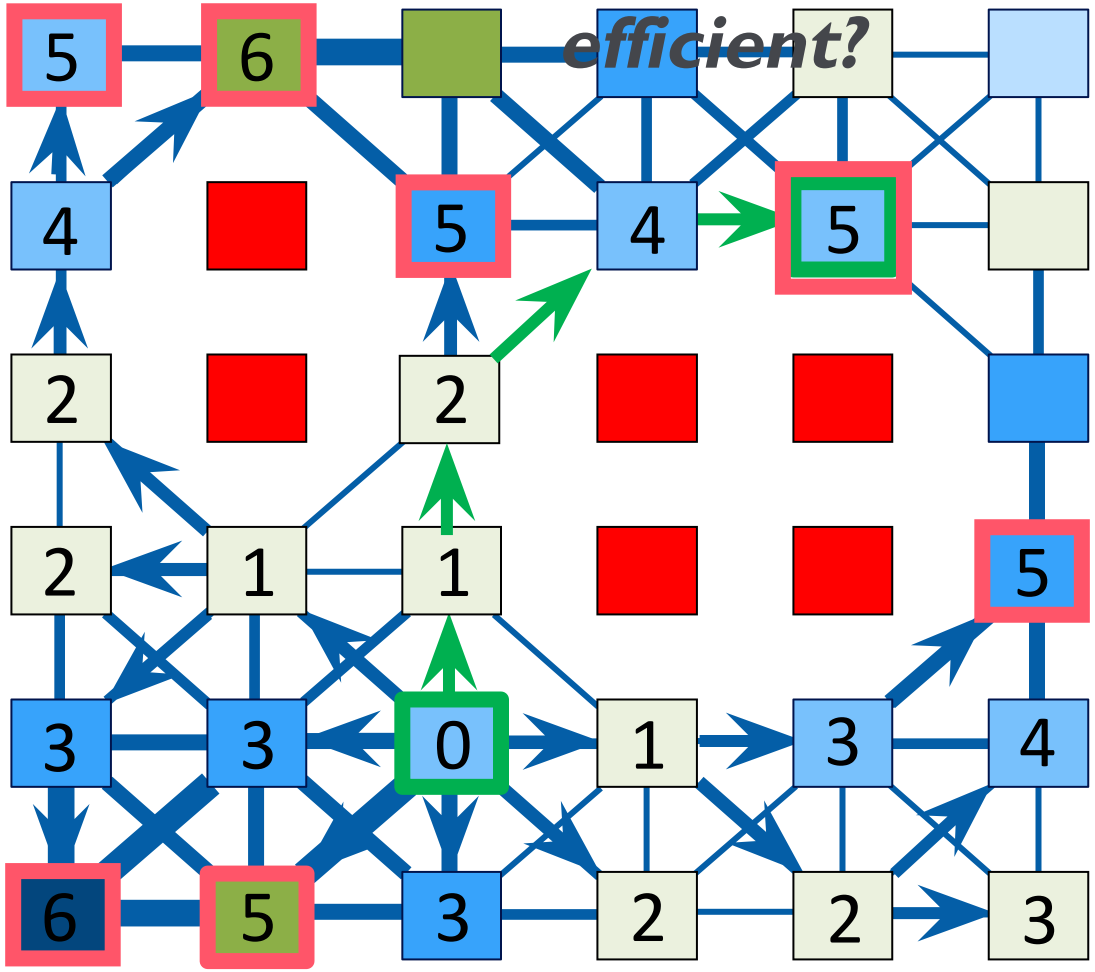
While ~empty(*boundary*)

- q = *boundary* cell with min cost
- Add all new neighbors to *boundary*
- Update costs of new neighbors
- Remove q from *boundary*
- If a neighbor is q_{end}, STORE
- If mincost(*boundary*) ≥ cost(q_{end}), STOP

Potentially search all cells:
Computation is $O(|V|log|V|+|E|)$

Dijkstra's Algorithm

Can we make this more efficient?



Pseudocode:

Start with i = 0 steps at q_{start}

Add neighbors of q_{start} to *boundary*

Update costs of neighbors

While ~empty(*boundary*)

 q = *boundary* cell with min cost

 Add all new neighbors to *boundary*

 Update costs of new neighbors

 Remove q from *boundary*

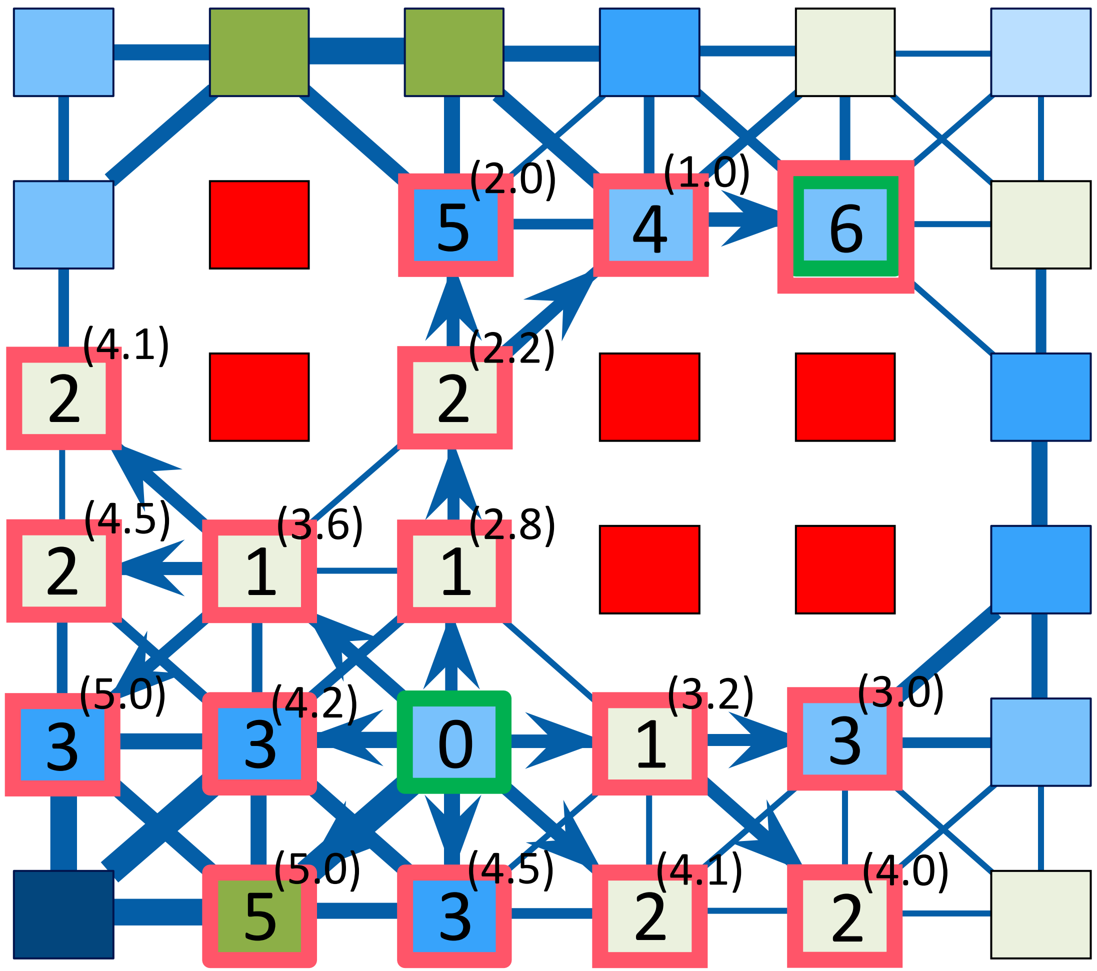
 If a neighbor is q_{end}, STORE

 If mincost(*boundary*) ≥ cost(q_{end}), STOP

Potentially search all cells:
Computation is $O(|V|log|V|+|E|)$

Penn Engineering

A* Search



- **Idea**: estimate remaining distance to the goal

  Order vertices based on estimated distance

$$f(i) = \underbrace{g(i)}_{\text{cost to come from start}} + \underbrace{h(i)}_{\text{heuristic: estimated cost to go to goal}}$$

Dijkstra's: $h(i) = 0$

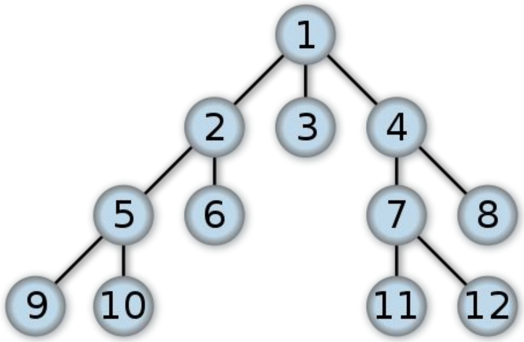
Let's try $h(i) =$ Euclidean distance to goal

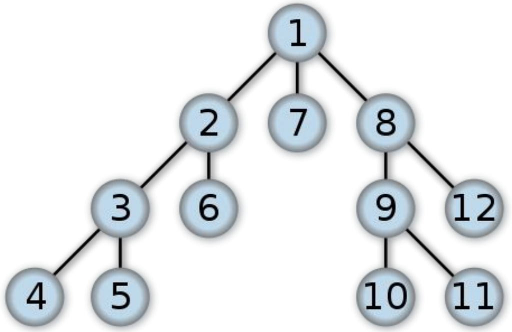
$h(i)$ must be **admissible**

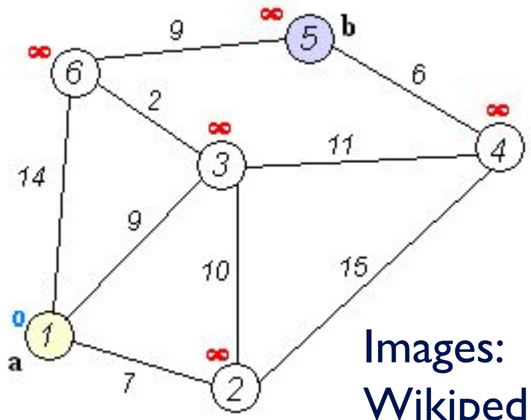
Worst case computational cost?

Comparison of Strategies

- Breadth First
  - Choose shallowest next, returns optimal path (when uniform edge weights)
- Depth First
  - Choose deepest next, first returned path may not be optimal
- Best First
  - E.g. Dijkstra (1959), A* (Hart 1968)
  - Choose "most promising" node next based on some rule



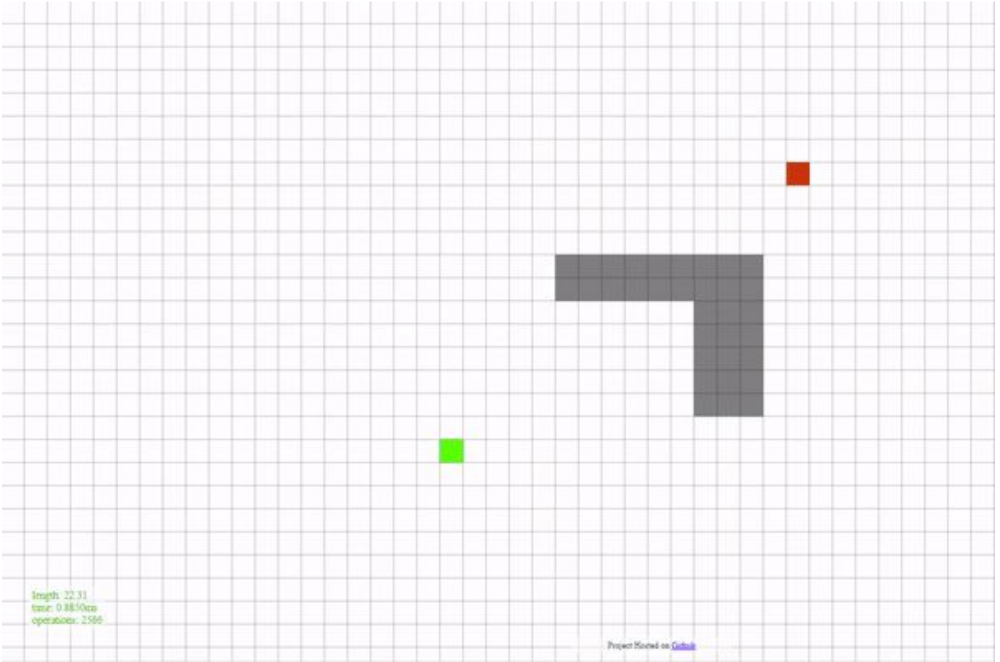




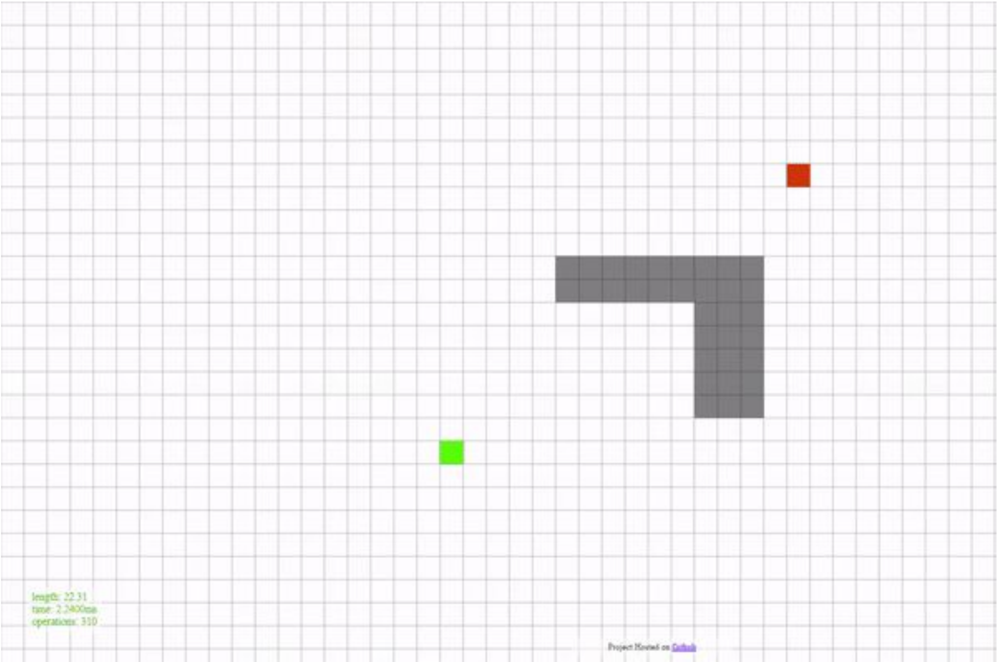
Images: Wikipedia

Dijkstra vs. A*

- Dijkstra



- A*



https://qiao.github.io/PathFinding.js/visual/

- Computational complexity of a trajectory planner grows with the size of the configuration space.

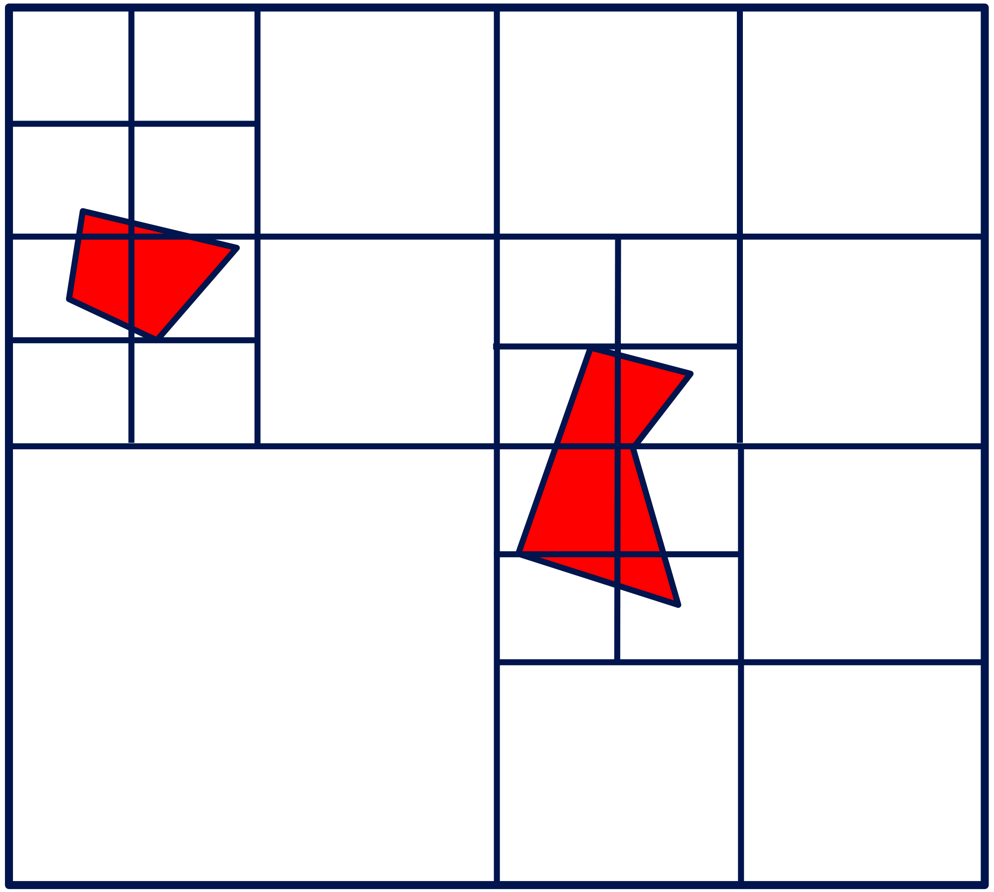
Complete planners have to search every cell of the discretized space in the worst case.

Worst case complexity is **exponential** in the robot dof (number of joints for a manipulator):

$$O(c^J)$$

Penn Engineering

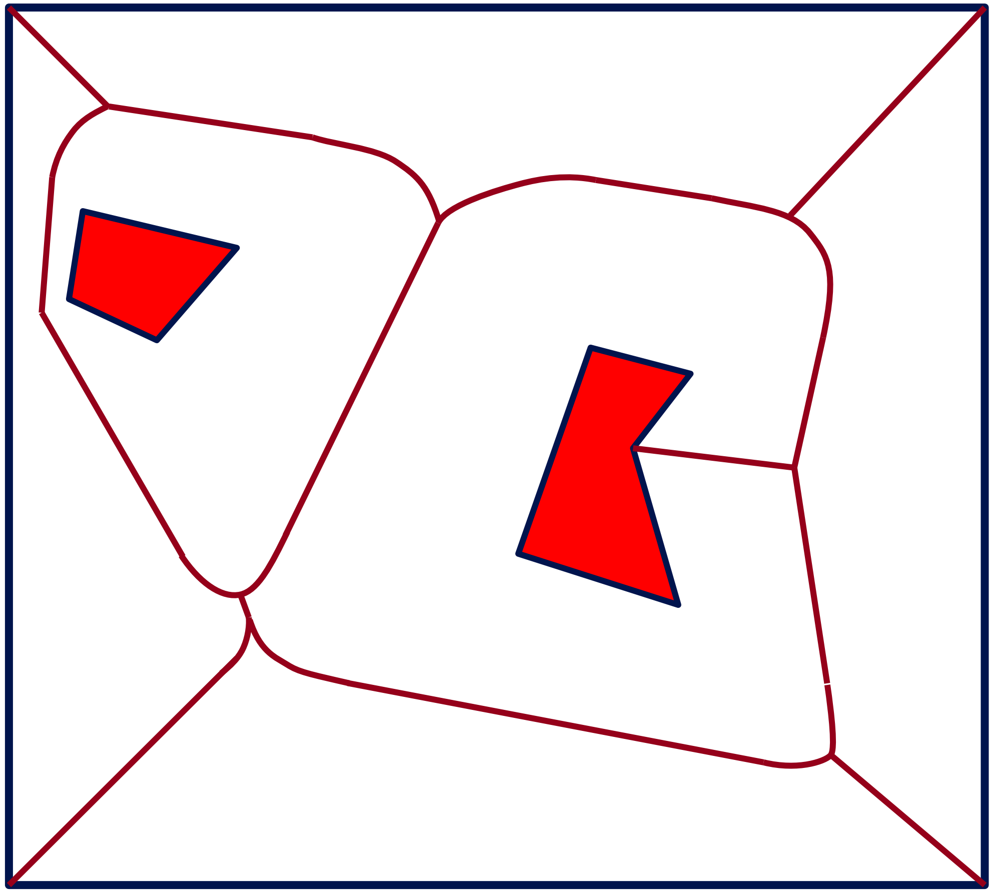
Can we do better?



Idea: Discretize only as much as necessary

This will depend on the number and geometric complexity of your obstacles

Can we do better?



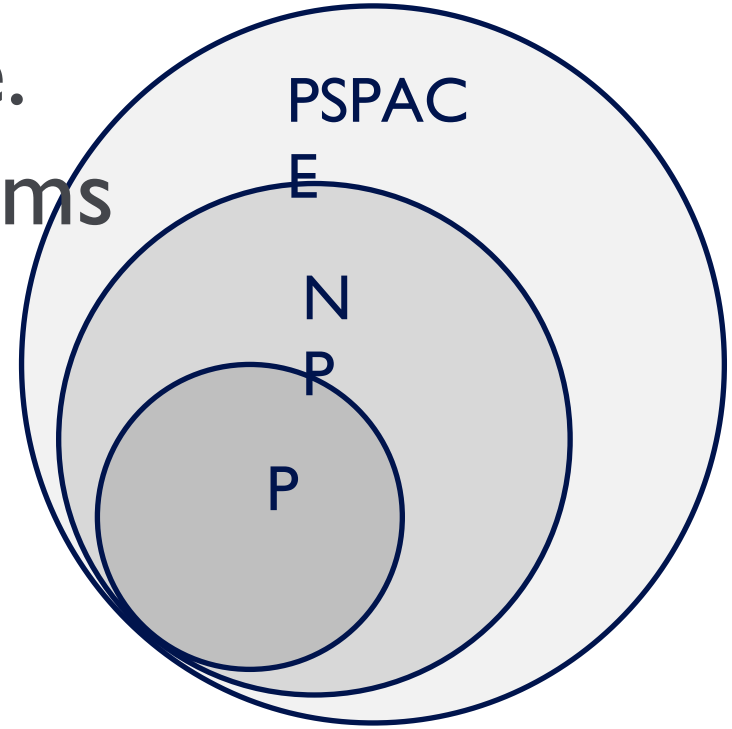
Idea: Map out the free space

This is called the Voronoi Diagram

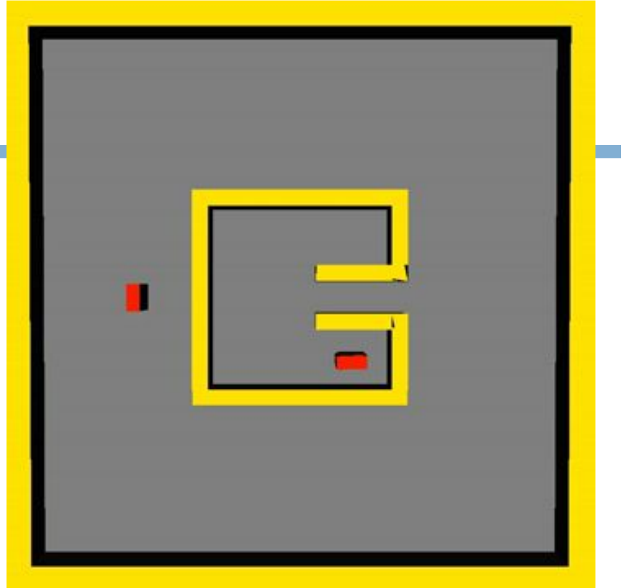
Can we do better?

Theoretically, no.

General motion planning is in a class of problems we call PSPACE-complete. These are some of the hardest problems in computer science.

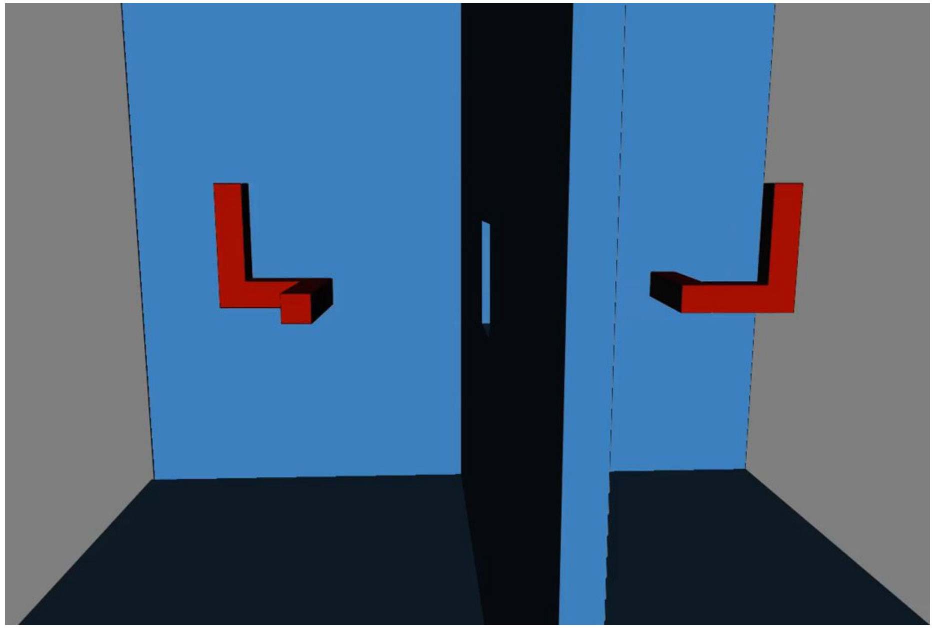


What makes planning hard?



https://vimeo.com/58686591

https://www.youtube.com/watch?v=UTbiAu8IXas



Complex obstacles
Narrow corridors in the free C-space

CHALLENGE: Map out the free C-Space

https://vimeo.com/58709589