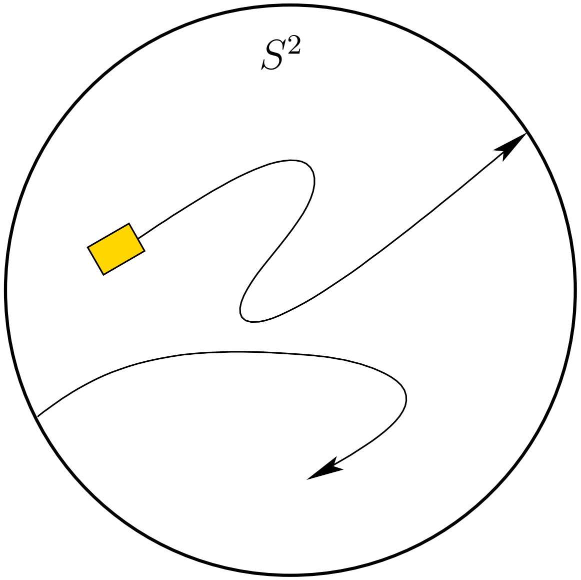
Motion Planning for Dynamic Environments

Part II: Motion Planning: Finding the Path

Steven M. LaValle
University of Illinois

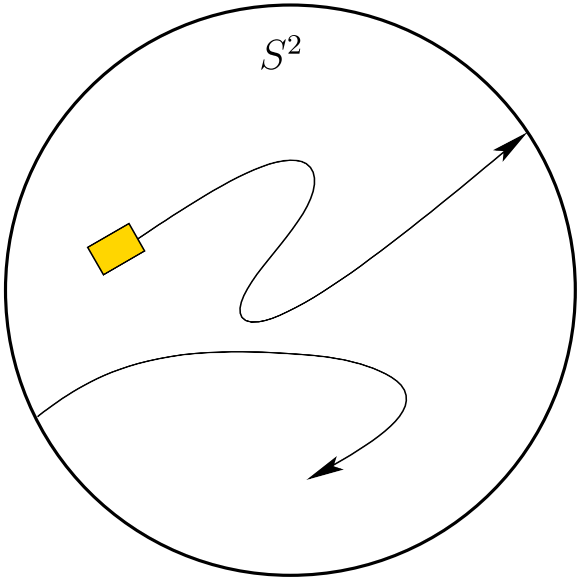
A car driving on a gigantic sphere:



The C-space is:

Solution to Homework 1

A car driving on a gigantic sphere:

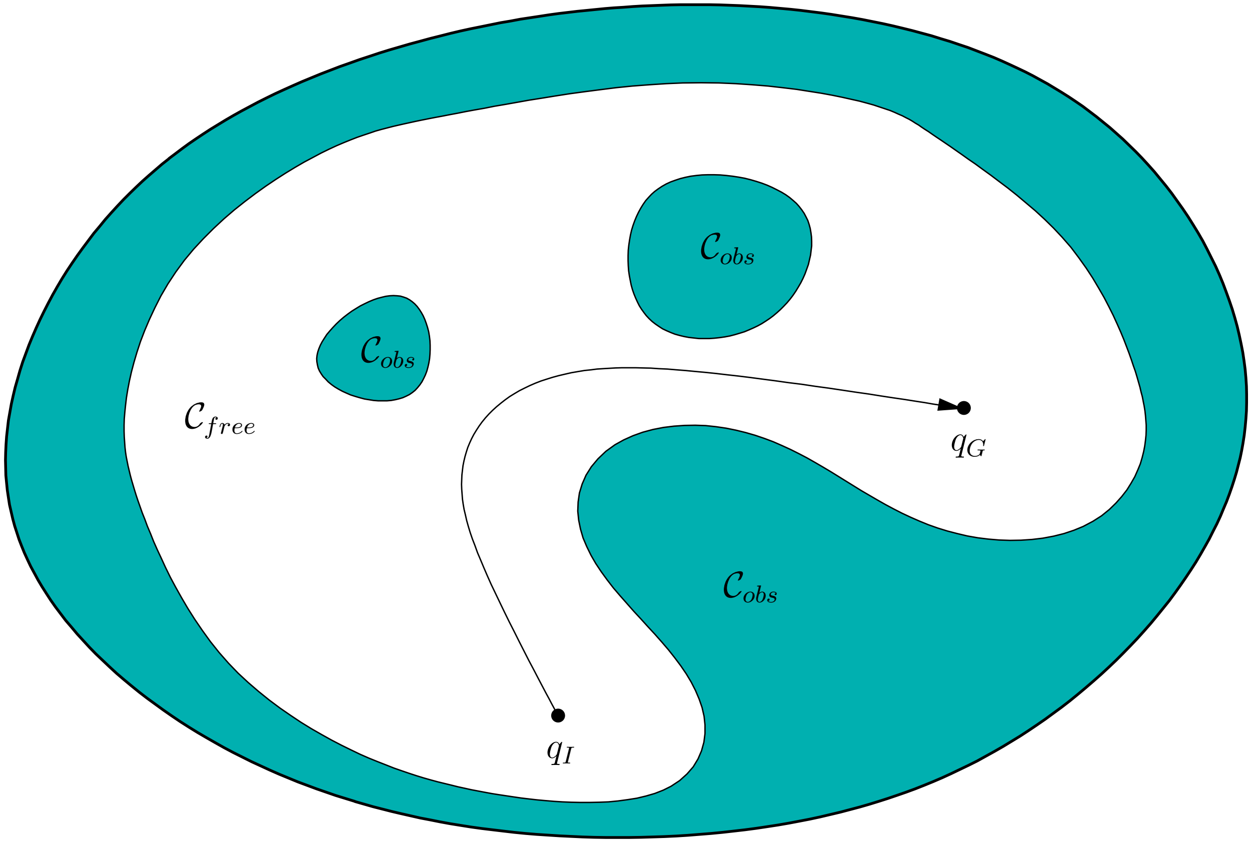


The C-space is: $SO(3) = \mathbb{R}P^3$
To see it, imagine car is painted on S^2 and rotate S^2 about its center.

It is not $S^2 \times S^1$: Cartesian product vs. fiber bundle (Hopf fibration)

Basic Motion Planning Problem

Given robot $\mathcal{A}$ and obstacle $\mathcal{O}$ models, C-space $\mathcal{C}$, and $q_I, q_G \in \mathcal{C}_{free}$.



Automatically compute a path $\tau : [0,1] \rightarrow \mathcal{C}_{free}$ so that $\tau(0) = q_I$ and $\tau(1) = q_G$.

Competing Paradigms

- **Combinatorial planning**
  (exact planning)
- **Sampling-based planning**
  (probabilistic planning, randomized planning)

The methods differ in the philosophy they use to *discretize* the problem.

Also: Approximate cell decompositions

Completeness Notions

A planning algorithm may be:

- **Complete:** If a solution exists, it finds one; otherwise, it reports failure.
- **Semi-complete:** If a solution exists, it finds one; otherwise, it may run forever.
- **Resolution complete:** If a solution exists, it finds one; otherwise, it terminates and reports that no solution within a specified resolution exists.
- **Probabilistically complete:** If a solution exists, the probability that it will be found tends to one as the number of iterations tends to infinity.

Combinatorial Planning

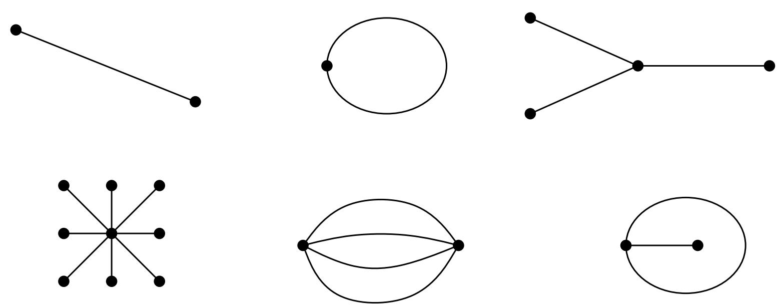
Combinatorial Planning Methods

- Mostly developed in the 1980s
- Influence from computational geometry and computational real algebraic geometry
- All algorithms are complete
- Usually produce a roadmap in $\mathcal{C}_{free}$
- Extremely efficient for low-dimensional problems
- Some are difficult to implement (numerical issues)

Topological Graph

Methods produce a *topological graph* $\mathcal{G}$:

- Each *vertex* is a configuration $q \in \mathcal{C}_{free}$.
- Each *edge* is a path $\tau : [0,1] \rightarrow \mathcal{C}_{free}$ for which $\tau(0)$ and $\tau(1)$ are vertices.



Sometimes, $\mathcal{C}_{free}$ may be replaced by $cl(\mathcal{C}_{free})$
(include the boundary of $\mathcal{C}_{free}$).

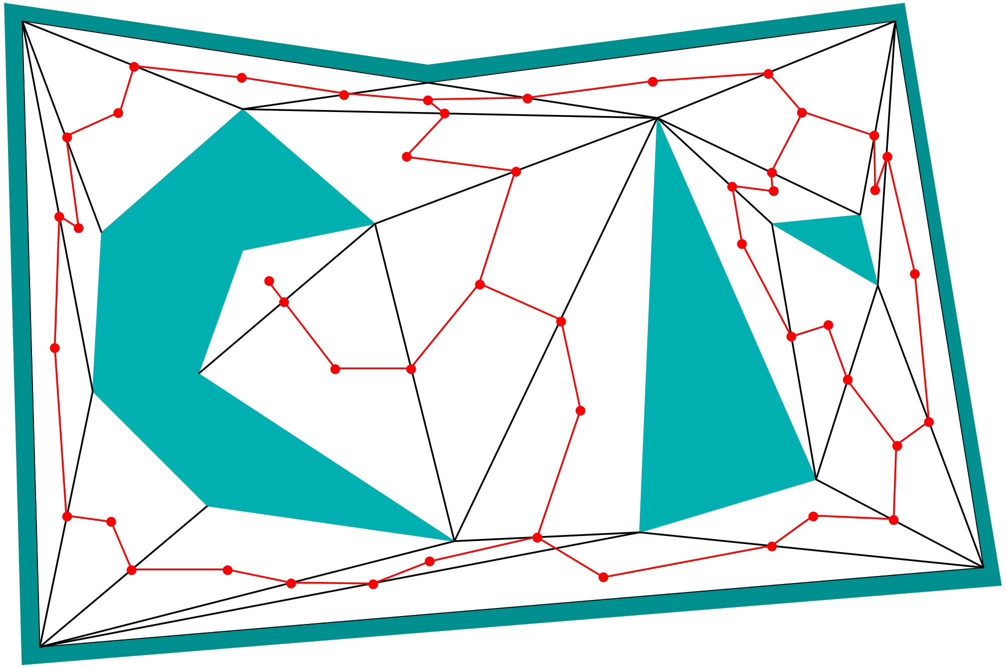
This allows the robot to "scrape" the obstacles.

Roadmap

A *roadmap* is a topological graph $\mathcal{G}$ with two properties:

1. **Accessibility:** From anywhere in $\mathcal{C}_{free}$ it is trivial to compute a path that reaches at least one point along any edge in $\mathcal{G}$.

2. **Connectivity-preserving:** If there exists a path through $\mathcal{C}_{free}$ from q_I to q_G, then there must also exist one that travels through $\mathcal{G}$.



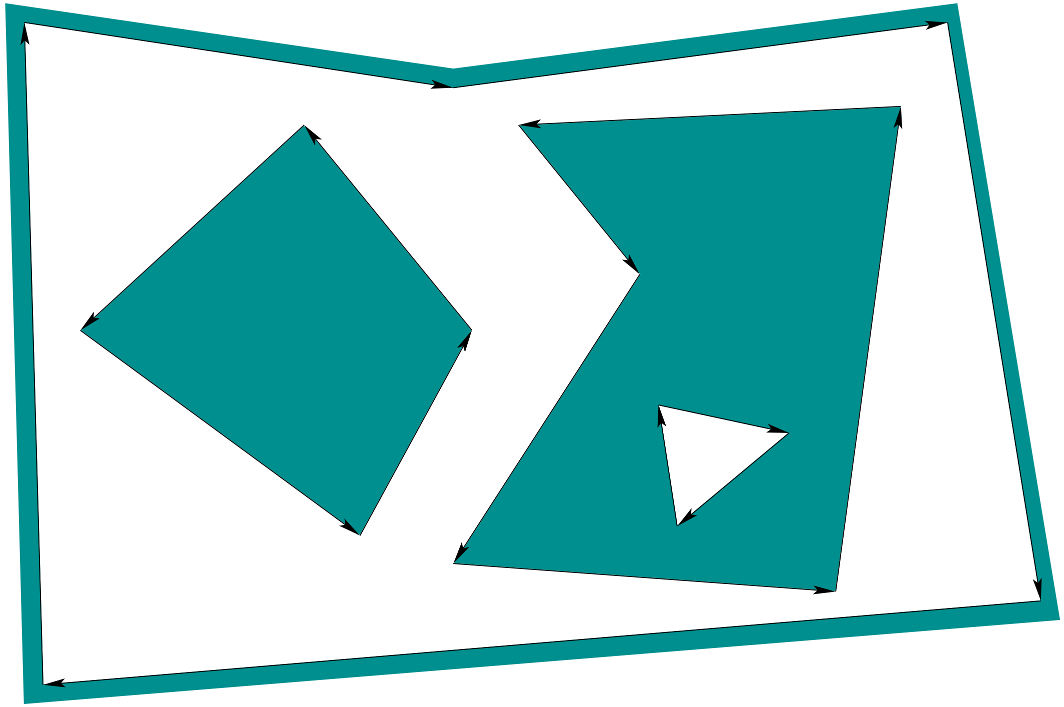
Planning in a Polygonal Obstacle Region

Assume that $\mathcal{C}_{obs}$ (and $\mathcal{C}_{free}$) are piecewise linear.

Could be a point robot among polygonal obstacles.

Could be a polygonal, translating robot among polygonal obstacles.

The methods tend to extend well to a disc robot.



Use clever data structures to encode vertices, edges, regions

Example: Doubly connected edge list

Planning in a Polygonal Obstacle Region

We consider four methods:

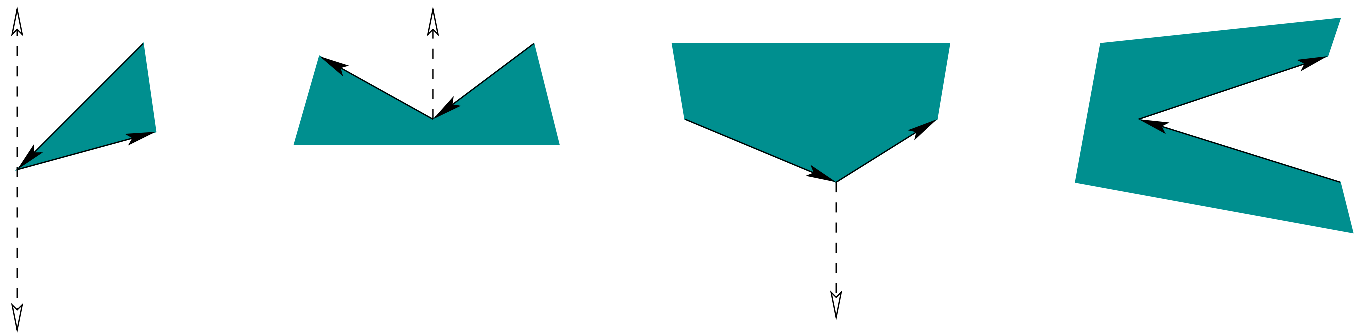
- Trapezoidal decomposition
- Triangulation
- Maximum-clearance roadmap (retraction method)
- Shortest-path roadmap (reduced visibility graph)

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

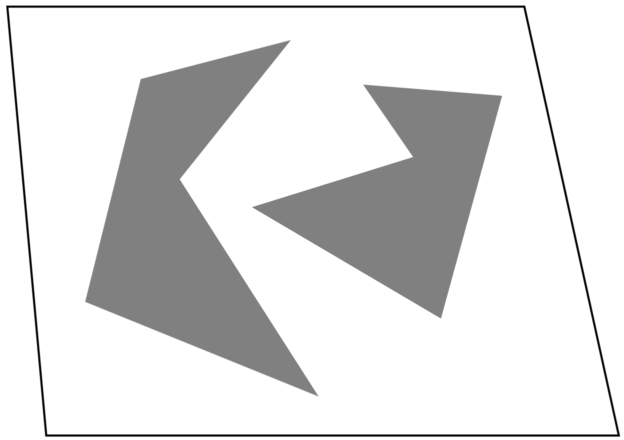
Trapezoidal Decomposition

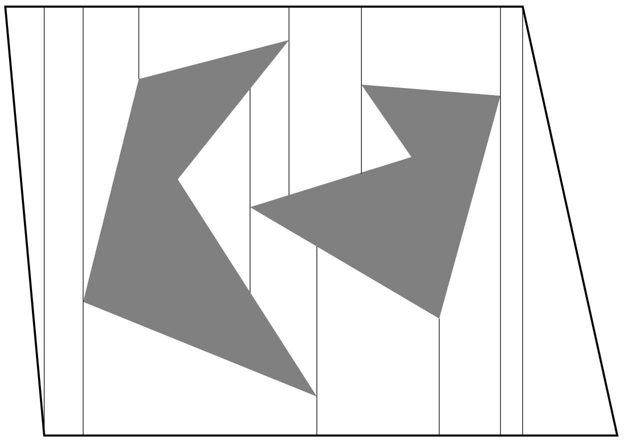
Try to extend a ray above or below every vertex.

There are four cases:



Trapezoidal Decomposition

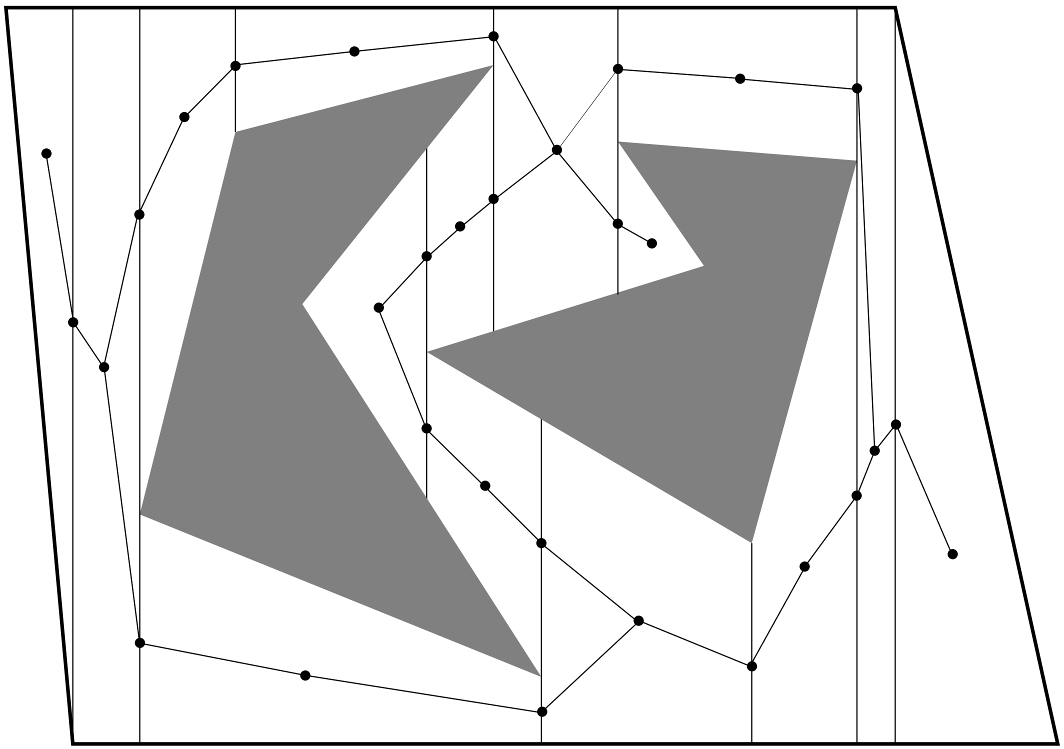




- Use the plane sweep principle to efficiently determine where the rays terminate.
- Sort vertices by x coordinate.
- Handle extensions from left to right, while maintaining a vertically sorted list of edges.
- Leads to $O(n \lg n)$ running time. Easy to implement.

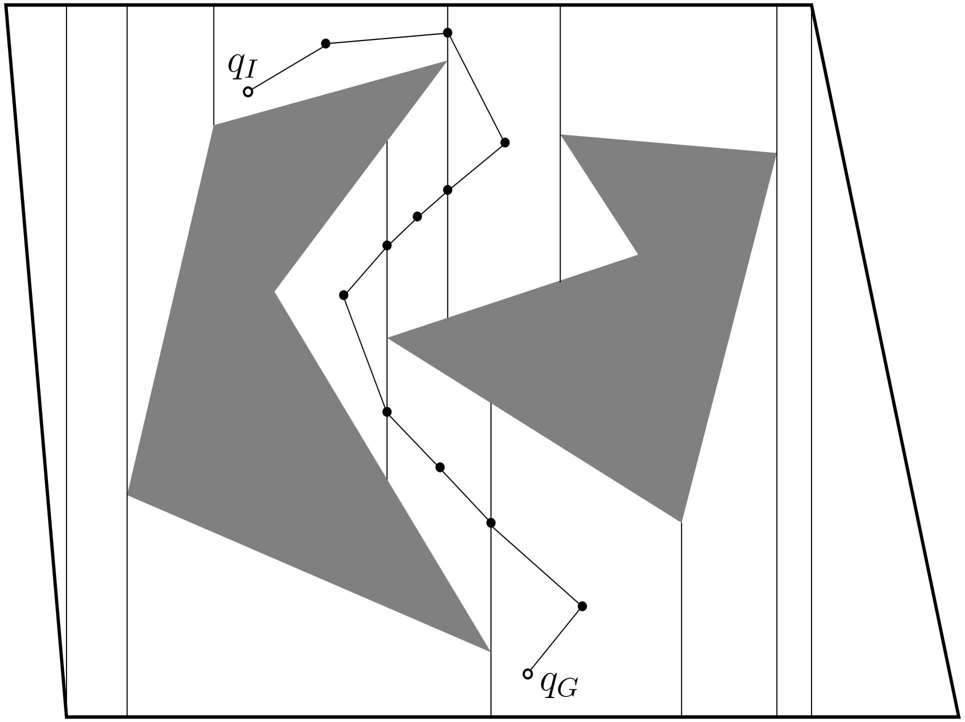
Trapezoidal Decomposition

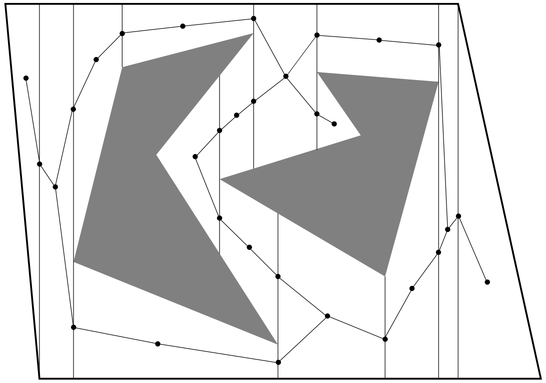
The resulting roadmap $\mathcal{G}$:



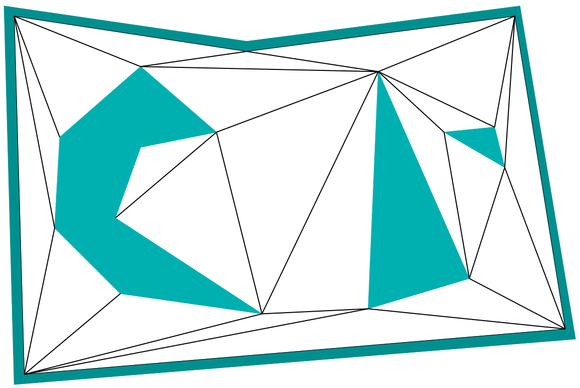
Trapezoidal Decomposition

Solving a query: Get from q_I to q_G.



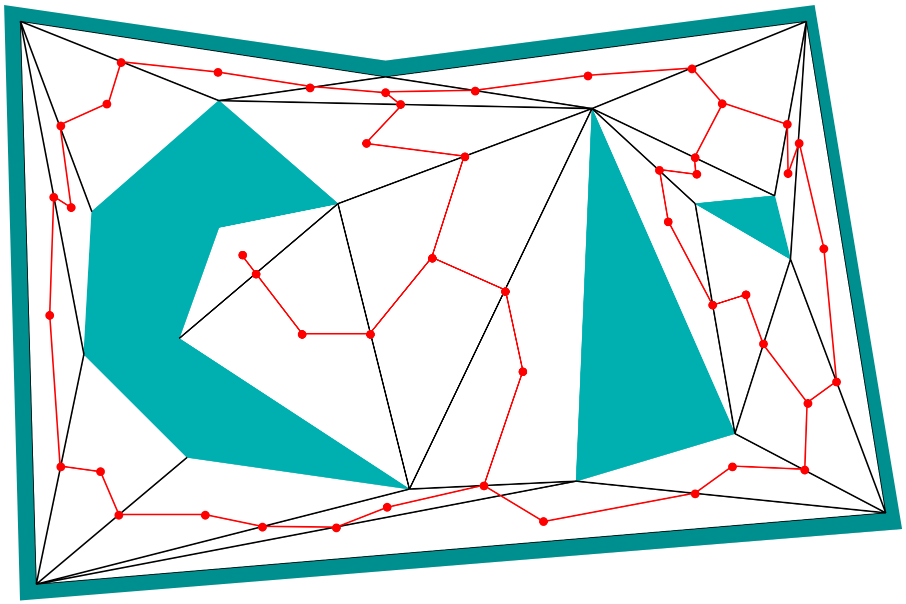


Triangulation

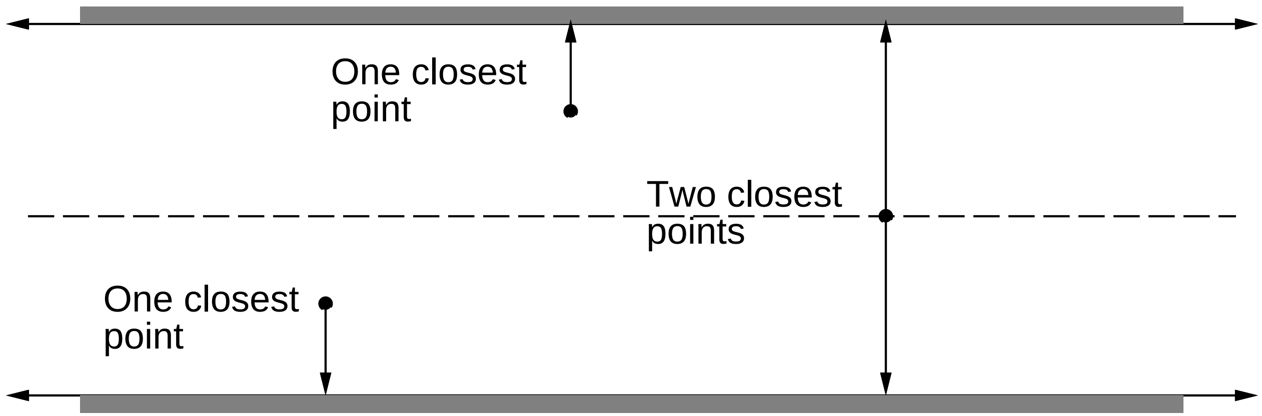


Compute triangulation:
$O(n^2)$ time naive, $O(n)$ optimal, $O(n \lg n)$ a good tradeoff.

Build easy roadmap from the triangulation:



Maximum Clearance Roadmap



Imagine obtaining a skeleton by gradually thinning $\mathcal{C}_{free}$.

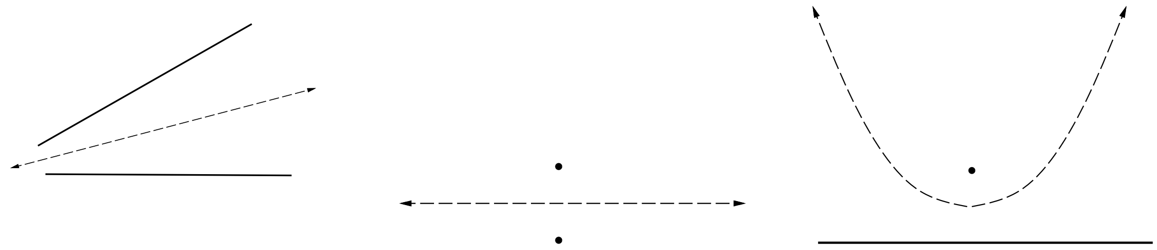
Based on *deformation retract* from topology.

Also is a kind of generalized Voronoi diagram.

O'Dunlaing, Yap, 1983

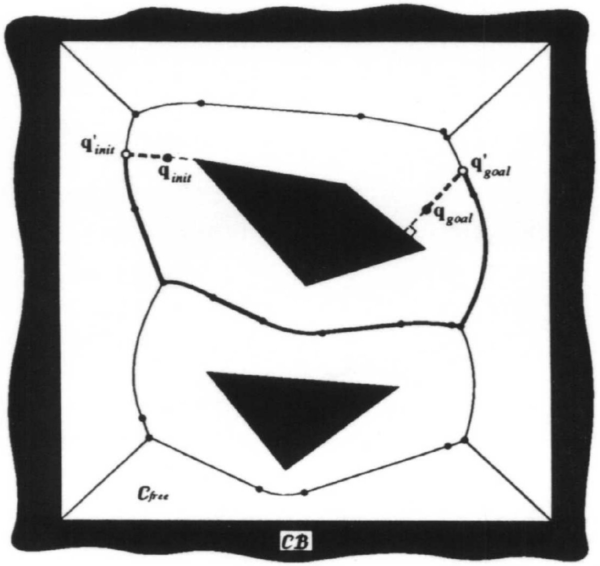
Maximum Clearance Roadmap

Three cases contribute to the roadmap:



Edge-Edge Vertex-Vertex Vertex-Edge

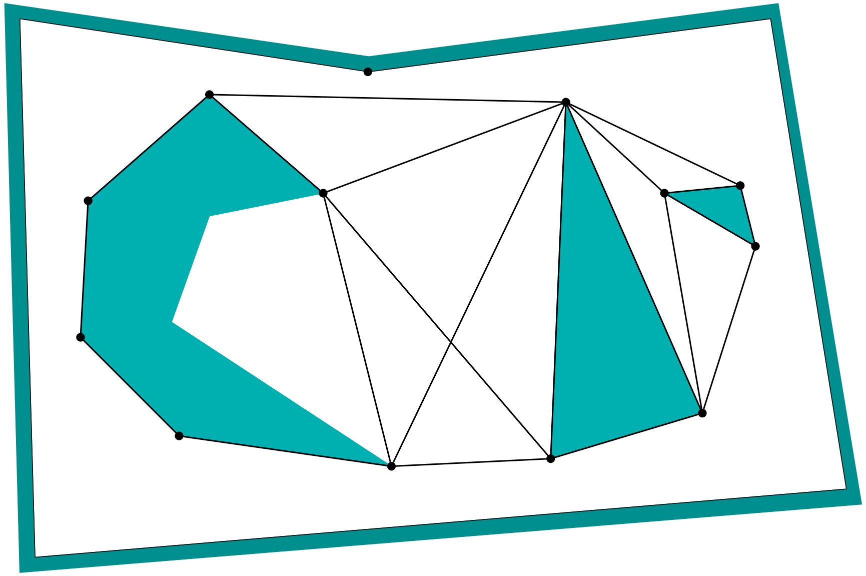
$O(n^4)$ time naive, $O(n \lg n)$ optimal.



Picture from Latombe, 1991

Shortest-Path Roadmap

Optimal planning is easy in polygonal environments.



The shortest-path roadmap contains all vertices and edges that optimal paths follow when obstructed.

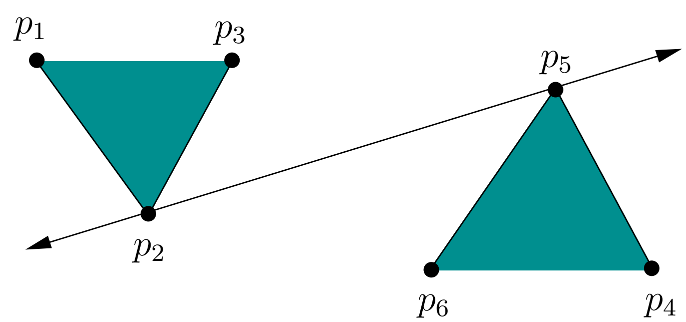
Imagine pulling a string tight between q_I and q_G.

Shortest-Path Roadmap

Every *reflex vertex* (interior angle $> \pi$) is a roadmap vertex.

Edges in the roadmap correspond to two cases:

1. Consecutive reflex vertices
2. Bitangent edges



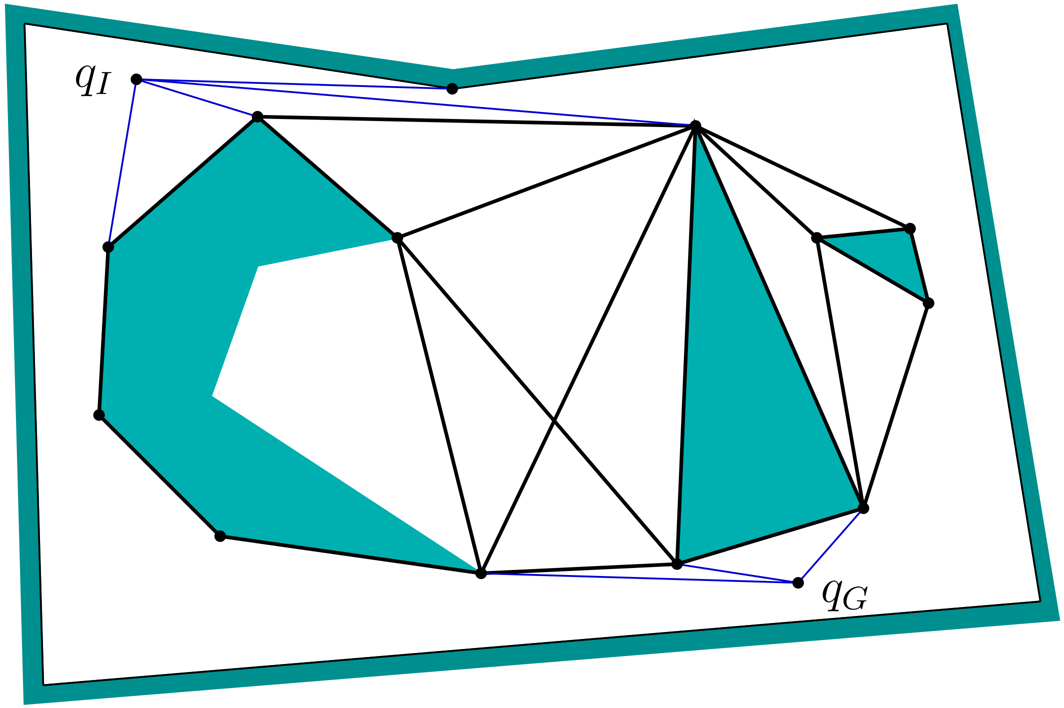
A bitangent edge is needed when this is true:

$$\big(f_l(p_1, p_2, p_5) \oplus f_l(p_3, p_2, p_5)\big) \vee \big(f_l(p_4, p_5, p_2) \oplus f_l(p_6, p_5, p_2)\big),$$

in which f_l is a *left-turn predicate*.

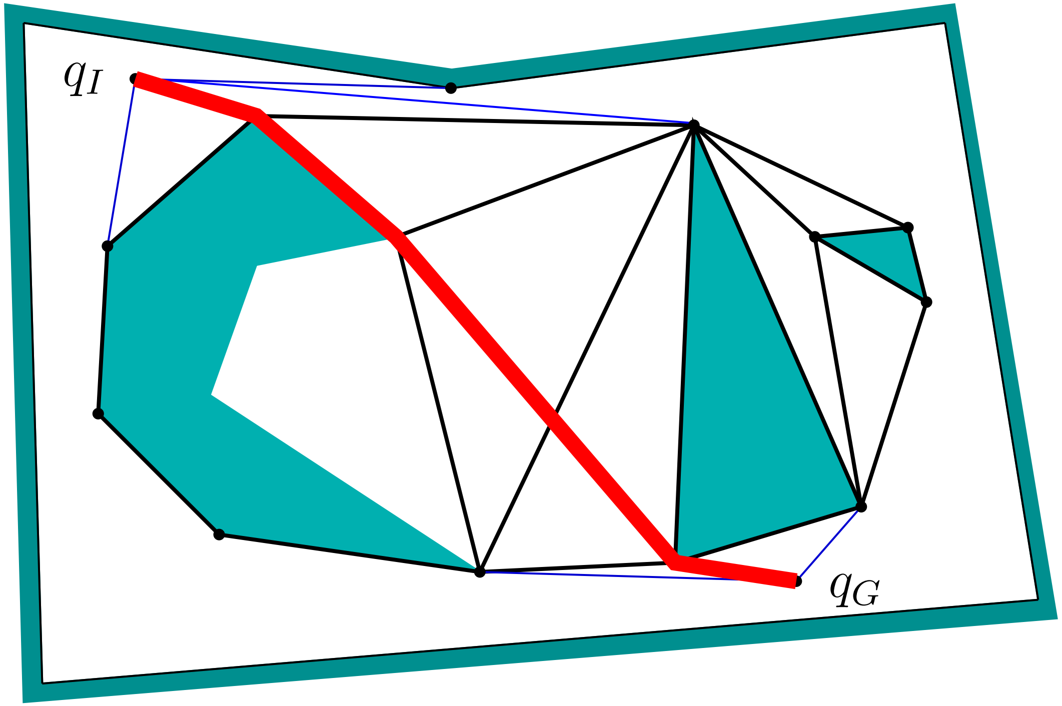
Shortest-Path Roadmap

To solve a query, connect q_I and q_G to the roadmap:



Shortest-Path Roadmap

Use Dijkstra's algorithm to search for a shortest path.



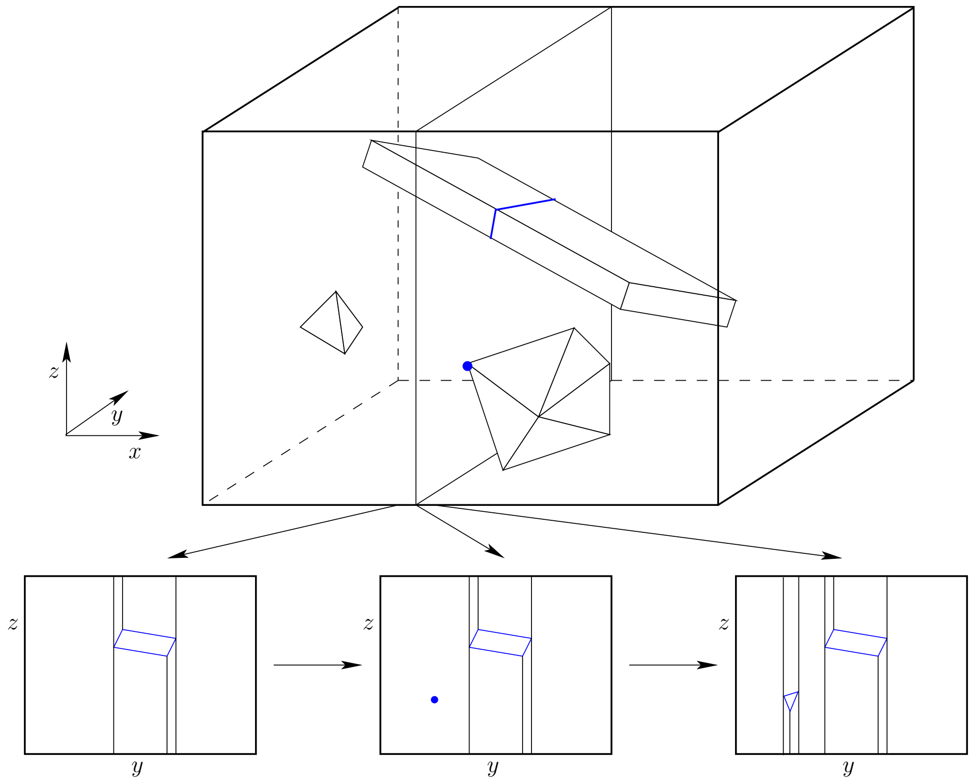
Higher Dimensions

If $\mathcal{C}$ is 3 or more dimensions, most methods do not extend.

Optimal path planning for 3D polyhedra is NP-hard.

Maximal clearance roadmaps become disconnected in 3D.

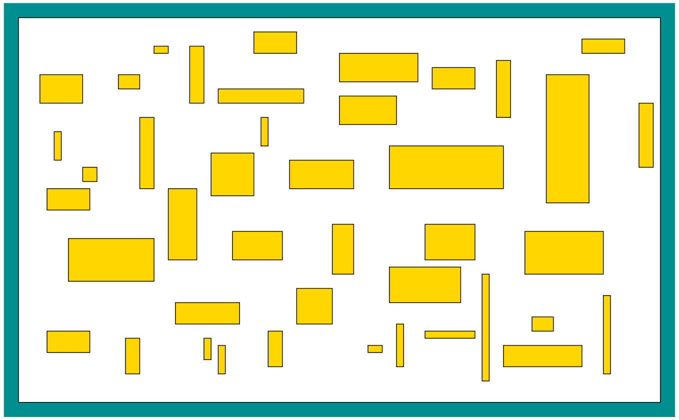
Trapezoidal decomposition extends:



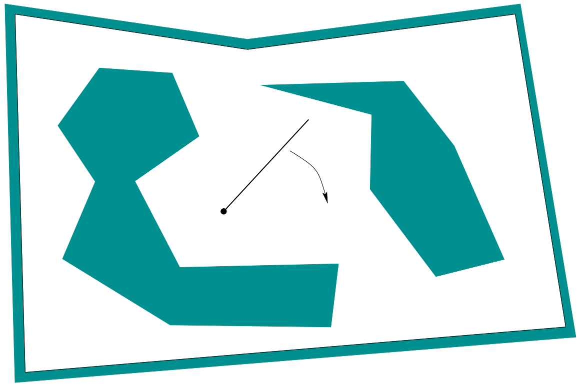
Higher Dimensions

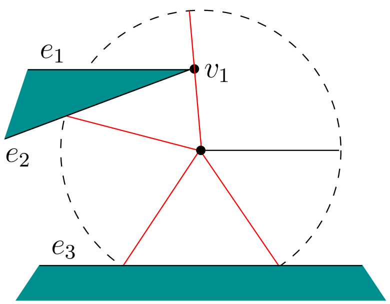
- Specialized decompositions: ladder, rigid planar robot, discs
- Cylindrical algebraic decomposition (Schwartz, Sharir, 1983)
- Canny's roadmap algorithm (1987)

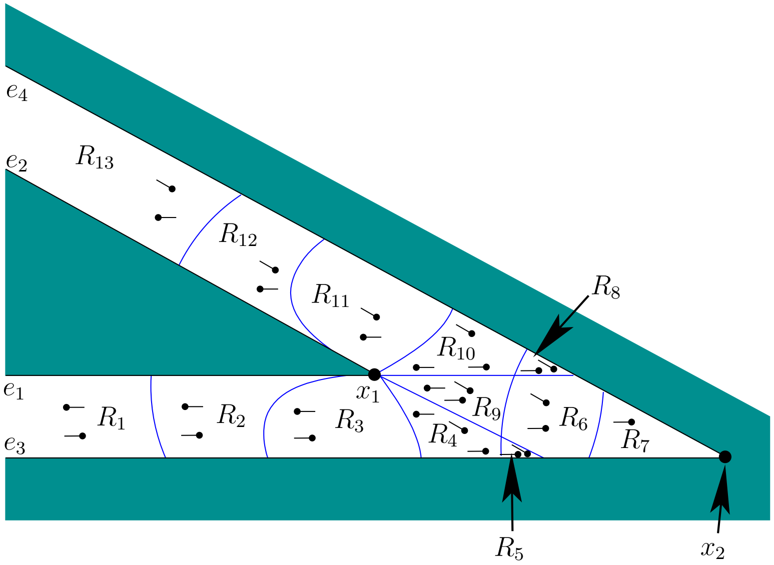
Rearranging a bunch of rectangles is PSPACE-hard:



Decomposition for a Ladder



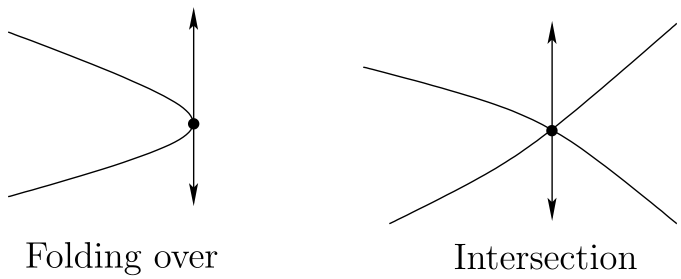




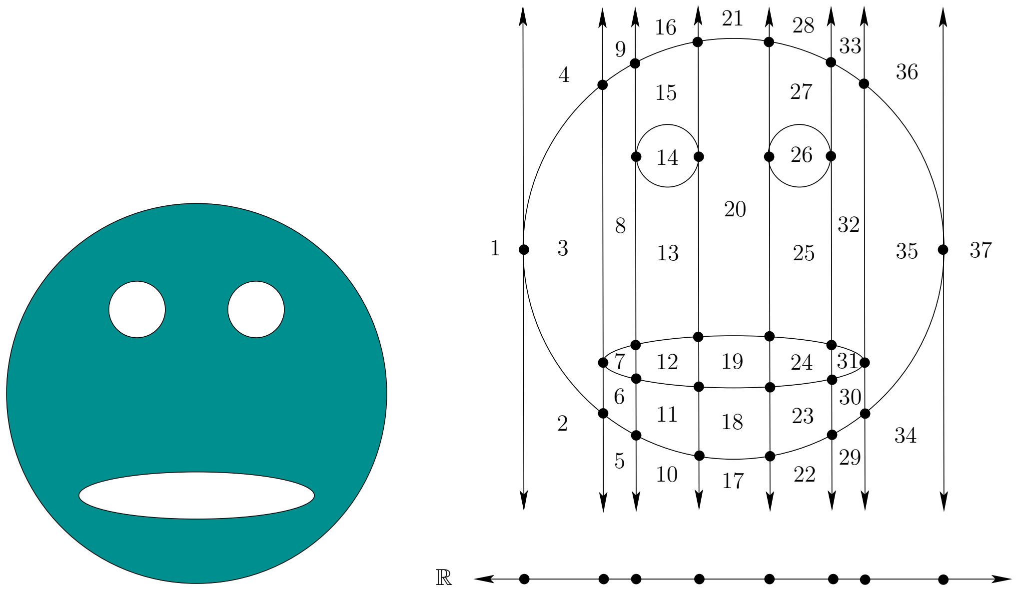
$O(n^5)$ time and space

Schwartz, Sharir, 1983

Cylindrical Algebraic Decomposition



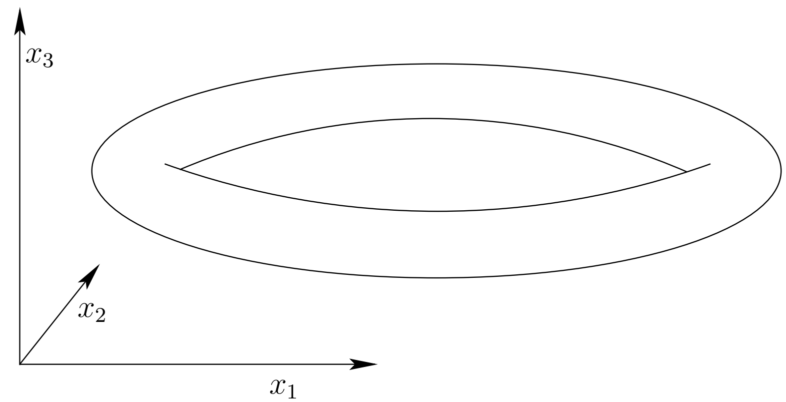
Developed by Collins to decide Tarski sentences

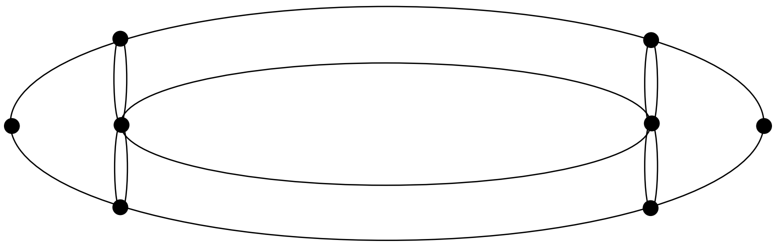


Doubly exponential time and space.

Schwartz, Sharir, 1983

Canny's Roadmap Algorithm





Singly exponential time.

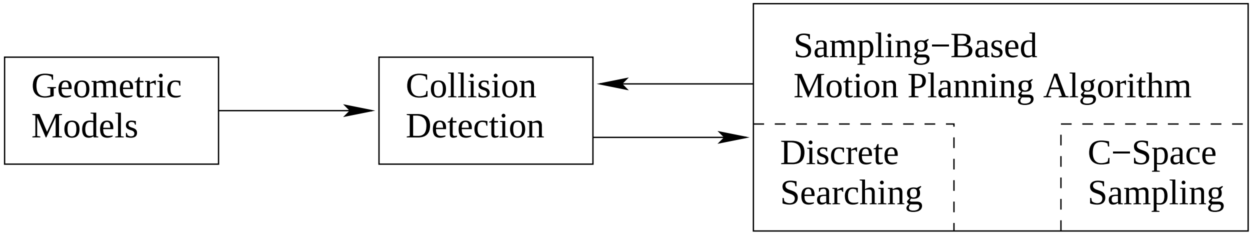
Canny, 1987

See *Algorithms in Real Algebraic Geometry* by Basu, Pollack, Roy, 2003.

More recently: Generalizations to o-minimal structures.

Sampling-Based Planning

Sampling-Based Planning: Philosophy



- Use collision detector to separate planning from input geometry
- Systematically sample (random vs. deterministic) the free space
- Single-query: Incremental sampling and searching
- Multiple-query: Precompute a sampling-based roadmap

Denseness

In topology, a set U is called *dense in* V if $cl(U) = V$.
Implication: Every open subset of V contains at least one point in U.

Example: The rational numbers $\mathbb{Q}$ are dense in $\mathbb{R}$
(every open interval contains some fractions)

If U is dense and countable, then a dense *sequence* can be formed:

$$\alpha : \mathbb{N} \to U$$

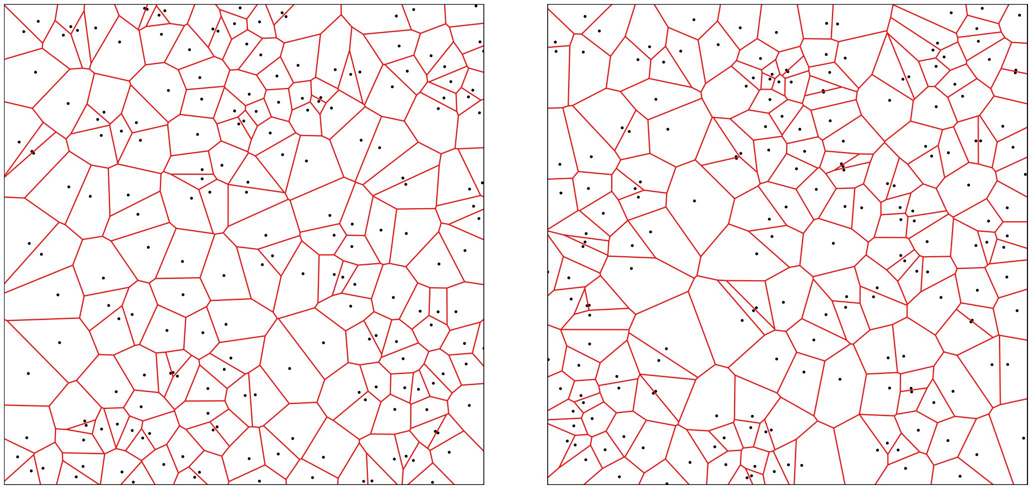
This imposes a linear ordering on U: $\alpha(1)$, $\alpha(2)$, $\ldots$

Example: A random sequence is dense with probability one.

(Pseudo-)Random Sequence

Uniform random samples seem easy to produce.

Statistical independence makes it easy to combine sample sets.



In reality, note that pseudo-random sequences are generated.

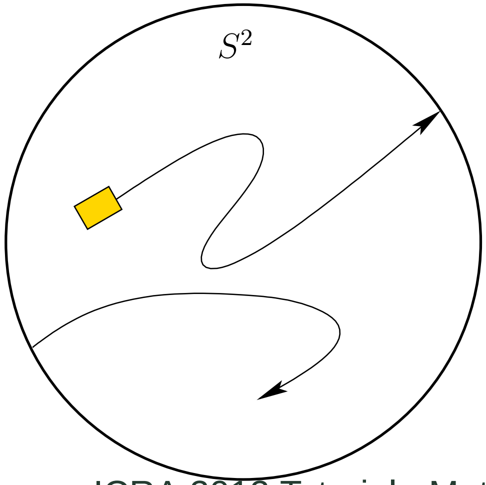
(Pseudo-)Random Sequence

Be careful in curved spaces!

To generate a random point on S^n: Generate n Guassian iid samples and normalize.

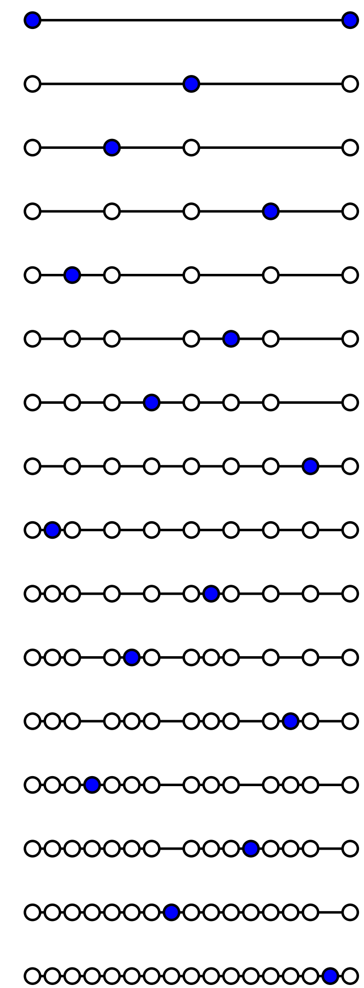
Uniform random rotation in $SO(3)$:
Choose three points $u_1, u_2, u_3 \in [0, 1]$ uniformly at random.
$(a, b, c, d) =$

$$(\sqrt{1-u_1}\sin 2\pi u_2,\ \sqrt{1-u_1}\cos 2\pi u_2,\ \sqrt{u_1}\sin 2\pi u_3,\ \sqrt{u_1}\cos 2\pi u_3).$$



Deterministic Alternative: van der Corput sequence

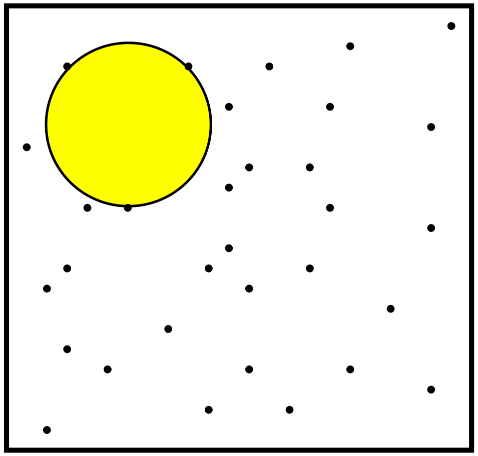
i	Naive Sequence	Binary	Reverse Binary	Van der Corput	Points in $[0,1]/\sim$
1	0	.0000	.0000	0	
2	1/16	.0001	.1000	1/2	
3	1/8	.0010	.0100	1/4	
4	3/16	.0011	.1100	3/4	
5	1/4	.0100	.0010	1/8	
6	5/16	.0101	.1010	5/8	
7	3/8	.0110	.0110	3/8	
8	7/16	.0111	.1110	7/8	
9	1/2	.1000	.0001	1/16	
10	9/16	.1001	.1001	9/16	
11	5/8	.1010	.0101	5/16	
12	11/16	.1011	.1101	13/16	
13	3/4	.1100	.0011	3/16	
14	13/16	.1101	.1011	11/16	
15	7/8	.1110	.0111	7/16	
16	15/16	.1111	.1111	15/16	



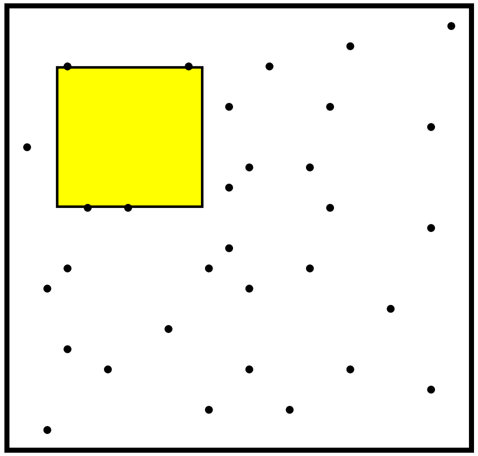
Dispersion

Let P be a finite set of points in metric space (X, ρ).
The *dispersion* of P is:

$$\delta(P) = \sup_{x \in X} \left\{ \min_{p \in P} \{ \rho(x, p) \} \right\}.$$



L_2 dispersion



L_∞ dispersion

In a bounded space, a dense sequence drives the dispersion to zero.

Deterministic Sequences

van der Corput is asymptotically optimal in terms of dispersion

Halton: Generalize van der Corput by using relatively prime bases $(2, 3, 5, 7, 11, ...)$ for each coordinate.

More uniform than random (which needs $O((\lg n)^{1/d})$ times as many samples needed to produce the same expected dispersion).

Other sequences produce better constants, and optimize discrepancy.
See *Random Number Generation and Quasi-Monte-Carlo Methods*, Niederreiter, 1992.

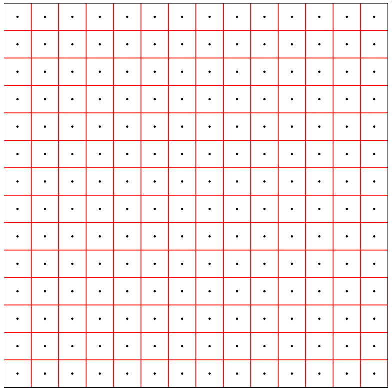
Best Possible Dispersion

Sukharev theorem:
For *any* set P of k samples in $[0,1]^d$:

$$\delta(P) \geq \frac{1}{2\left\lfloor k^{\frac{1}{d}} \right\rfloor}, \tag{1}$$

in which δ is the L_∞ dispersion.

The best possible placement of k points:



Think: “points per axes” for any sample set
Holding the dispersion fixed requires exponentially many points in dimension.

Johnson-Lindenstrauss Lemma

Maybe you didn't need all of those dimensions anyway...

Pick any positive $\epsilon < 1$ and any set P of k points in $\mathbb{R}^n$, there exists a function $f : \mathbb{R}^n \to \mathbb{R}^m$ so that for all $x, y \in P$,

$$(1-\epsilon)\|x-y\|^2 \leq \|f(x)-f(y)\|^2 \leq (1+\epsilon)\|x-y\|^2,$$

and $m = 4 \ln n / (\epsilon^2/2 - \epsilon^3/3)$.

In other words, a low-distortion, low-dimensional embedding exists.

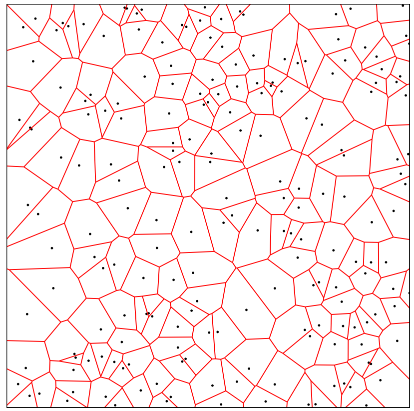
The basis of many dimensionality reduction methods, in machine learning, compressed sensing, computational geometry, ...

A Spectrum of Sample Sequences

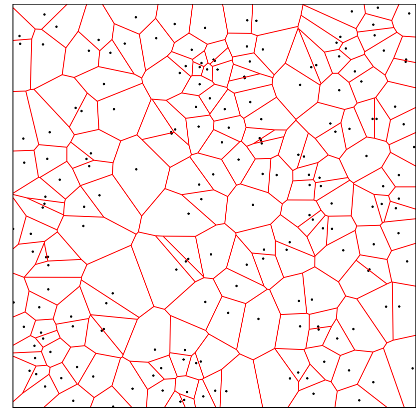
1. Random sample sequence – can these really be generated?
2. Pseudo-random sequence
3. Low-dispersion sequence
4. Multiresolution grid

Irregularity Does Not Help

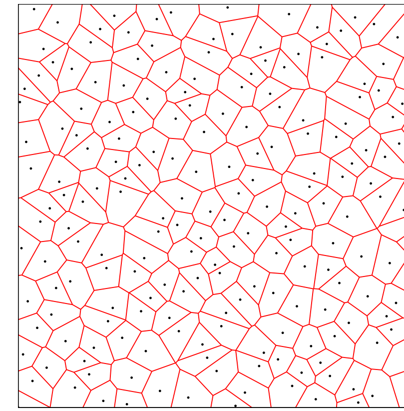
196 points in each square region:



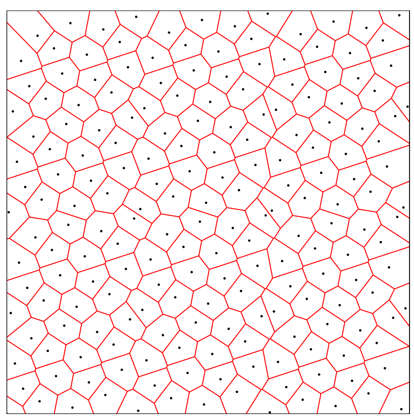
Pseudo-random points



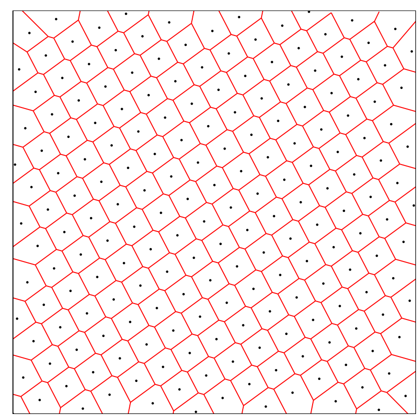
Pseudo-random points



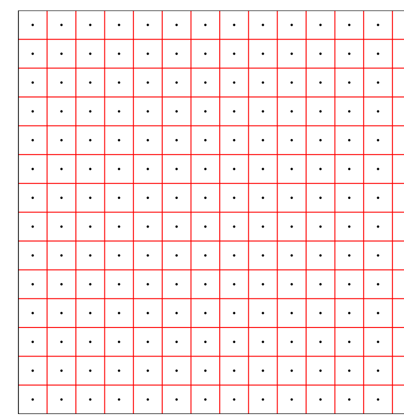
Halton points



Hammersley points



Lattice points



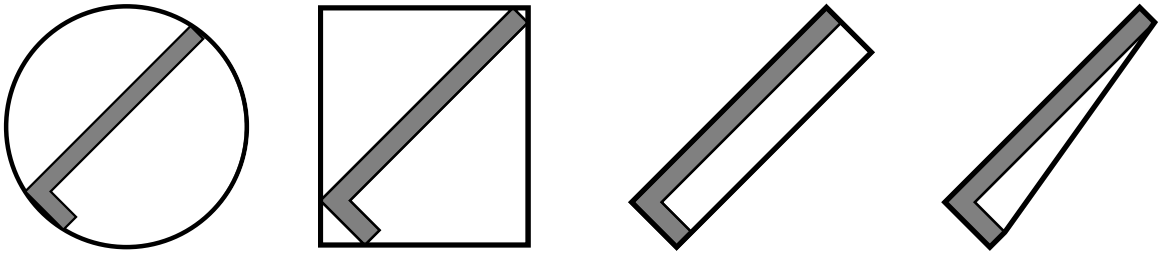
Sukharev grid

Collision Detection

Maintain a hierarchy of bounding regions

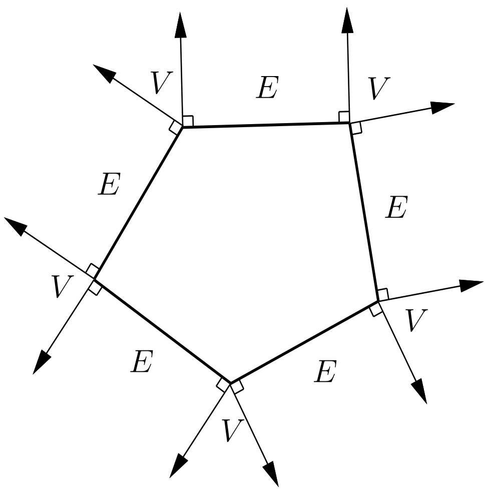
Two opposing criteria:

1. The region should fit the intended body points as tightly as possible.
2. The intersection test for two regions should be as efficient as possible.



Popular packages from UNC: PQP, I-Collide, ...

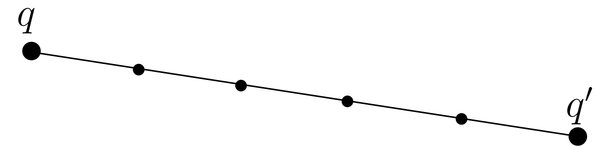
Important Issue: Incremental Methods



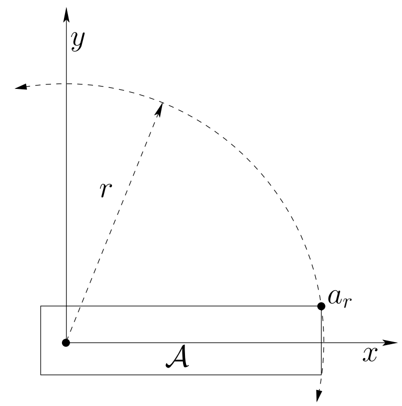
- **Vertex-Vertex** Each point of the closest pair is a vertex of a polygon.
- **Edge-Vertex** One point of the closest pair lies on an edge, and the other lies on a vertex.
- **Edge-Edge** Each point of the closest pair lies on an edge. In this case, the edges must be parallel.

Important Issue: Collision Checking a Segment

How many collision checks should be performed along an edge?



Using workspace distance information, may be able to guarantee collision-free segments.



Let $a(q) \in \mathcal{A}(q)$ denote a point on the robot.
Find a constant $c > 0$ so that

$$\|a(q) - a(q')\| < c\|q - q'\| \tag{3}$$

over robot points and configuration pairs q, q'.

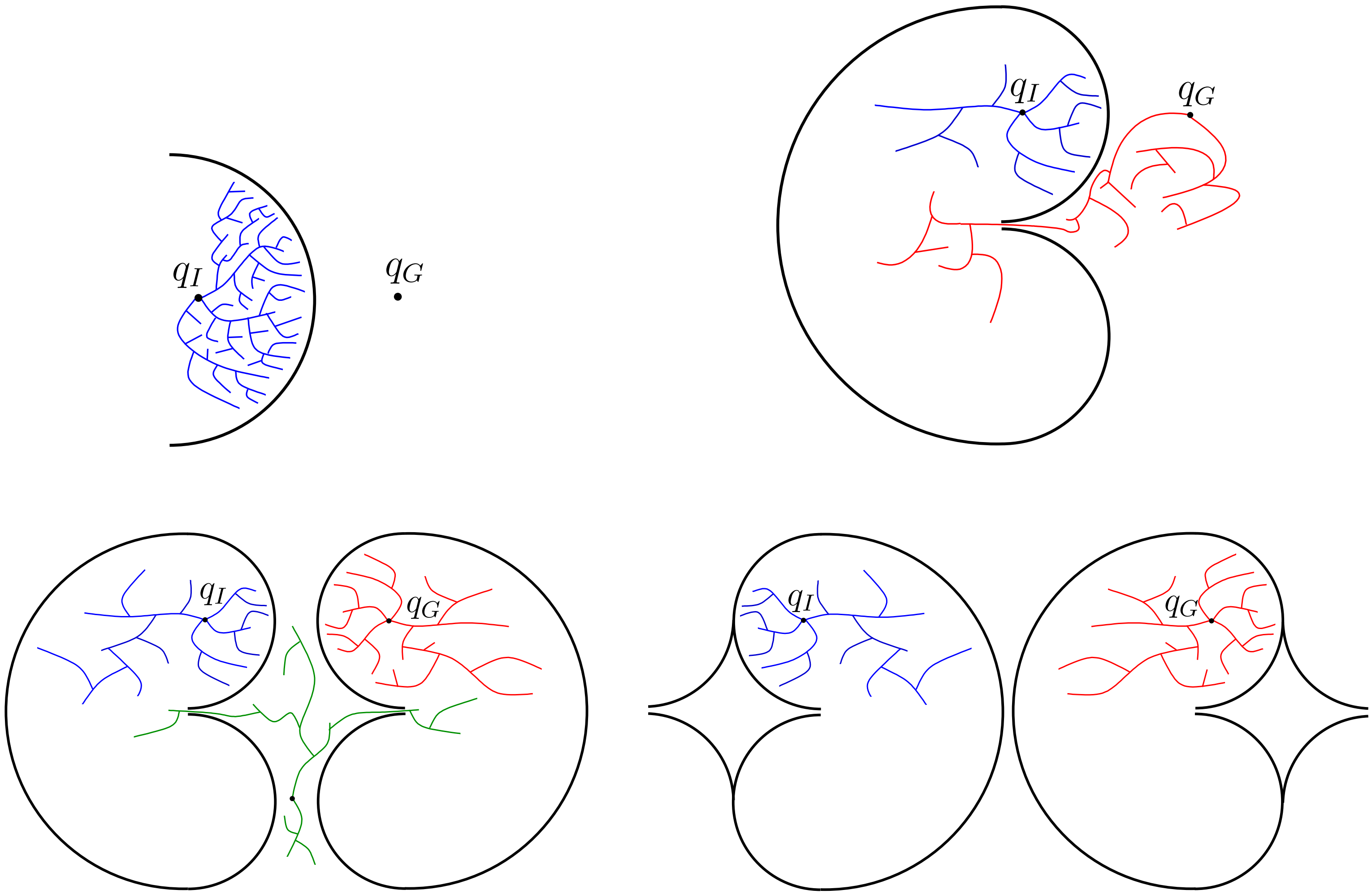
Framework: Incremental Sampling and Searching

Given a single query: $q_I, q_G \in \mathcal{C}_{free}$

---

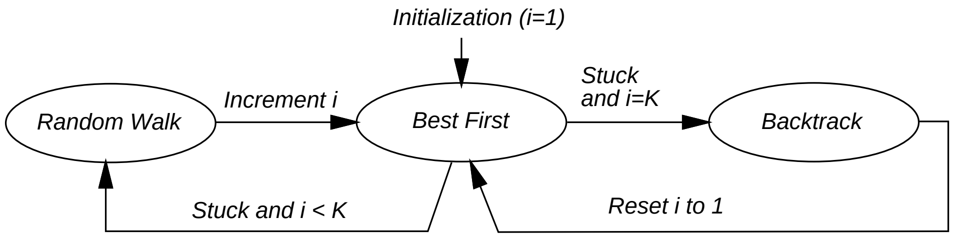
1. **Initialization:** Form $\mathcal{G}(V, E)$ with vertices q_I, q_G and no edges.
2. **Vertex Selection Method (VSM):** Choose a vertex $q_{cur} \in V$ for expansion.
3. **Local Planning Method (LPM):** For some $q_{new} \in \mathcal{C}_{free}$, attempt to construct a path $\tau_s : [0,1] \rightarrow \mathcal{C}_{free}$ such that $\tau(0) = q_{cur}$ and $\tau(1) = q_{new}$.
4. **Insert an Edge in the Graph:** Insert τ_s into E, as an edge from q_{cur} to q_{new}. If q_{new} is not already in V, then it is inserted.
5. **Check for a Solution:** Determine whether $\mathcal{G}$ encodes a solution path.
6. **Return to Step 2:** Iterate unless a solution has been found or the algorithm reports failure.

A convenient way to express the challenges of incremental sampling and searching

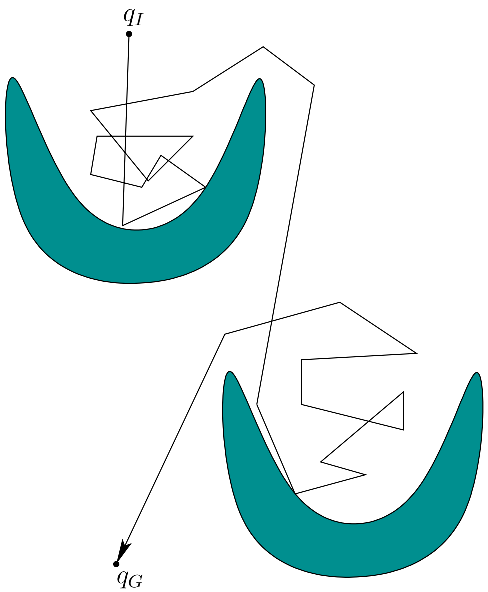


Barraquand, Latombe, 1989

Use BFS on an implicit, high resolution grid. Use random walks to escape local minima.



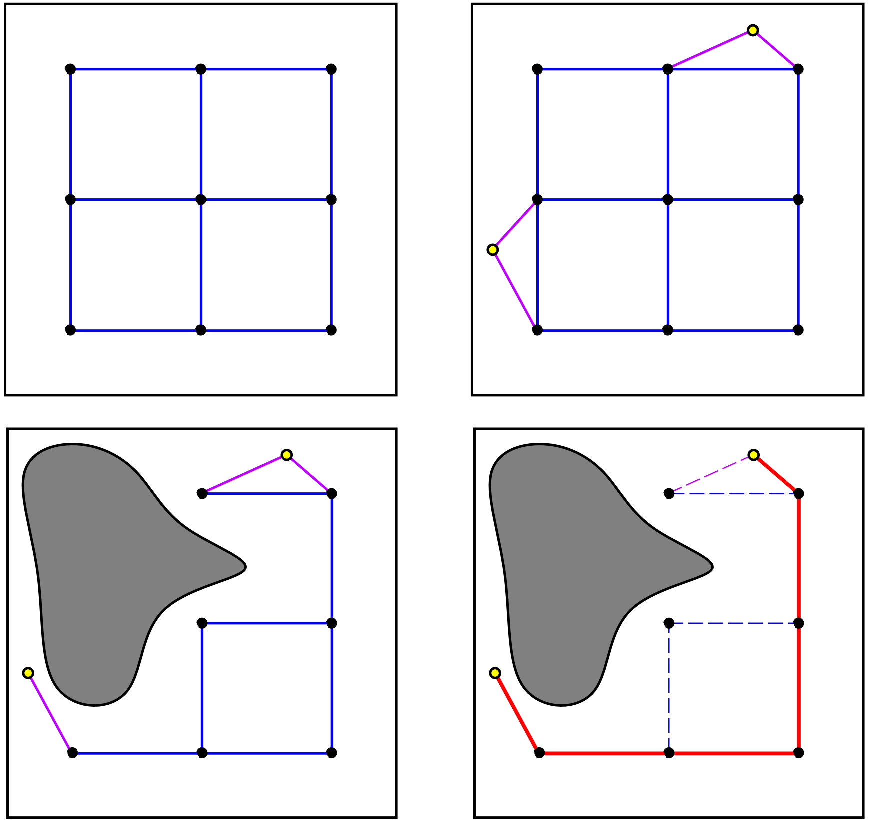
It was able to solve high dimensional problems, but required too much parameter tuning.



Implicit Low Resolution Grid

Connect q_I and q_G to the grid.

Apply classical grid search: BFS, DFS, Dijkstra, A^*



Some Other Incremental Planners

- Ariadne's clew algorithm (Mazer et al. 1992)
- Expansive space planner (Hsu et al., 1997)
- Rapidly exploring Random Trees (LaValle, Kuffner, 1998)
- SBL planning (Sanchez, Latombe, 2001)
- Adaptive random walk planner (Carpin, Pillonetto, 2005)

Making a “Random” Tree

Suppose $X = [-25, 25]^2$ and $q_I = (0, 0)$.

Pick a vertex a random, extend one unit in a random direction repeat, ...

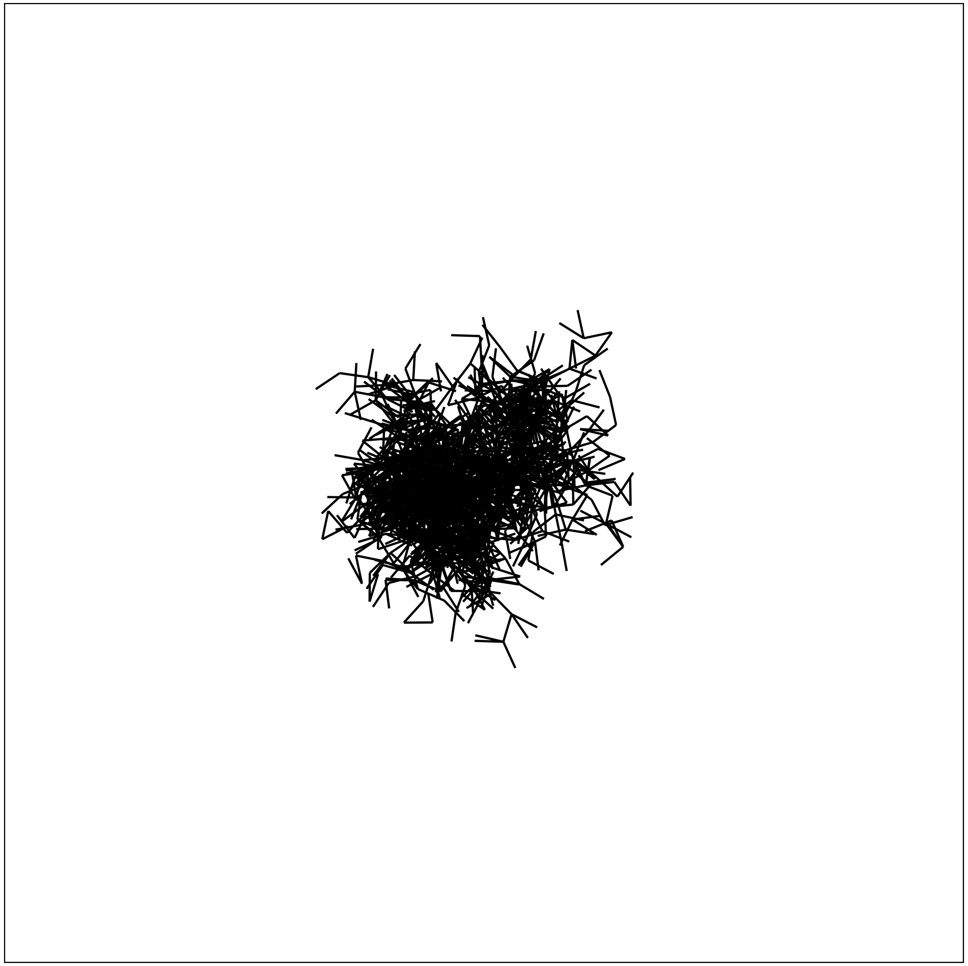
What happens?

Making a “Random” Tree

Suppose $X = [-25, 25]^2$ and $q_I = (0, 0)$.

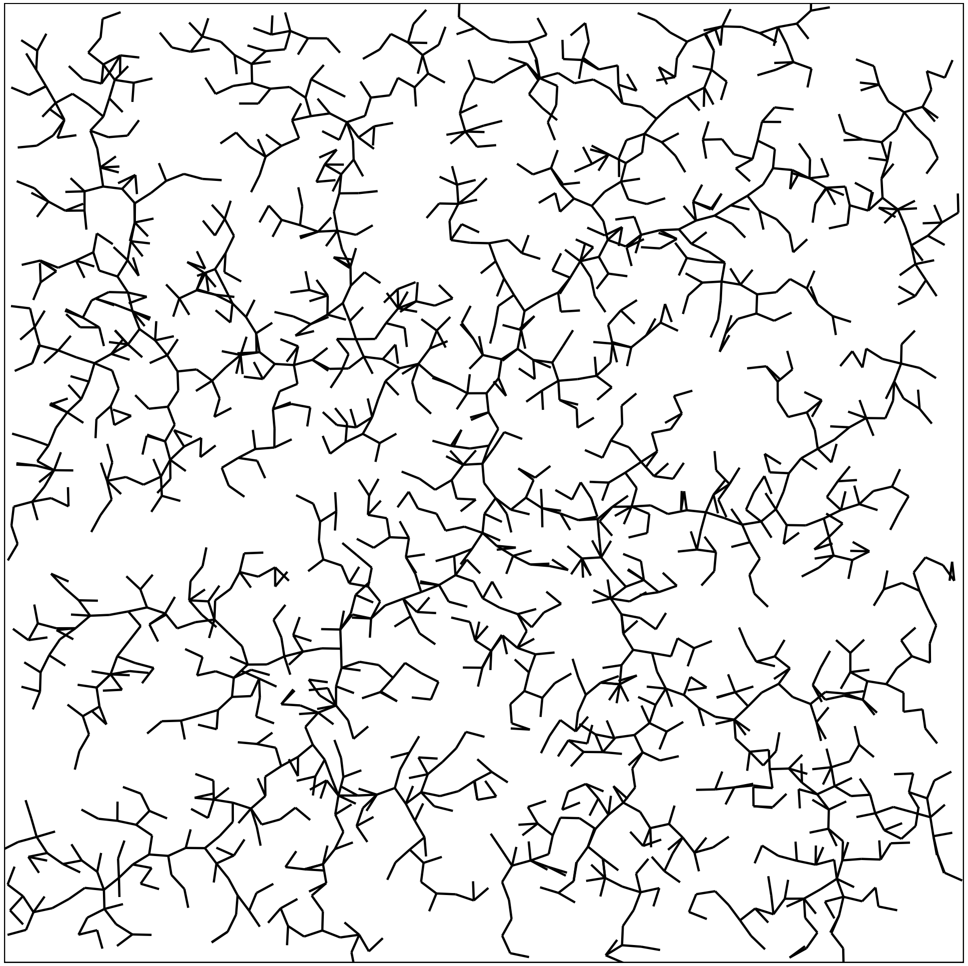
Pick a vertex a random, extend one unit in a random direction repeat, ...

What happens?



Rapidly exploring Random Tree

By changing the vertex selection method, we obtain this:



Rather than pick a *vertex* at random, pick a *configuration* at random.

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

RRT


```
SIMPLE_RRT(q_0)
  G.init(q_0);
  for i = 1 to k do
      q_r ← RandomConf(i);
      G.add_vertex(q_r);
      q_n ← NEAREST(S(G), q_r);
      G.add_edge(q_n, q_r);
```

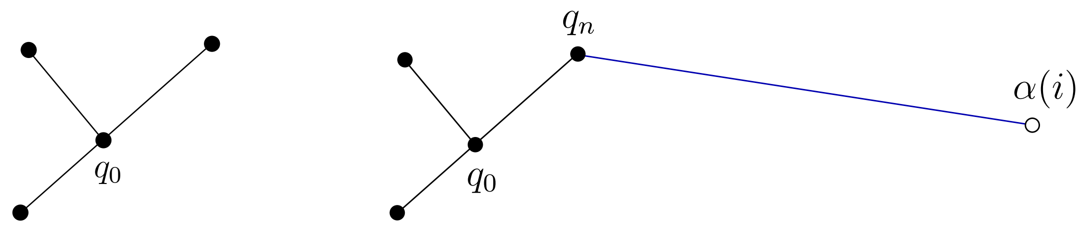
SIMPLE_RRT(q_0)

To bias toward the goal, q_G can be substituted for $RandomConf(i)$ in some (e.g., every 100) iterations.

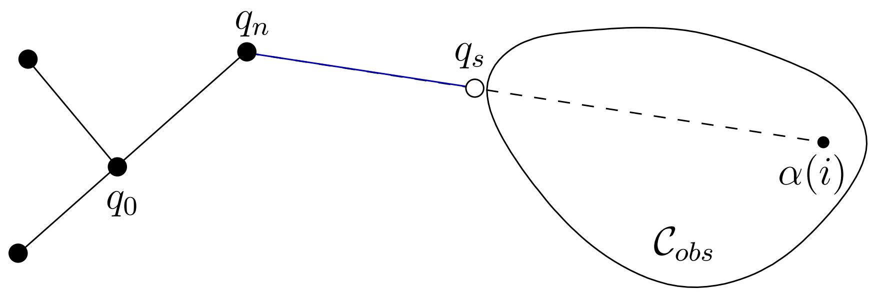
RandomConf(i) can be replaced by any dense sequence $\alpha(i)$ to obtain Rapidly exploring Dense Trees (RDTs).

RRT: Implementation Issues

Extend the nearest vertex (using the metric!) to the random point:

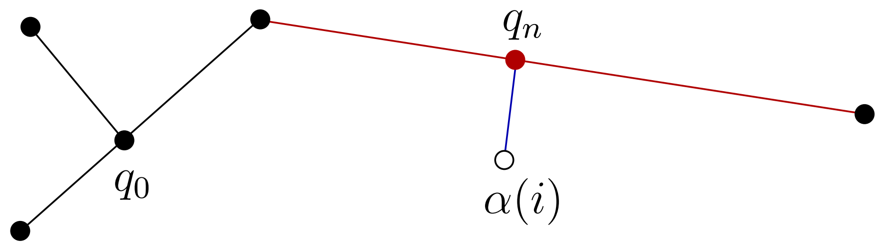


If there is an obstacle, then stop short:

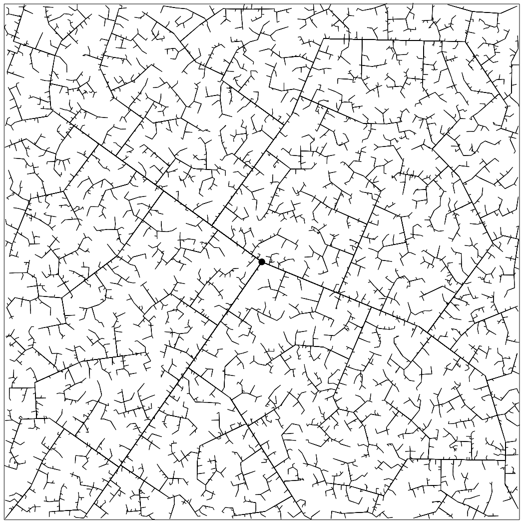


RRT: Implementation Issues

If the nearest RRT point lies in an edge, it is better to extend from there:



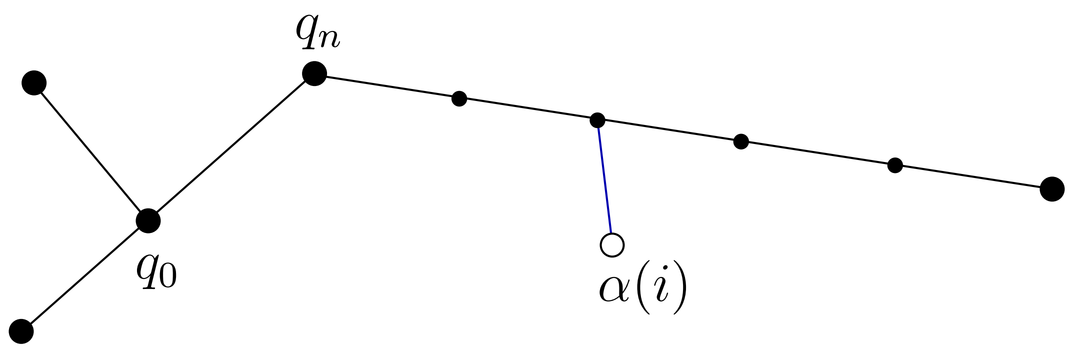
45 iterations



2345 iterations

RRT: Implementation Issues

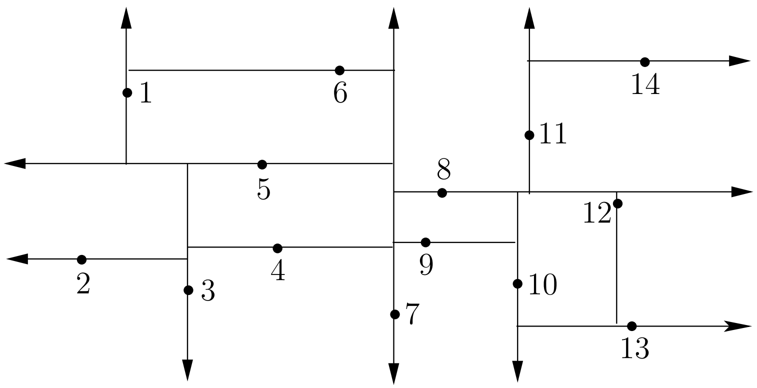
An approximate solution: Insert intermediate vertices.

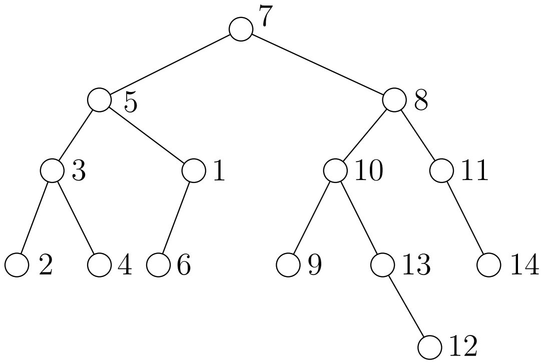


RRT: Implementation Issues

For large RRTs (thousands of nodes), nearest-neighbor requests dominate.

In some settings, Kd-trees can dramatically improve performance.



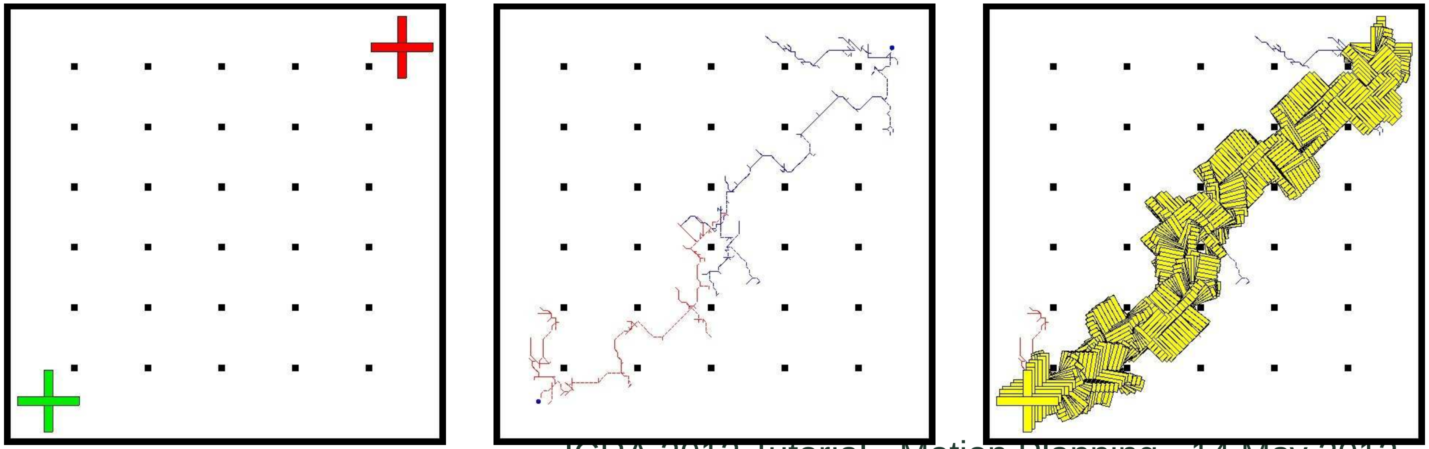


Bidirectional Search

To solve a q_I, q_G query with RRTs.
Grow two trees: 1) T_i from q_I and 2) T_g from q_G.

Repeat the following four steps:

1. Extend T_i using $\alpha(i)$, making q_{new}.
2. Extend T_g using q_{new}. If connected, then solution found.
3. Extend T_g using $\alpha(i+1)$, making q_{new}.
4. Extend T_i using q_{new}. If connected, then solution found.

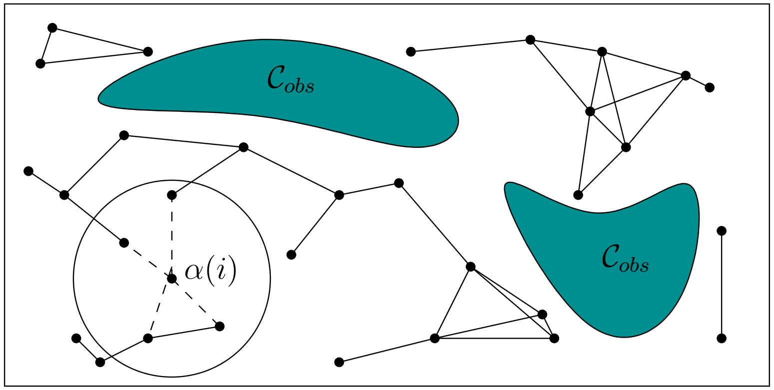


Single vs. Multiple Query

If there are multiple queries in the same $\mathcal{C}_{free}$, then precomputing a roadmap may pay off.

BUILD_ROADMAP

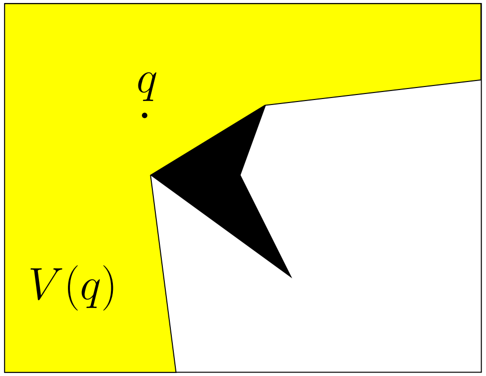
$\mathcal{G}$.init(); $i \leftarrow 0$;
while $i < N$
  if $\alpha(i) \in \mathcal{C}_{free}$ **then**
    $\mathcal{G}$.add_vertex($\alpha(i)$); $i \leftarrow i + 1$;
    for each $q \in$ NEIGHBORHOOD($\alpha(i)$,$\mathcal{G}$)
      if ((**not** $\mathcal{G}$.same_component($\alpha(i), q$)) **and** CONNECT($\alpha(i), q$)) **then**
        $\mathcal{G}$.add_edge($\alpha(i), q$);



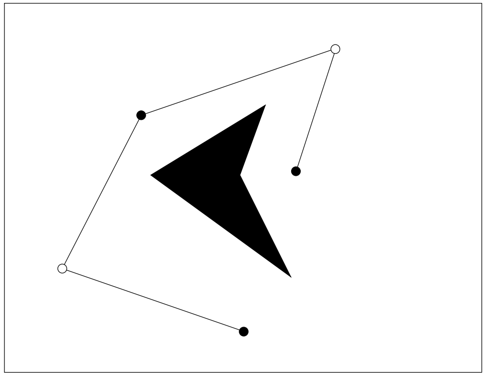
Connection rules:

- **Nearest K:** The K closest points to $\alpha(i)$ are considered. This requires setting the parameter K (a typical value is 15).
- **Component K:** Try to obtain up to K nearest samples from each connected component of $\mathcal{G}$.
- **Radius:** Take all points within a ball of radius r centered at $\alpha(i)$.
- **Visibility:** Try connecting α to all vertices in $\mathcal{G}$.

Sampling strategies: Gaussian, medial axis, bridge-test, ...
See Karaman, Frazzoli, IJRR 2011 for PRM connection theory.



Visibility definition



(b) Visibility roadmap

Simeon, Laumond, Nissoux, 2000

Define two different kinds of vertices in $\mathcal{G}$:

Guards: To become a *guard*, a vertex, q must not be able to see other guards.

Connectors: To become a *connector*, a vertex, q, must see at least two guards.

Differential Constraints

Planning Under Differential Constraints

Due to robot kinematics and dynamics, most systems are *locally* constrained, in addition to *global* obstacles.

Let $\dot{q}$ represent the C-space velocity.
In ordinary planning, any "direction" is allowed and the magnitude does not matter.

Thus, we could say

$$\dot{q} = u \tag{4}$$

and $u \in \mathbb{R}^n$ may be any velocity vector so that $\|u\| \leq 1$.

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Planning Under Differential Constraints

More generally, a *control system* (or *state transition equation*) constrains the velocity:

$$\dot{q} = f(q, u)$$

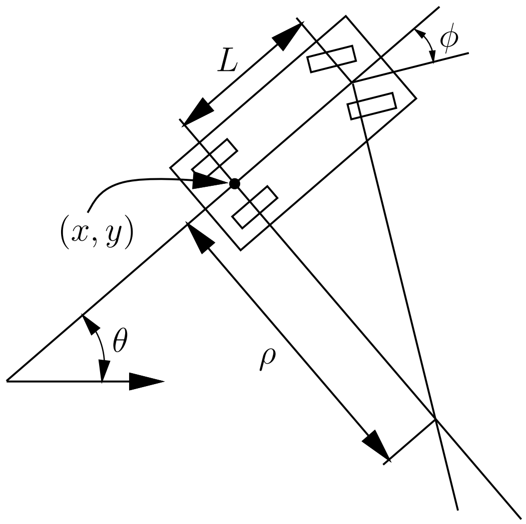
and u belongs to some set U (usually bounded).

A function $\tilde{u} : T \to U$ is applied over a time interval $T = [0, t_f]$ and the configuration $q(t)$ at time t is given by the state at time t is given by

$$q(t) = q(0) + \int_0^t f(q(t'), \tilde{u}(t'))dt'.$$

in which $q(0)$ is the initial configuration.

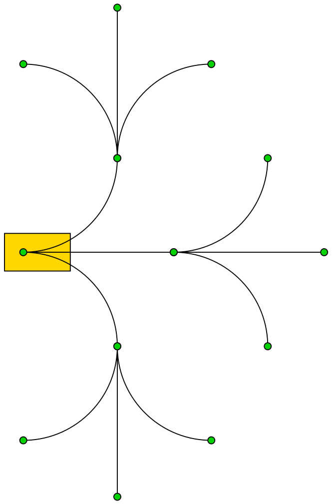
This car drives forward only:



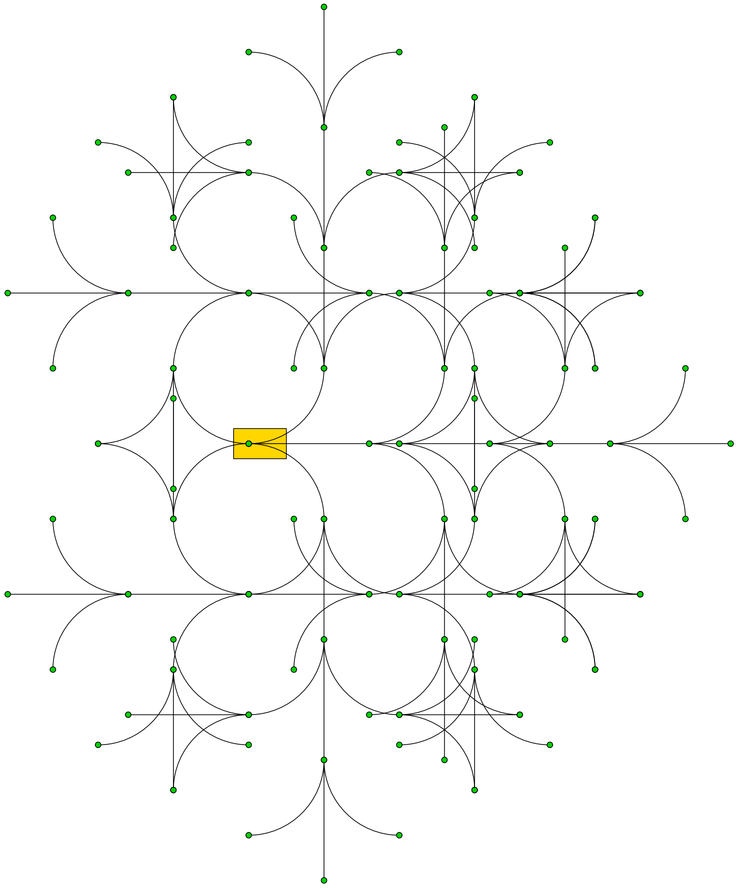
$\mathcal{C} = \mathbb{R}^2 \times S^1$.
Let $u = (u_s, u_\phi)$ and $U = [0, 1] \times [-\phi_{max}, \phi_{max}]$.
Control system of the form $\dot{q} = f(q, u)$:

$$\begin{aligned} \dot{x} &= \cos\theta \\ \dot{y} &= \sin\theta \\ \dot{\theta} &= \frac{u_s}{L}\tan u_\phi. \end{aligned} \tag{5}$$

Stepping forward in the Dubins car



Two stages



Four stages

Moving Into the Phase Space

Handling higher order derivatives on $\mathcal{C}$ allows dynamical system models.
This includes accelerations, momentum, drift.

Let X be a *state space* (or *phase space*).
Typically, $X = \mathcal{C} \times \mathbb{R}^n$, in which n is the dimension of $\mathcal{C}$.
Each $x \in X$ represents a $2n$ dimensional vector $x = (q, \dot{q})$.

A control system then becomes

$$\dot{x} = f(x, u)$$

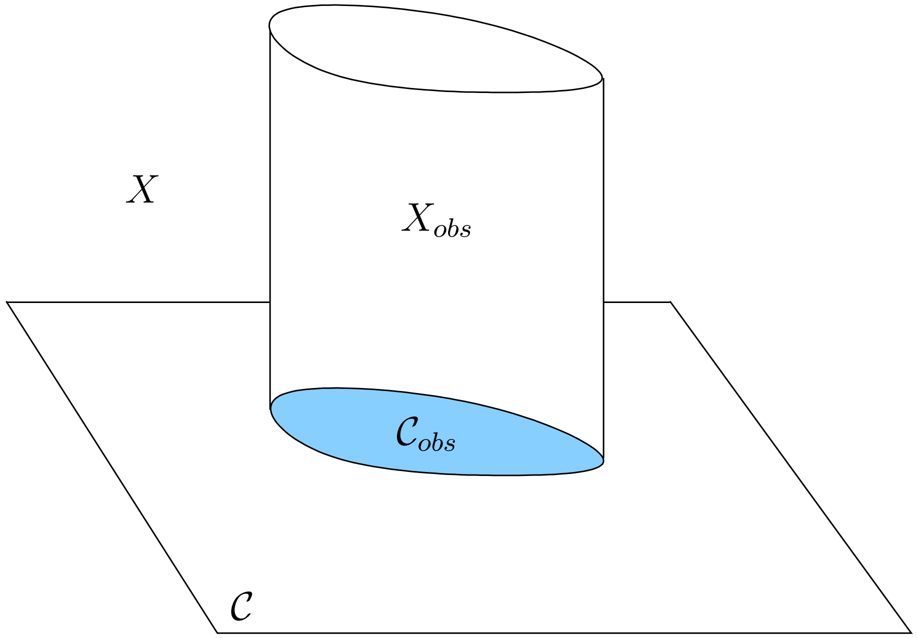
Note that $\dot{x}$ includes $\ddot{q}$ components (hence, acceleration constraints).

Moving Into the Phase Space

The obstacle region in X is usually:

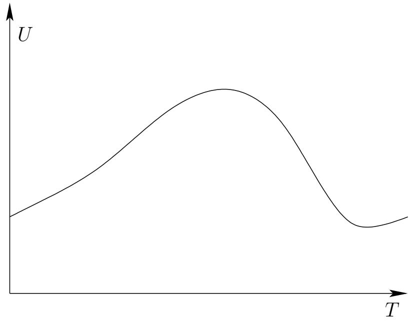
$$X_{obs} = \{x \in X \mid \kappa(x) \in \mathcal{C}_{obs}\},$$

This has cylindrical structure:

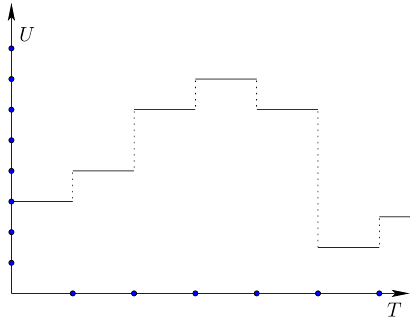


Resolution Complete Planning

Discretize time and action space:

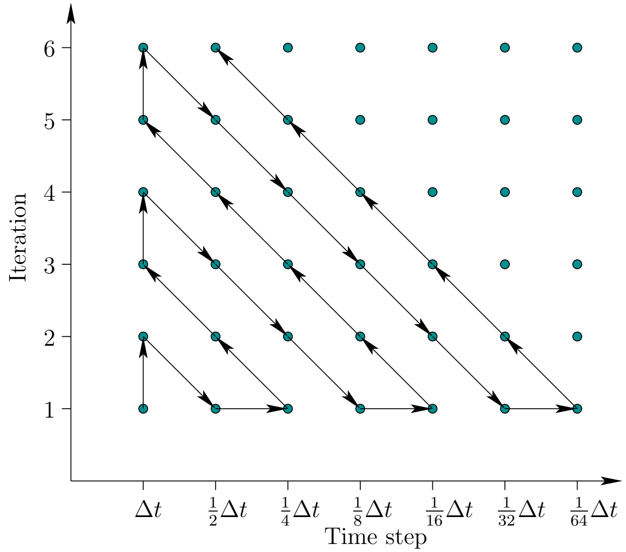


A trajectory in $\mathcal{U}$

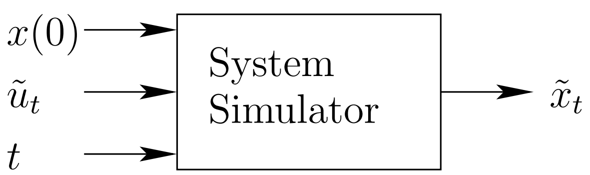


A trajectory in $\mathcal{U}_d$

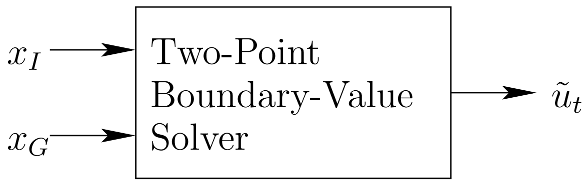
Using countability to try all sequences:



Local Planning

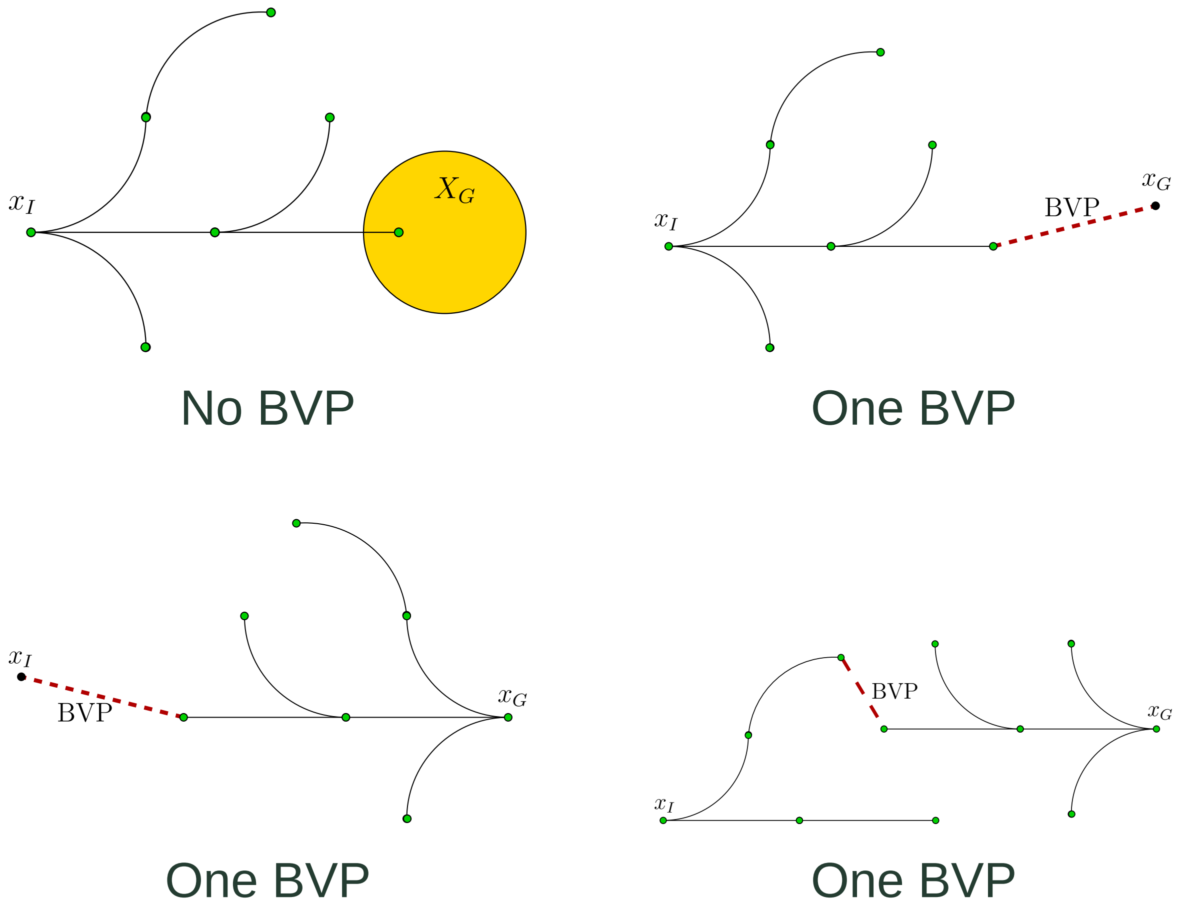


Easy

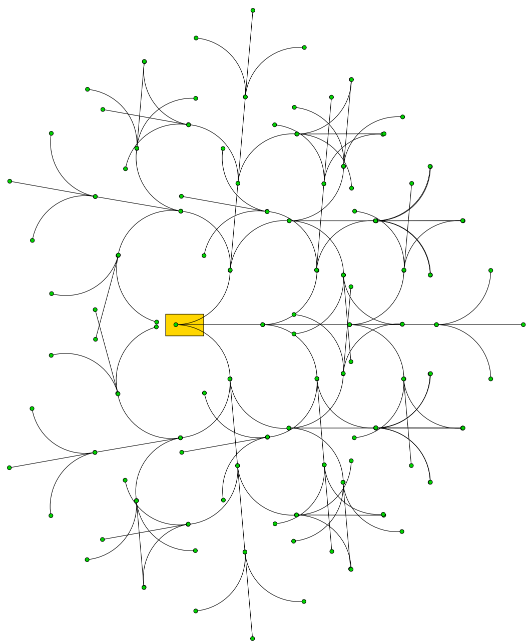


Usually Hard

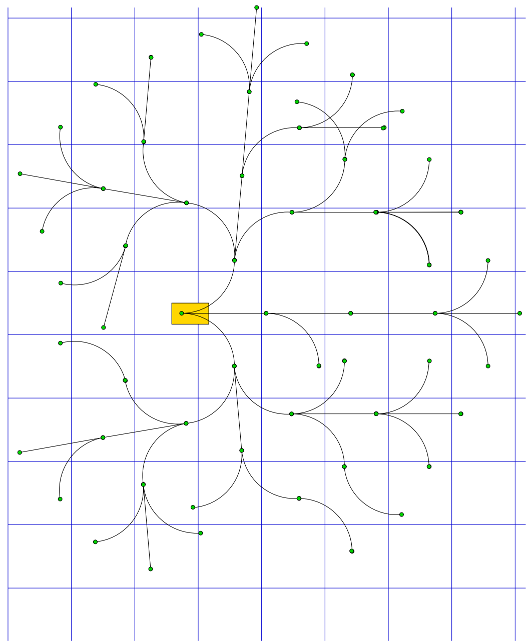
Boundary Value Problems



Limiting Vertices Per Cell



Four stages for Dubins



Limiting one vertex per cell

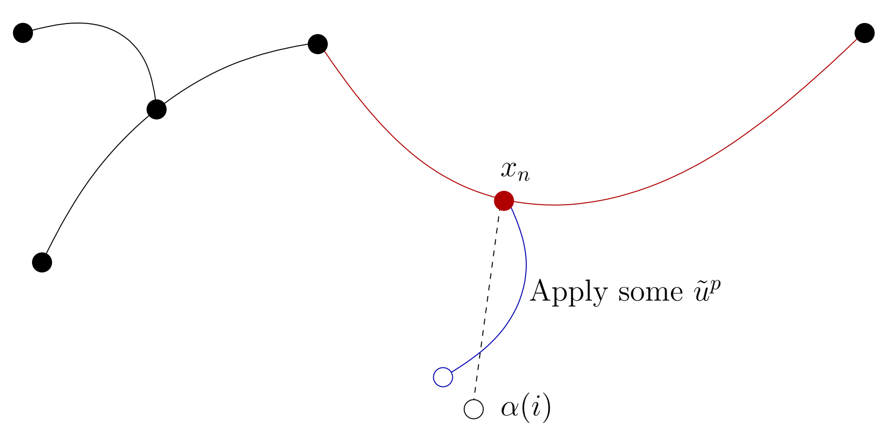
Barraquand, Latombe, 1993

Incremental Sampling and Searching

For an RRT, just replace the “straight line” connection with a local planner.

SIMPLE_RRT_WITH_DIFFERENTIAL_CONSTRAINTS(x_0)

$\mathcal{G}$.init(x_0);
for $i = 1$ **to** k **do**
    $x_n \leftarrow$ NEAREST($S(\mathcal{G}), \alpha(i)$);
    $(\tilde{u}^p, x_r) \leftarrow$ LOCAL_PLANNER($x_n, \alpha(i)$);
    $\mathcal{G}$.add_vertex(x_r);
    $\mathcal{G}$.add_edge($\tilde{u}^p$);



Problems: Need good metrics and primitives

Summary of Part II

- Combinatorial vs. sampling based.
- For some problems, combinatorial is far superior.
- For most “industrial problems” sampling-based works well.
- Weaker notions of completeness are tolerated.
- Dimensionality always an issue (Sukharev).

More details: *Planning Algorithms*, Chapters 5 and 6.

Homework 2: Solve During Lunch Break

For an infinite sample sequence $\alpha : \mathbb{N} \to X$, let α_k denote the first k samples.

Find a metric space $X \subseteq \mathbb{R}^n$ and α so that:

1. The dispersion of α_k is ∞ for all k.
2. The dispersion of α is 0.

Hint: Do not make it too complicated.