Robot Programming

Finding Neighbors
KD-Trees

G. Grisetti

grisetti}@diag.uniroma1.it

Department of Computer, Control, and Management Engineering
Sapienza University of Rome

Neighbor Search

Given:

- a collection of vectors $\mathcal{P} = \{\mathbf{p}_n\}_{n=1:N} \qquad \mathbf{p}_n \in \Re^k$
- a query vector $\mathbf{p}_q \in \Re^k$
- a distance metric $d(\mathbf{p}_n, \mathbf{p}_q) \in \Re^+$

Find either:

- the point in the collection that is the ***closest*** to the query, according to the metric

$$\mathbf{p}_i = \underset{\mathbf{p}_n \in \mathcal{P}}{\operatorname{argmin}}\, d(\mathbf{p}_q, \mathbf{p}_n)$$

- the points in the collection whose distance from the query is smaller than a value ϵ

$$\mathcal{P}' = \{\mathbf{p}_i \in \mathcal{P}, d(\mathbf{p}_q, \mathbf{p}_i) < \epsilon\}$$

Distance Metrics

Examples:

- Squared Norm

$$\|\mathbf{p}_i - \mathbf{p}_j\|^2 = (\mathbf{p}_i - \mathbf{p}_j)^T (\mathbf{p}_i - \mathbf{p}_j)$$

- Omega Norm

this should look familiar

$$\|\mathbf{p}_i - \mathbf{p}_j\|_{\mathbf{\Omega}}^2 = (\mathbf{p}_i - \mathbf{p}_j)^T \mathbf{\Omega} (\mathbf{p}_i - \mathbf{p}_j)$$

- Hamming distance (for binary descriptors)

  Integer valued distance between two bit strings having the same dimension. Its value is the number of different bits.

  example:

```
hamming(100100,101000)= 2
```

Trivial Approach

Brute Force:

compute the distance metric between the query point and *each* of the points in the collection and update the minimum.

Complexity: `O(N*cost_distance_metric)`

If we need to perform many queries, this results in unacceptable delays.

Idea: *use auxiliary search structures.*

KD-Trees

What if the points are K-dimensional, with `K` large?

- Exploiting the structure of the data or of the search
- Distance Maps (if dim small)
- KD-Tree:
  - search structure that partitions the query point according to their spatial distribution
  - If the tree is balanced, a search takes `log(N)` with `N` the number of points

KD-Trees

Constructing KD-trees can be done with a trivial recursion.

At each time, the set is split in two parts until the number of points in a leaf is smaller than a threshold.

Question:

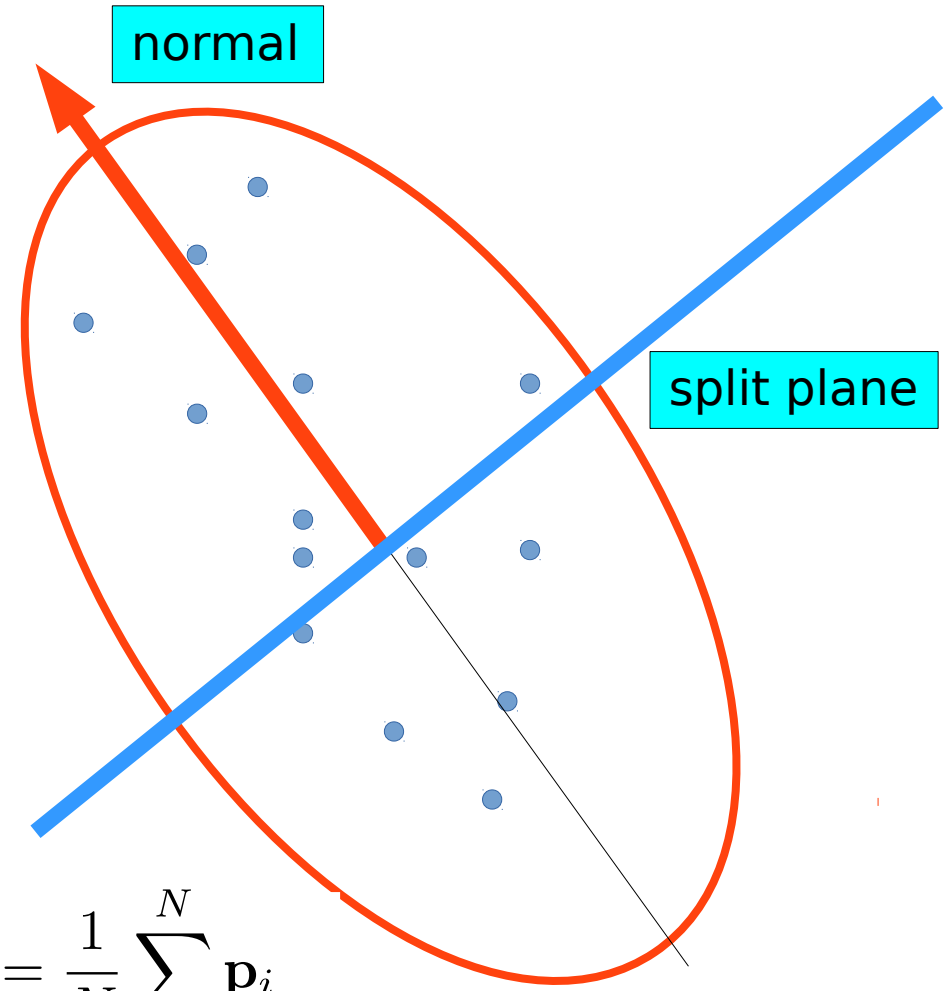
how to split?

KD-Trees: simple splitting

Consider the K-dimensional points as if they were normally distributed.

Compute the **covariance** matrix $\sum$ of the distribution.

Chose a splitting (hyper) plane that passes through the mean μ and has a normal aligned with the longest axis of the **covariance** matrix.`



$$\mu = \frac{1}{N}\sum_{i=1}^{N}\mathbf{p}_i$$

$$\mathbf{S} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{p}_i\mathbf{p}_i^T$$

$$\Sigma = \mathbf{S} - \mu\mu^T$$

KD-Trees

A split plane is characterized by

- a *normal* $\mathbf{n}_i$
- a *mean* μ_i

We can check if a point lies on the one side of the plane by evaluating

$$\mathbf{n}_i^T(\mathbf{p}_q - \mu_i) > 0$$

KD-Trees

Each intermediate node of the tree contains

- The normal of the splitting plane
- The mean
- Pointers to the children nodes

A query requires traversing the tree from the top to the bottom and at each time going left or right.

The query result is an approximation, *i.e.* the tree is a heuristic that might return not the real minimum, depending on how the tree was built.

Efficient randomized variants (see ANN C++ library)