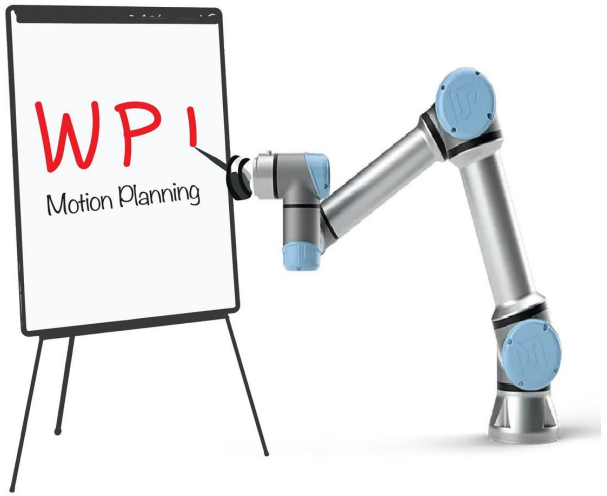
RBE550
Motion Planning
Probabilistic Roadmaps



Constantinos Chamzas
www.cchamzas.com
www.elpislab.org

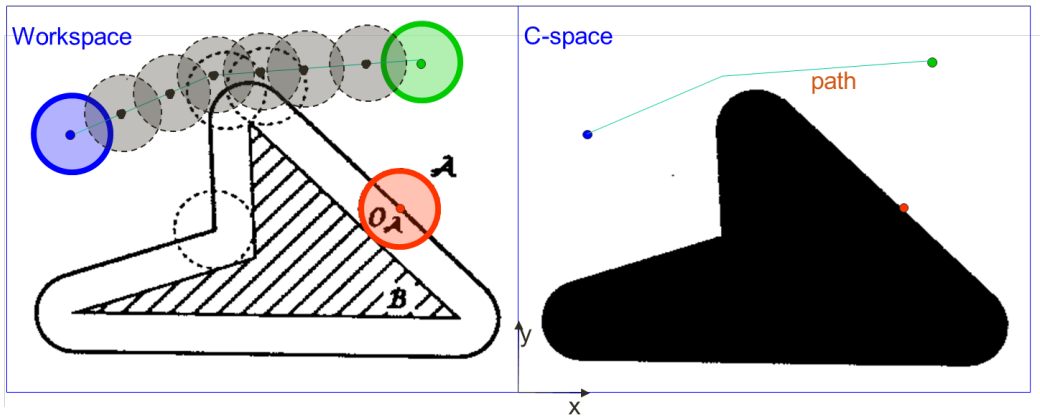
Disclaimer and Acknowledgments

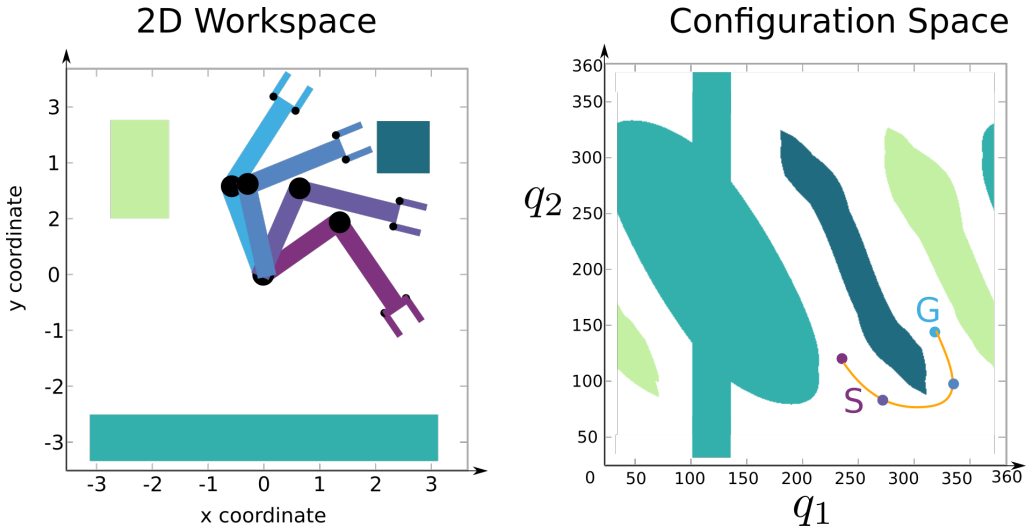
The slides are a compilation of work based on notes and slides mainly from from Erion Plaku, but also Constantinos Chamzas, Lydia Kavraki, Howie Choset, Morteza Lahijanian, Constantinos Chamzas, David Hsu, Greg Hager, Mark Moll, G. Ayorkor Mills-Tetty, Hyungpil Moon, Zack Dodds, Zak Kingston, Nancy Amato, Steven Lavalle, Seth Hutchinson, George Kantor, Dieter Fox, Vincent Lee-Shue Jr., Prasad Narendra Atkar, Kevin Tantiseviand, Bernice Ma, David Conner, and students taking comp450/comp550 at Rice University.

Last time: Configuration space obstacles

- Configuration Space
  - Obstacles
  - Paths

- 2-D disc robot (translation only)



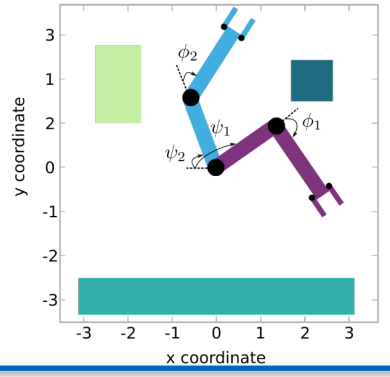


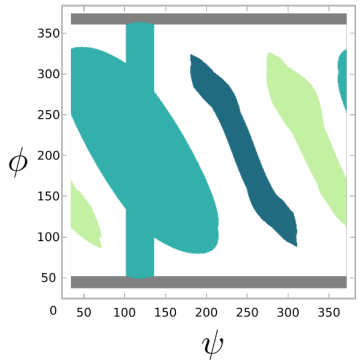
Last time: Configuration space obstacles

Workspace

Configuration Space

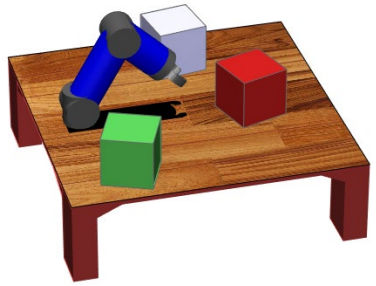
2-Dof Robot



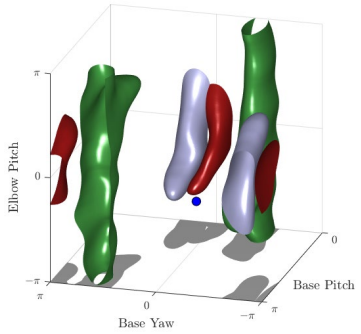


2 Dimensions

3-Dof Robot

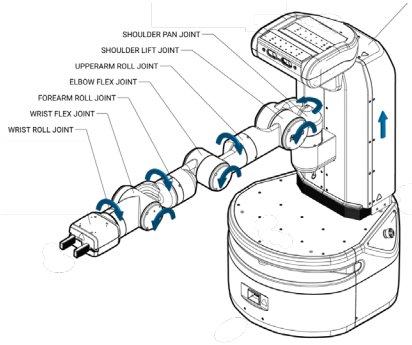


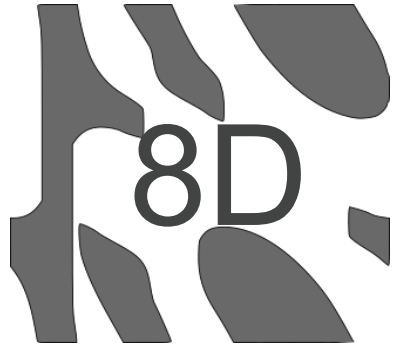
Das et al, *TRO* 2020



3 Dimensions

8-Dof Robot





8 Dimensions

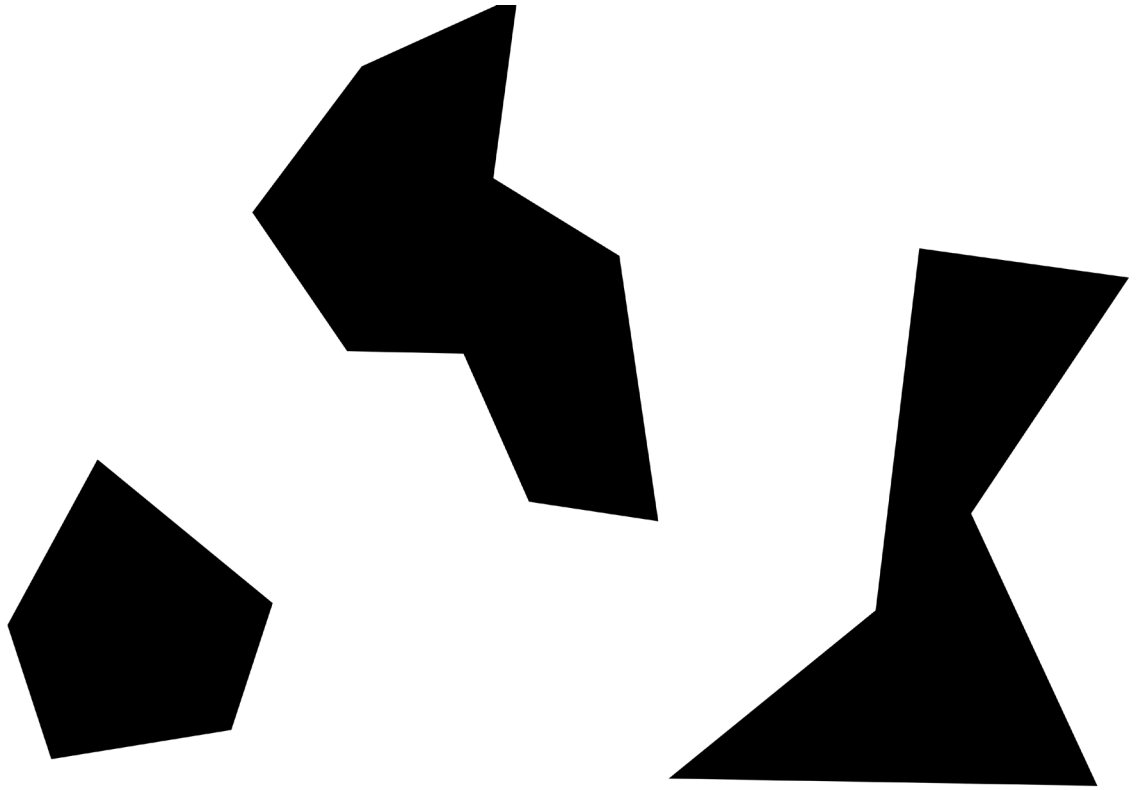
Can we compute these high-Dof obstacles?

Overview

- Probabilistic Roadmaps
- Examples
- Path Post Processing
- Nearest –Neighbor Strategy
- Lazy PRM
- Sampling Strategy

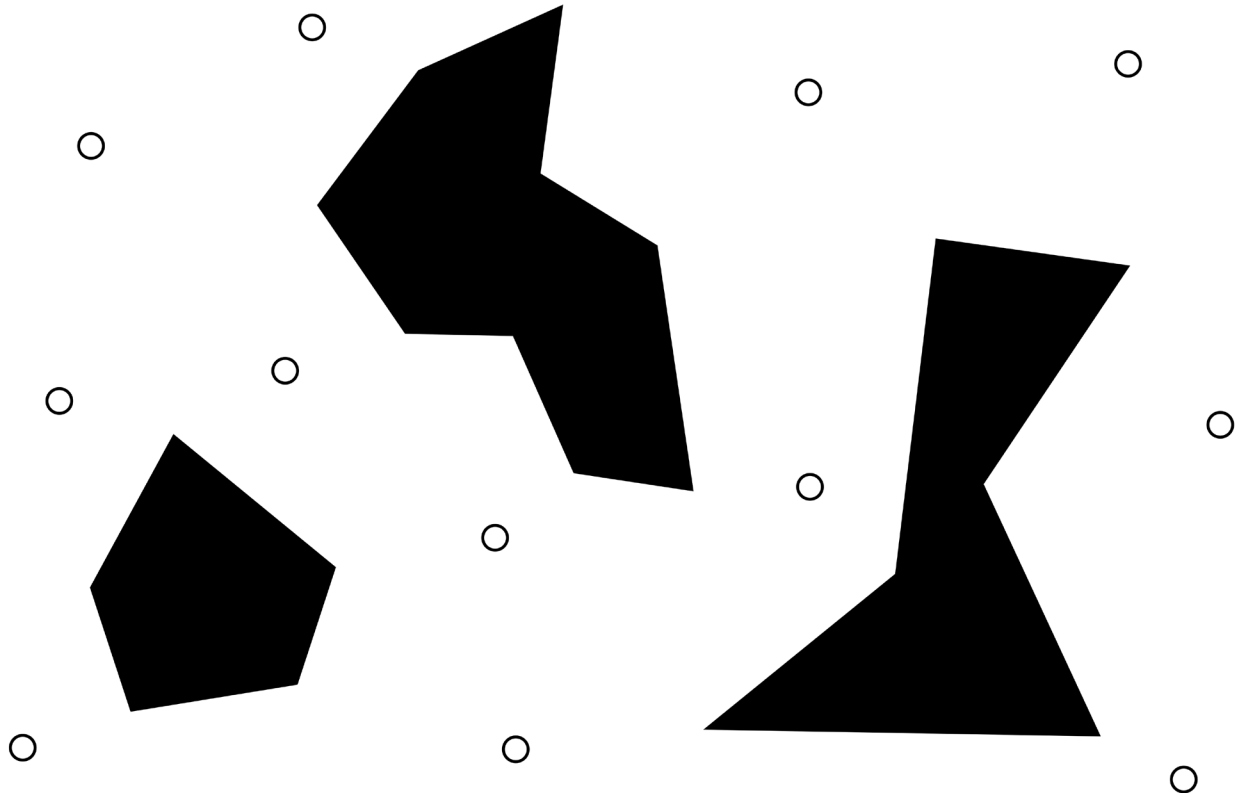
Probabilistic RoadMap – Main Idea

Main Idea: Sample random points in the configuration space without creating the c-space obstacles!



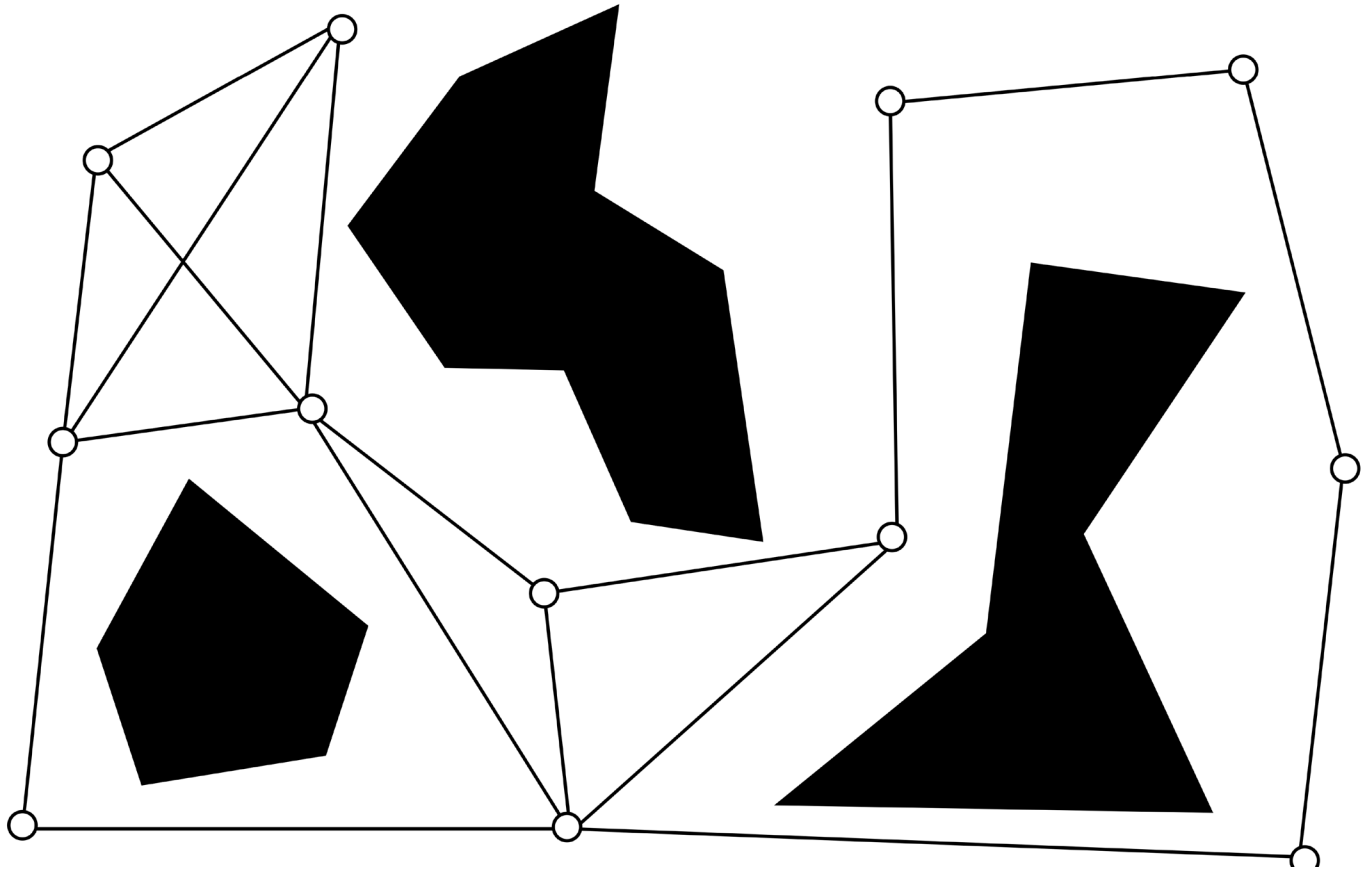
Probabilistic RoadMap – Main Idea

Main Idea: Sample random points in the configuration space without creating the c-space obstacles!



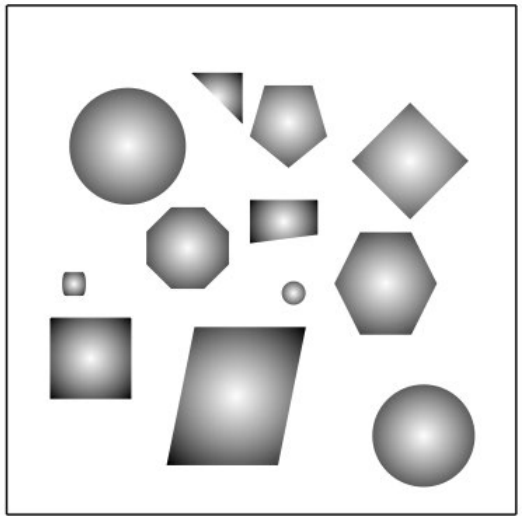
Probabilistic RoadMap – Main Idea

Main Idea: Sample random points in the configuration space without creating the c-space obstacles!

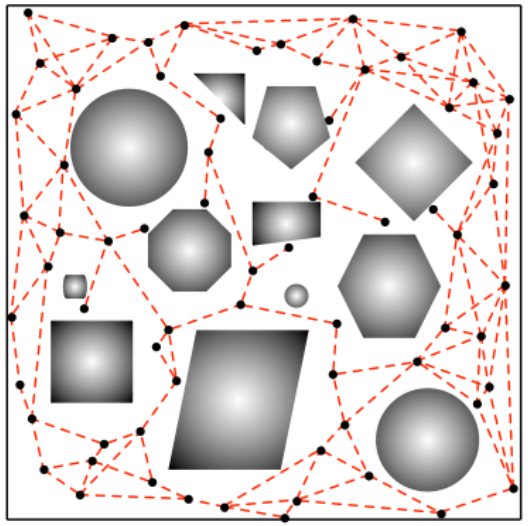


Probabilistic RoadMap

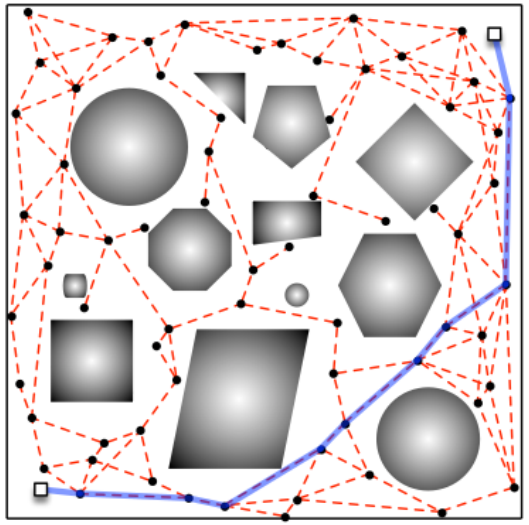
- Probabilistic Roadmap (PRM) is sampling-based technique for multi-query planning problems



PRM consists of 2 phases



1. Learning phase
   - Given robot and workspace
   - Generate a graph



2. Query phase
   - Given start and goal configuration
   - Graph search (A*)

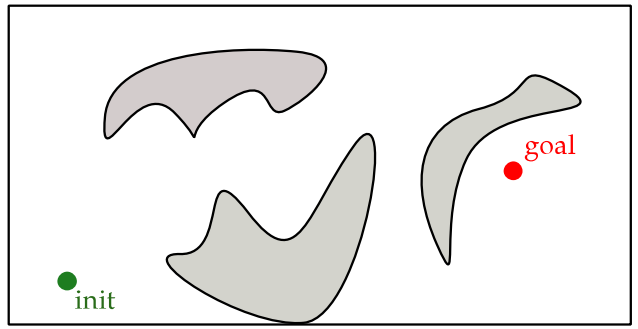
Multi-query Planner: For the same robot, workspace, only they Query Phase needs to be executed

Probabilistic RoadMap

- Construction of the Roadmap:

1. **Initialization**

   add q_{init} and q_{goal} to roadmap (graph) vertex set V if you have these. Otherwise V={}.



2. **Sampling**

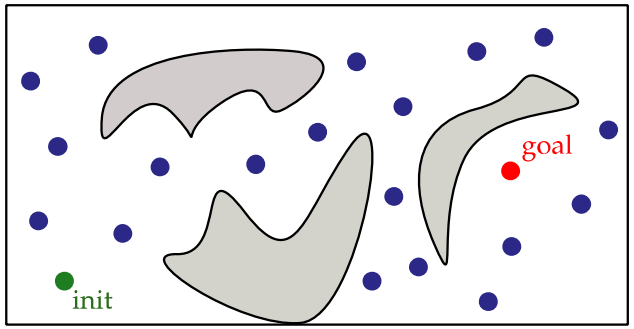
   repeat several times

   $q \leftarrow Sample()$

   if $\text{IsCollisionFree}(q) = true$

   add q to roadmap vertex set V

   end if



Probabilistic RoadMap

3. Connect Samples

for each pair of neighboring samples

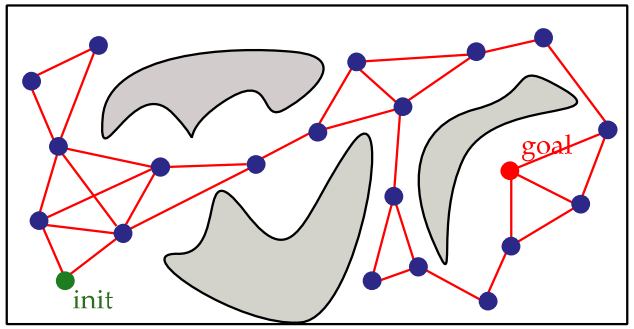
$(q_a, q_b) \in V \times V$

$path \leftarrow \text{GenerateLocalPath}(q_a, q_b)$

if $\text{IsCollisionFree}(path) = true$

add (q_a, q_b) to roadmap edge set E

end if



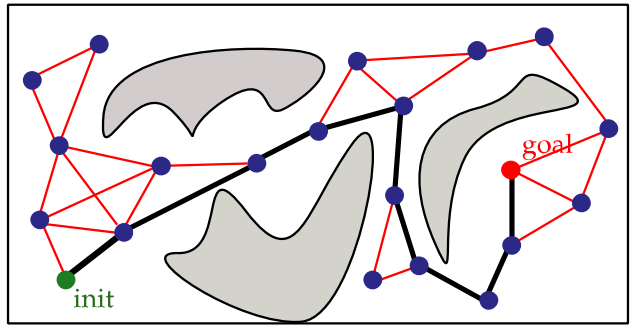
- Query Phase:

Graph Search

search graph (V, E) for path from q_{init} to q_{goal}

if q_{init}, q_{goal} are in V. Otherwise connect

q_{init}, q_{goal} to the graph and search.



Sampling-based Path Planning

Advantages

- Computationally efficient
- Solves high-dimensional problems (with hundreds of DOFs)
- Easy to implement
- Applications in many different areas

Disadvantages

- Does not guarantee completeness (a complete planner always finds a solution if there exists one, or reports that no solution exists)

Is it then just greedy search ?

It offers **probabilistic completeness**

- When a solution exists, a probabilistically complete planner finds a solution with probability 1 as time goes to infinity.
- When a solution does not exist, a probabilistically complete planner may not be able to determine that a solution does not exist.

Overview

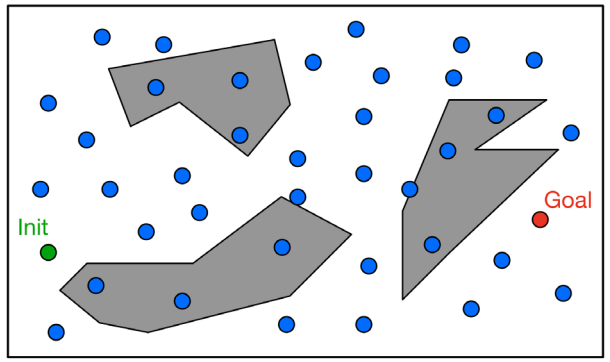
- Probabilistic Roadmaps
- Examples
- Nearest –Neighbor Strategy
- Lazy PRM
- Sampling Strategy

PRM

- **Example 1:** a point robot in 2-D space with polygon obstacles

$q = (x, y) \leftarrow Sample()$

- $x \leftarrow rand(x_{min}, x_{max})$
- $y \leftarrow rand(y_{min}, y_{max})$

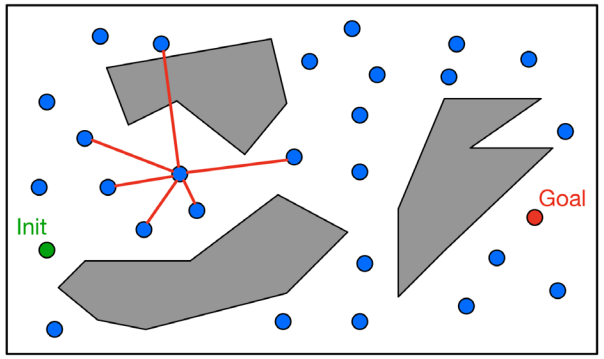


$\text{IsSampleCollisionFree}(q)$

- Point inside/outside polygon test

$path \leftarrow \text{GenerateLocalPath}(q_a, q_b)$

- Straight-line segment from point q_a to point q_b



$\text{IsPathCollisionFree}(path)$

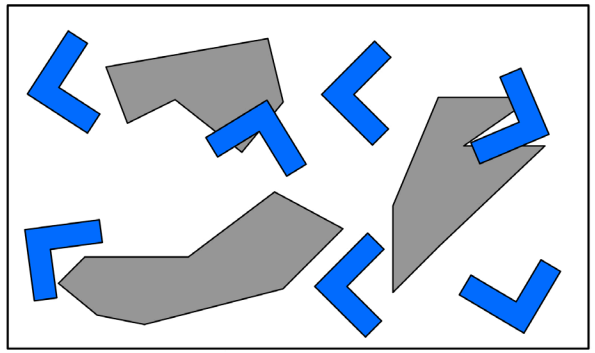
- Line segment-polygon intersection test

PRM

- **Example 2:** rigid (polygon) body robot in 2-D space

$q = (x, y, \theta) \leftarrow Sample()$

- $x \leftarrow rand(x_{min}, x_{max})$
- $y \leftarrow rand(y_{min}, y_{max})$
- $\theta \leftarrow rand\ (-\pi, \pi)$



$\mathrm{IsSampleCollisionFree}(q)$

- Place rigid body in position and orientation specified by q (FK)
- Polygon-polygon intersection test

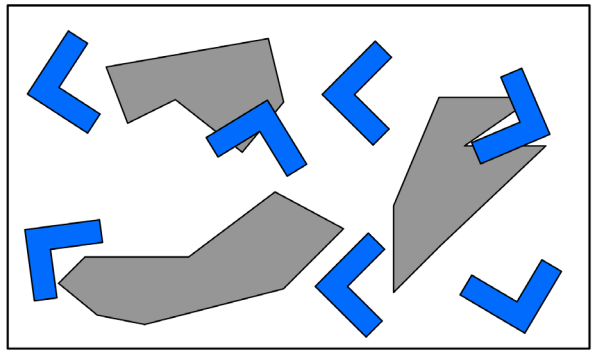
PRM

- **Example 2:** rigid (polygon) body robot in 2-D space

$path \leftarrow \text{GenerateLocalPath}(q_a, q_b)$

- Continuous function path : $[0,1] \rightarrow Q$
- Starts at q_a and ends at q_b

$$path(0) = q_a \ , \ path(1) = q_b$$

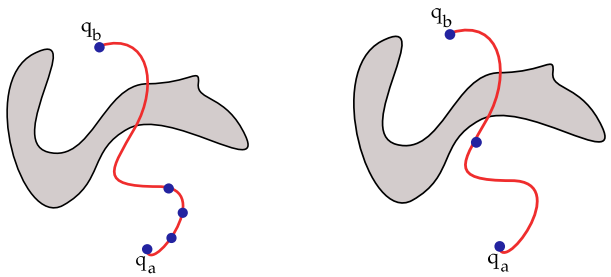


- Many possible ways of defining it, e.g., by linear interpolation

$$path(t) = (1-t)q_a + t\, q_b$$

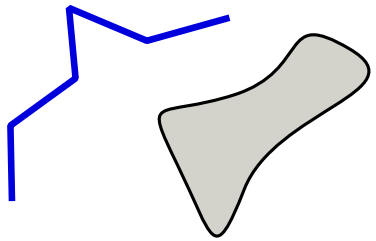
$\text{IsPathCollisionFree}(path)$

- Incremental approach
- Subdivision Approach



PRM

- **Example 3:** articulated chain

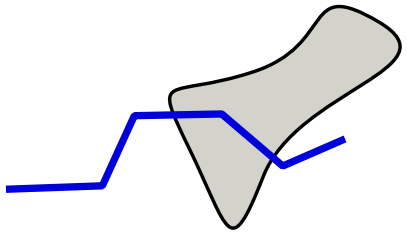


$q = (\theta_1, \theta_2, \ldots, \theta_n) \leftarrow Sample()$

- $\theta_i \leftarrow rand\ (-\pi, \pi) \quad \forall\ i \in \{1, \ldots, n\}$

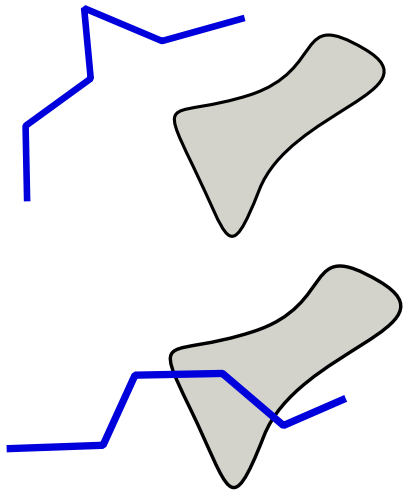
$\mathrm{IsSampleCollisionFree}(q)$

- Place chain in configuration q (forward kinematics)
- Check for collision with obstacles



PRM

- **Example 3:** articulated chain



$path \leftarrow \text{GenerateLocalPath}(q_a, q_b)$

- Continuous function path : $[0,1] \rightarrow Q$
- Starts at q_a and ends at q_b

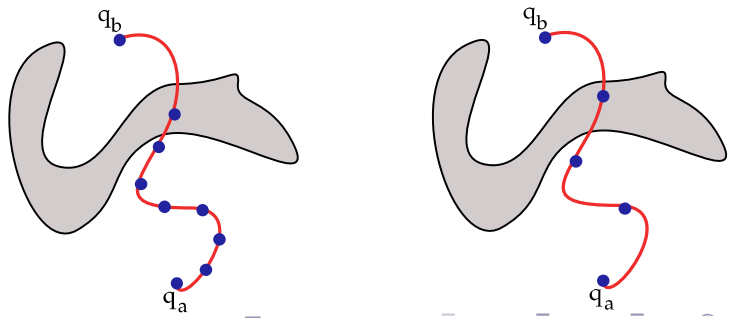
$$path(0) = q_a\,, \quad path(1) = q_b$$

- Many possible ways of defining it, e.g., by linear interpolation

$$path(t) = (1-t)q_a + t\,q_b$$

$\text{IsPathCollisionFree}(path)$

- Incremental approach
- Subdivision approach



Note on Interpolation for High-Dof Spaces

Compact Representation for Rotations of Vectors in 3-Dimensions

- 3×3 Matrices -- 9 Entries
- Unit Quaternions -- 4 Coefficients

Avoids Distortions

- After several matrix multiplications, rotation matrices may no longer be orthogonal due to floating point inaccuracies.
- Non-Orthogonal matrices are difficult to renormalize -- leads to distortions.
- Quaternions are easily renormalized -- avoids distortions.

Interpolation Between Rotations

- Linear Interpolation between two rotation matrices R_1 and R_2 fails to generate another rotation matrix.

  $Lerp(R_1, R_2, t) = (1-t)R_1 + tR_2$ -- not necessarily orthogonal matrices.

- Spherical Linear Interpolation between two unit quaternions always generates a unit quaternion.

  $Slerp(q_1, q_2, t) = \frac{\sin((1-t)\phi)}{\sin(\phi)} q_1 + \frac{\sin(t\phi)}{\sin(\phi)} q_2$ -- always a unit quaternion.

Note on Sampling for High-Dof spaces

Read the Kuffner paper of 2004: The paper was published out of frustration

In Proc. 2004 IEEE Int'l Conf. on Robotics and Automation (ICRA 2004)

Effective Sampling and Distance Metrics for 3D Rigid Body Path Planning

James J. Kuffner

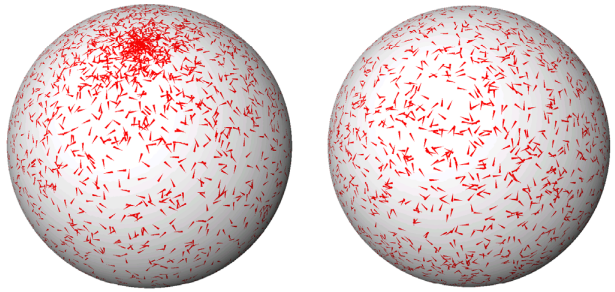
The Robotics Institute
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213, USA
email: kuffner@cs.cmu.edu

Digital Human Research Center
National Institute of Advanced
Science and Technology (AIST)
2-41-6 Aomi, Koto-ku, Tokyo, Japan 135-0064

***Abstract*— Important implementation issues in rigid body path planning are often overlooked. In particular, sampling-based motion planning algorithms typically require a *distance metric* defined on the configuration space, a *sampling function*, and a method for *interpolating sampled points*. The configuration space of a 3D rigid body is identified with the Lie group SE(3). Defining proper metrics, sampling, and interpolation techniques for SE(3) is not obvious, and can become a hidden source of failure for many planning algorithm implementations. This paper examines some of these issues and presents techniques which have been found to be effective experimentally for Rigid Body path planning.**

I. Introduction

The configuration space ($\mathcal{C}$-space) of a 3D rigid body is usually defined as the set of all possible positions and orientations of a body-fixed frame relative to a stationary world frame. This identifies the $\mathcal{C}$-space with the *Lie group* $SE(3)$, the special Euclidean group in three-dimensions. Geometric path planning problems defined on $SE(3)$ arise in a number of important application domains including mechanical assembly planning and part removability analysis, control of free-flying robots and UAVs, satellite motion, and biochemical



(a) Naïve sampling (b) Uniform sampling

Fig. 1. Naïve sampling (a) and uniform sampling (b) of $SO(3)$ using euler angles. There are a total of 5000 rotation samples, with each sample visualized as an oriented arrow on the surface of the unit sphere.

Overview

- Probabilistic Roadmaps
- Examples
- Nearest –Neighbor Strategy
- Lazy PRM
- Sampling Strategy

Two critical issues in PRM

- Several 100s implementations of PRMs exist

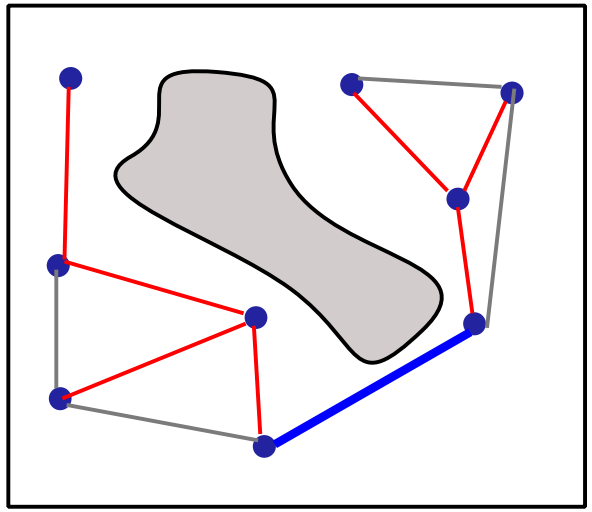
- There are two major sets of choices that affect the performance roadmap-planners:
  1. What node to connect to what other
  2. Where to sample

Let’s start with connections (which nearest neighbors to choose?)

Roadmaps with no Cycles

- Edge in cycle does not improve roadmap connectivity
- Edge is added to roadmap only if it connects two different roadmap components

1: ***if*** $\text{SameRoadmapComponent}(q_a, q_b) = false$ **then**
2: $path \leftarrow \text{GeneratePath}(q_a, q_b)$
3: **if** $\text{IsPathCollisionFree}(\text{path}) = true$ **then**
4: $(q_a, q_b).path \leftarrow path$
5: $E \leftarrow E \cup \{(q_a, q_b)\}$



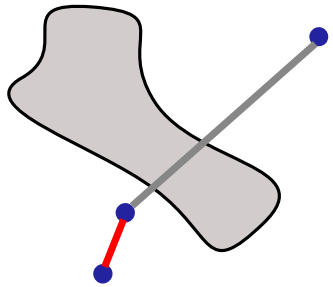
- *Disjoint-set data structure* is used to speed up computation of

$$\text{SameRoadmapComponent}(q_a, q_b)$$

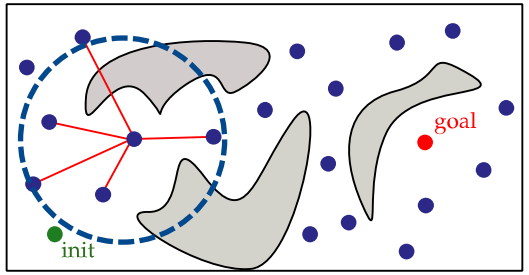
Check *Cormen et al* for disjoint-set (union/find) data structure O(n) time

Connecting Roadmap Nodes to Nearest Neighbors

- Edges between neighboring nodes are more likely to be collision free than edges between far away nodes
- Common practice is to attempt to connect each node to k of its nearest neighbors



- Nearest neighbors defined by some distance metric

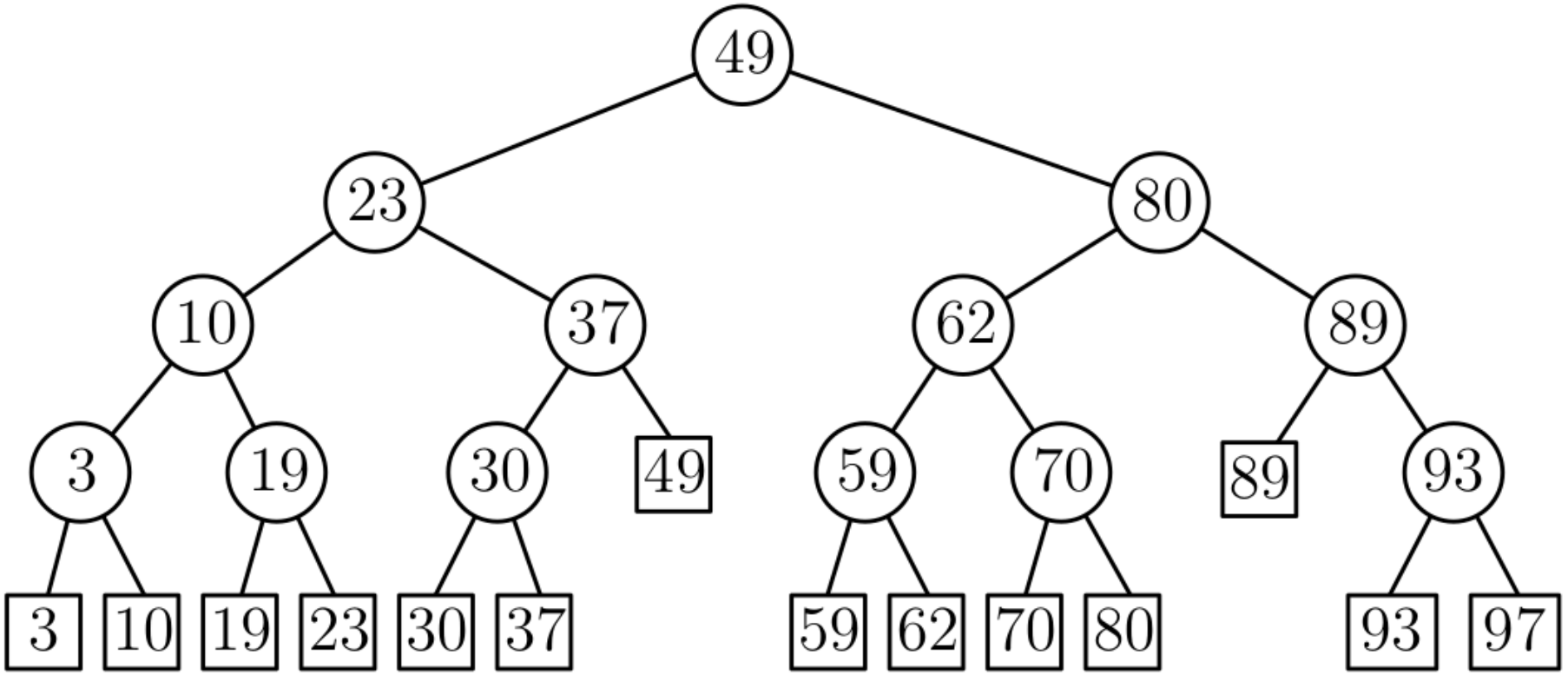


$$\rho : Q \times Q \rightarrow \mathbb{R}^{\geq 0},$$

- geodesic distances, i.e., shortest-path distances according to topology
- weighted combination of translation and rotation components
- Euclidean distance between selected robot points
- Good distance metrics reflect the likelihood of successful connections

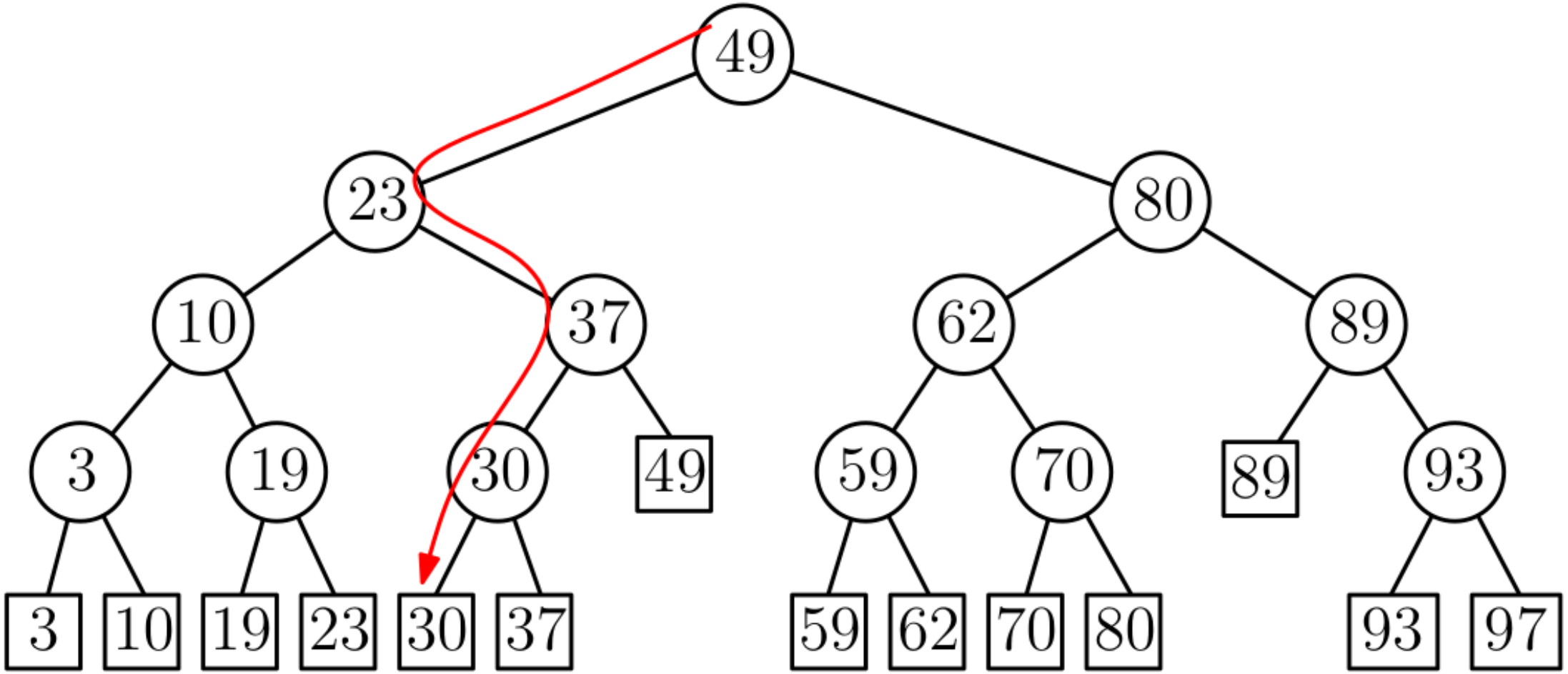
KD-Tree Background : A balanced Binary Tree

A balanced binary search tree with the points in the leaves



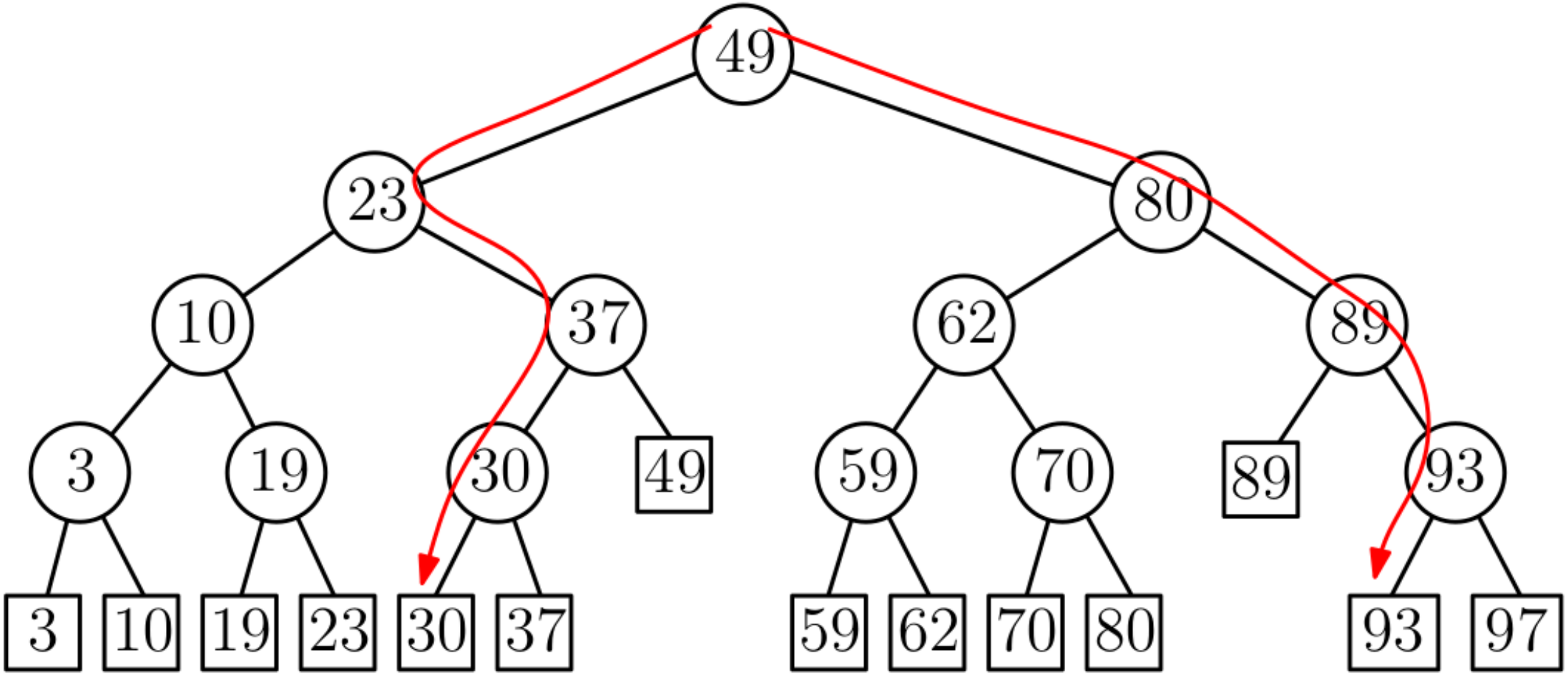
KD-Tree Background : A balanced Binary Tree

Searching if 25 is part of the tree



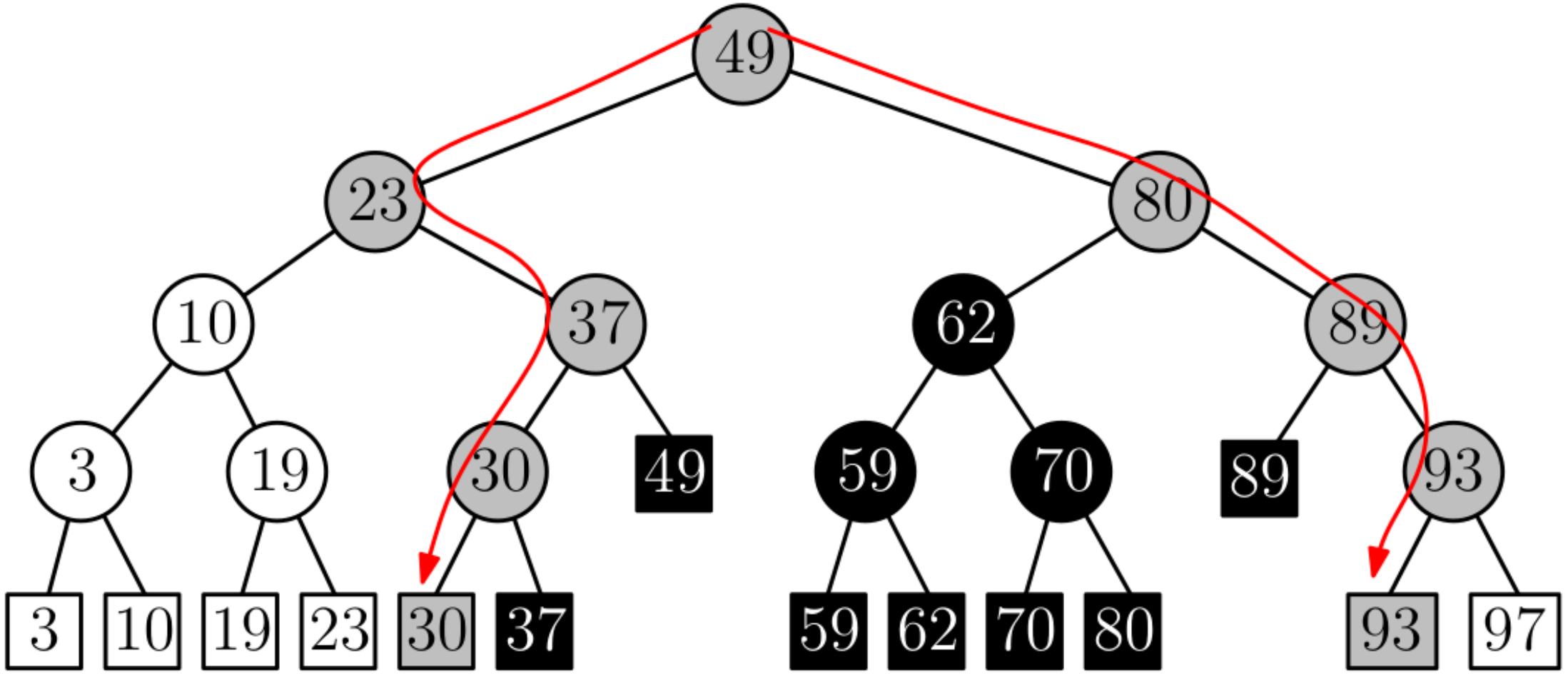
KD-Tree Background : A balanced Binary Tree

Searching if 25 and 90 is part of the tree



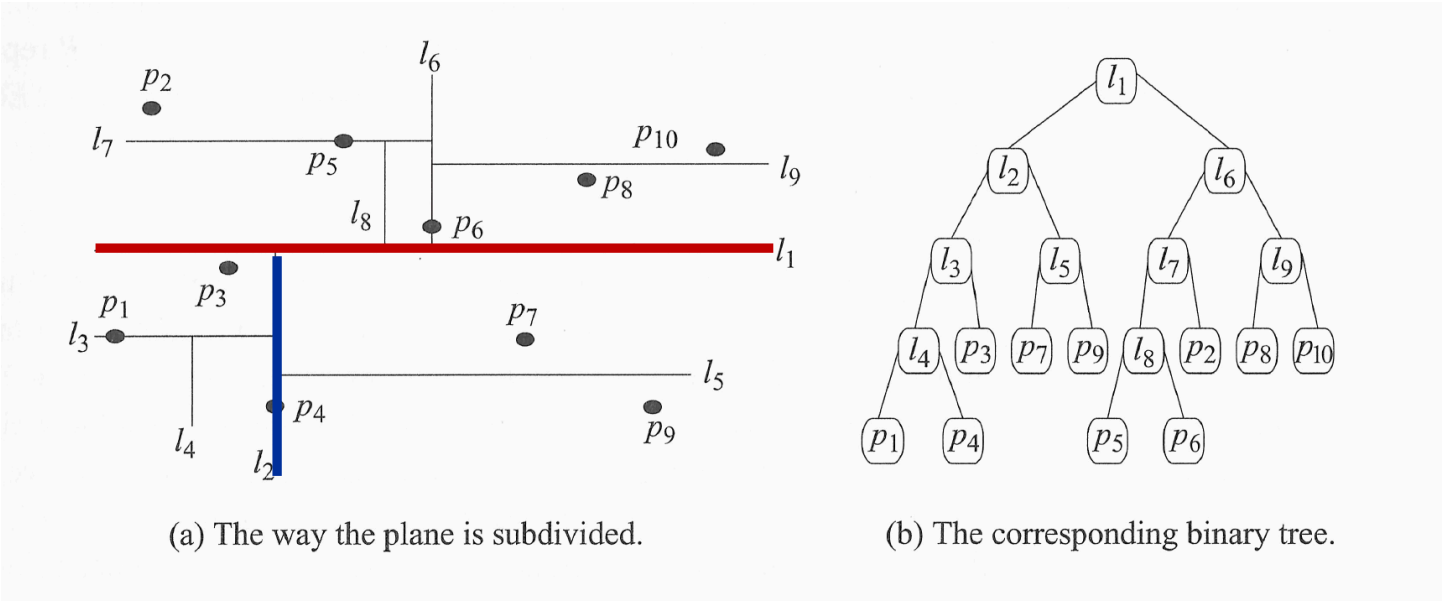
KD-Tree Background : A balanced Binary Tree

A 1-dimensional range query with [25, 90]



KD-Trees : extend the same idea in higher dimentions

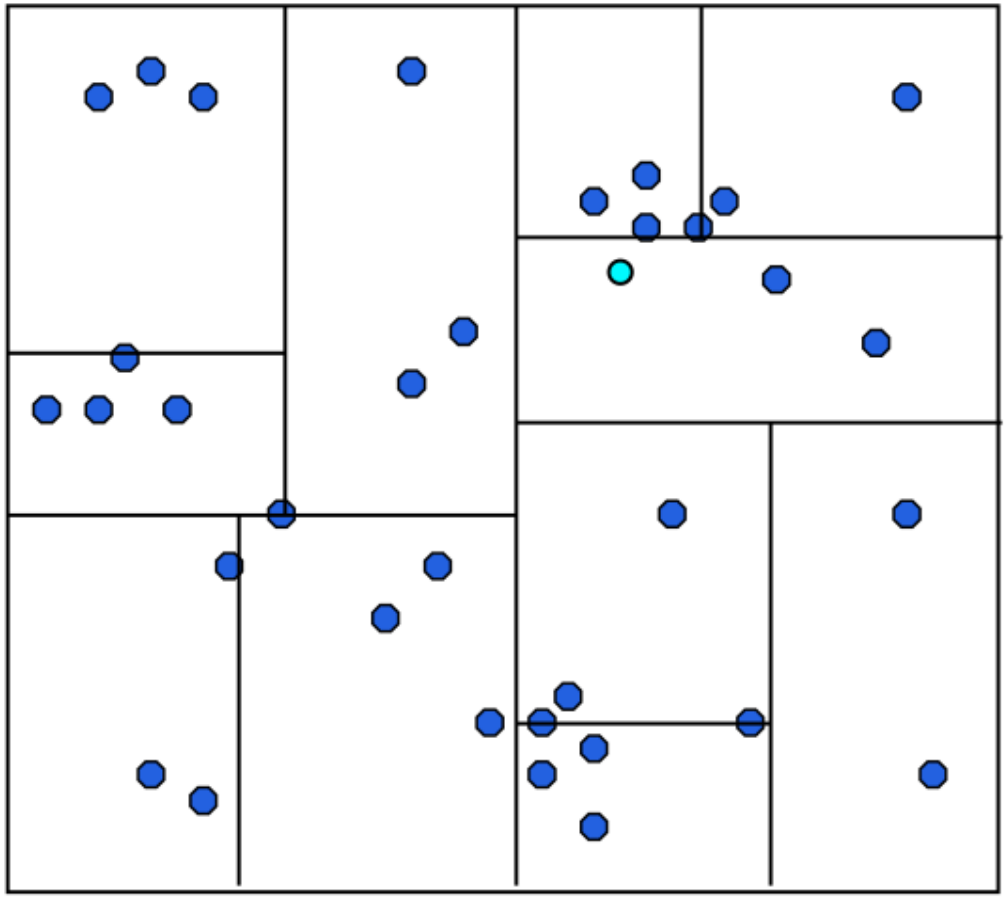
- We alternate splitting between different dimensions e.g., L1, splits along y dimentions L2, L6 along x dimensions
- A d-dimensional kd-tree uses as input a set S of n points in d dimensions and constructs a binary tree that decomposes the space into cells such that no cell contains too many points.
- O(dn) storage and the tree can be built in O(dn logn) time. A rectangular range query takes $O\left(n^{d-1/d}+m\right)$ time where m is the number of reported neighbors

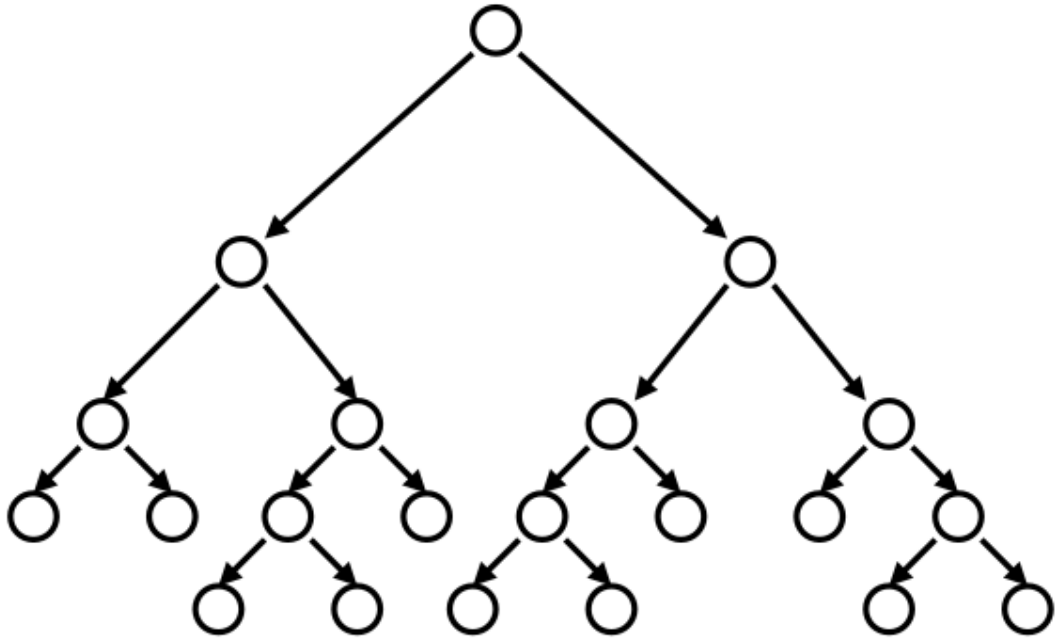


(a) The way the plane is subdivided.

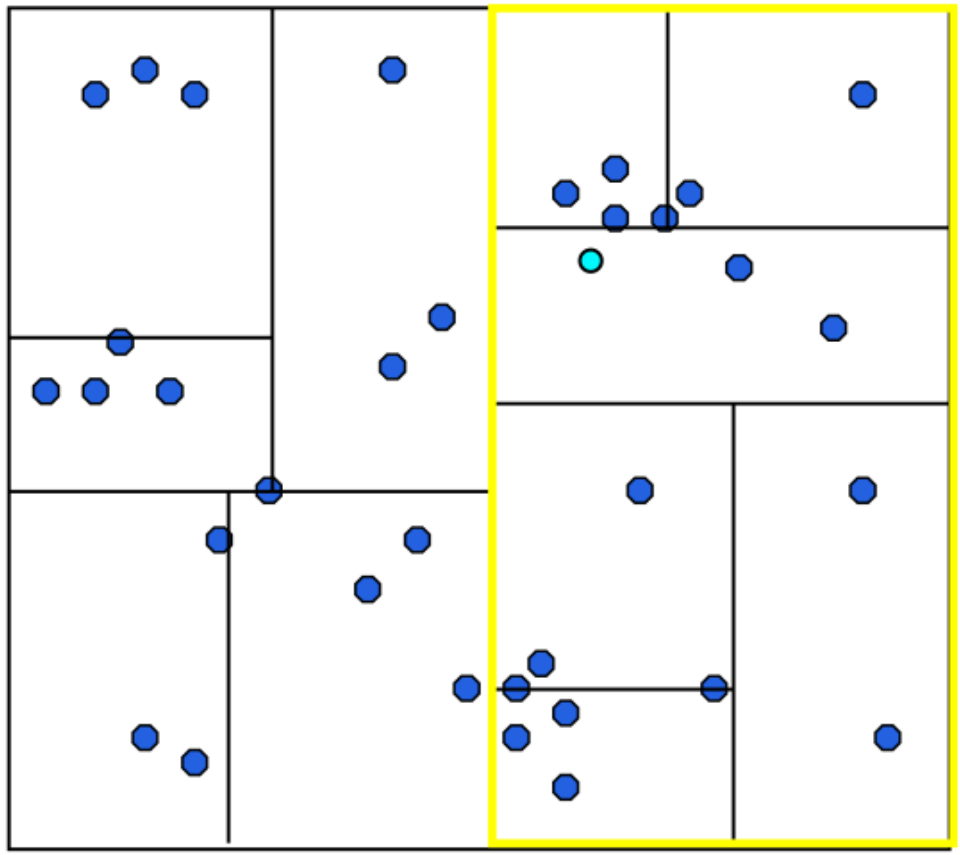
(b) The corresponding binary tree.

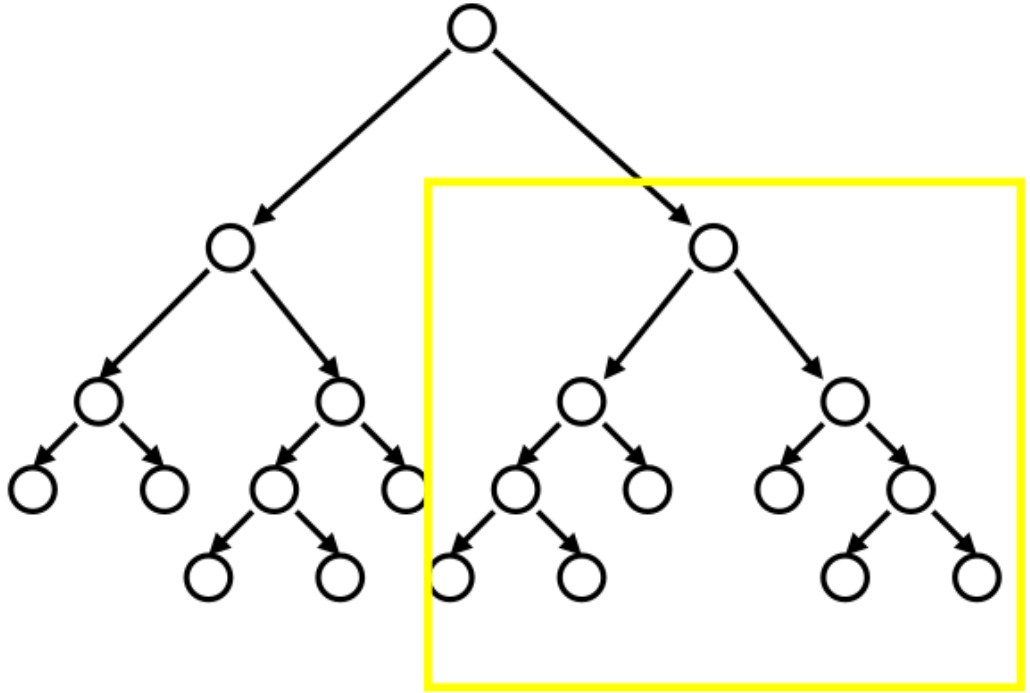
KD-Tree find region near query point



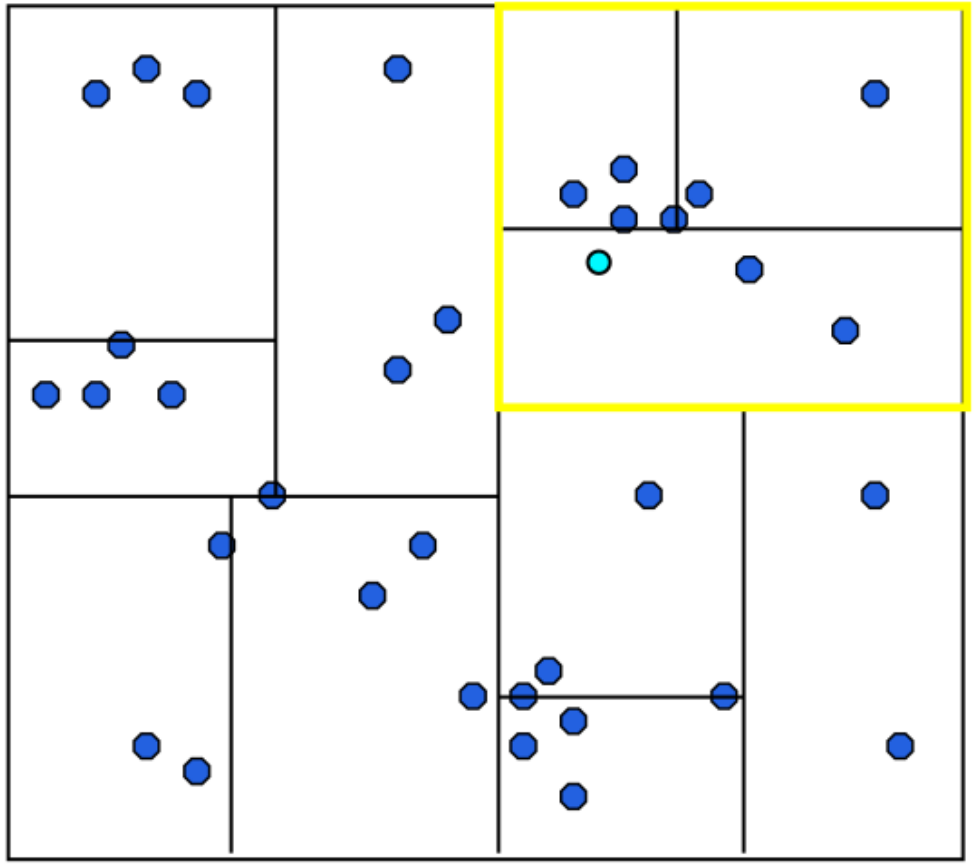


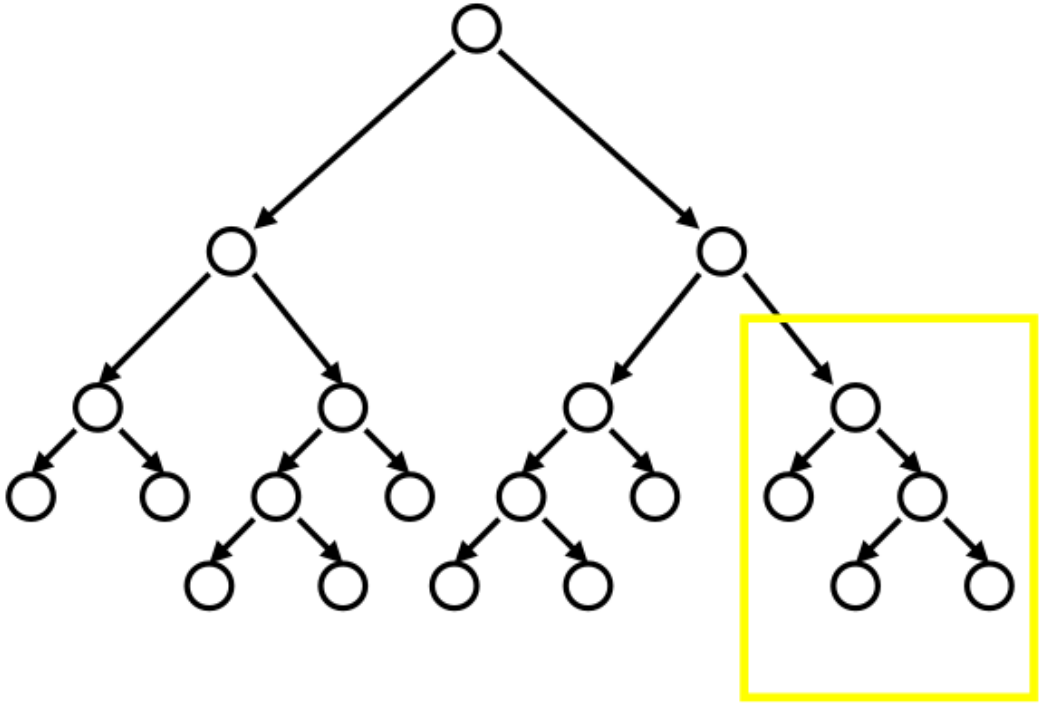
KD-Tree find region near query point



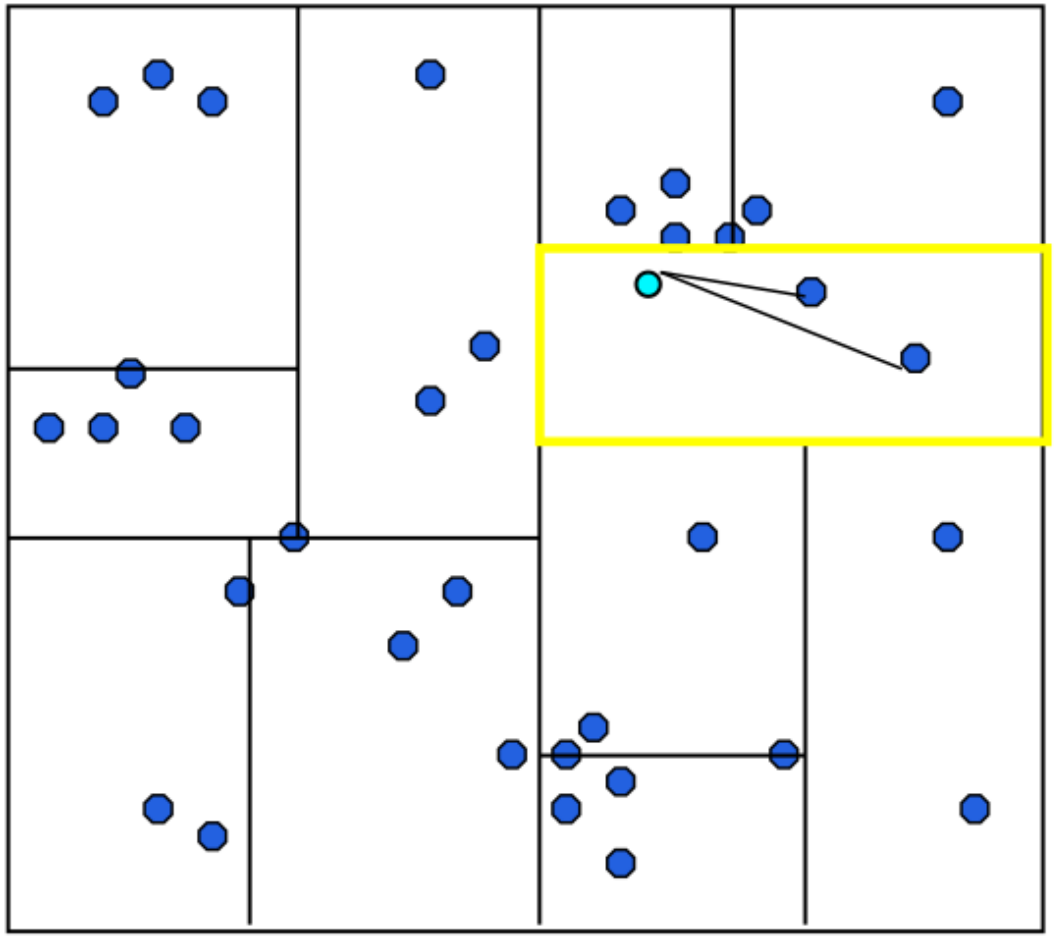


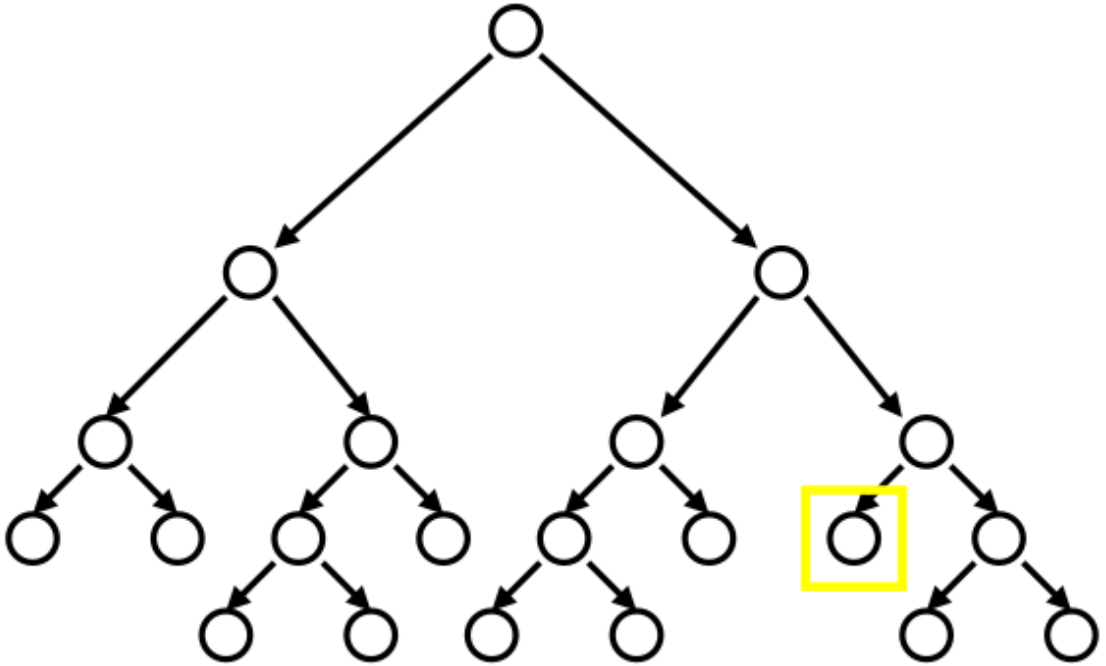
KD-Tree find region near query point



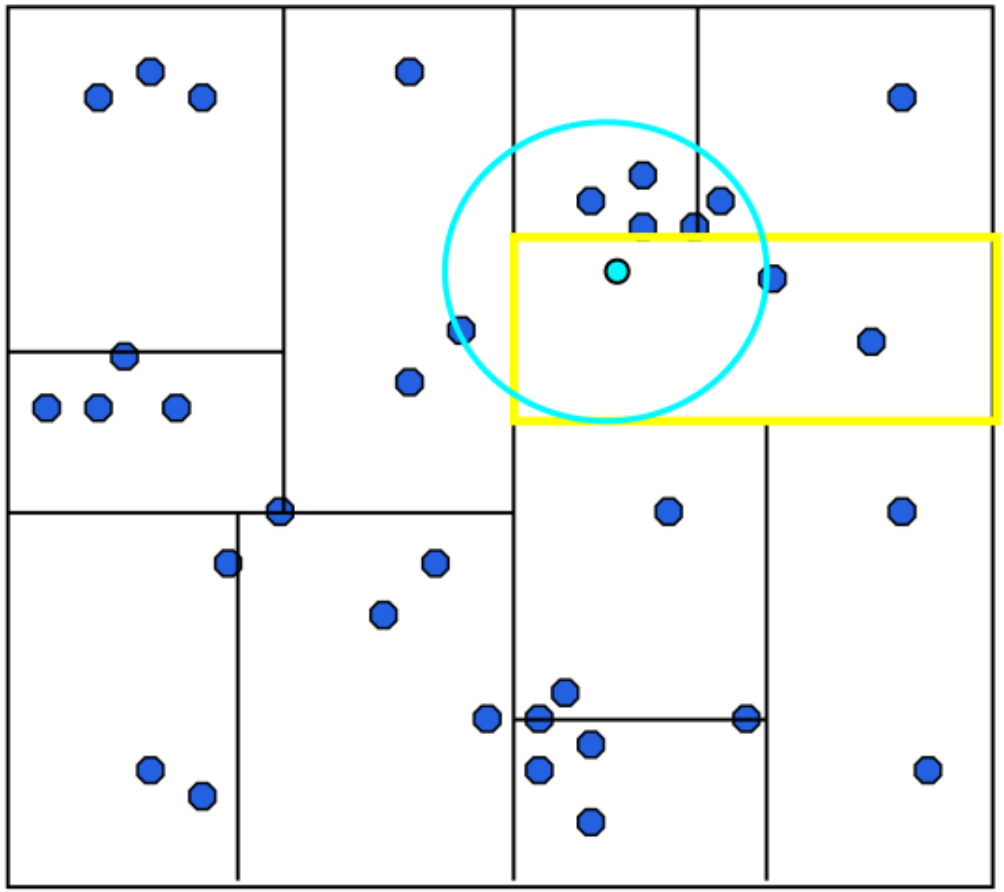


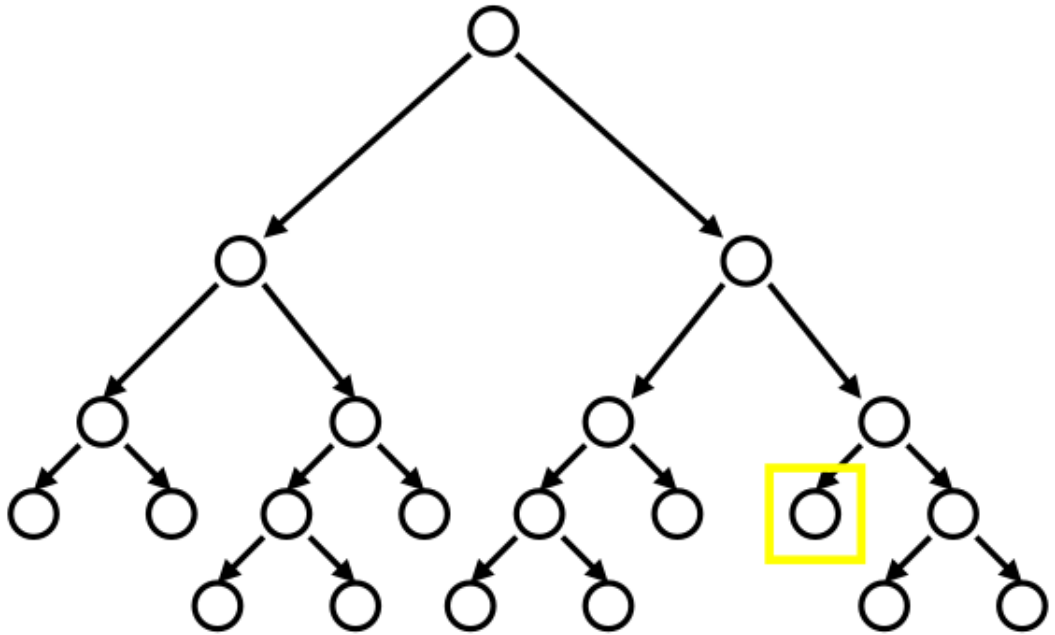
KD-Tree find region near query point



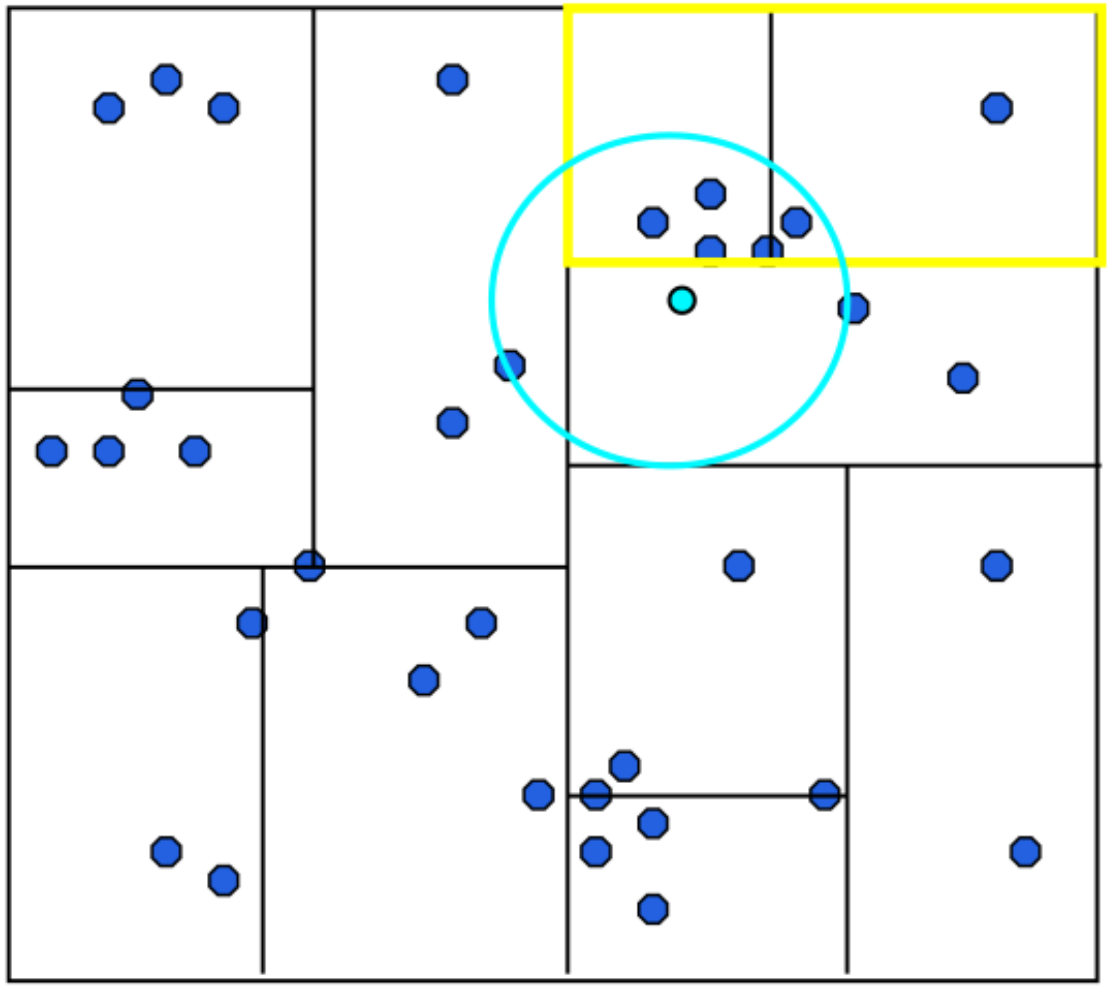


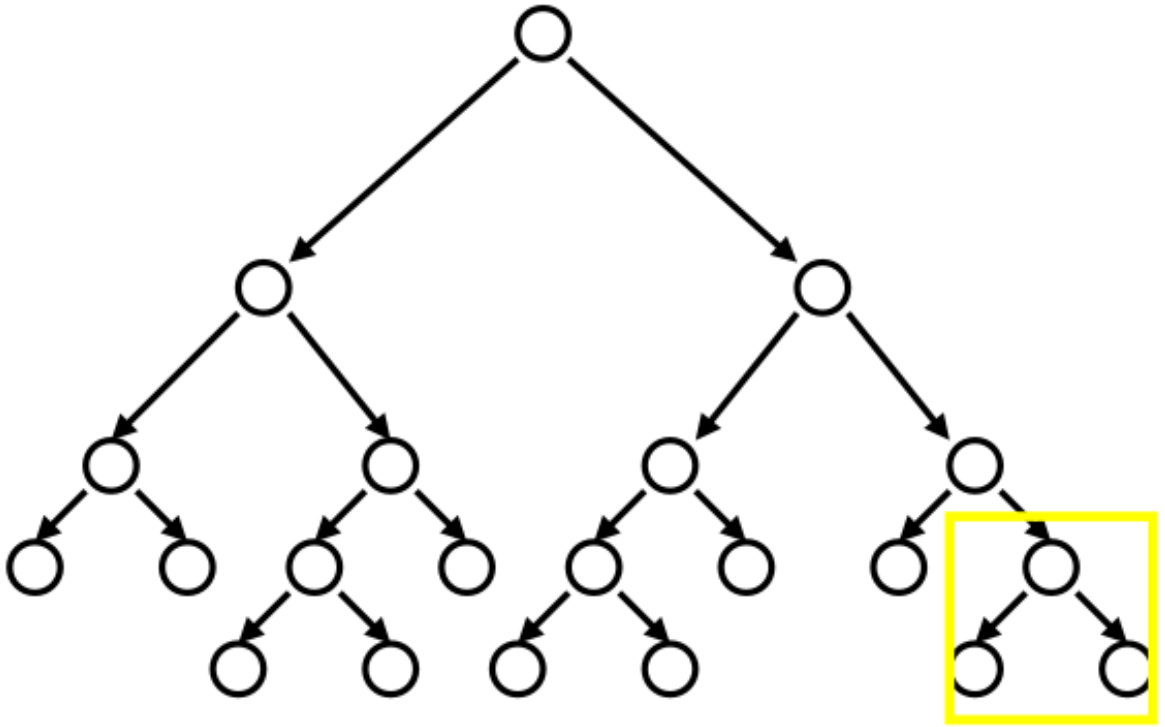
KD-Tree: At leaf node compute distance to each point in the node



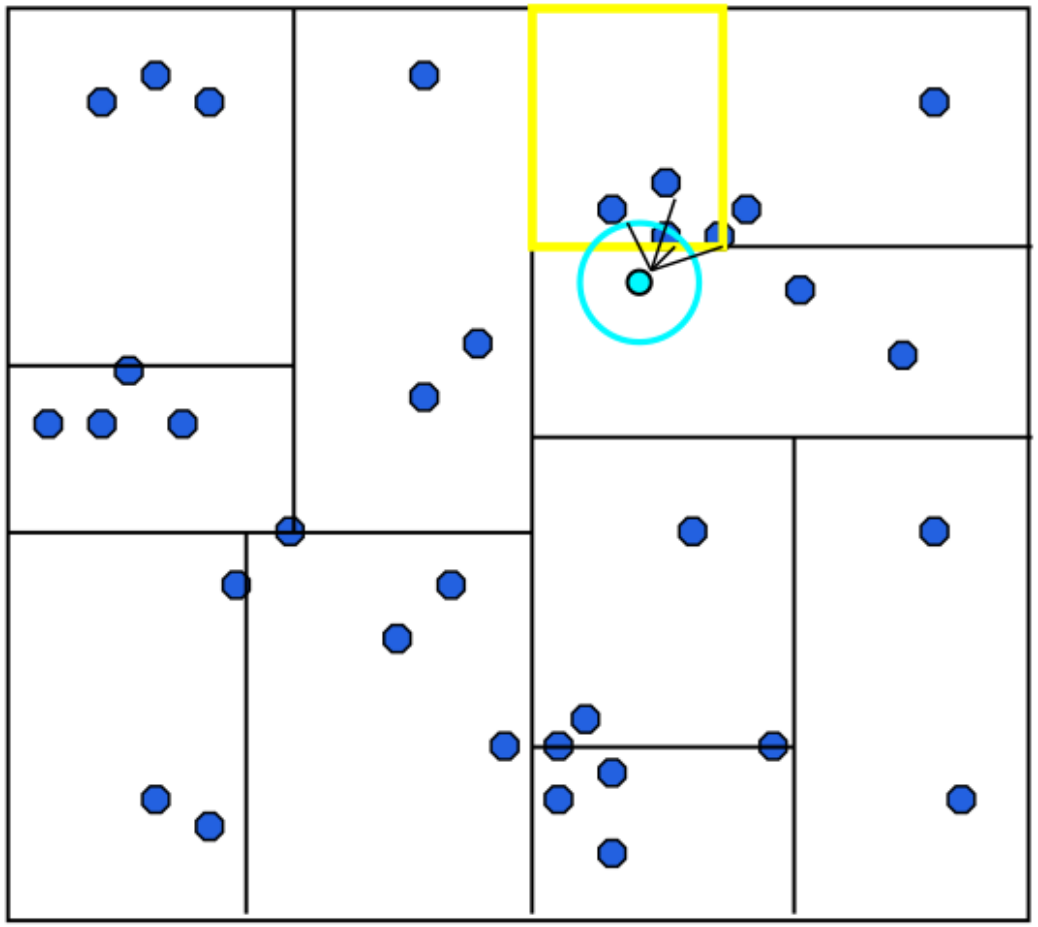


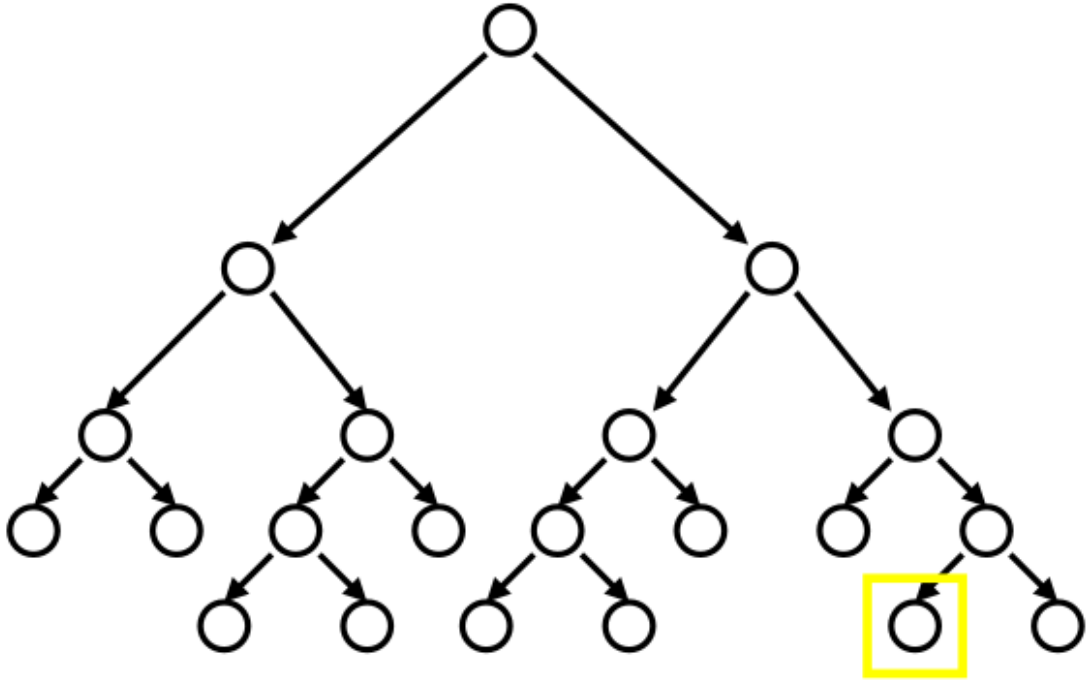
KD-Tree: Back -Track to Siblings



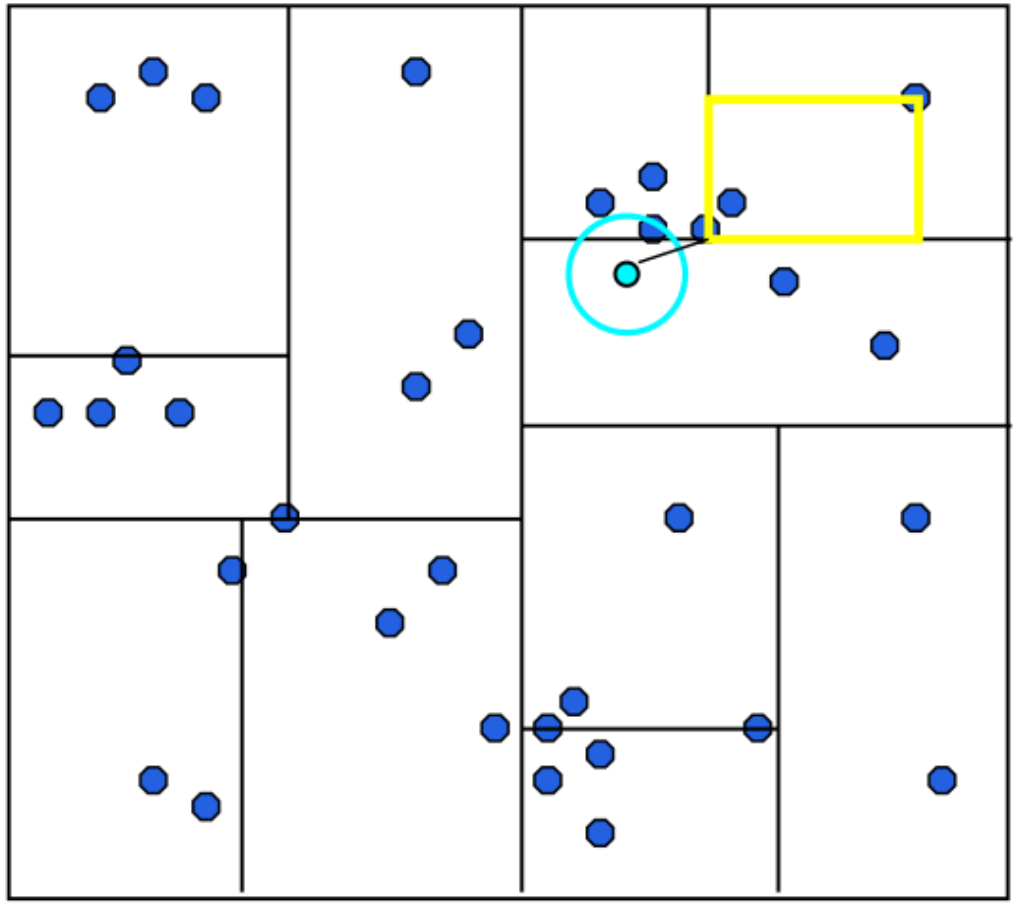


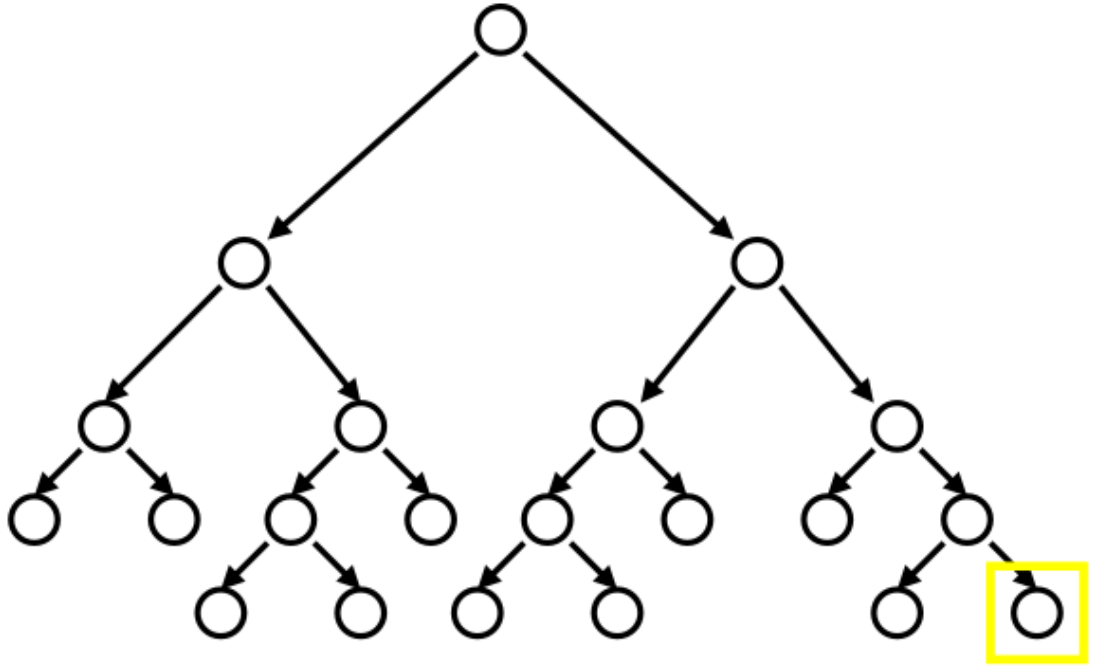
KD-Tree: Back -Track to Siblings



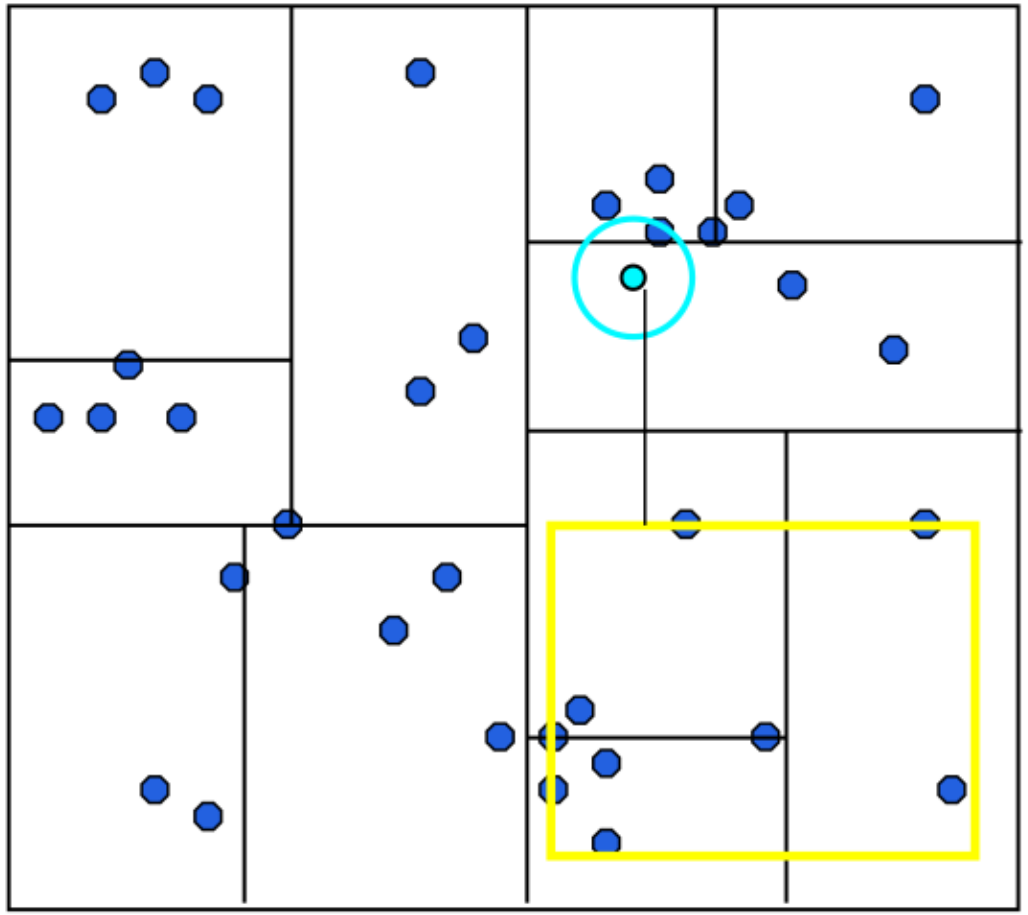


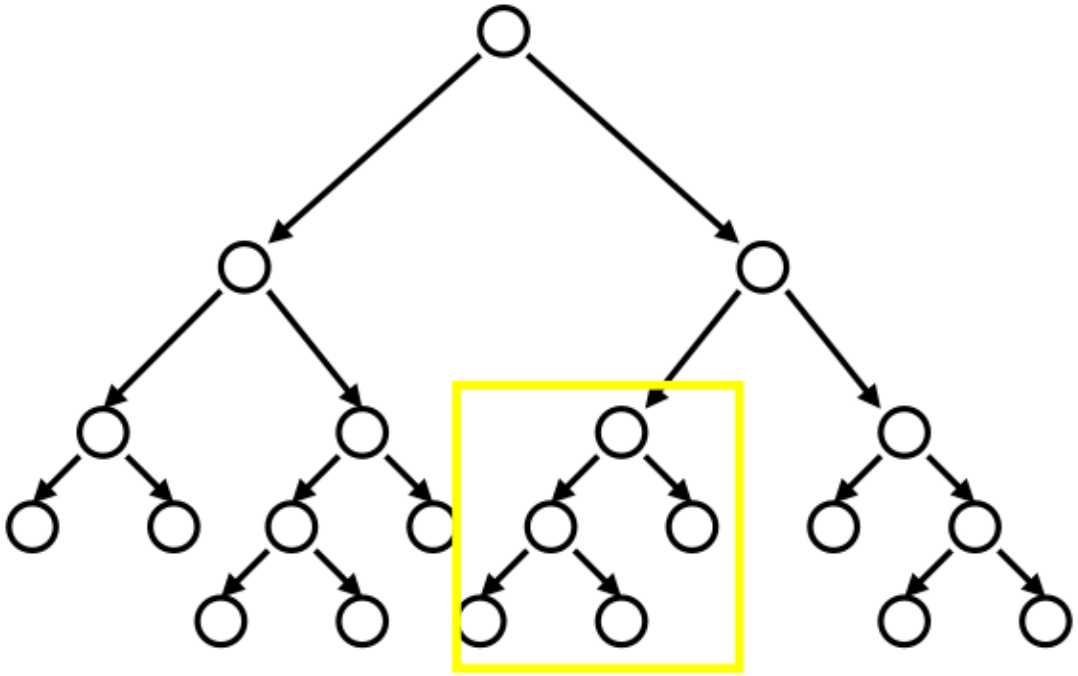
KD-Tree: Back -Track to Siblings



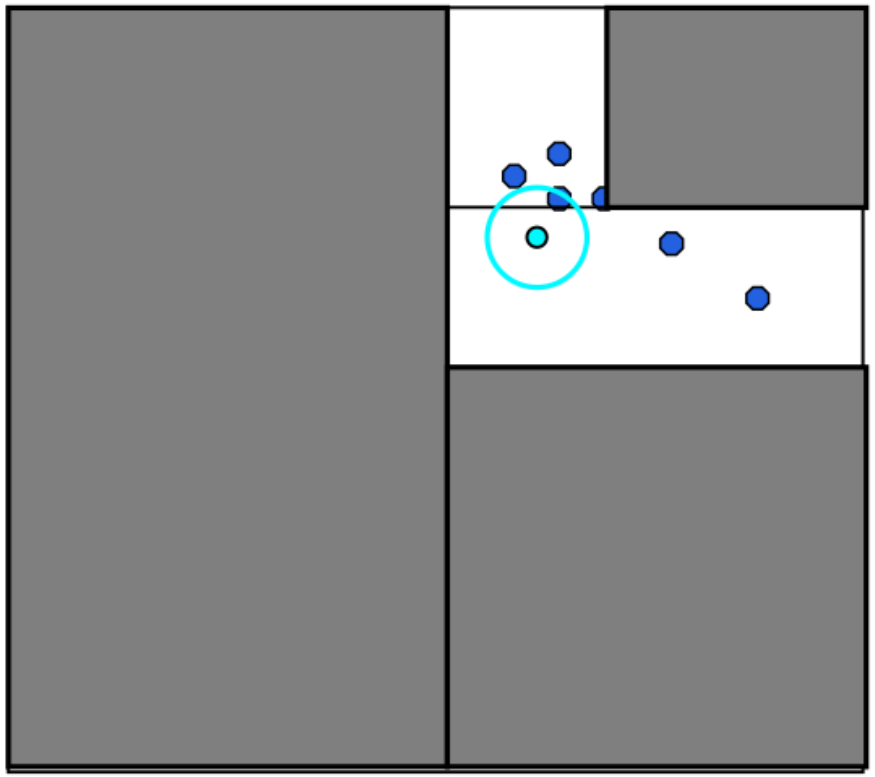


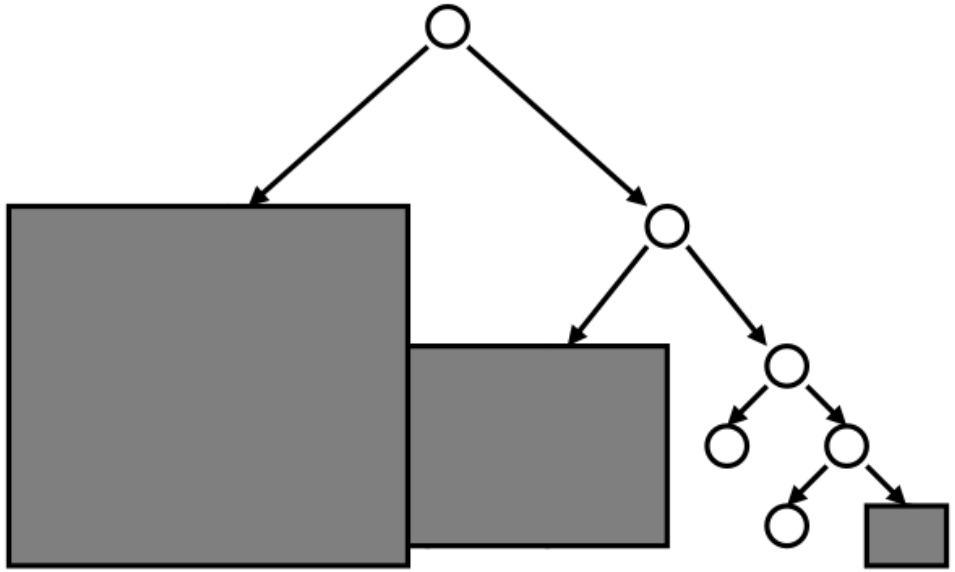
KD-Tree find region near query point





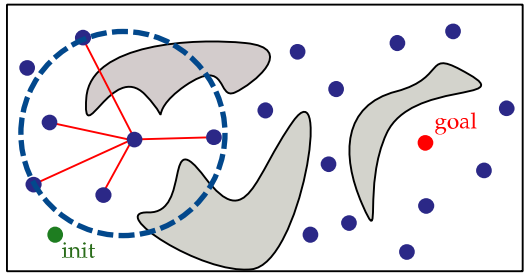
KD Tree: Prune search area based on region bounds and distance bounds





Connecting Roadmap Nodes to Nearest Neighbors

- Numerous algorithms/data structures for nearest-neighbors computations,
  - e.g., KD-tree, R-tree, M-tree, V-tree, PR-tree, GNAT, iDistance, CoverTree
    - Computational challenges of nearest neighbors in high-dimensional spaces
      - Efficiency deteriorates rapidly
      - Not much better than brute-force approach



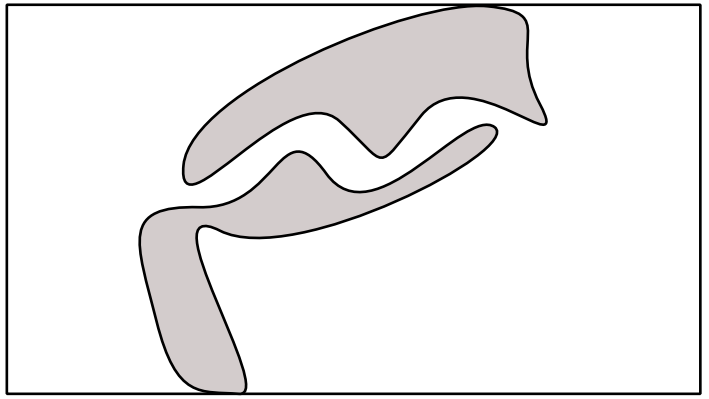
- Alternative approach is to compute approximate nearest neighbors [Plaku, Kavraki: WAFR 2006, SDM 2007]
  - Minimal losses in accuracy of neighbors
  - No loss in accuracy of overall path planner
  - Significant computational gains

Overview

- Probabilistic Roadmaps
- Examples
- Path Post Processing
- Nearest –Neighbor Strategy
- Sampling Strategy
- Lazy PRM

Narrow Passage Problem

- Probability of generating samples via uniform sampling in a narrow passage is low due to the small volume of the narrow passage



- Generating samples inside a narrow passage may be critical to the success of the path planner

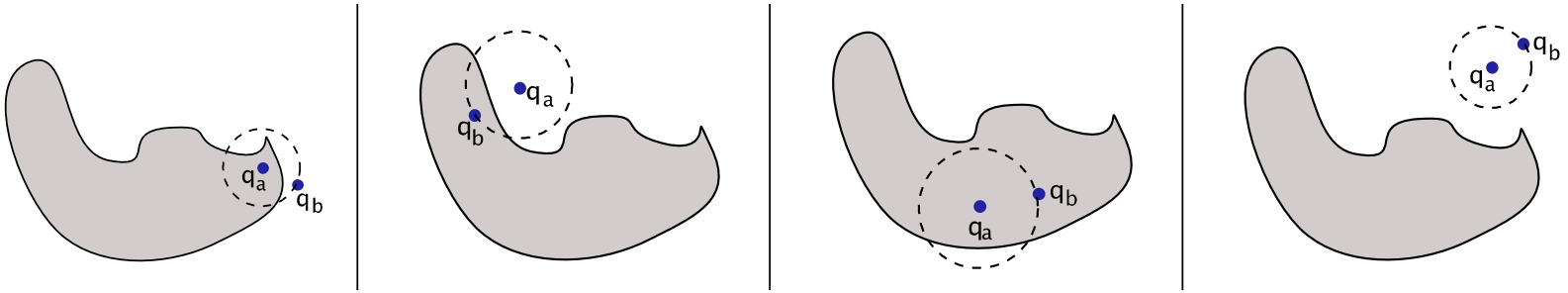
- Objective is then to design sampling strategies that can increase the probability of generating samples inside narrow passages

Gaussian Sampling in PRM

- **Objective:** Increase Sampling Inside/Near Narrow Passages
- **Approach:** Sample from a Gaussian distribution biased near the obstacles

GenerateCollisionFreeConfig

1: $q_a \leftarrow$ generate config uniformly at random
2: $r \leftarrow$ generate distance from Gaussian distribution
3: $q_b \leftarrow$ generate config uniformly at random at distance r from q_a
4: $ok_a \leftarrow \text{IsConfigCollisionFree}(q_a)$
5: $ok_b \leftarrow \text{IsConfigCollisionFree}(\text{q}_\text{b})$
6: **if** $ok_a = true$ **and** $ok_b = false$ **then return** q_a
7: **if** $ok_a = false$ **and** $ok_b = true$ **then return** q_b
8: **return** null

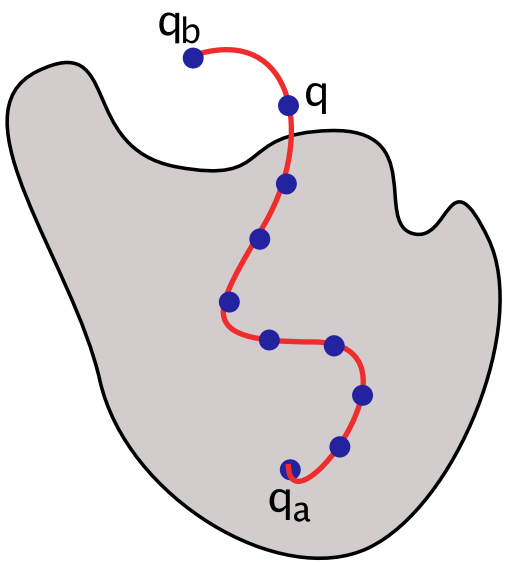


Obstacle-Based Sampling in PRM

- **Objective:** Increase Sampling Inside/Near Narrow Passages
- **Approach:** Move samples in collision outside obstacle boundary

GenerateCollisionFreeConfig

1: $q_a \leftarrow$ generate config uniformly at random
2: **if** IsConfigCollisionFree$(q_a) = true$ **then**
3: **return** q_a
4: **else**
5: $q_b \leftarrow$ generate config uniformly at random
6: $path \leftarrow$ GeneratePath(q_a, q_b)
7: **for** $t = \delta$ to $|path|$ by δ **do**
8: **if** IsConfigCollisionFree$\big(path(t)\big)$ **then**
9: **return** $path(t)$
10: **return** null

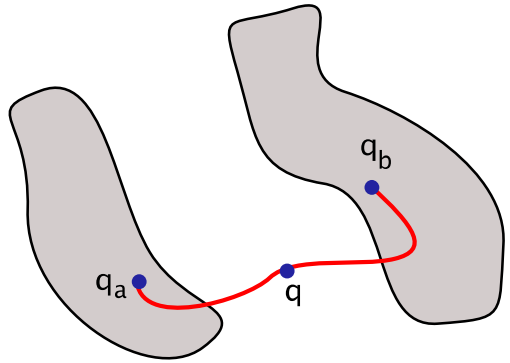


Bridge-Based Sampling in PRM

- **Objective:** Increase Sampling Inside/Near Narrow Passages
- **Approach:** Create “bridge” between samples in collision

GenerateCollisionFreeConfig

1: $q_a \leftarrow$ generate config uniformly at random
2: $q_b \leftarrow$ generate config uniformly at random
3: $ok_a \leftarrow \text{IsConfigCollisionFree}(q_a)$
4: $ok_b \leftarrow \text{IsConfigCollisionFree}(q_b)$
5: **if** $ok_a = false$ **and** $ok_b = false$ **then**
6: $path \leftarrow \text{GeneratePath}(q_a, q_b)$
7: $q \leftarrow path(0.5\ |path|)$
8: **if** $\text{IsConfigCollisionFree}(q)$ **then**
9: **return** q
10: **return** null

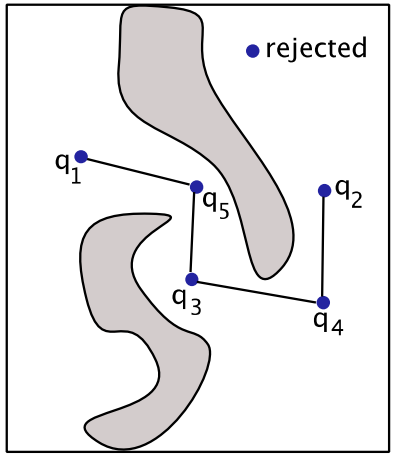


Visibility Based Sampling in PRM

- **Objective:** Capture connectivity of configuration space with few samples
- **Approach:** Generate samples that create new components or join existing components

GenerateCollisionFreeConfig

1: $q \leftarrow$ generate config uniformly at random
2: **if** IsConfigCollisionFree(q) **then**
3: **if** q belongs to a new roadmap component **then**
4: **return** q
5: **if** q connects two roadmap components **then**
6: **return** q
7: **return** null



- q_1: creates new roadmap component
- q_2: creates new roadmap component
- q_3: creates new roadmap component
- q_4: connects two roadmap components
- q_5: connects two roadmap components

[Nisseoux, Simeon, Laumond: Advanced Robotics J 2000]

Importance Sampling

- **Objective:** Increase Sampling Inside/Near Narrow Passages
- **Approach:** Improve roadmap connectivity

Method algorithm

- Associate weight $w(q)$ with each configuration q in the roadmap
- Weight $w(q)$ indicates difficulty of region around q
  - $w(q) = \frac{1}{1+\deg(q)}$, $\deg(q)$: number of neighbors of q
  - $w(q)$ = number of times connections from/to q have failed
  - combination of different strategies
- Select sample with probability $\frac{w(q)}{\sum_{q' \in V} w(q')}$
- Generate more samples around q
- Connect new samples to neighboring roadmap nodes

Combine Different Sampling Strategies

- Each sampling strategy has its strengths and weakness
- Objective is to identify the appropriate sampling strategy for a given region

- One common strategy is to assign a weight w_i to each sampler S_i
- A sampler S_i is then selected with probability

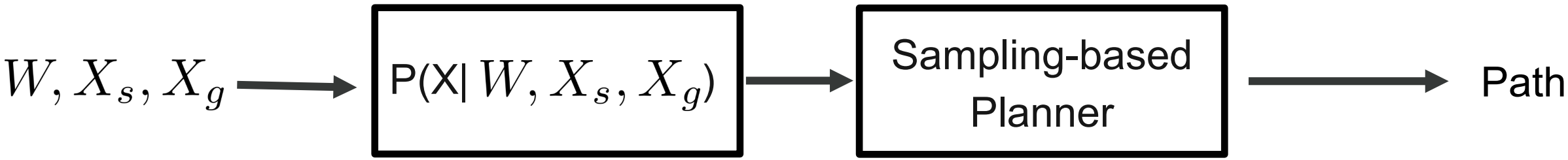
$$\frac{w_i}{\sum_j w_j}$$

- Sampler weight is updated based on quality of performance
- Balance between being “smart and slow” and “dumb and fast”

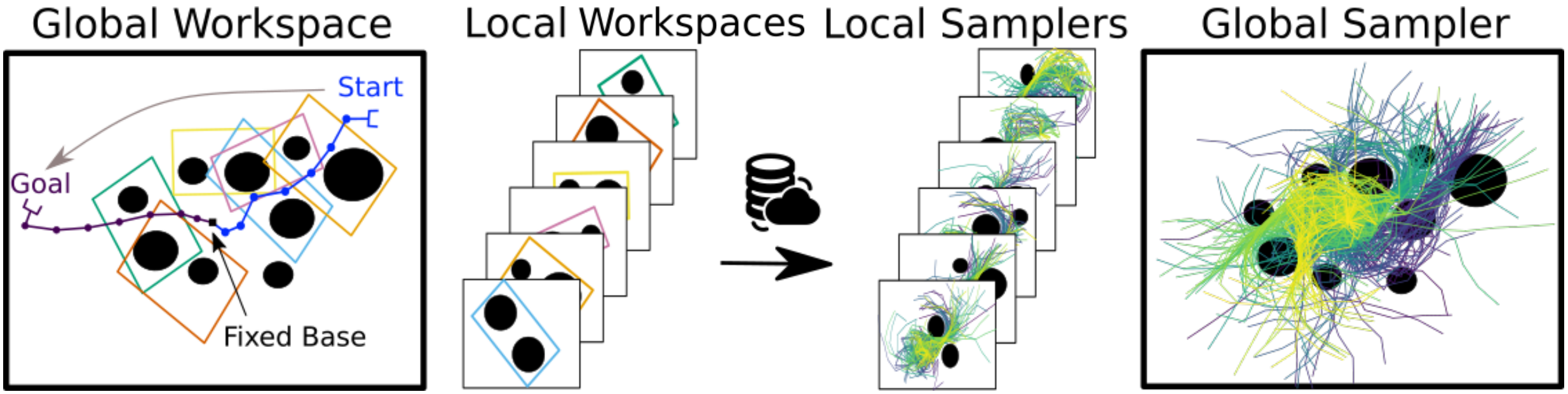
Experience-Based Samplers

Training/Learning Phase

Testing/Inference



Biased Sampling for a Kinematic Chain



- Decompose workspace in local workspaces
- Compute an efficient local sampler for each local workspaces
- During inference retrieve from DB local samplers
- Synthesize the local samplers in a global distribution

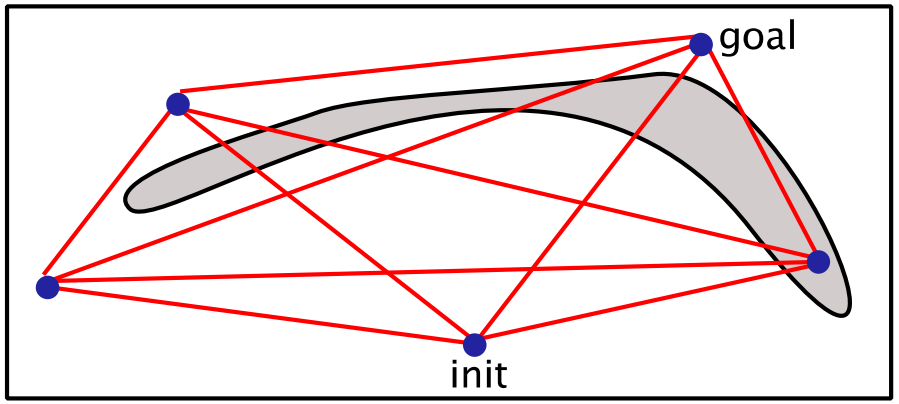
Chamzas et al, "Using local experiences for global motion planning." *International Conference on Robotics and Automation (ICRA)*. IEEE, 2019.

Overview

- Probabilistic Roadmaps
- Examples
- Path Post Processing
- Nearest –Neighbor Strategy
- Sampling Strategy
- Lazy PRM

Lazy PRM

- Lazy PRM: perform collision checking only when necessary



LazyRoadmapConstruction

1: $V \leftarrow V \cup \{q_{init}, q_{goal}\}$; $E \leftarrow \emptyset$

2: **for** several times **do**

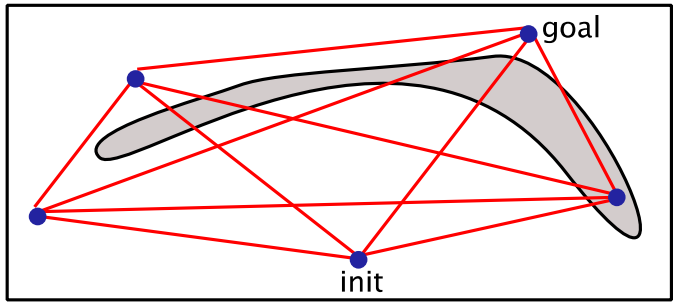
3: $q \leftarrow$ generate config uniformly at random; $q.checked \leftarrow false$; $V \leftarrow V \cup \{q\}$

4: **for** each pair $(q_a, q_b) \in V \times V$ **do**

5: $(q_a, q_b).res \leftarrow 1.0$; $(q_a, q_b).path \leftarrow \text{GeneratePath}(q_a, q_b)$; $E \leftarrow E \cup \{(q_a\ q_b)\}$

Lazy PRM

- Lazy PRM: perform collision checking only when necessary

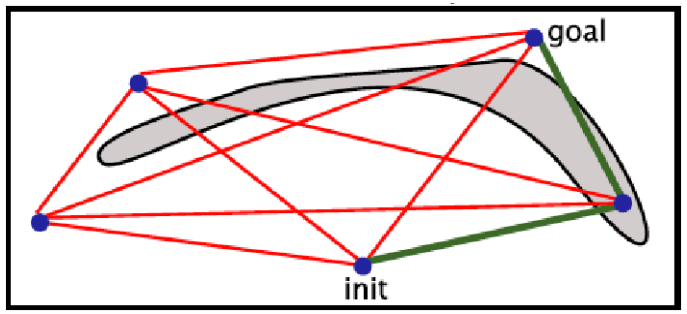


LazyRoadmapCollissionChecking

1: **for** several times **do**
2: $[q1, q2, \ldots, qn] \leftarrow search\ G(V, E)$ for sequence of edges connecting q_{init} to q_{goal}
3: **for** $i = 1,2,\ldots,n$ **do**
4: **if** $q_i.checked = false$ **and** IsConfigCollisionFree$(q_i) = false$ **then**
5: remove q_i from roadmap; **goto** line 2
6: **else**
7: $q_i.checked \leftarrow true$
8: **while** no edge collisions are found **and** minimum resolution not reached **do**
9: **for** $i = 1, 2, \ldots, n - 1$ **do**
10: $(q_i, q_{i+1}).res \leftarrow (q_i, q_{i+1}).res/2$; check $(q_i, q_{i+1}).path$ at resolution $(q_i, q_{i+1}).res$
11: **if** collision found in $(q_i, q_{i+1}).path$ **then**
12: remove (q_i, q_{i+1}) from roadmap; **goto** line 2
13: **return** $(q_1, q_2).path \circ \cdots \circ (q_{n-1}, q_n).path$

Lazy PRM

- Lazy PRM: perform collision checking only when necessary

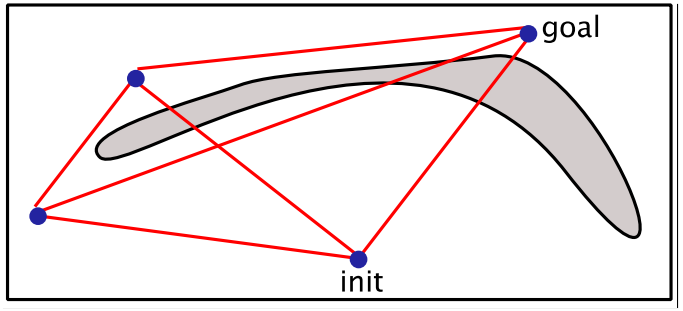


LazyRoadmapCollissionChecking

1: **for** several times **do**
2: $[q1, q2, \ldots, qn] \leftarrow search\ G(V, E)$ for sequence of edges connecting q_{init} to q_{goal}
3: **for** $i = 1,2,\ldots,n$ **do**
4: **if** $q_i.checked = false$ **and** IsConfigCollisionFree$(q_i) = false$ **then**
5: remove q_i from roadmap; **goto** line 2
6: **else**
7: $q_i.checked \leftarrow true$
8: **while** no edge collisions are found **and** minimum resolution not reached **do**
9: **for** $i = 1, 2, \ldots, n - 1$ **do**
10: $(q_i, q_{i+1}).res \leftarrow (q_i, q_{i+1}).res/2$; check $(q_i, q_{i+1}).path$ at resolution $(q_i, q_{i+1}).res$
11: **if** collision found in $(q_i, q_{i+1}).path$ **then**
12: remove (q_i, q_{i+1}) from roadmap; **goto** line 2
13: **return** $(q_1, q_2).path \circ \cdots \circ (q_{n-1}, q_n).path$

Lazy PRM

- Lazy PRM: perform collision checking only when necessary

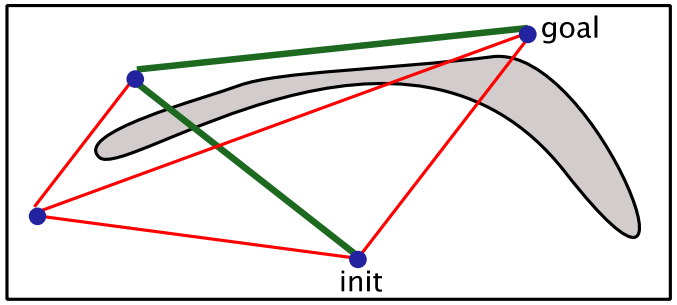


LazyRoadmapCollissionChecking

1: **for** several times **do**
2: $[q1, q2, \ldots, qn] \leftarrow search\ G(V, E)$ for sequence of edges connecting q_{init} to q_{goal}
3: **for** $i = 1,2,\ldots,n$ **do**
4: **if** $q_i.checked = false$ **and** $\text{IsConfigCollisionFree}(q_i) = false$ **then**
5: remove q_i from roadmap; **goto** line 2
6: **else**
7: $q_i.checked \leftarrow true$
8: **while** no edge collisions are found **and** minimum resolution not reached **do**
9: **for** $i = 1, 2, \ldots, n - 1$ **do**
10: $(q_i, q_{i+1}).res \leftarrow (q_i, q_{i+1}).res/2$; check $(q_i, q_{i+1}).path$ at resolution $(q_i, q_{i+1}).res$
11: **if** collision found in $(q_i, q_{i+1}).path$ **then**
12: remove (q_i, q_{i+1}) from roadmap; **goto** line 2
13: **return** $(q_1, q_2).path \circ \cdots \circ (q_{n-1}, q_n).path$

Lazy PRM

- Lazy PRM: perform collision checking only when necessary

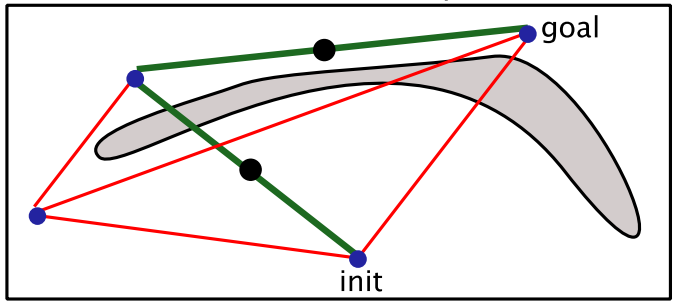


LazyRoadmapCollissionChecking

1: **for** several times **do**
2: $[q1, q2, \ldots, qn] \leftarrow search\ G(V, E)$ for sequence of edges connecting q_{init} to q_{goal}
3: **for** $i = 1,2,\ldots,n$ **do**
4: **if** $q_i.checked = false$ **and** IsConfigCollisionFree$(q_i) = false$ **then**
5: remove q_i from roadmap; **goto** line 2
6: **else**
7: $q_i.checked \leftarrow true$
8: **while** no edge collisions are found **and** minimum resolution not reached **do**
9: **for** $i = 1, 2, \ldots, n - 1$ **do**
10: $(q_i, q_{i+1}).res \leftarrow (q_i, q_{i+1}).res/2$; check $(q_i, q_{i+1}).path$ at resolution $(q_i, q_{i+1}).res$
11: **if** collision found in $(q_i, q_{i+1}).path$ **then**
12: remove (q_i, q_{i+1}) from roadmap; **goto** line 2
13: **return** $(q_1, q_2).path \circ \cdots \circ (q_{n-1}, q_n).path$

Lazy PRM

- Lazy PRM: perform collision checking only when necessary

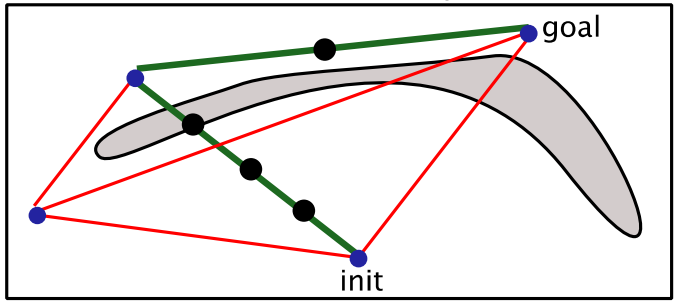


LazyRoadmapCollissionChecking

1: **for** several times **do**
2: $[q1, q2, \ldots, qn] \leftarrow search\ G(V, E)$ for sequence of edges connecting q_{init} to q_{goal}
3: **for** $i = 1,2,\ldots,n$ **do**
4: **if** $q_i.checked = false$ **and** IsConfigCollisionFree$(q_i) = false$ **then**
5: remove q_i from roadmap; **goto** line 2
6: **else**
7: $q_i.checked \leftarrow true$
8: **while** no edge collisions are found **and** minimum resolution not reached **do**
9: **for** $i = 1, 2, \ldots, n - 1$ **do**
10: $(q_i, q_{i+1}).res \leftarrow (q_i, q_{i+1}).res/2$; check $(q_i, q_{i+1}).path$ at resolution $(q_i, q_{i+1}).res$
11: **if** collision found in $(q_i, q_{i+1}).path$ **then**
12: remove (q_i, q_{i+1}) from roadmap; **goto** line 2
13: **return** $(q_1, q_2).path \circ \cdots \circ (q_{n-1}, q_n).path$

Lazy PRM

- Lazy PRM: perform collision checking only when necessary

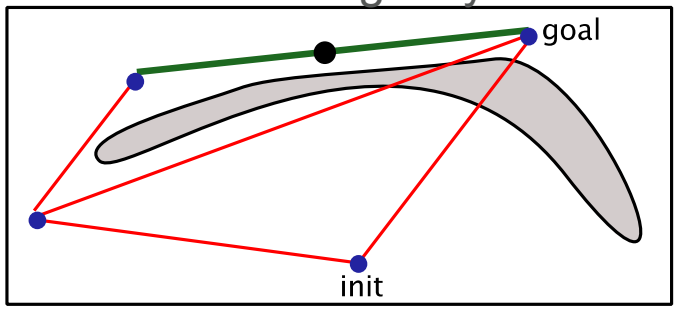


LazyRoadmapCollissionChecking

1: **for** several times **do**
2: $[q1, q2, \ldots, qn] \leftarrow search\ G(V, E)$ for sequence of edges connecting q_{init} to q_{goal}
3: **for** $i = 1,2,\ldots,n$ **do**
4: **if** $q_i.checked = false$ **and** IsConfigCollisionFree$(q_i) = false$ **then**
5: remove q_i from roadmap; **goto** line 2
6: **else**
7: $q_i.checked \leftarrow true$
8: **while** no edge collisions are found **and** minimum resolution not reached **do**
9: **for** $i = 1, 2, \ldots, n - 1$ **do**
10: $(q_i, q_{i+1}).res \leftarrow (q_i, q_{i+1}).res/2$; check $(q_i, q_{i+1}).path$ at resolution $(q_i, q_{i+1}).res$
11: **if** collision found in $(q_i, q_{i+1}).path$ **then**
12: remove (q_i, q_{i+1}) from roadmap; **goto** line 2
13: **return** $(q_1, q_2).path \circ \cdots \circ (q_{n-1}, q_n).path$

Lazy PRM

- Lazy PRM: perform collision checking only when necessary

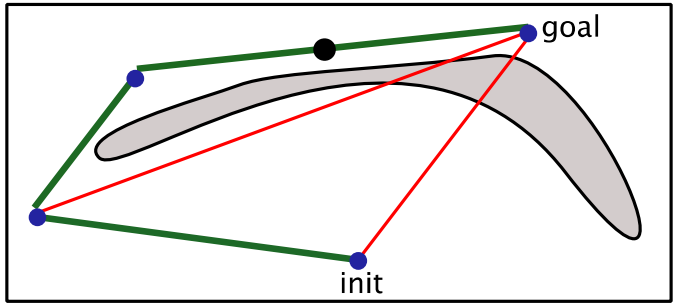


LazyRoadmapCollissionChecking

1: **for** several times **do**
2: $[q1, q2, \ldots, qn] \leftarrow search\ G(V, E)$ for sequence of edges connecting q_{init} to q_{goal}
3: **for** $i = 1,2,\ldots,n$ **do**
4: **if** $q_i.checked = false$ **and** IsConfigCollisionFree$(q_i) = false$ **then**
5: remove q_i from roadmap; **goto** line 2
6: **else**
7: $q_i.checked \leftarrow true$
8: **while** no edge collisions are found **and** minimum resolution not reached **do**
9: **for** $i = 1, 2, \ldots, n - 1$ **do**
10: $(q_i, q_{i+1}).res \leftarrow (q_i, q_{i+1}).res/2$; check $(q_i, q_{i+1}).path$ at resolution $(q_i, q_{i+1}).res$
11: **if** collision found in $(q_i, q_{i+1}).path$ **then**
12: remove (q_i, q_{i+1}) from roadmap; **goto** line 2
13: **return** $(q_1, q_2).path \circ \cdots \circ (q_{n-1}, q_n).path$

Lazy PRM

- Lazy PRM: perform collision checking only when necessary

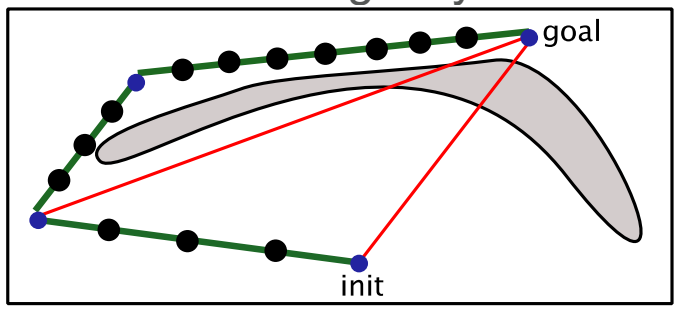


LazyRoadmapCollissionChecking

1: **for** several times **do**
2: $[q1, q2, \ldots, qn] \leftarrow search\ G(V, E)$ for sequence of edges connecting q_{init} to q_{goal}
3: **for** $i = 1,2,\ldots,n$ **do**
4: **if** $q_i.checked = false$ **and** IsConfigCollisionFree$(q_i) = false$ **then**
5: remove q_i from roadmap; **goto** line 2
6: **else**
7: $q_i.checked \leftarrow true$
8: **while** no edge collisions are found **and** minimum resolution not reached **do**
9: **for** $i = 1, 2, \ldots, n - 1$ **do**
10: $(q_i, q_{i+1}).res \leftarrow (q_i, q_{i+1}).res/2$; check $(q_i, q_{i+1}).path$ at resolution $(q_i, q_{i+1}).res$
11: **if** collision found in $(q_i, q_{i+1}).path$ **then**
12: remove (q_i, q_{i+1}) from roadmap; **goto** line 2
13: **return** $(q_1, q_2).path \circ \cdots \circ (q_{n-1}, q_n).path$

Lazy PRM

- Lazy PRM: perform collision checking only when necessary



LazyRoadmapCollissionChecking

1: **for** several times **do**
2: $[q1, q2, \ldots, qn] \leftarrow search\ G(V, E)$ for sequence of edges connecting q_{init} to q_{goal}
3: **for** $i = 1,2,\ldots,n$ **do**
4: **if** $q_i.checked = false$ **and** IsConfigCollisionFree$(q_i) = false$ **then**
5: remove q_i from roadmap; **goto** line 2
6: **else**
7: $q_i.checked \leftarrow true$
8: **while** no edge collisions are found **and** minimum resolution not reached **do**
9: **for** $i = 1, 2, \ldots, n - 1$ **do**
10: $(q_i, q_{i+1}).res \leftarrow (q_i, q_{i+1}).res/2$; check $(q_i, q_{i+1}).path$ at resolution $(q_i, q_{i+1}).res$
11: **if** collision found in $(q_i, q_{i+1}).path$ **then**
12: remove (q_i, q_{i+1}) from roadmap; **goto** line 2
13: **return** $(q_1, q_2).path \circ \cdots \circ (q_{n-1}, q_n).path$

Two Critical Issues with PRMS

- Several 100s implementations of PRMs exist
- There are two major sets of choices that affect the performance of these planners:
  - What node to connect to what other
  - Where to sample

Let’s now move to where to sample