AA 274
Principles of Robotic Autonomy

Combinatorial motion planning

Stanford University

Autonomous Systems Lab

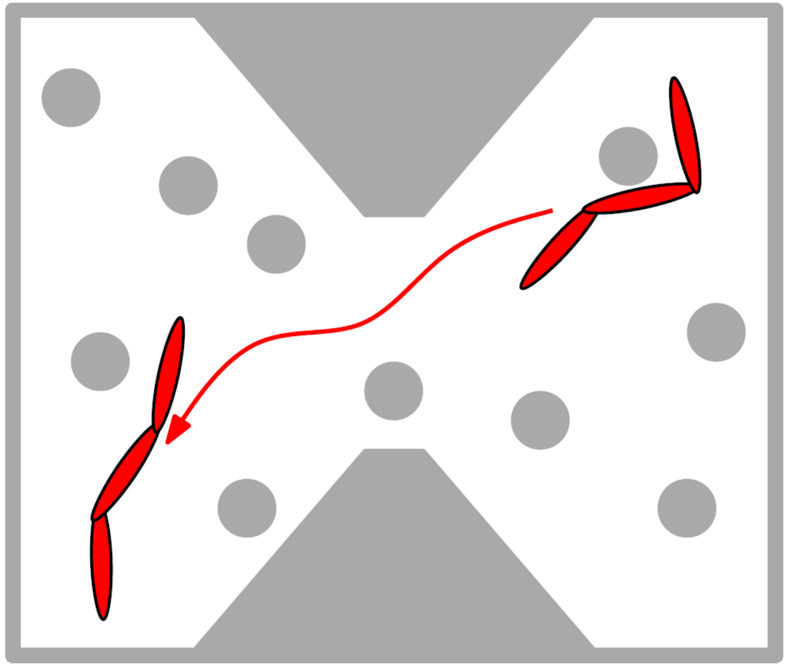
Today's lecture

- Aim
  - Introduction to motion planning & specifically, combinatorial motion planning

- Readings:
  - Bertsekas, *Dynamic Programming and Optimal Control, Vol I, 3rd ed.,* Section 2.3
  - LaValle, *Planning Algorithms*, Sections 6.1-6.3, 6.5

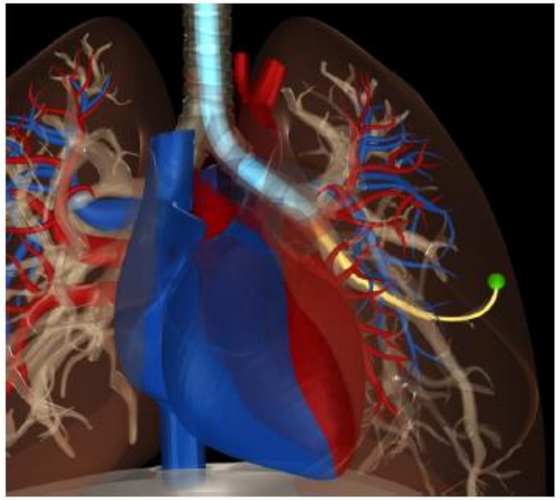
Robot Motion Planning

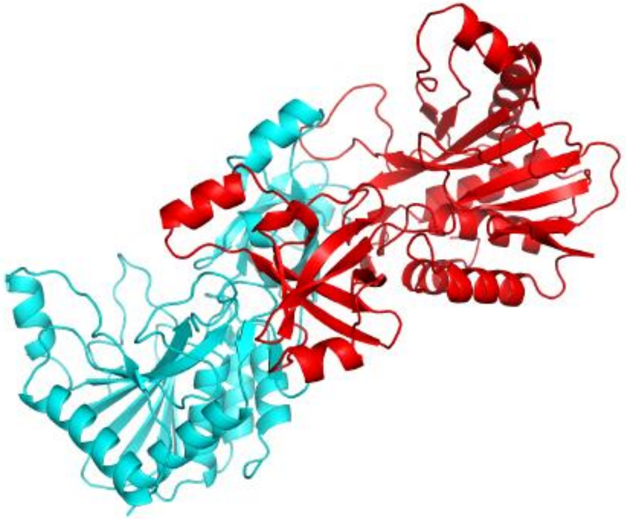
- Robot motion planning: compute a sequence of actions that drives a robot from an initial condition to a terminal condition while avoiding obstacles, respecting kinematic/dynamical constraints, and possibly optimizing an objective function



More examples of motion planning

- Game puzzles
- Steering autonomous vehicles
- Controlling humanoid robot
- Protein docking
- Surgery planning
- …





Some History

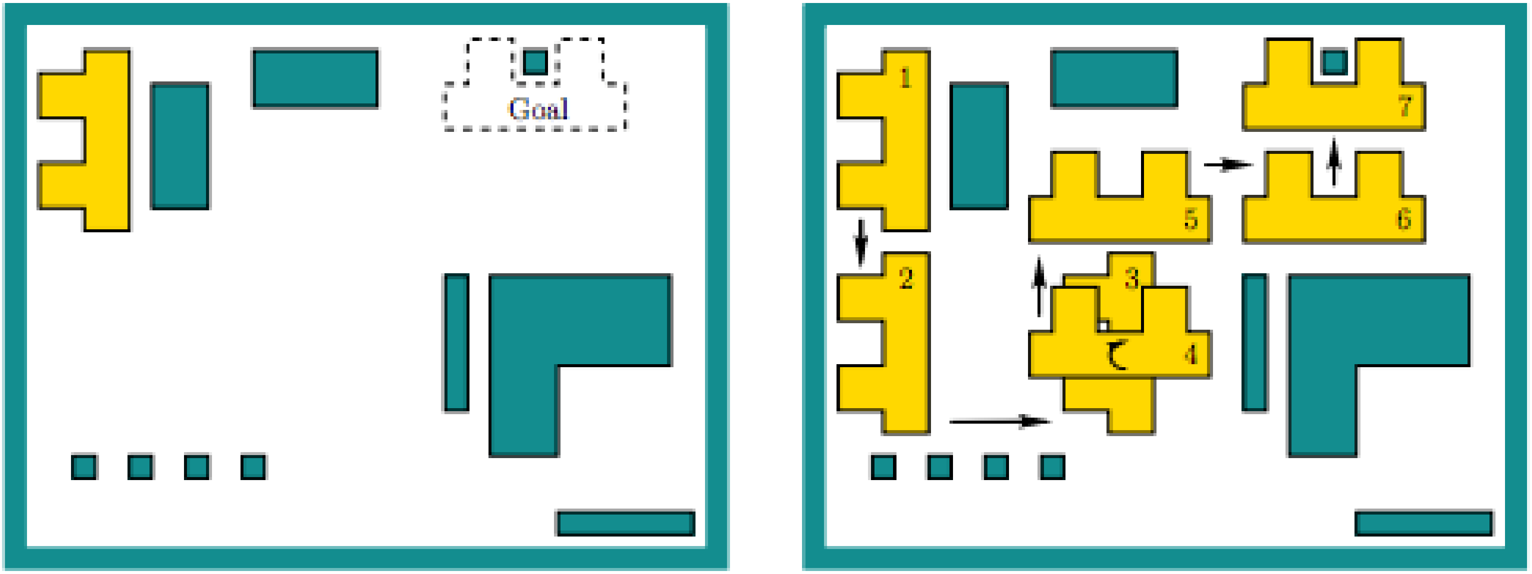
- Formally defined in the 1970s
- Development of exact, combinatorial solutions in the 1980s
- Development of sampling-based methods in the 1990s and deployment on real-time systems in the 2000s
- Current research: inclusion of differential constraints, planning under uncertainty, parallel implementation, feedback plans

Popular Methods:

- Potential functions [Rimon, Koditschek, '92]
- Grid-based search (A*, D*) [Stentz, '94]
- Geometric algorithms (visibility graphs, cell decomposition) [LaValle, '06]
- Sampling-based [Kavraki et al, '96; LaValle, Kuffner, '06,]

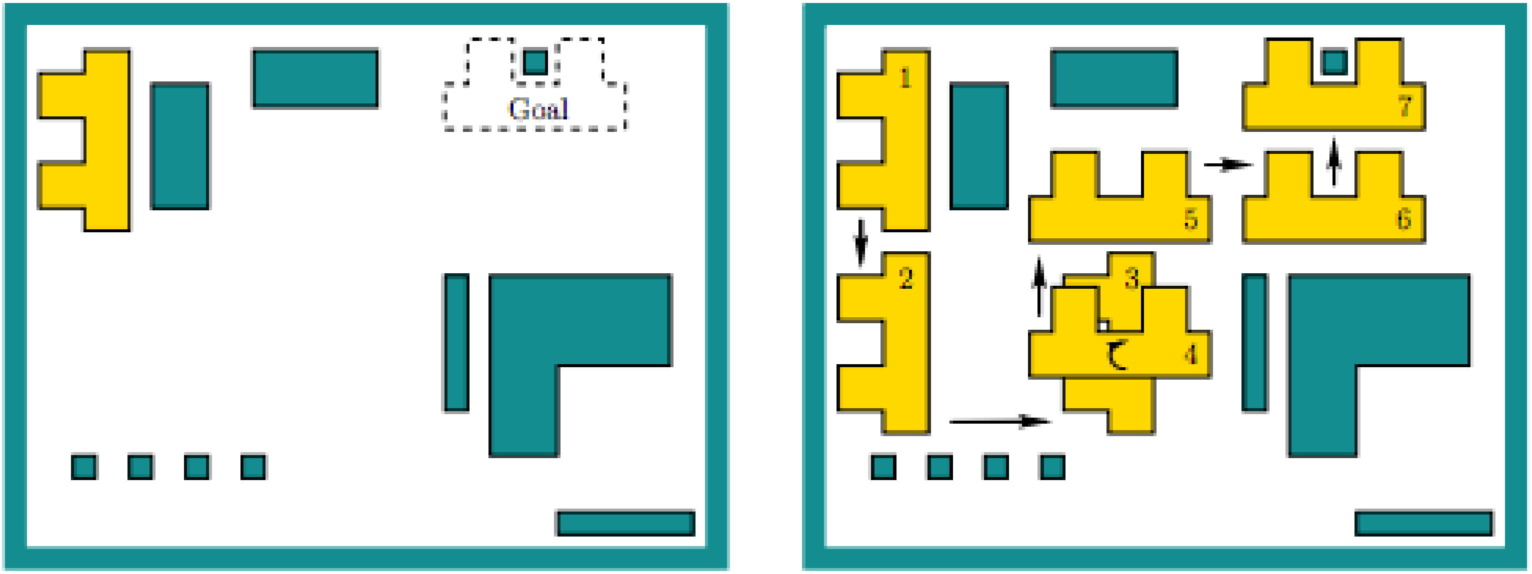
Simplest Setup

- Assume 2D workspace: $\mathcal{W} \subseteq \mathbb{R}^2$
- $\mathcal{O} \subset \mathcal{W}$ is the obstacle region with polygonal boundary
- Robot is a rigid polygon
- Problem: Given initial placement of robot, compute how to gradually move it into a desired goal placement so that it never touches the obstacle region



Simplest Setup

Key point: motion planning problem described in the real-world, but it really lives in a another space -- the configuration (*C*-) space!



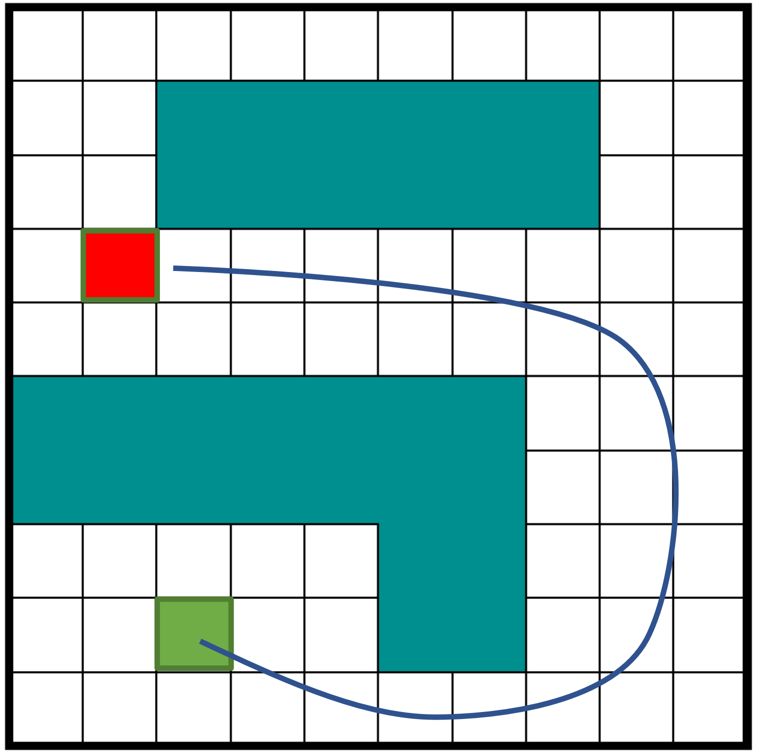
Solution Approaches

Three main approaches to motion planning:

- *Grid-based planning:* Discretize problem into grid and run a graph-search algorithm (Dijkstra, A*, …)
- *Combinatorial planning:* constructs structures in the *C*-space that discretely and completely capture all information needed to perform planning
- *Sampling-based planning:* uses collision detection algorithms to probe and incrementally search the C-space for a solution, rather than completely characterizing all of the C_{free} structure

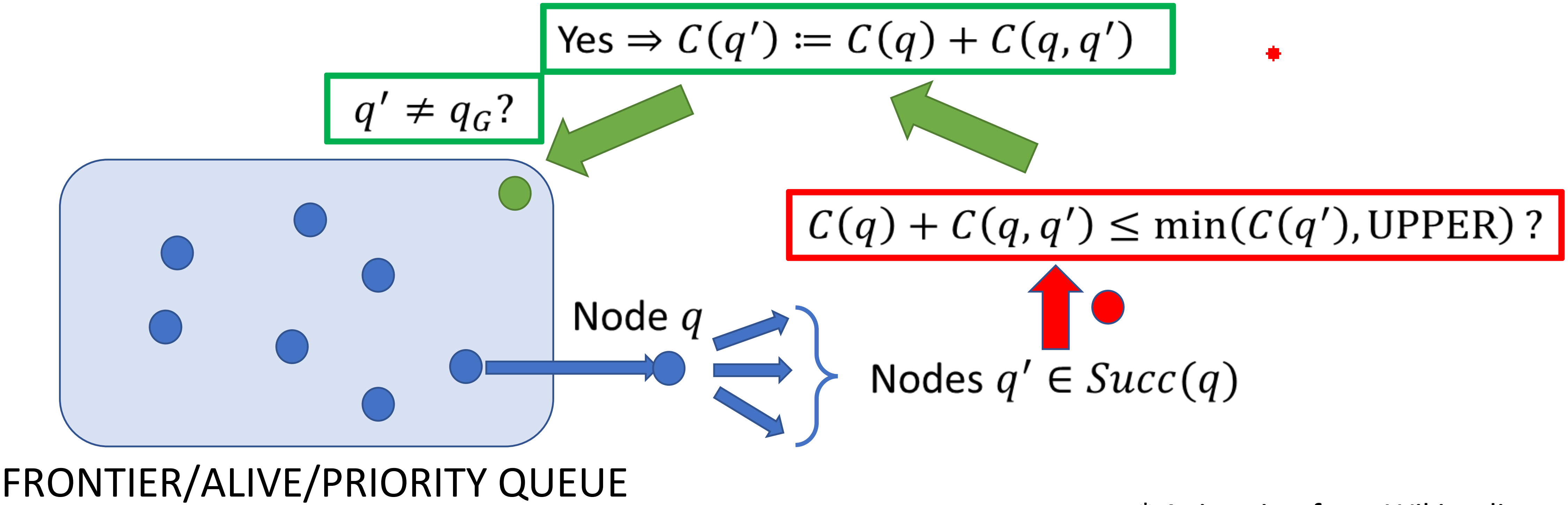
Grid-based approaches

- Discretize the continuous world into a grid
  - Each grid cell is either free or forbidden
  - Robot moves between adjacent free cells
  - **Goal:** find a sequence of free cells from start to goal
- Mathematically, this corresponds to pathfinding in a discrete graph $G = (V, E)$
  - Each vertex $v \in V$ represents a free cell
  - Edges $(v, u) \in E$ connect adjacent grid cells



Graph Search Algorithms

- Having determined decomposition, how to find "best" path?
- **Label-Correcting Algorithms**: $C(q)$: *cost-of-arrival* from q_I to q



* Animation from Wikipedia

Label Correcting Algorithm

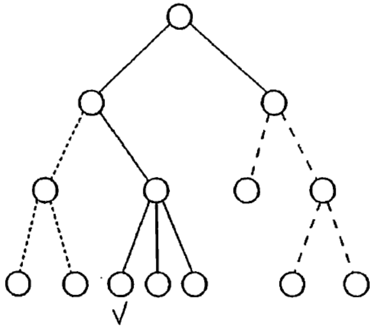
Step 1. Remove a node q from frontier queue and for each child q' of q, execute step 2

Step 2. If $C(q) + C(q, q') \leq \min(C(q'), \mathrm{UPPER})$, set $C(q') \coloneqq C(q) + C(q, q')$ and set q to be the parent of q'. In addition, if $q' \neq q_G$, place q' in the frontier queue if it is not already there, while if $q' = q_G$, set UPPER to the new value $C(q) + C(q, q_G)$

Step 3. If the frontier queue is empty, terminate, else go to step 1.

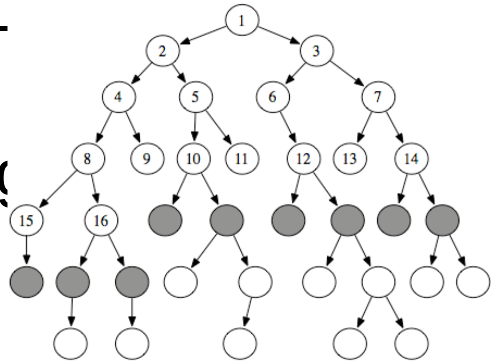
Initialization: set the labels of all nodes to ∞, except for the label of the origin node, which is set to 0.

GetNext() ?



Depth-First-Search (DFS): Maintain Q as a **stack** – Last in/first out

- Lower memory requirement (only need to store part of graph)



Breadth-First-Search (BFS, Bellman-Ford): Maintain Q as a **list** – First in/first first out

- Update cost for all edges up to current depth before proceeding greater depth
- Can deal with negative edge (transition) costs

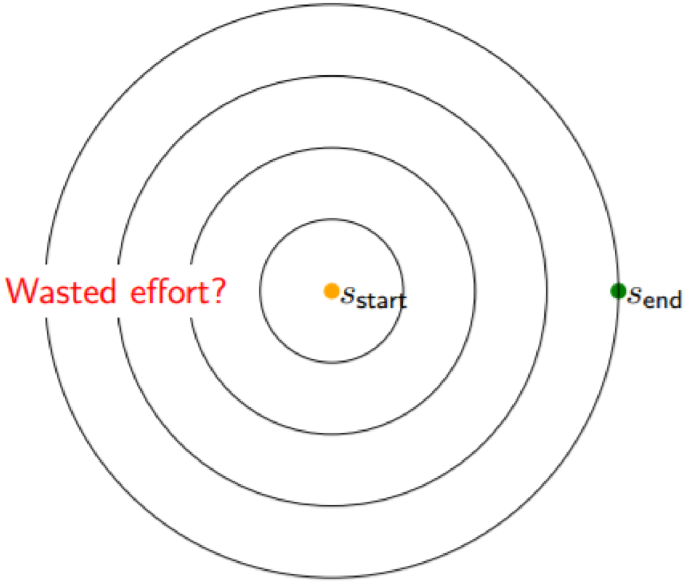
Best-First (BF, Dijkstra): Greedily select next q: $q = \operatorname{argmin}_{q \in Q} C(q)$

- Node will enter the frontier queue at most *once*
- Requires costs to be non-negative

Correctness & Improvements

Theorem

If a feasible path exists from q_I to q_G, then algorithm terminates in finite time with $C(q_G)$ equal to the optimal cost of traversal, $C^*(q_G)$.



A*: Improving Dijkstra

- Dijkstra orders by optimal “*cost-to-arrival*”
- Faster results if order by “*cost-to-arrival*”+ *(approximate) “cost-to-go”*
- That is, strengthen test

$$C(q) + C(q, q') \leq \text{UPPER}$$

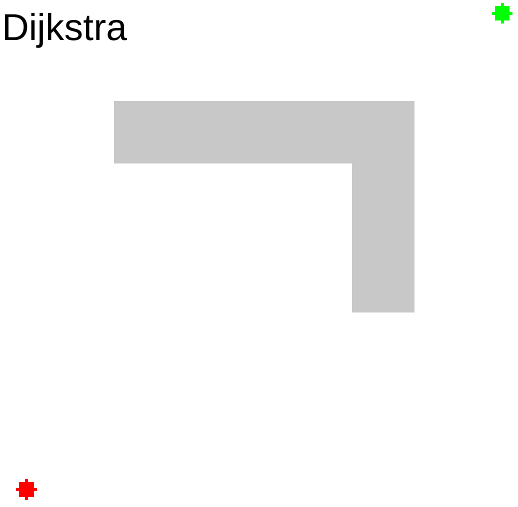
to

$$C(q) + C(q, q') + h(q') \leq \text{UPPER}$$

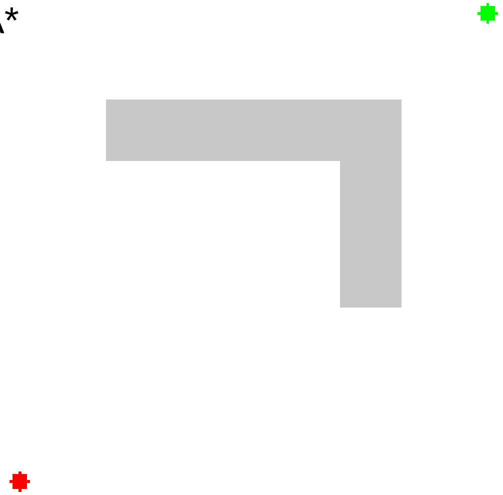
where $h(q)$ is heuristic for optimal cost-to-go (specifically, a positive *underestimate*)

- In this way, fewer nodes will be placed in the frontier queue
- This modification still guarantees that the algorithm will terminate with a shortest path

Dijkstra



A*



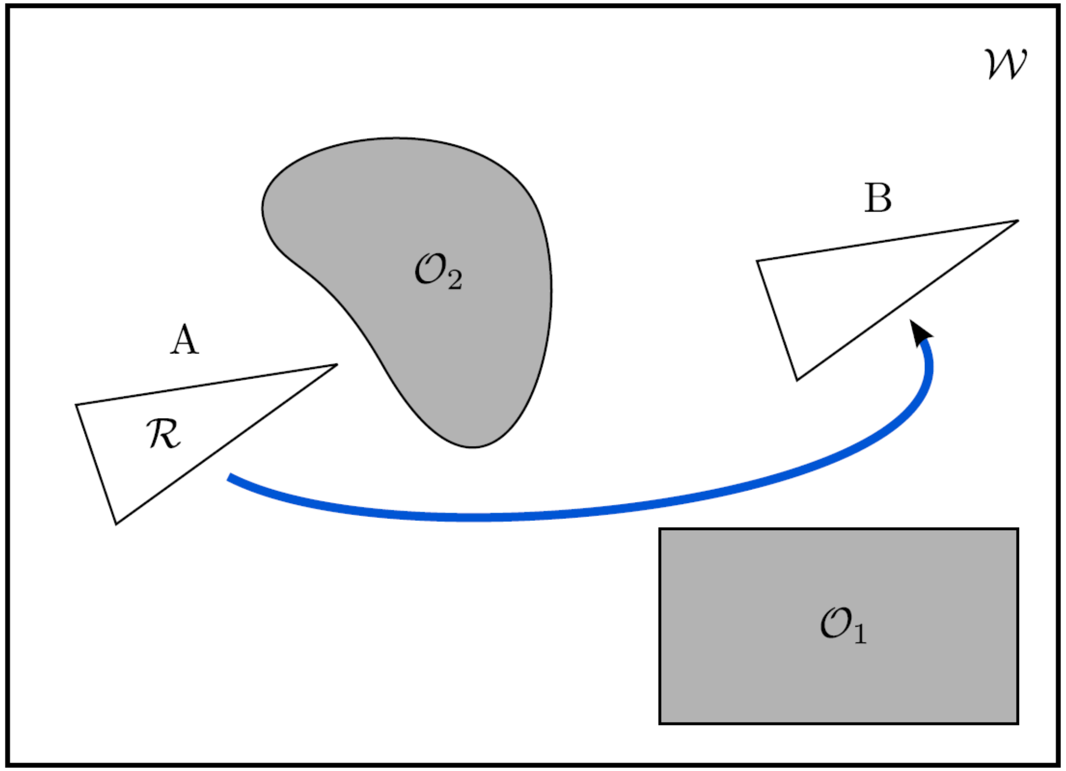
Graph-based approach: summary

- Pros:
  - Simple and easy to use
  - Fast (for some problems)
- Cons:
  - Resolution dependent
    - Not guaranteed to find solution if grid resolution is not small enough
  - Limited to simple robots (at most 3 DOFs)
    - Grid size is exponential in the number of DOFs

We shall move on to more “realistic” approaches…

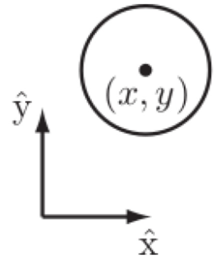
Back to (continuous) motion planning

- Our robot is a geometric entity operating in continuous space
- *Combinatorial techniques* for motion planning capture the structure of this continuous space
  - Particularly, the regions in which the robot is not in collision with obstacles
- Such approaches are typically complete
  - I.e., guaranteed to find a solution;
  - and sometimes even an optimal one

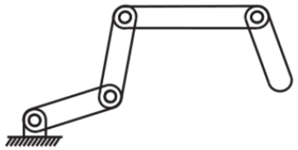


Configuration Space

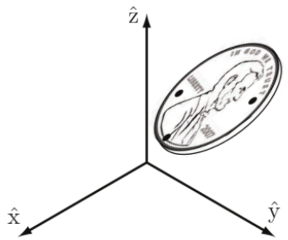
- A robot’s **configuration** is a complete specification of the position of every point of the robot
- A robot's **degrees of freedom** (dof) is the smallest number $d \geq 1$ of coordinates needed to represent its configuration
- The d-dimensional space $\mathcal{C}$ containing all possible configurations of the robot is called the configuration space (C-space)
- For instance, a polygonal robot translating and rotating in the plane has $d = 3$ DOFs and its configuration space is $\mathbb{R}^2 \times \mathcal{S}$



$d = 2$



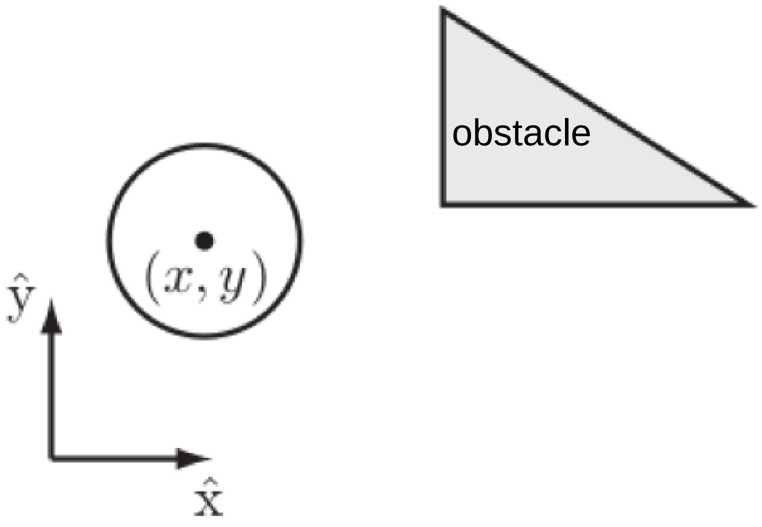
$d = 4$

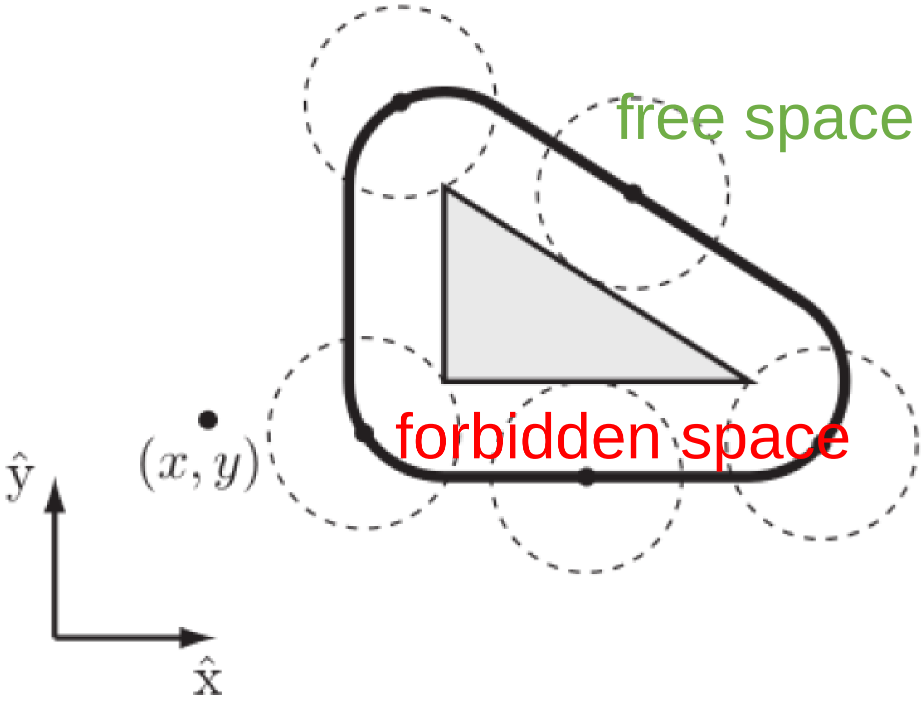


$d = 6$

Configuration space

- The subset $\mathcal{F} \subseteq \mathcal{C}$ of all collision free configurations is the **free space**

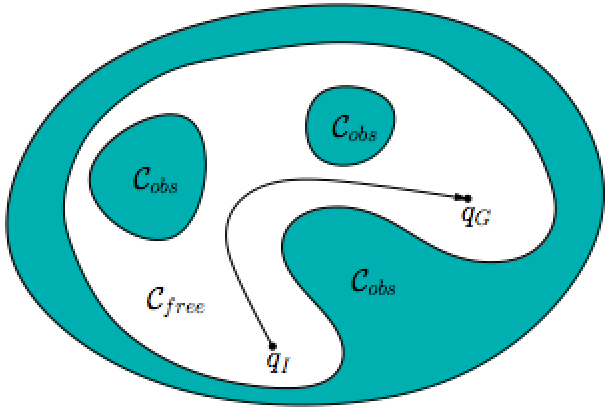




Planning in *C*- space

- Let $R(q) \subset W$ denote set of points in the world occupied by robot when in configuration q
- Robot in collision $\Leftrightarrow R(q) \cap O \neq \emptyset$
- Accordingly, *free* space is defined as: $C_{free} = \{q \in C | R(q) \cap O = \emptyset\}$
- Path planning problem in *C*-space: compute a **continuous** path: $\tau: [0,1] \rightarrow C_{free}$

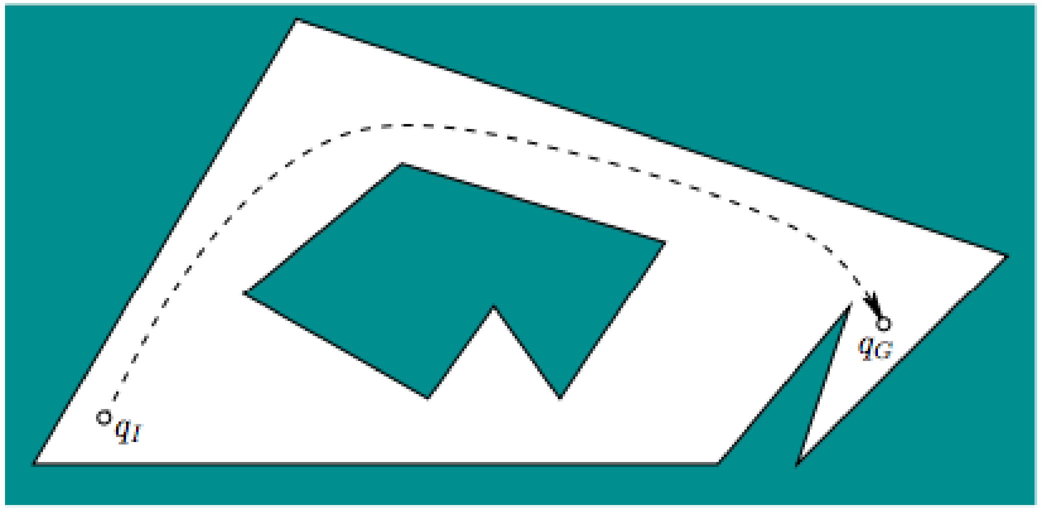
with $\tau(0) = q_I$ and $\tau(1) = q_G$



Combinatorial Planning

Example:

- Point robot in the plane



Key idea: compute a roadmap, which is a graph in which each vertex is a configuration in C_{free} and each edge is a path through C_{free} that connects a pair of vertices

Free-space roadmaps

Given a complete representation of the free space, we compute a roadmap that captures its connectivity.

A roadmap should preserve:

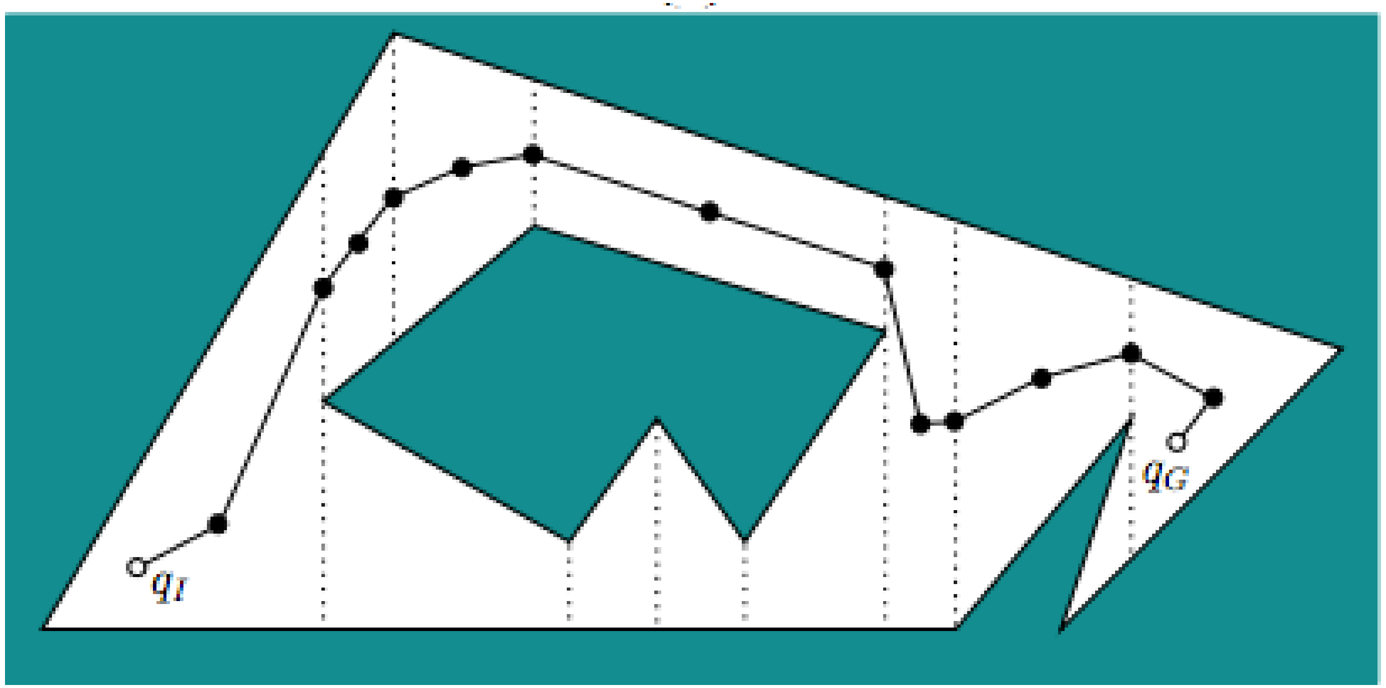
1. Accessibility: it is always possible to connect some q to the roadmap (e.g., $q_I \rightarrow s_1, q_G \rightarrow s_2$)
2. Connectivity: if there exists a path from q_I to q_G, there exists a path on the roadmap from s_1 to s_2

- Key idea: a roadmap provides a discrete representation of the continuous motion planning problem without losing any of the original connectivity information needed to solve it

Cell Decomposition

Typical approach: cell decomposition. General requirements:

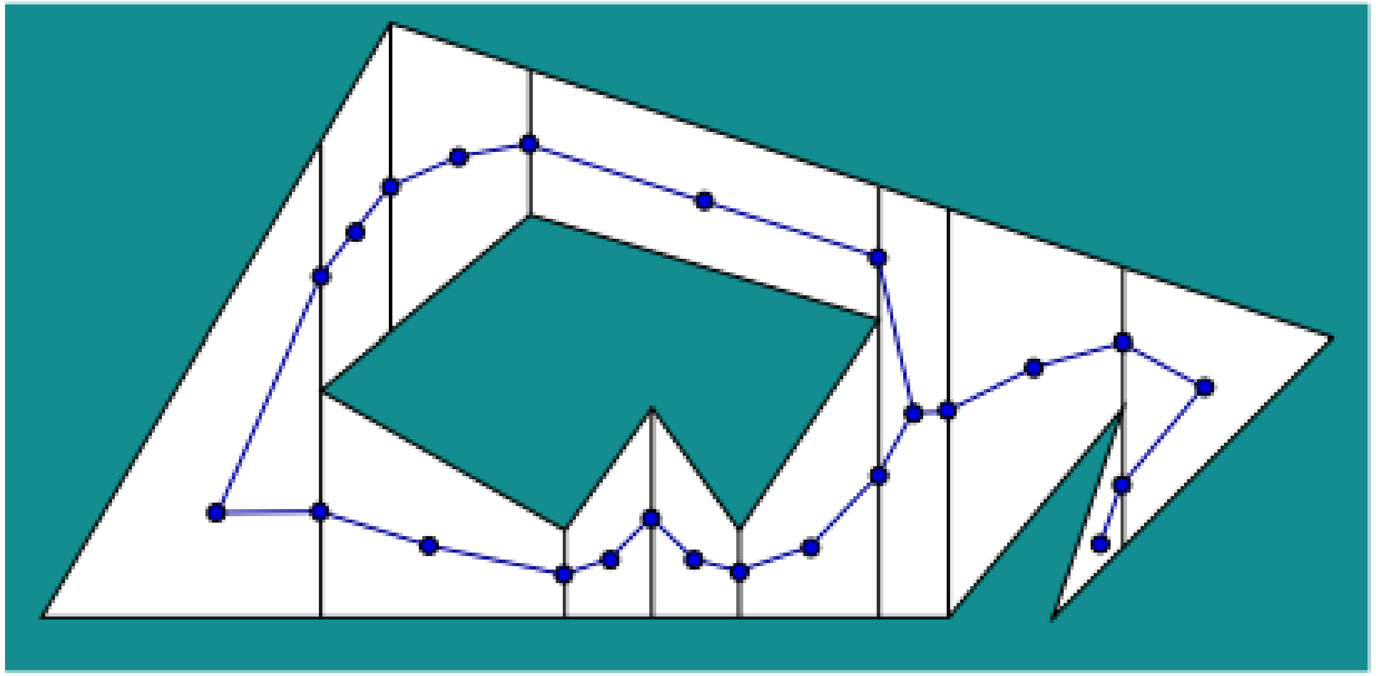
- Each cell should be easy to traverse (ideally convex)
- Decomposition should be easy to compute
- Adjacencies between cells should be straightforward to determine



Computing a trapezoidal decomposition:

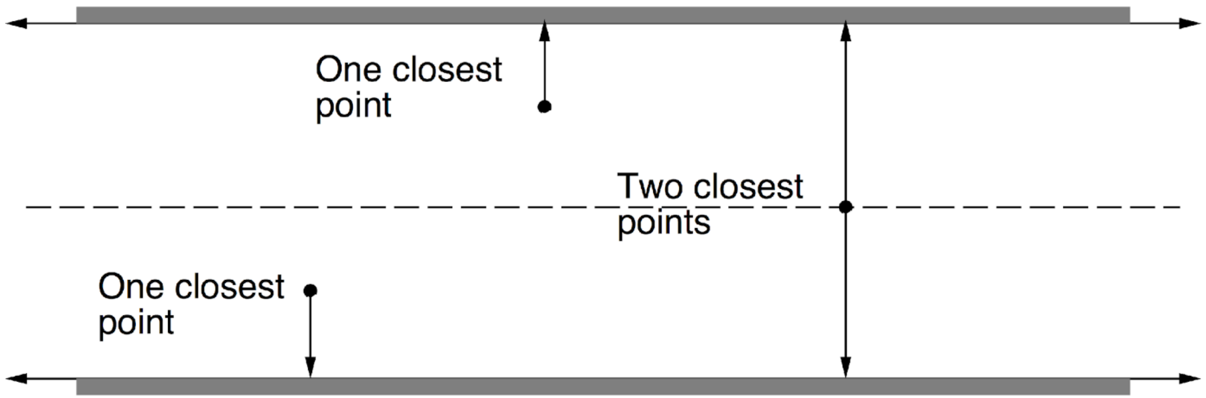
For every vertex (corner) of the forbidden space:

- Extend a vertical ray until it hits the first edge from top and bottom
- Implementation details:
  - Compute intersection points with all edges, and take the closest ones
  - More efficient approaches exists

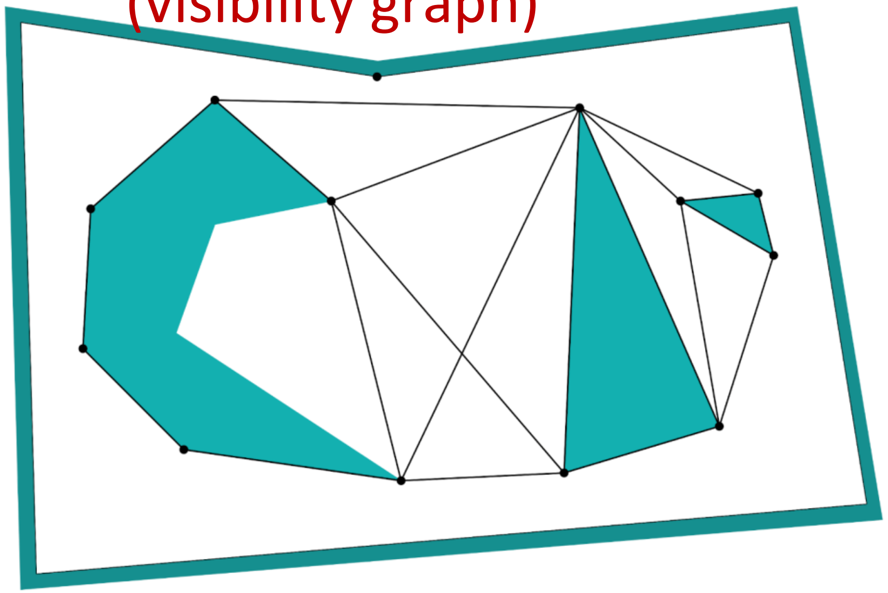


Other roadmaps

Maximum clearance (medial axis)



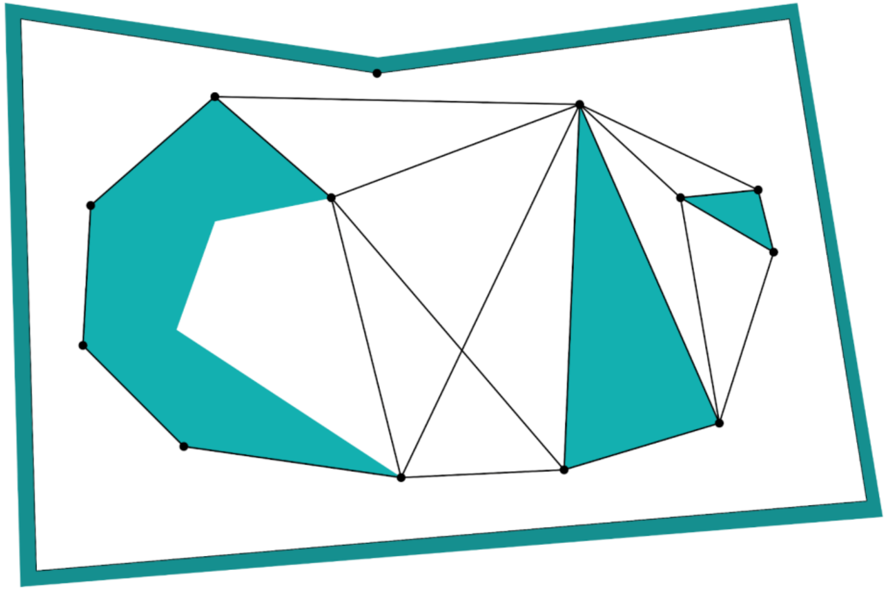
Minimum distance (visibility graph)



Note: No loss in optimality for a proper choice of discretization

Visibility graph

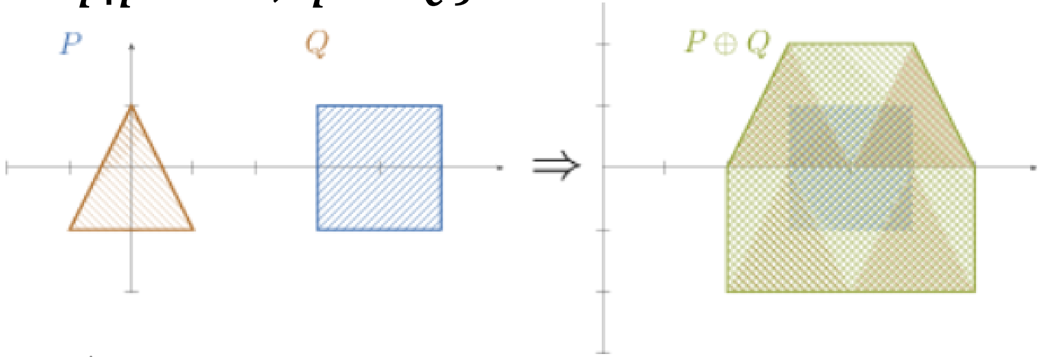
- Allows to compute shortest collision free paths
- Connect by an edge every two vertices of the forbidden space that are *visible* from each other
  - The straight-line path between them is collision- free
- Given query points, connect them to the graph in a similar fashion



Free-space computation

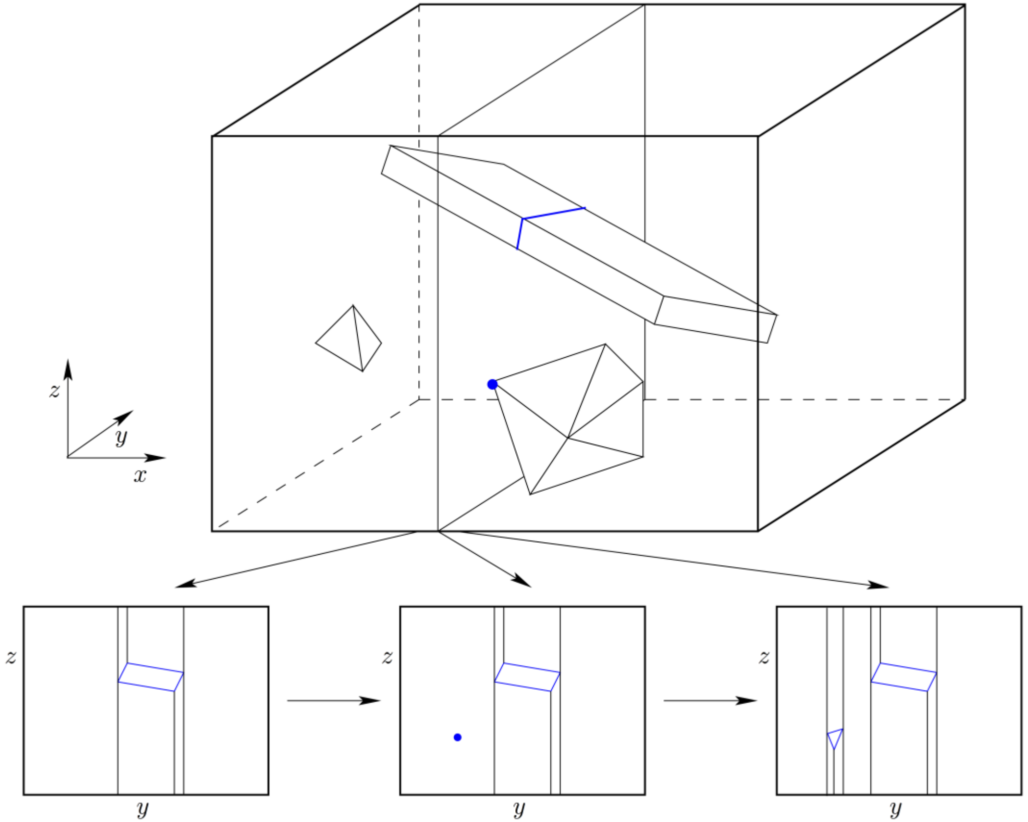
- The free space is not known in advance
- We need to compute this space given the ingredients
  - Robot representation, i.e., its shape (polygon, polyhedron, …)
  - Representation of obstacles
- To achieve this we do the following:
  - Contract the robot into a point
  - In return, inflate (or stretch) obstacles by the shape of the robots
  - Also known as **Minkowski sum**

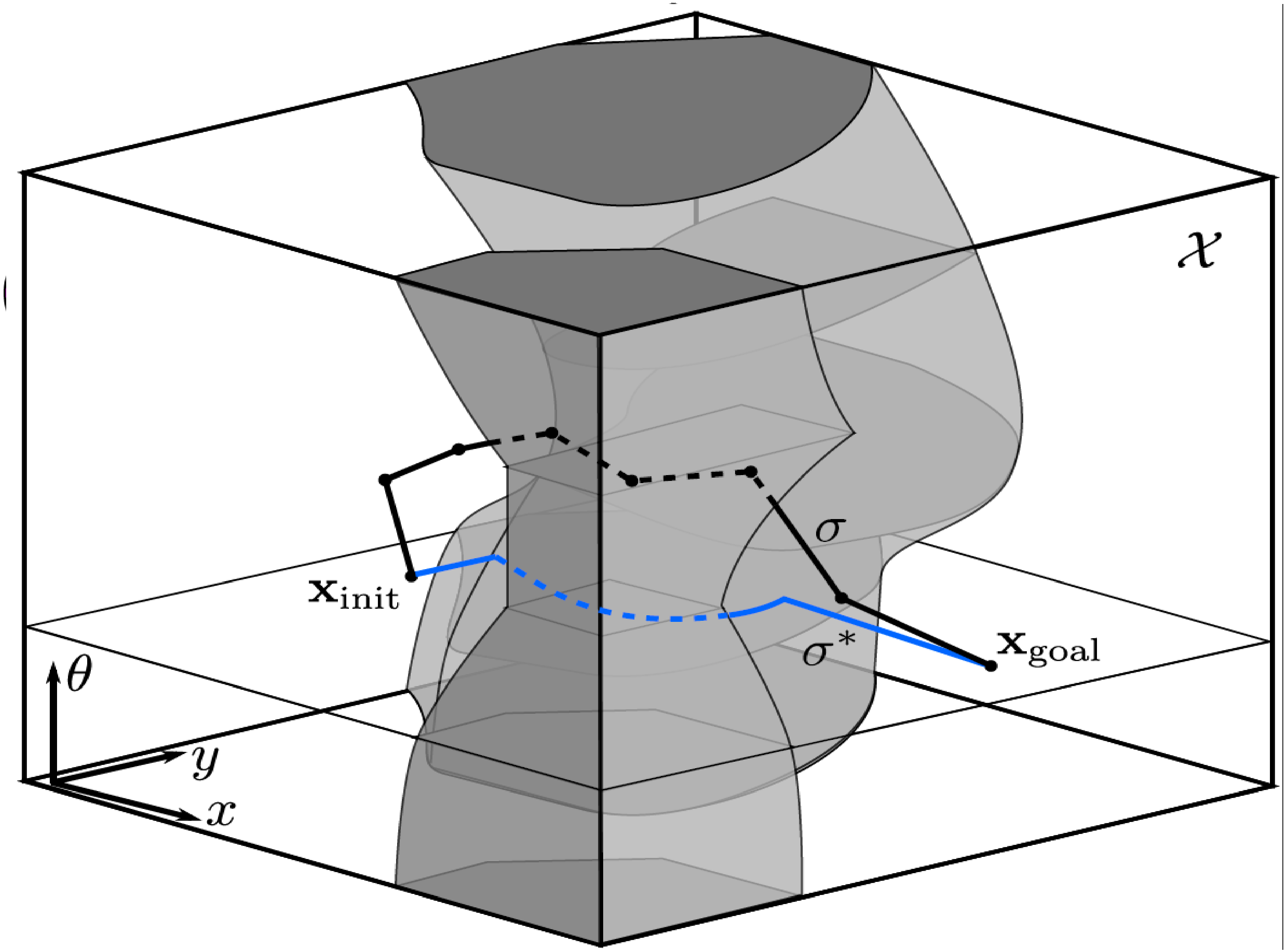
$$P \oplus Q := \{p + q | p \in P, q \in Q\}$$



Higher Dimensions

- Extensions to higher dimensions is challenging ⇒ algebraic decomposition methods

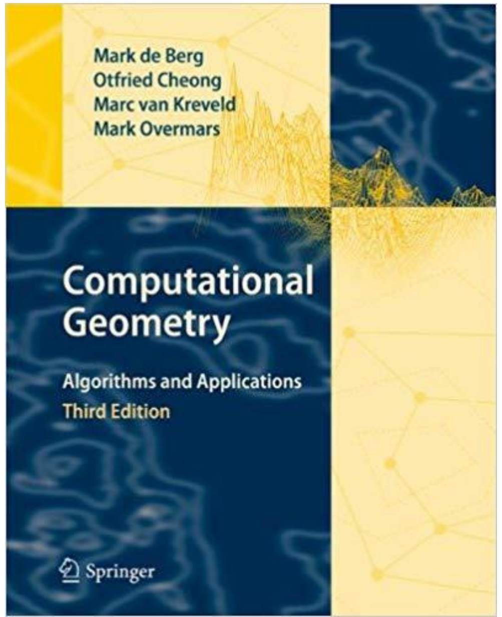




$\mathcal{X}$
σ
$\mathbf{x}_{\text{init}}$
σ^*
$\mathbf{x}_{\text{goal}}$
θ
y
x

Additional resources on combinatorial planning

- Visualization of C-space for polygonal robot: https://www.youtube.com/watch?v=SBFwgR4K1Gk
- Algorithmic details for Minkowski sums and trapezoidal decomposition: de Berg et al., “Computational geometry: algorithms and applications”, 2008
- Implementation in C++: Computational Geometry Algorithms Library



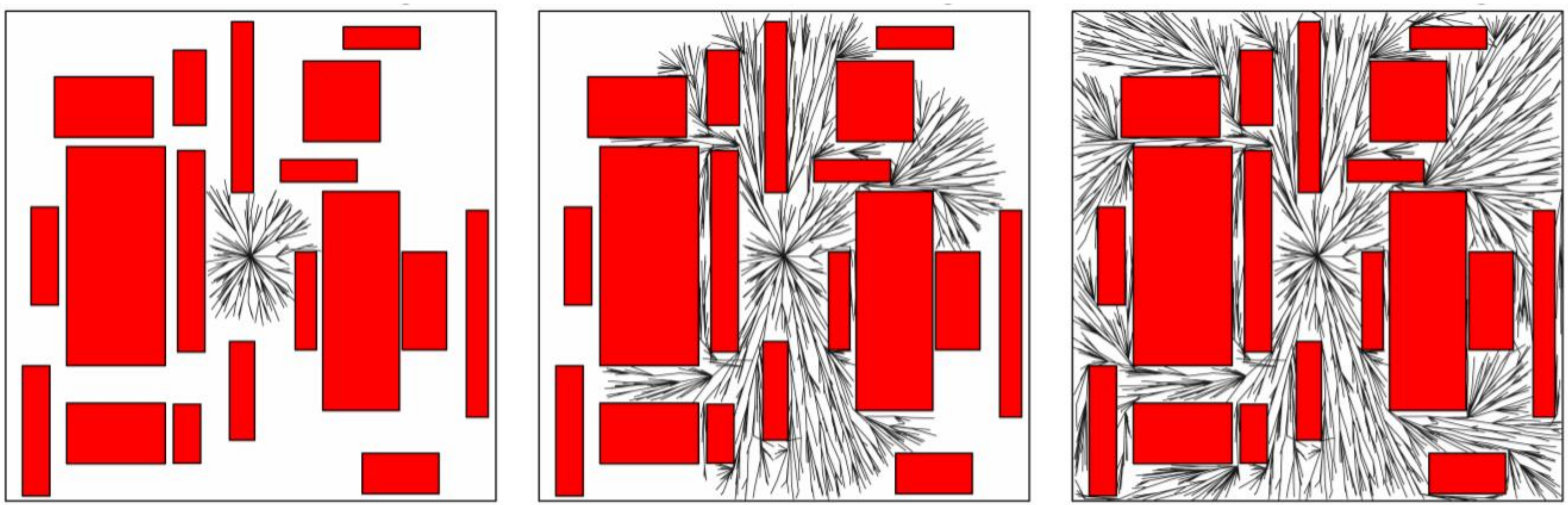
Combinatorial planning: summary

- These approaches are complete and even optimal in some cases
  - Do not discretize or approximate the problem
- Have theoretical guarantees on the running time
  - I.e., computational complexity is known
- Usually limited to small number of DOFs
  - Computationally intractable for many problems
- Problem specific: each algorithm applies to a specific type of robot/problem
- Difficult to implement: require special software to reason with geometric data structures (CGAL)

Summary

- Search: Dijkstra (Uniform cost), A* (cost-to-go heuristic)
- Combinatorial planning: discretize *C*-space as a graph, then search for shortest path
- Decomposition depends on cost function
- Popular method: cell decomposition – non intuitive in higher dimensions
- In general, motion planning is (PSPACE-)hard!

Next time: sampling-based planning

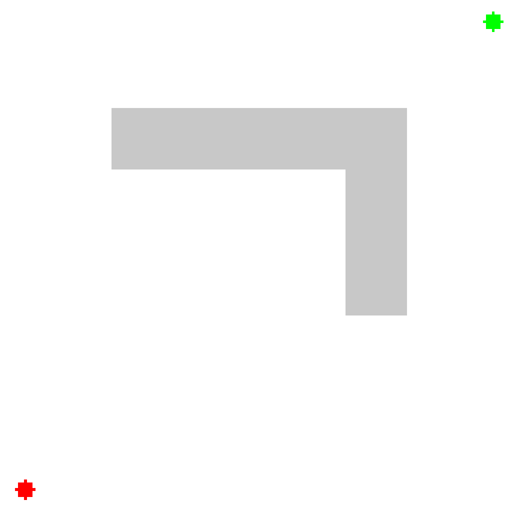


Backup: additional details for A*

- Basic ingredients of A*
  - q_{init}: start vertex; q_{goal}: target vertex
  - OPEN: list of known vertices that have not been expanded yet
  - CLOSED: expanded vertices
  - $C(q, q')$: real cost of edge from q to q'
  - $C(q)$: upper bound on "cost-to-come" from q_{init} to q
  - $h(q)$: lower bound on "cost-to-go" from q to q_{goal}
  - $f(q) = C(q) + h(q)$: upper bound on total cost

A* algorithm

$C(q) = \infty, f(q) = \infty$ for all q;
$f(q_{init}) = h(q_{goal}); C(q_{init}) = 0$
OPEN = $\{q_{init}\}$; CLOSED := { };
while (OPEN not empty)
    $q \coloneqq q$ in OPEN that minimizes $f(q)$
    if $q == q_{goal}$ return path
    OPEN.remove(q); CLOSED.add(q)
    for all q' in $\{q' \mid (q, q')$ in G, q' not in CLOSED$\}$ // expansion
        OPEN.add(q')
        if $C(q') \leq C(q) + C(q, q')$
            continue;
        q'.parent = q; $C(q') = C(q) + C(q, q')$
        $f(q') = C(q') + h(q')$
Return failure



A*: theory

- h is **admissible** if for every q, $h(q)$ is at most the actual cost from q to q_{goal}
- h is **monotonic** if for every two adjacent vertices q, q', it holds that $h(q) \leq C(q, q') + h(q')$
- **Theorem:** If h is admissible and monotonic the algorithm returns the shortest path
- Common heuristic:
  - Euclidean distance in configuration space