Motion Planning Lecture 5

Sampling-Based Geometric Motion Planning: PRMs

Wolfgang Hönig (TU Berlin) and Andreas Orthey (Realtime Robotics)

May 22, 2024

Recap

Foundations

2 Weeks (problem formulation, terminology, collision checking)

Search-based

2 Weeks (A* and variants; state-lattice-based planning)

Sampling-based

5 Weeks (RRT, PRM, OMPL, Sampling Theory)

Optimization-based

2 Weeks (SCP, TrajOpt)

Current and Advanced Topics

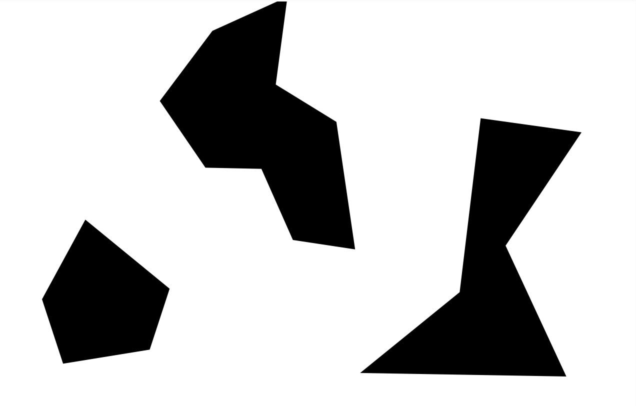
3 Weeks (Comparative Analysis, Machine Learning and Motion Planning, Hybrid- and Multi-Robot approaches)

PRM: Probabilistic Roadmaps

Recap: Lecture 3: Probabilistic roadmap

Probabilistic roadmap

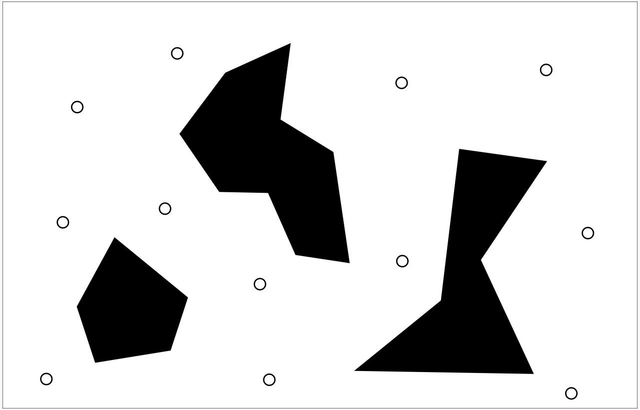
Idea: Sample random points in configuration space



Recap: Lecture 3: Probabilistic roadmap

Probabilistic roadmap

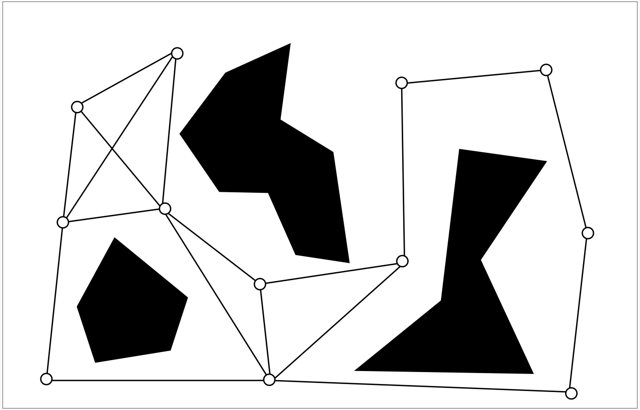
Idea: Sample random points in configuration space



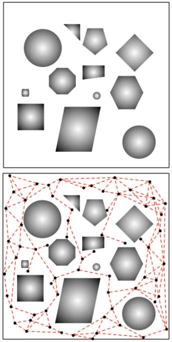
Recap: Lecture 3: Probabilistic roadmap

Probabilistic roadmap

Idea: Sample random points in configuration space



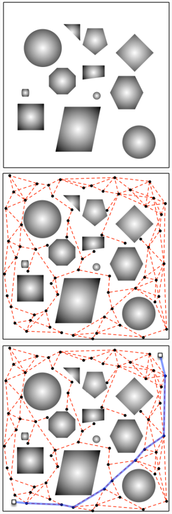
Probabilistic Roadmap [1]



- Randomized algorithm, published in 1996
- Two stages
  1. Pre-processing (given environment and robot)
     - Generate a weighted graph (roadmap)

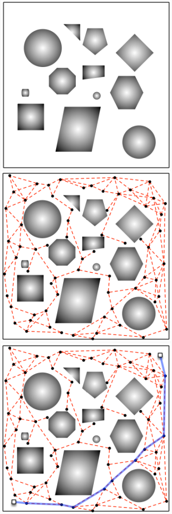
Probabilistic Roadmap [1]

- Randomized algorithm, published in 1996
- Two stages
  1. Pre-processing (given environment and robot)
     - Generate a weighted graph (roadmap)
  2. Query (given start and goal configuration)
     - Graph search



Probabilistic Roadmap [1]

- Randomized algorithm, published in 1996
- Two stages
  1. Pre-processing (given environment and robot)
     - Generate a weighted graph (roadmap)
  2. Query (given start and goal configuration)
     - Graph search

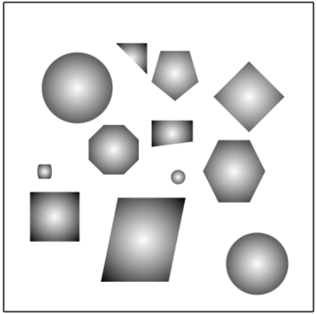


Multi-Query Planning

For the same environment and robot, only state 2 needs to be executed. Thus, PRM is a multi-query planner.

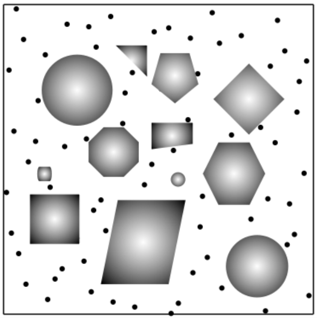
Graph Generation

```
def GenPRM(𝒬, 𝒲_free, ℬ(·), N):
  𝒢 = (𝒱, ℰ) = (∅, ∅)
  # Generate N vertices
  while |𝒱| < N:
    q = Sample(𝒬)
    if ℬ(q) ⊂ 𝒲_free:
      𝒱 = 𝒱 ∪ {q}
  # Connect vertices
  for q in 𝒱:
    for p in {p ∈ 𝒱 : isNeighbor(p, q)}:
      if path q to p feasible:
        ℰ = ℰ ∪ {path q to p}
  return 𝒢
```



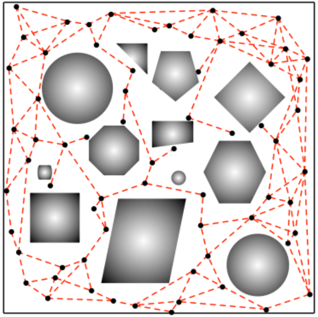
Graph Generation

```
def GenPRM(𝒬, 𝒲_free, ℬ(·), N):
    𝒢 = (𝒱, ℰ) = (∅, ∅)
    # Generate N vertices
    while |𝒱| < N:
        q = Sample(𝒬)
        if ℬ(q) ⊂ 𝒲_free:
            𝒱 = 𝒱 ∪ {q}
    # Connect vertices
    for q in 𝒱:
        for p in {p ∈ 𝒱 : isNeighbor(p, q)}:
            if path q to p feasible:
                ℰ = ℰ ∪ {path q to p}
    return 𝒢
```



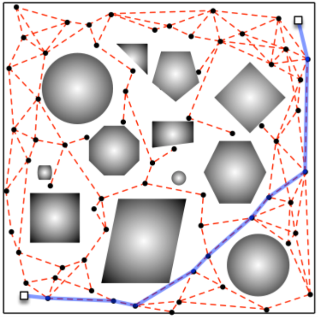
Graph Generation

```
def GenPRM(𝒬, 𝒲_free, ℬ(·), N):
    𝒢 = (𝒱, ℰ) = (∅, ∅)
    # Generate N vertices
    while |𝒱| < N:
        q = Sample(𝒬)
        if ℬ(q) ⊂ 𝒲_free:
            𝒱 = 𝒱 ∪ {q}
    # Connect vertices
    for q in 𝒱:
        for p in {p ∈ 𝒱 : isNeighbor(p, q)}:
            if path q to p feasible:
                ℰ = ℰ ∪ {path q to p}
    return 𝒢
```



Query

1. Add $\mathbf{q}_{start}$ and $\mathbf{q}_{goal}$ to $\mathcal{V}$
2. Connect $\mathbf{q}_{start}$ to at least one $\mathbf{q} \in \mathcal{V}$
3. Connect $\mathbf{q}_{goal}$ to at least one $\mathbf{q} \in \mathcal{V}$
4. Use A* to find shortest path from $\mathbf{q}_{start}$ to $\mathbf{q}_{goal}$



Ideal Roadmap Properties (1)

What constitutes a "good" roadmap?

- Accessible: For any $\mathbf{q}_{start} \in \mathcal{Q}_{free}$ we can compute a path to some $\mathbf{q} \in \mathcal{V}$
- Departable: For any $\mathbf{q}_{goal} \in \mathcal{Q}_{free}$ we can compute a path from some $\mathbf{q} \in \mathcal{V}$
- Connectivity-Preserving: For any $\mathbf{q}, \mathbf{p} \in \mathcal{Q}_{free}$ that can be connected, there is a path in the roadmap
- Efficient with factor ϵ: For any $\mathbf{q}, \mathbf{p} \in \mathcal{Q}_{free}$ that can be connected with cost c^*, there is a path in the roadmap with a cost of ϵc^* or less
- Sparse: As few vertices and edges as possible

Ideal Roadmap Properties (2)

```
def GenPRM(𝒬, 𝒲_free, ℬ(·), N):
    𝒢 = (𝒱, ℰ) = (∅, ∅)
    # Generate N vertices
    # ...
    # Connect vertices
    # ...
    return 𝒢
```

When is the roadmap accessible, departable, connectivity-preserving, efficient, sparse?

$N \to \infty$

$N \to 0$

Ideal Roadmap Properties (2)

```
def GenPRM(𝒬, 𝒲_free, ℬ(·), N):
    𝒢 = (𝒱, ℰ) = (∅, ∅)
    # Generate N vertices
    # ...
    # Connect vertices
    # ...
    return 𝒢
```

When is the roadmap accessible, departable, connectivity-preserving, efficient, sparse?

$N \to \infty$

There will be $\mathbf{q} \in \mathcal{V}$ for almost every configuration in $\mathcal{Q}_{free}$.

\+ Accessible, Departable, Connectivity-Preserving, Efficient

\- Not sparse; very slow computation (both pre-processing and query)

$N \to 0$

\- Not Accessible, Not Departable, Not Connectivity-Preserving, Not Efficient

\+ Sparse

Practical PRM Considerations

- Can run pre-processing and query in parallel
  - Incrementally increase roadmap size
  - Periodically check if a solution can be found
  - Avoids picking N explicitly
- Many variants are possible based on choice of Sample, isNeighbor, and feasible path computation
- Result is in the correct homeomorphism class, but often far from optimal ($\Rightarrow$ Path Smoothing)

Sampling

Rejection Sampling

Rejection Sampling

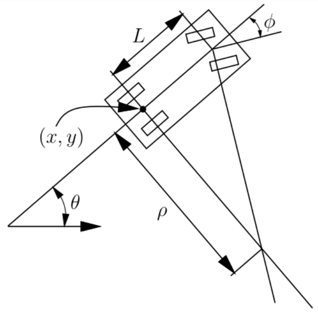
1. Sample $\mathbf{q} \in \mathcal{Q}$ from a given distribution
2. Repeat until $\mathcal{B}(\mathbf{q}) \subset \mathcal{W}_{free}$ (or: $\mathbf{q} \in \mathcal{Q}_{free}$)

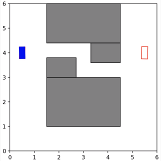
Does not require $\mathcal{Q}_{free}$ explicitly

Inefficient in highly constrained spaces

Common distributions: Uniform distribution, Gaussian/Normal distribution, Deterministic Sequence

Composite Sampling (1)





- Configuration $\mathbf{q} = (x, y, \theta) \in \mathcal{Q}$
- All components are independent $\Rightarrow$ we can sample all components independently:
  1. $x \sim \mathcal{U}(0, 8)$
  2. $y \sim \mathcal{U}(0, 8)$
  3. $\theta \sim \mathcal{U}(0, 2\pi)$
  4. Resulting state $\mathbf{q}$ is still drawn from a uniform distribution

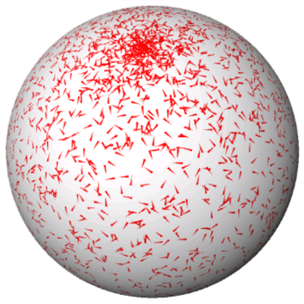
Composite Sampling (2)

What about $SO(3)$ (orientation in 3D)?

Idea 0

- Sample three Euler angles uniformly between $[0, 2\pi)$
- Construct rotation (rotation matrix, quaternion, ...)

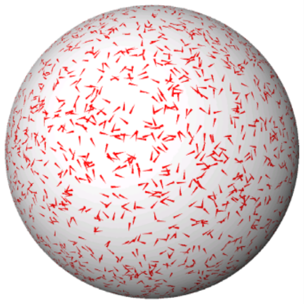
not uniform, because Euler angles are not independent



Source: [2]

Composite Sampling (3)

What about $SO(3)$ (orientation in 3D)?



Source: [2]

Better idea

- Sample three numbers from $\mathcal{U}(0,1)$
- Quaternion

$$(qw, qx, qy, qz) = \\ (\sqrt{1-u_1}\sin 2\pi u_2, \sqrt{1-u_1}\cos 2\pi u_2, \\ \sqrt{u_1}\sin 2\pi u_3, \sqrt{u_1}\cos 2\pi u_3)$$

More: Section 5.5.2 of [3] and [2]

Deterministic Sampling (1)

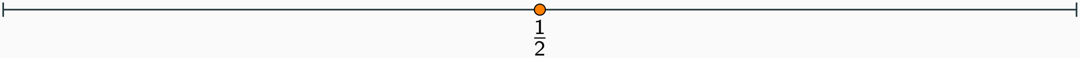
- Halton sequence (1960): better uniformity than pseudo-random numbers and incremental
- Examples:

$p = 2$:

Deterministic Sampling (1)

- Halton sequence (1960): better uniformity than pseudo-random numbers and incremental
- Examples:

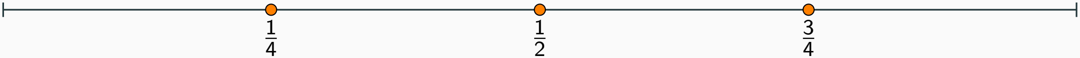
$p = 2$:



Deterministic Sampling (1)

- Halton sequence (1960): better uniformity than pseudo-random numbers and incremental
- Examples:

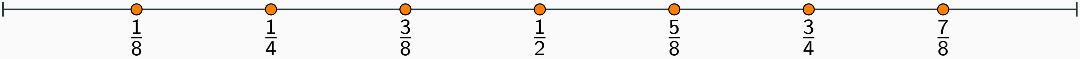
$p = 2$:



Deterministic Sampling (1)

- Halton sequence (1960): better uniformity than pseudo-random numbers and incremental
- Examples:

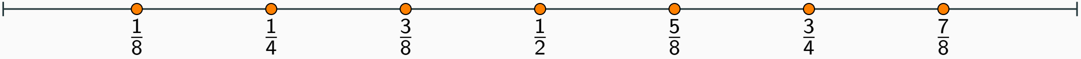
$p = 2$:



Deterministic Sampling (1)

- Halton sequence (1960): better uniformity than pseudo-random numbers and incremental
- Examples:

$p = 2$:

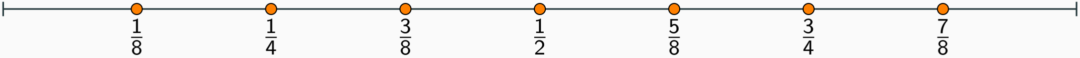


$p = 3$:

Deterministic Sampling (1)

- Halton sequence (1960): better uniformity than pseudo-random numbers and incremental
- Examples:

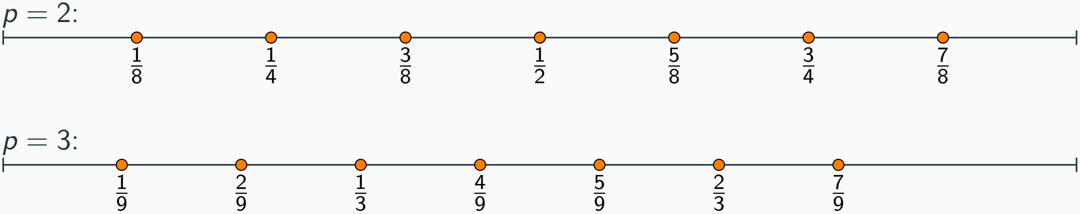
$p = 2$:



$p = 3$:

Deterministic Sampling (1)

- Halton sequence (1960): better uniformity than pseudo-random numbers and incremental
- Examples:



Deterministic Sampling (2)

- Key idea: The i^{th} number written using prime number p as base

```
def GenHalton(p, i):
  h = 0
  j = i
  f = 1 / p
  while j > 0:
    quotient q, remainder r = j / p
    h = h + f * r
    j = q
    f = f / p
  return h
```

GenHalton(2, 6)

$h = 0; j = 6; f = \frac{1}{2}$

$q = 3; r = 0; h = 0; j = 3; f = \frac{1}{2 \cdot 2}$

$q = 1; r = 1; h = \frac{1}{4}; j = 1; f = \frac{1}{4 \cdot 2}$

$q = 0; r = 1; h = \frac{1}{4} + \frac{1}{8}; j = 0; f = \frac{1}{8 \cdot 2}$

$\frac{3}{8}$

Note: 6 (base 10) is 110 (base 2);
$\frac{3}{8} = 0\frac{1}{2} + 1\frac{1}{4} + 1\frac{1}{8}$

Deterministic Sampling (3)

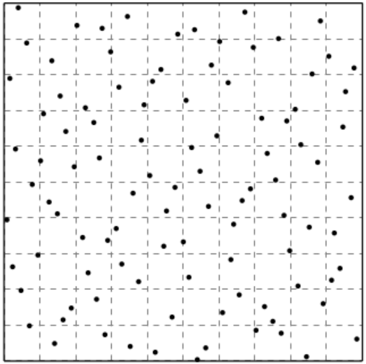
- Example sequence ($p = 2, i = 1, \ldots, 7$):

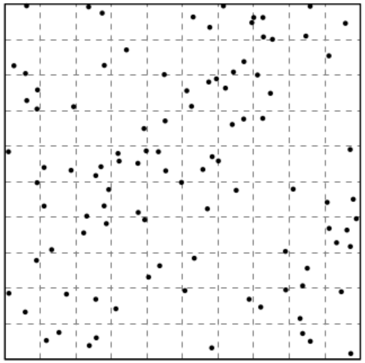
$$\frac{1}{2}, \qquad \frac{1}{4}, \frac{3}{4}, \qquad \frac{1}{8}, \frac{5}{8}, \frac{3}{8}, \frac{7}{8}$$

- For higher-dimensional configuration spaces, use different prime numbers per dimension
  - $\mathcal{Q} \subset \mathbb{R}^2$: Use $p = 2$ for x and $p = 3$ for y

Deterministic Sampling (4)

2D Case, 100 Samples

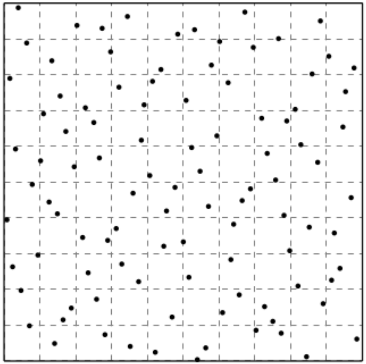


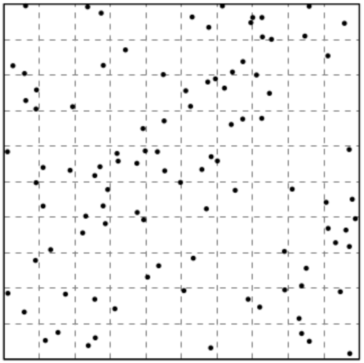


Which one is pseudo-random uniform and which one is Halton?

Deterministic Sampling (4)

2D Case, 100 Samples





Halton

Uniform

Dispersion: How well does a sampler cover the space? (1)

Dispersion

Let $\mathcal{P} \subset \mathcal{Q}$ be a set of points in the configuration space and $d : \mathcal{Q} \times \mathcal{Q} \to \mathbb{R}_{\geq 0}$ be a metric. The dispersion is the maximum distance from a configuration $\mathbf{q} \in \mathcal{Q}$ to its nearest sample $\mathbf{p} \in \mathcal{P}$:

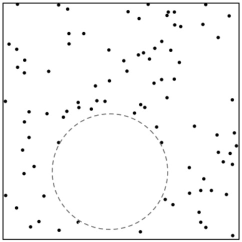
$$dispersion_d(\mathcal{P}) = \max_{\mathbf{q} \in \mathcal{Q}} \min_{\mathbf{p} \in \mathcal{P}} d(q, p).$$

Dispersion (2)

Dispersion

$$dispersion_d(\mathcal{P}) = \max_{\mathbf{q}\in\mathcal{Q}} \min_{\mathbf{p}\in\mathcal{P}} d(x, p).$$

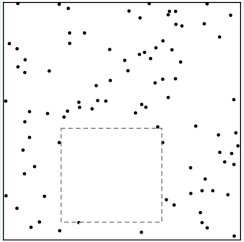
L2-Norm



Source: [4]

$$\mathcal{Q} = [0, 1]^2 \subset \mathbb{R}^2;$$
$$d(x, p) = \sqrt{(x_1 - p_1)^2 + (x_2 - p_2)^2}$$

L∞-Norm

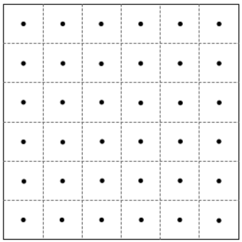


Source: [4]

$$\mathcal{Q} = [0, 1]^2 \subset \mathbb{R}^2;$$
$$d(x, p) = \max\left(|x_1 - p_1|, |x_2 - p_2|\right)$$

Dispersion (3)

Minimal L∞ Dispersion: A Grid



Source: [4]

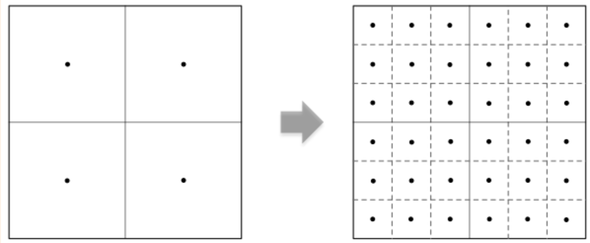
$$\mathcal{Q} = [0,1]^2 \subset \mathbb{R}^2;\ \mathit{dispersion}_{L\infty} = \frac{1}{2\sqrt[d]{n}} = \frac{1}{2\sqrt[2]{36}} = \frac{1}{12}$$

Dispersion (4)

Why do we not use a grid for sampling?

Dispersion (4)

Why do we not use a grid for sampling?



Source: [4]

A grid is not incremental (a resolution change requires exponentially many additional points).

Here: $n = 4 \Rightarrow n \cdot 3^d = 4 \cdot 3^2 = 36$ configurations

Sampling Method Pros and Cons

Sampling property	Uniform grids	Random sampling	Halton sequences
dispersion	$O(\frac{1}{\sqrt[d]{n}})$	$O(\frac{\ln^{1/d}(n)}{\sqrt[d]{n}})$	$O(\frac{f(d)}{\sqrt[d]{n}})$
incremental	no	yes	yes
lattice	yes	no	yes (more complex)

Lattice Property

Neighbors can be computed directly

Nearest-Neighbor Computation

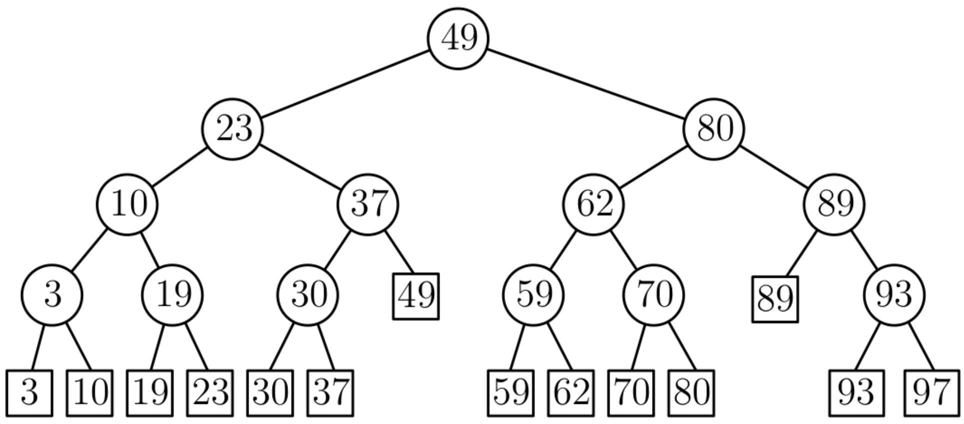
Goal

Efficient Nearest-Neighbor calculation with the following interface:

- `addConfiguration(q) -> None`: Adding a configuration $\mathbf{q} \in \mathcal{Q}$ to the datastructure
- `queryK(q, k) -> [Configuration]`: Return the k nearest (with respect to a distance metric) configurations of $\mathbf{q}$
- `queryR(q, r) -> [Configuration]`: Return the configurations that are within a given distance r of $\mathbf{q}$

Background: Balanced binary search trees (1)

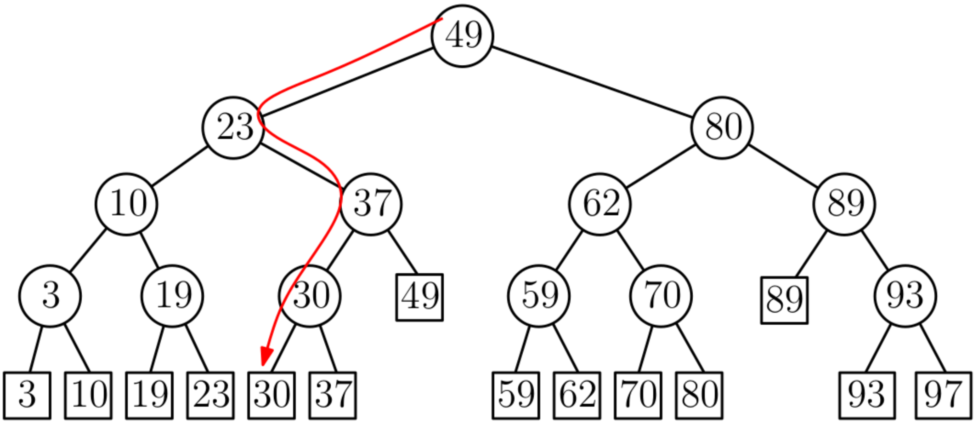
A balanced binary search tree with the points in the leaves



Source: [5]

Background: Balanced binary search trees (2)

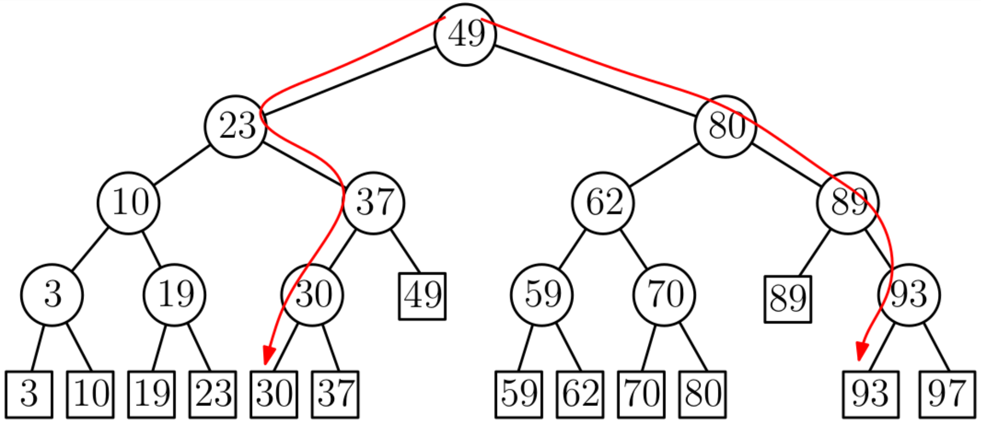
Searching if 25 is part of the tree



Source: [5]

Background: Balanced binary search trees (3)

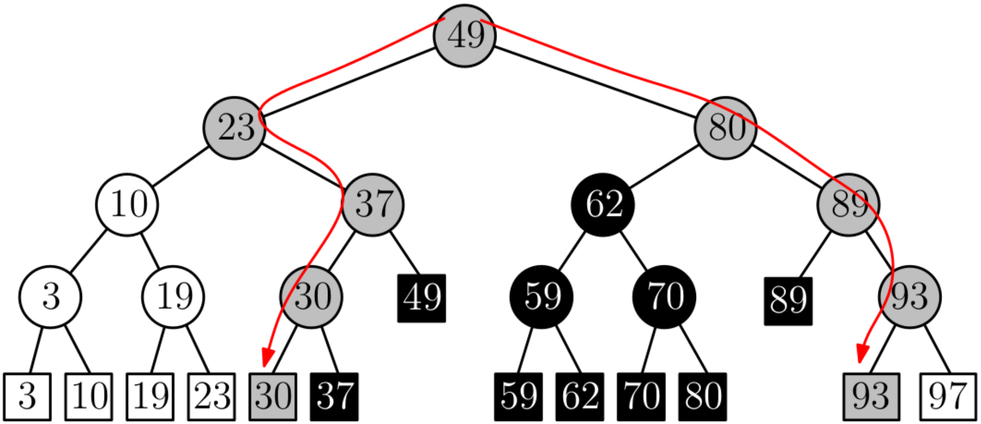
Search path for 25 and 90



Source: [5]

Background: Balanced binary search trees (4)

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



Source: [5]

Background: Balanced binary search trees (5)

- Build a binary search tree of N numbers (time: $\mathcal{O}(N \log N)$; space: $\mathcal{O}(N)$)
- Finding an entry: time: $\mathcal{O}(\log N)$
- Range query of K numbers: time $\mathcal{O}(\log N + K)$

What is the naive Range Query Time Complexity?

Background: Balanced binary search trees (5)

- Build a binary search tree of N numbers (time: $\mathcal{O}(N \log N)$; space: $\mathcal{O}(N)$)
- Finding an entry: time: $\mathcal{O}(\log N)$
- Range query of K numbers: time $\mathcal{O}(\log N + K)$

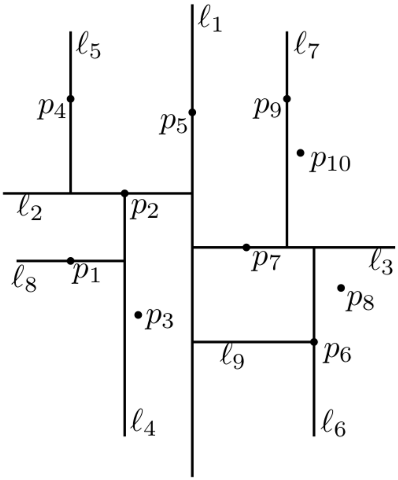
What is the naive Range Query Time Complexity?

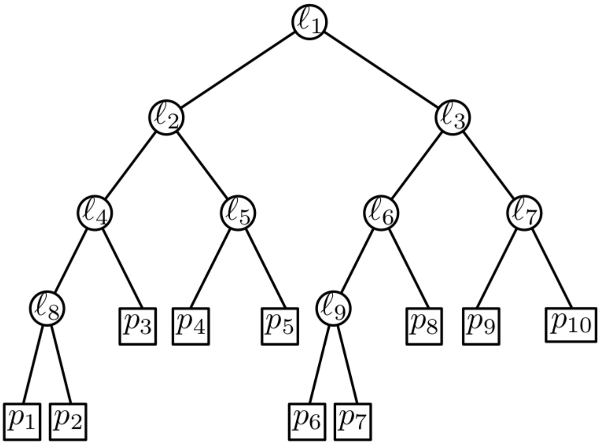
We have to consider all N numbers $\Rightarrow \mathcal{O}(N)$

Kd-Trees

- Extend the same idea to multi-dimensional data
- 2D case:
  - Split the point set alternatingly by x-coordinate and by y-coordinate
  - Split by x-coordinate: split by a vertical line that has half the points left or on, and half right
  - Split by y-coordinate: split by a horizontal line that has half the points below or on, and half above

Kd-Tree Example





Source: [5]

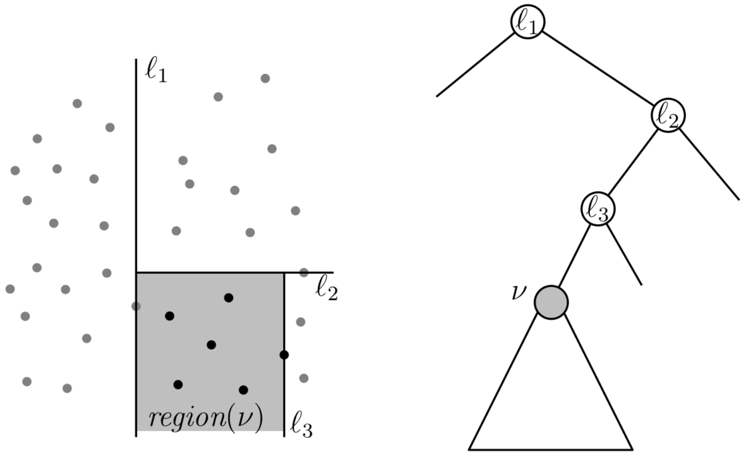
Kd-Tree Construction

Algorithm $\textsc{BuildKdTree}(P, depth)$

if P contains only one point
  then return a leaf storing this point
  else if $depth$ is even
    then Split P with a vertical line ℓ through the median x-coordinate into P_1 (left of or on ℓ) and P_2 (right of ℓ)
    else Split P with a horizontal line ℓ through the median y-coordinate into P_1 (below or on ℓ) and P_2 (above ℓ)
  $v_{\text{left}} \leftarrow \textsc{BuildKdTree}(P_1, depth + 1)$
  $v_{\text{right}} \leftarrow \textsc{BuildKdTree}(P_2, depth + 1)$
  Create a node v storing ℓ, make v_{left} the left child of v, and make v_{right} the right child of v.
  return v

Source: [5]

Kd-Tree Regions of Nodes

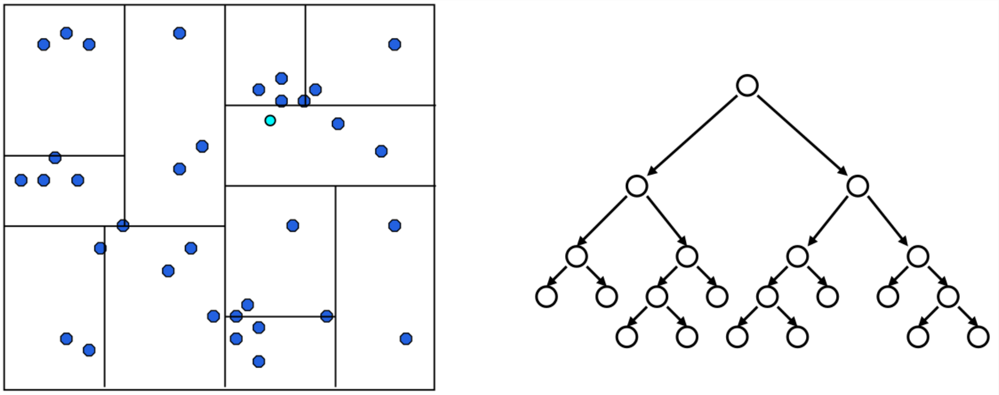


Source: [5]

Region can be:

- Stored explicitly at every node, OR
- Computed on-the-fly, while traversing from the root

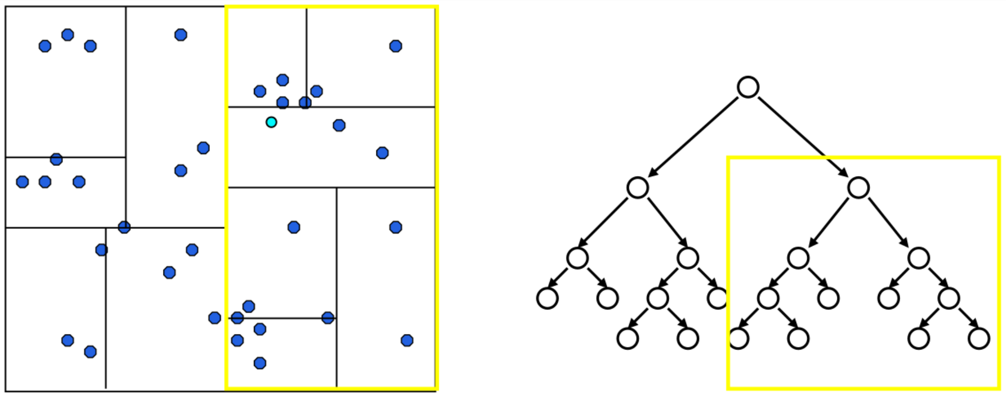
Kd-Tree Query (1)



Source: [6]

Traverse existing tree to find region of query point (leaf in tree)

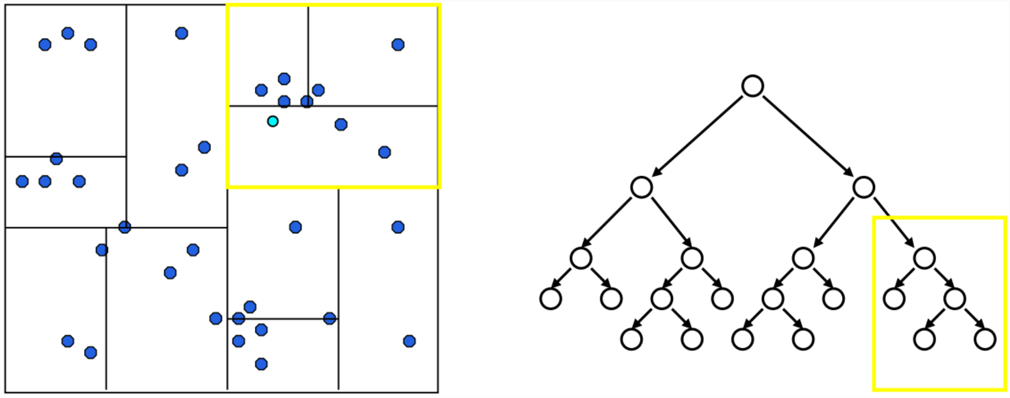
Kd-Tree Query (1)



Source: [6]

Traverse existing tree to find region of query point (leaf in tree)

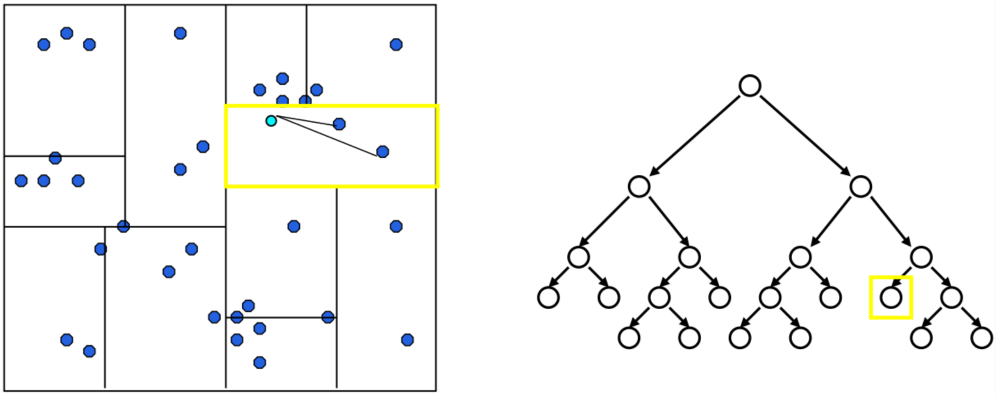
Kd-Tree Query (1)



Source: [6]

Traverse existing tree to find region of query point (leaf in tree)

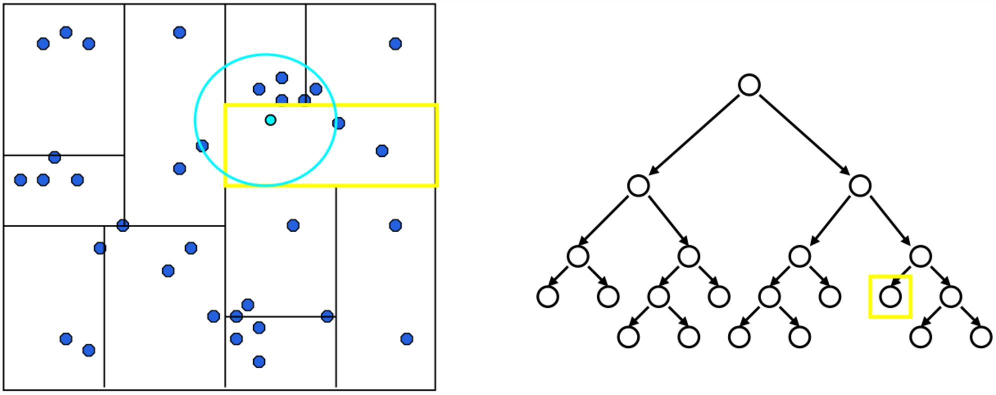
Kd-Tree Query (2)



Source: [6]

At leaf node: compute distance to each point in the node

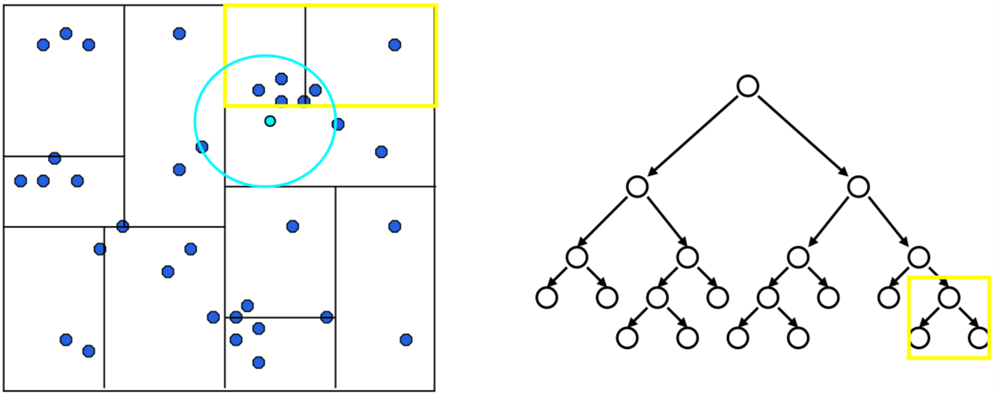
Kd-Tree Query (2)



Source: [6]

At leaf node: compute distance to each point in the node

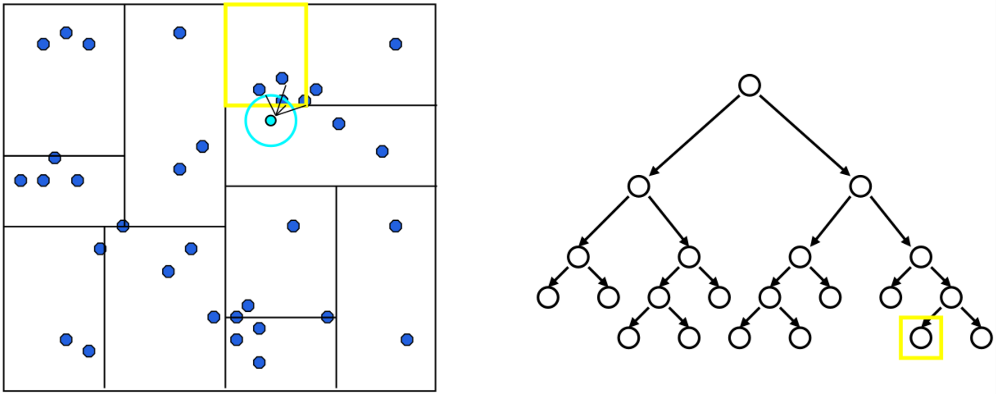
Kd-Tree Query (3)



Source: [6]

Backtrack to sibling nodes

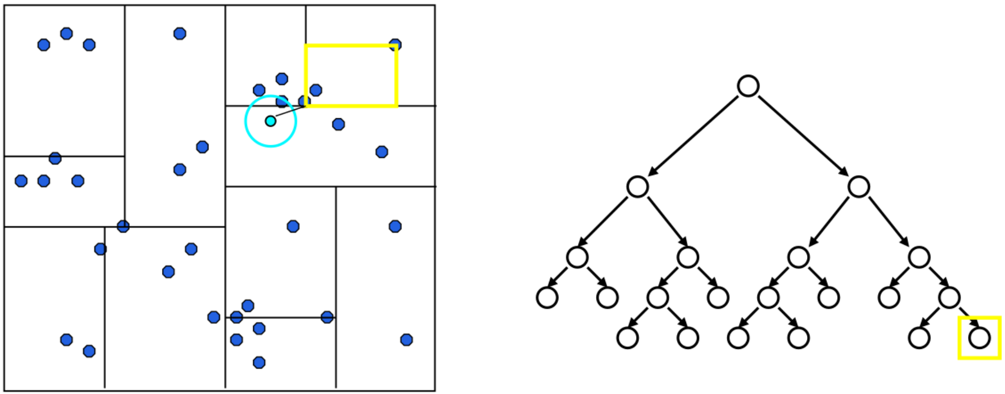
Kd-Tree Query (4)



Source: [6]

Update distance bounds, when a new nearest neighbor is found

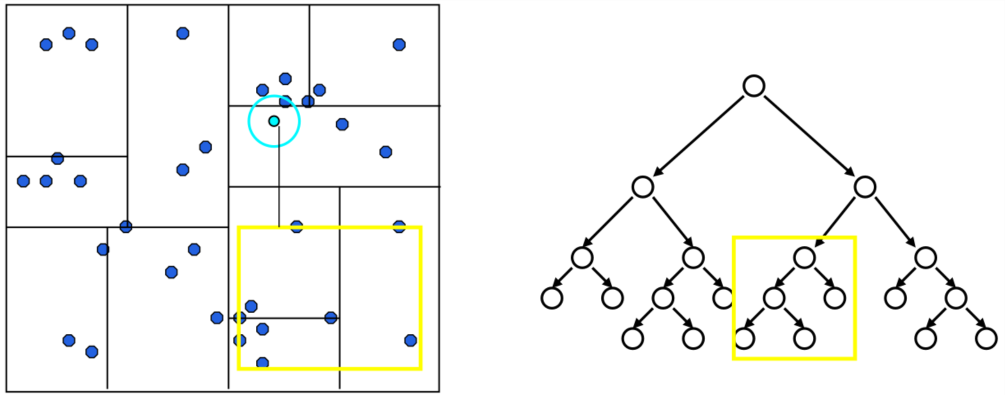
Kd-Tree Query (5)



Source: [6]

Prune search area based on region bounds and distance bounds

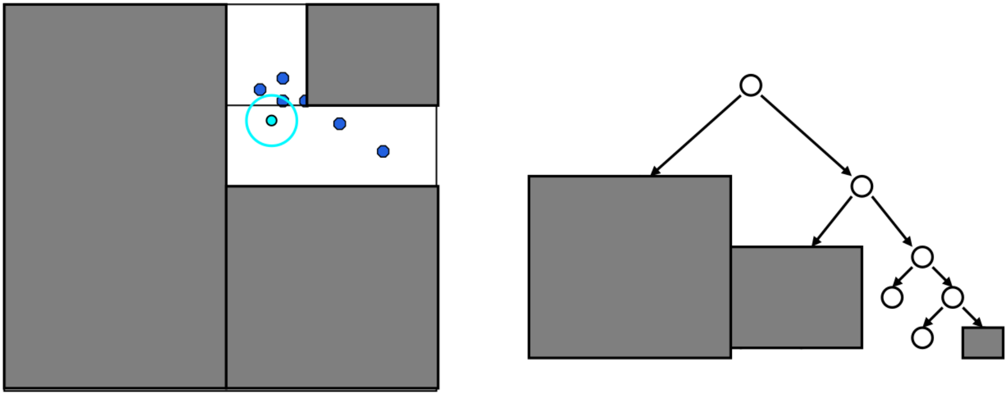
Kd-Tree Query (5)



Source: [6]

Prune search area based on region bounds and distance bounds

Kd-Tree Query (5)



Source: [6]

Prune search area based on region bounds and distance bounds

Kd-Tree: Practical Notes

- KD-trees are much faster than a naive implementation (time complexity: $\mathcal{O}(N^{1-1/d} + K)$
  - This is poor for high-dimensional spaces
  - Approximate Nearest Neighbor Algorithms (ANN) can help (but theoretical implications for sampling-based planners unknown)
- Possible to support more complicated configuration spaces (such as $SE(3)$)
- KD-Trees have a construction and query stage!

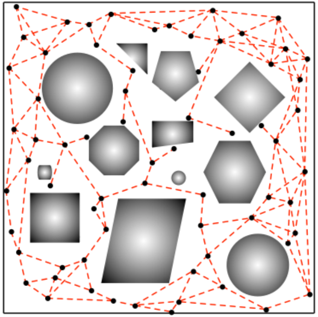
How can KD-trees be used incrementally?

- Reconstruct the tree only every 1000 nodes, or so
- Keep Kd-tree and plain list since last reconstruction
- For a query, search both Kd-tree and list

PRM - Revisited

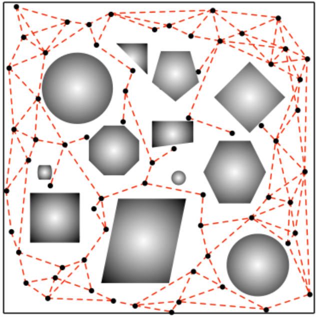
PRM

```
def GenPRM($\mathcal{Q}$, $\mathcal{W}_{free}$, $\mathcal{B}(\cdot)$, $N$):
    $\mathcal{G} = (\mathcal{V}, \mathcal{E}) = (\emptyset, \emptyset)$
    # Generate N vertices
    while $|\mathcal{V}| < N$:
        $\mathbf{q}$ = Sample($\mathcal{Q}$)
        if $\mathcal{B}(\mathbf{q}) \subset \mathcal{W}_{free}$:
            $\mathcal{V} = \mathcal{V} \cup \{\mathbf{q}\}$
    # Connect vertices
    for $\mathbf{q}$ in $\mathcal{V}$:
        for $\mathbf{p}$ in $\{\mathbf{p} \in \mathcal{V} : \text{isNeighbor}(\mathbf{p}, \mathbf{q})\}$:
            if path $\mathbf{q}$ to $\mathbf{p}$ feasible:
                $\mathcal{E} = \mathcal{E} \cup \{\text{path } \mathbf{q} \text{ to } \mathbf{p}\}$
    return $\mathcal{G}$
```



PRM

```
def GenPRM(𝒬, 𝒲_free, ℬ(·), N):
    𝒢 = (𝒱, ℰ) = (∅, ∅)
    # Generate N vertices
    while |𝒱| < N:
        q = Sample(𝒬)
        if ℬ(q) ⊂ 𝒲_free:
            𝒱 = 𝒱 ∪ {q}
    # Connect vertices
    for q in 𝒱:
        for p in {p ∈ 𝒱 : isNeighbor(p, q)}:
            if path q to p feasible:
                ℰ = ℰ ∪ {path q to p}
    return 𝒢
```



Motion Collision Checking

- Flexible Collision Library has continuous collision checking
  - Works as long as the motion is a linear interpolation of two configurations
- Alternative:
  - Compute intermediate configurations during the motion
  - Check the collision for each intermediate configuration
  - Note, that this is significantly slower than continuous collision checking

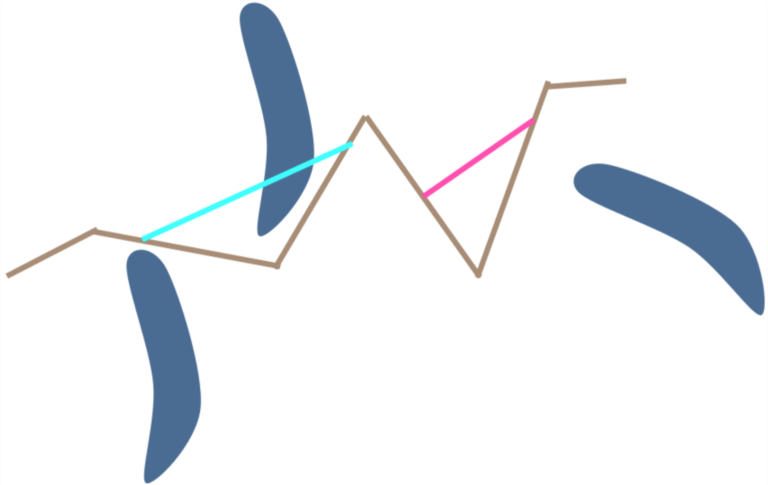
Path Smoothing

- The output of PRM is often far from optimal
- Post-processing of the data can help

Path Shortening

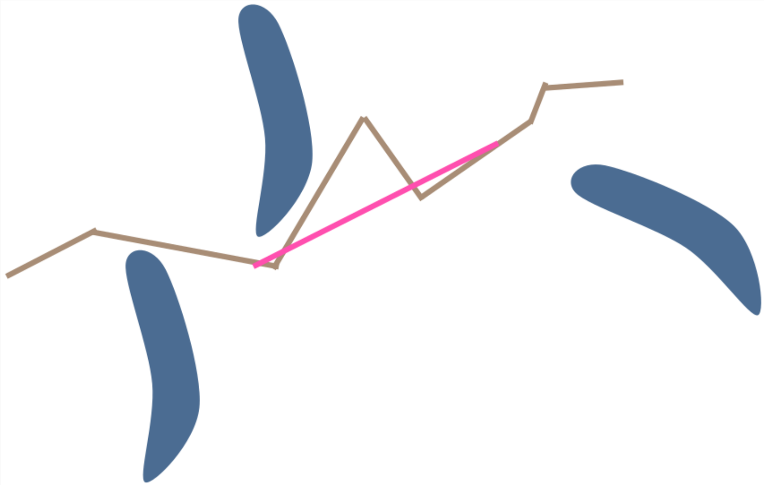
```
def PathShortening(⟨q_1, q_2, ..., q_N⟩):
  Pick q_i, q_j randomly
  if (q_i, q_j) can be connected by a line:
    replace path between q_i, q_j using the line
```

Path Shortening Example



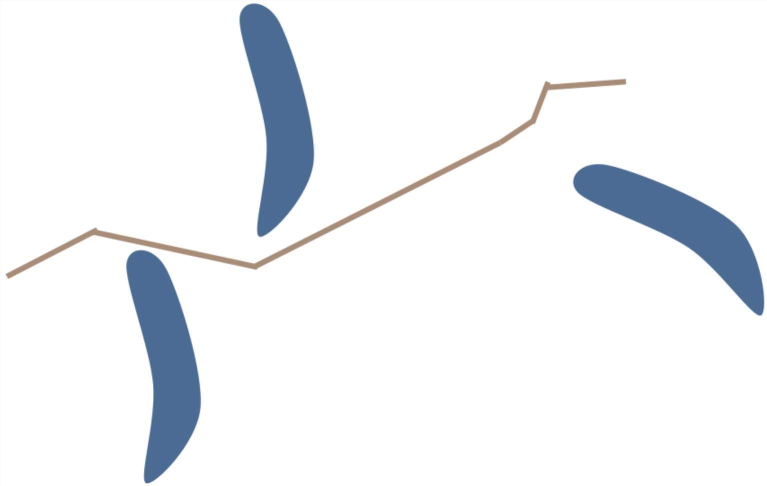
Source: [6]

Path Shortening Example



Source: [6]

Path Shortening Example



Source: [6]

Theoretical Properties

PRMs are probabilistically complete

Guaranteed to find a solution, if one exists, but only in the limit of the number of samples (that is, only as the number of samples approaches infinity).

- This is a milder form of completeness
- Typically no convergence guarantee, so not that helpful in practice
- Sampling-based planning works very well in high-dimensional spaces

Conclusion

- PRM are multi-query planners
- Kd-trees allow fast nearest-neighbor computation
- Concept simple, but difficult in details
  - Choice of sampling
  - Choice of nearest-neighbor computation
  - Handling of metric spaces correctly throughout

Next Time

- Tree-based Planners; Optimizing Planners

Suggested Reading

1. Oren Salzman. "Sampling-Based Robot Motion Planning". In: *Communications of the ACM* 62.10 (Sept. 24, 2019), pp. 54–63. ISSN: 0001-0782. DOI: 10.1145/3318164
2. F. Bullo and S. L. Smith. *Lectures on Robotic Planning and Kinematics*. 2019. URL: http://motion.me.ucsb.edu/book-lrpk/, Chapter 4
3. Steven M. LaValle. *Planning algorithms*. Cambridge University Press, 2006. ISBN: 978-0-521-86205-9. URL: http://planning.cs.uiuc.edu, Section 5.6
4. Kevin M. Lynch and Frank C. Park. *Modern Robotics*. Cambridge University Press, 2017. ISBN: 978-1-107-15630-2. URL: http://hades.mech.northwestern.edu/index.php/Modern_Robotics, Section 10.5

[1] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces". In: *IEEE Transactions on Robotics and Automation* 12.4 (Aug. 1996), pp. 566–580. ISSN: 2374-958X. DOI: 10.1109/70.508439.

[2] J.J. Kuffner. "Effective Sampling and Distance Metrics for 3D Rigid Body Path Planning". In: *IEEE International Conference on Robotics and Automation*. Vol. 4. 2004, pp. 3993–3998. DOI: 10.1109/ROBOT.2004.1308895.

[3] Steven M. LaValle. *Planning algorithms*. Cambridge University Press, 2006. ISBN: 978-0-521-86205-9. URL: http://planning.cs.uiuc.edu.

[4] F. Bullo and S. L. Smith. *Lectures on Robotic Planning and Kinematics*. 2019. URL: http://motion.me.ucsb.edu/book-lrpk/.

[5] Frank Staals. *Geometric Algorithms*. 2021. URL: http://www.cs.uu.nl/docs/vakken/ga/2021/.

[6] Dmitry Berenson. "Motion Planning: Robotics and Beyond". In: (2021). URL: https://web.eecs.umich.edu/~dmitryb/courses/winter2021motionplanning/index.html.

[7] Oren Salzman. "Sampling-Based Robot Motion Planning". In: *Communications of the ACM* 62.10 (Sept. 24, 2019), pp. 54–63. ISSN: 0001-0782. DOI: 10.1145/3318164.

[8] Kevin M. Lynch and Frank C. Park. *Modern Robotics*. Cambridge University Press, 2017. ISBN: 978-1-107-15630-2. URL: http://hades.mech.northwestern.edu/index.php/Modern_Robotics.