Nearest Neighbors Algorithms in Euclidean and Metric Spaces: Algorithms and Data Structures

February 7, 2023

Frederic.Cazals@inria.fr

Nearest Neighbors Algorithms in Euclidean and Metric Spaces: Algorithms and Data Structures

Nearest Neighbors Algorithms in Euclidean and Metric Spaces: Algorithms and Data Structures

Applications

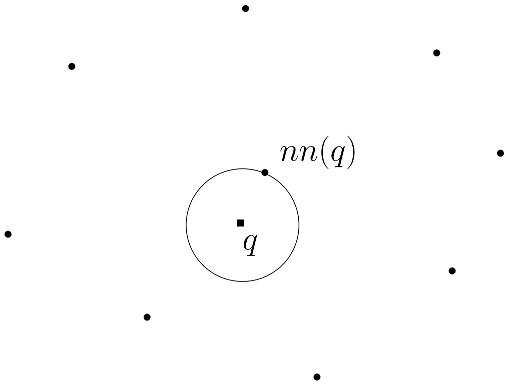
▷ A core problem in the following applications:

- clustering, k-means algorithms
- information retrieval in databases
- information theory : vector quantization encoding
- classification in learning theory
- ...

Nearest Neighbors: Getting Started

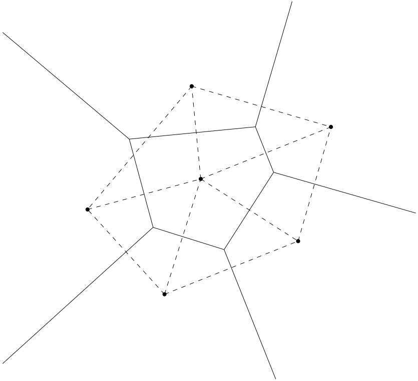
- ▷ Input: a set of points (aka sites) P in $\mathbb{R}^d$, a query point q
- ▷ Output: $nn(q, P)$, the point of P nearest to q

$$d(q, P) = d\left(q, nn(q, P)\right). \tag{1}$$



The Euclidean Voronoi Diagram and its Dual the Delaunay Triangulation

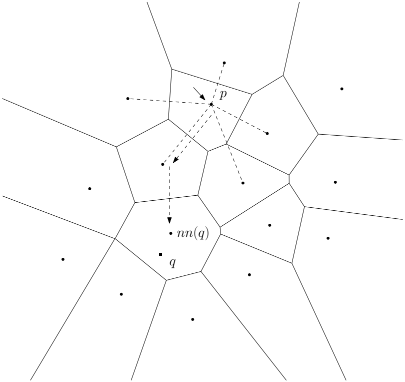
▷ Voronoi and Delaunay diagrams



▷ Key properties:

- Voronoi cells of all dimensions
- Voronoi - Delaunay via the nerve construction
- Duality : cells of dim. $d-k$ vs cells of dimension k
- The *empty ball property*

Nearest Neighbors Using Voronoi Diagrams



▷ Nearest neighbor by walking
- start from any point $p \in P$
- while $\exists$ a neighbor $n(p)$ of p in Vor(P)
  closer to q than p,
  step to it: $p = n(p)$
- done $nn(q) = p$

▷ Argument: the Delaunay neighborhood of a point is *complete*
Vor(p, P)= cell of p in Vor(P)
$N(p)$ = set of neighbors of p in Vor(P)
$N'(p) = \{p\} \bigcup N(p)$

$$\text{Vor}(p, N'(p)) = \text{Vor}(p, P)$$

▷ Exercise: specify the algorithm using DT

The Nearest Neighbors Problem: Overview

▷ Strategy: prepocess point set P of n points in $\mathbb{R}^d$ into a data structure (DS) for fast nearest neighbor queries answer.

▷ Ideal wish list:

- The DS should have linear size
- A query should have sub-linear complexity i.e. $o(n)$
  - When $d = 1$: balanced binary search trees yield $O(\log n)$

▷ Core difficulties:

- *Curse of dimensionality in* $\mathbb{R}^d$: for high d, it is difficult to outperform the linear scan
- Interpretation: meaningfull-ness of distances in high dimensional spaces – distance concentration phenomena.

The Nearest Neighbors Problem: Elementary Options

▷ The trivial solution :
$O(dn)$ space, $O(dn)$ query time

▷ Voronoi diagram

$d = 2$, $O(n)$ space $O(\log n)$ query time
$d > 2$, $O\left(n^{\lceil \frac{d}{2} \rceil}\right)$ space

$\rightarrow$ Under locally uniform condition on point distribution
the 1-skeleton Delaunay hierarchy achieves :
$O(n)$ space, $O(c^d \log n)$ expected query time.

▷ Spatial partitions based on trees

The Nearest Neighbors Problem: Variants

- Variants:
  - k-nearest neighbors: find the k points in P that are nearest to q
  - given $r > 0$, find the points in P at distance less than r from q
  - Various metrics
    - L_2, L_p, L_∞
    - String: Hamming distance
    - Images, graphs: distance based on optimal transportation
    - Point sets: distances via optimal alignment
  - Non metric spaces – cf metric trees
- Main contenders in metric spaces:
  - Tree like data structures:
    - quad-trees – and its variant ANN
    - (randomized) kd-trees
    - k-means trees – partition derived from k-means with k=2
  - Locally Sensitive Hashing

Comparison and appetizer: setup

▷ Contenders: various hierarchical methods for approximate NN

- randomized kd-trees: hierarchical partition with split direction chosen at random
- k-means trees: hierarchical partition with split direction derived from k-means
- ANN
- LSH

▷ Assessment for the accuracy of the approximation: precision i.e. fraction of queries for which the correct NN is found

▷ Two main questions addressed:

- Question 1: for a fixed database, which algorithm is best?
- Question 2: are the performances stable when the size of the DB changes?

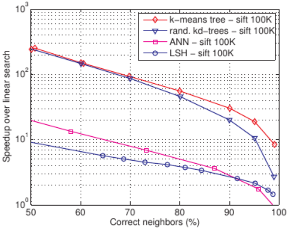
▷Ref: Muja and Lowe, VISAPP 2009

▷Ref: O'Hara and Draper, Applications of Computer Vision (WACV), 2013

Main Contenders: Typical Results for Approximate NN

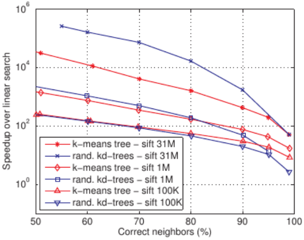
▷ DB used : Scale-Invariant Feature Transform (SIFT) for images: $\{(x_i, y_i, \sigma_i)\}$

▷ Question 1: best algorithm



(a)

▷ Question 2 – for winners only
i.e. for rand. kd-trees and k-means trees



(b)

▷ Take-home messages:

- Randomized kd-trees and k-means trees win
  - splits must exploit the variance in the dataset
- Speed-ups consistent when DB size increases

▷Ref: Muja and Lowe, VISAPP 2009
▷Ref: O'Hara and Draper, Applications of Computer Vision (WACV), 2013

Nearest Neighbors Algorithms in Euclidean and Metric Spaces: Algorithms and Data Structures

Performances of geolocalization

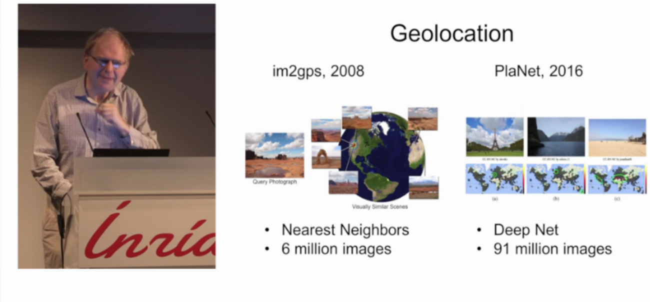
a tale of data, features, and algorithms

▷ Source: Inria Colloquim talk by Alexei Efros, UC Berkeley, see `https://iww.inria.fr/colloquium/fr/alexei-alyosha-efros-self-supervised-visual-learning-and-synthesis/`

▷ Problem: geolocalize an image

- **Solution one:**
  - DB of 6M images; (SIFT) features
  - Answer: derived from the NN of the query image
- **Solution two:** DeepNet trained on DB of 91 M images
- **Nb:** correctness assessed at a given scale (in kilometers)

Localization from images: two (antipodal) strategies



Performances: a matter of DB size

▷ img2gps:

- Original im2gps: localization from NN in the database using simple SIFT, DB of 6.5 Flickr images
- Revamped im2gps: more engineering on features but same DB size

▷ img2gps versus Planet:

Algorithm vs. Data

Method	Street 1 km	City 25 km	Region 200 km	Country 750 km	Continent 2500 km
Im2GPS (orig) [19]		12.0%	15.0%	23.0%	47.0%
Im2GPS (new) [20]	2.5%	21.9%	32.1%	35.4%	51.9%
PlaNet (900k)	0.4%	3.8%	7.6%	21.6%	43.5%
PlaNet (6.2M)	6.3%	18.1%	30.0%	45.6%	65.8%
PlaNet (91M)	**8.4%**	**24.5%**	**37.6%**	**53.6%**	**71.3%**

- Im2GPS: wins on city and region levels
- PlaNet 6.2M: wins on on street, country and continent levels.

▷Ref: `Weyand et al, ECCV 2016`

The lesson: data, features, algorithms

https://iww.inria.fr/colloquium/fr/alexei-alyosha-efros-self-supervised-visual-learning-and-synthesis/

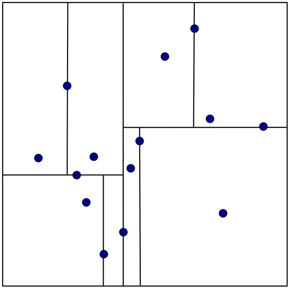
Take home messages

- Do not underestimate the data
- DeepLand is not the only sweet spot...

Nearest Neighbors Algorithms in Euclidean and Metric Spaces: Algorithms and Data Structures

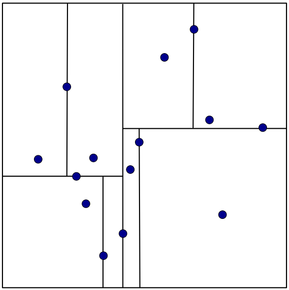
kd-tree for a collection of points (sites) P

- Definition:
  - A binary tree
  - Any internal node implements a spatial partition induced by a hyperplane H, splitting the point cloud into two equal subsets
    - right subtree: points p on one side of H
    - left subtree: remaining points
  - The process halts when a node contains $\leq n_0$ points



Nb: the point realizing the median is stored in the node performing the split

kd-tree for a collection of points P



▷ Algorithm build_kdTree(S)

$n \leftarrow newNode$
if $\mid S \mid \leq n_0$ **then**
  Store the point of S into a container of n
  return n
else
  $dir = depth \mod d$
  Project the points of S along direction dir
  Compute the median m
  {Split into two equal subsets}
  $n.sample \leftarrow$ sample v realizing the median
  $L \leftarrow$ point from $S \backslash \{v\}$ whose dirth coord is $< m$
  $R \leftarrow$ point from $S \backslash \{v\}$ whose dirth coord is $\geq m$
  $n.left \leftarrow$ build_kdTree(L)
  $n.right \leftarrow$ build_kdTree(R)
  return n

kd-tree: search

- Main considerations:
  - Exact versus approximate NN
  - No free lunch: complexity matters
- Three main search strategies:
  - (Approx.) the defeatist search: simple, but may fail
    (Nb: see later, distance concentration phenomema)
  - (Exact) the descending search: always succeeds, but may take time
  - (Exact) the priority search: strikes a compromise between the defeatist and descending strategies

kd-tree search: the defeatist search

- **Key idea:** recursively visit the subtree containing the query point
- **Algorithm defeatist_search_kdTree: the defeatist search in a kd tree.**

Require: Maintains $nn(q)$ of q, and $\tau = d(q, nn(q))$

  $n \leftarrow root$; $\tau \leftarrow d(q, n.sample)$

  while $n \neq NIL$ **do**

    Possibly update $nn(q)$ using $n.sample$, and τ

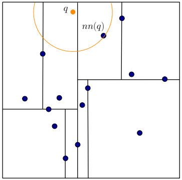
    if $q \in$ Domain of L **then**

      defeatist_search_kdTree($n.left$)

    if $q \in$ Domain of R **then**

      defeatist_search_kdTree($n.right$)

- **Complexity:** assuming leaves of size n_0 – depth satisfies $2^h n_0 = n$
  - search cost: $O(n_0 + \log(n/n_0))$
- **Caveat:** failure

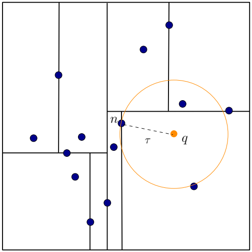


kd-tree search: the exhaustive descending search

▷ Key idea: visit one or two subtree, depending on the distance $d(q, nn(q))$ computed

▷ Algorithm descending_search_kdTree: the descending search in a kd tree.

Require: Maintains $nn(q)$ of q, and $\tau = d(q, nn(q))$
Require: Uses the domain of a node n (an intersection of half-spaces)
  $n \leftarrow root$
  $\tau \leftarrow d(q, n.sample)$
  while $n \neq NIL$ **do**
    Possibly update $nn(q)$ using $n.sample$
    if $Sphere(q, \tau) \cap$ Domain of L **then**
      descending_search_kdTree($n.left$)
    if $Sphere(q, \tau) \cap$ Domain of R **then**
      descending_search_kdTree($n.right$)



The value of τ ensures that the top cell will be visited.

kd-tree search: the priority search (idea)

- Priority search, key ideas:
  - Uses a priority queue to store nodes (regions), with a priority inversely proportional to the distance to q.
  - Upon popping a node, the corresponding subtree is descended to visit the node closest to q. Upon descending, $nn(q)$ is updated.
  - While descending, the child not visited is possibly enqueued,

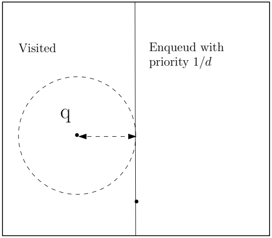
kd-tree search: priority search (algorithm)

▷ Uses a priority queue Q to enumerate nodes by increasing distance to query q

Ensure: Maintains $nn(q)$ of q, and $\tau = d(q, nn(q))$

$nn(q) \leftarrow root.sample$
Q.insert(root)
while True **do**
  if Q.empty() **then**
    return
  { Node with highest priority}
  $r \leftarrow Q.pop()$
  {The nearest box is too far wrt $nn(q)$}
  if $d(bbox(r), q) > \tau$ **then**
    return
  {Descend into box **nearest to** q,} {and possibly enqueue the second node}
  for Nodes n on the path from r to the box nearest to q **do**
    {Possibly update $nn(q)$ and τ}
    $d \leftarrow d(q, n.sample)$
    if $d < \tau$ **then**
      $nn(q) \leftarrow n.sample$; $\tau \leftarrow d$
    {Possibly enqueue the second subtree}
    $f \leftarrow$ brother of n
    if $d(bbox(f), q) \leq \tau$ **then**
      {Insert with priority inverse to distance to q}
      $Q.insert(f, 1/d)$

Box of current node r



NB: Enquing criterion can be adapted to report an $(1 + \varepsilon)$ approx. of the exact NN

kd-tree search: priority search (analysis)

- Pros and cons:
  - + nn always found
  - + linear storage
  - – nn often found at an early stage ... then time spent in useless recursion
  - – In the worst-case, all nodes are visited.
  - – Maintaining the priority queue Q has a cost
- Variants and improvements:
  - Initially the Q with all nodes from root to leaf containing the query
  - Stopping the recursion once a fraction of nodes has been visited
  - Backing up defeatist search with overlapping cells
  - Combining multiple randomized kd-trees

References

Sam06 H. Samet. Foundations of multidimensional and metric data structures. Morgan Kaufmann, 2006.

SDE05 G. Shakhnarovich, T. Darrell, and P. Indyk (Eds). Nearest-Neighbors Methods in Learning and Vision. Theory and Practice. MIT press, 2005.

Nearest Neighbors Algorithms in Euclidean and Metric Spaces: Algorithms and Data Structures

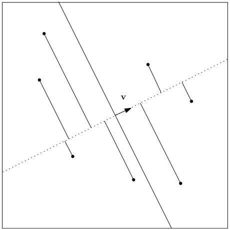
Improvements aiming at fixing the defeatist search

- Defeatist search: (early) choice of one side is risky
- Simple improvements:
  - Use several trees, and pick the best neighbor(s)
  - Allow overlap between cells in a node: selected points stored twice $\rightarrow$ spill trees
  - Use randomization to obtain different partitions rescuing the defeatist search
    - different permutations of coordinate axis
    - directions aiming at maximizing the variance
  - Next: randomization captures information on directions carrying variance

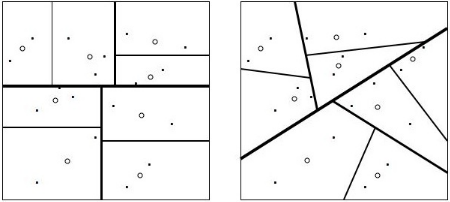
Random projection trees (RPTrees)

Aka Random partition trees (RPTrees!)

- kd-tree: axis parallel splits
- Splitting along a random direction $U \in S^{d-1}$: project onto U and split at the (perturbed) median



- Resulting spatial partition



Random projection trees: generic algorithm with jitter

- Below: version where one also jitters the median defining the split
- Algorithm build_RPTree(S)

Ensure: Build the RPTree of a point set S

  if If $|S| \leq n_0$ **then**
    $n \leftarrow newNode$
    Store S into n
    return n
  Pick U uniformly at random from the unit sphere
  Pick β uniformly at random from $[1/4, 3/4]$
  Let v be the β-fractile point on the projection of S onto U
  $Rule(x) = ($left if $\langle x, U \rangle < v$, otherwise right$)$
  $left\_tree \leftarrow$ build_RPTree($\{x \in S : Rule(x) = left\}$)
  $right\_tree \leftarrow$ build_RPTree($\{x \in S : Rule(x) = right\}$)
  return $(Rule(\cdot), left\_tree, right\_tree)$

- Remark: RP trees have the following property – more later: diameter of the cells decrease down the tree at a rate depending on the *intrinsic dimension* of the data.

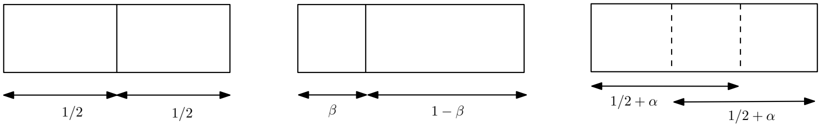
RPTrees: varying splits and their applications

▷ Various types of splits possible

Randomized partition tree:
- exact split

Randomized partition tree:
- perturbed split

Spill tree with overlapping split:
- regular spill tree
- virtual spill tree



▷ NB: splits monitor the tree structure and the search route

▷ Spill trees:

– Regular spill trees:
  overlapping cells yield redundant storage of points

– Virtual spill trees:
  median splits used – no redundant storage
  query routed in multiple leaves using overlapping splits

▷ Summary: tree creation versus search

	Routing data	Routing queries (defeatist style)
RP tree	Perturbed split	Perturbed split
Regular spill tree	Overlapping split	Median split
Virtual spill tree	Median split	Overlapping split

Regular spill trees: size

- Tree depth: assume that
  - the number of nodes transmitted to a son decreases by a factor at least $\beta = 1/2 + \alpha$,
  - a leaf accommodates up to n_0 points

Then: the tree depth l satisfies $\beta^l n \leq n_0$ i.e. $l = O(\log_{1/\beta} \frac{n}{n_0})$.

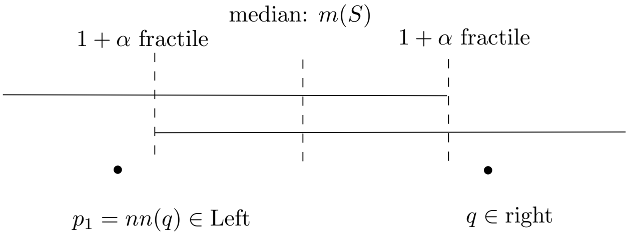
- Tree size:

$$n_0 2^l = n_0 2^{\log_{1/\beta} \frac{n}{n_0}}.$$

- Examples:
  - $\alpha = 0.05$: $O(n^{1.159})$.
  - $\alpha = 0.1$: $O(n^{1.357})$.

Spill trees: compromising Storage vs NN searches

- Spill trees: overlapping splits yield superlinear storage
- Yet, search mail fail too:



  - $p_1 = nn(q)$: routed in left subtree only
  - query point q: routed in right subtree only

Failure of the defeatist search

▷ Goal: probability that a defeatist seach does not return the exact nearest neighbor(s)?

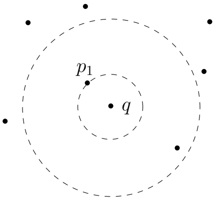
▷ The event to be analyzed, denoted **Err**:

- $k = 1$:the NN query does not return $p_{(1)}$
- $k > 1$: the NN query does not return $p_{(1)}, \ldots, p_{(k)}$

Qualifying the hardness of nearest neighbor queries

▷ Notations:

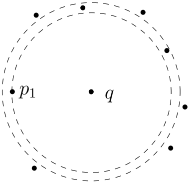
- Dataset $P = p_1, \ldots, p_n$
- Sorted dataset wrt q: $p_{(1)}, \ldots, p_{(n)}$



$$\Phi(q, P) = \frac{1}{n} \sum_{i=2}^{n} \frac{\left\| q - p_{(1)} \right\|_2}{\left\| q - p_{(i)} \right\|_2}. \qquad (2)$$

▷ Extreme cases:

- $\Phi \sim 0$: p_1 *isolated*, finding it should be easy
- $\Phi \sim 1$: points equidistant from q; finding $p_{(1)}$ should be hard



▷ Rationale: in using RPT and spill trees with the defeatist search, the probability of success should depend upon Φ.

Generalizations of the function Φ

▷ Rationale: function Φ shall be used for nodes containing a subset of the database

▷ For a cell containing m points – evaluate the remaining points in that cell:

$$\Phi_m(q, P) = \frac{1}{m} \sum_{i=2}^{m} \frac{\left\| q - p_{(1)} \right\|_2}{\left\| q - p_{(i)} \right\|_2}. \tag{3}$$

▷ If one is interested in the k nearest neighbors – evaluate the remaining points too:

$$\Phi_{k,m}(q, P) = \frac{1}{m} \sum_{i=k+1}^{m} \frac{\left\| q - p_{(1)} \right\|_2 + \cdots + \left\| q - p_{(k)} \right\|_2}{\left\| q - p_{(i)} \right\|_2}. \tag{4}$$

Theoretical results on the performances

- Agenda for the next lecture:
  - RPTrees: success/failure probability to report NN
  - Random projections and adaptation to intrinsic dimension
  - NN, distances and concentration phenomena

Nearest Neighbors Algorithms in Euclidean and Metric Spaces: Algorithms and Data Structures

A geometric distance: the Hausdorff distance

▷ Hausdorff distance. Consider a metric space (M, d). The *Hausdorff distance* of two non-empty subsets X and Y is defined by

$$d_H(X, Y) = \max(H(X, Y), H(Y, X)), \text{ with } H(X, Y) = \sup_{x \in X} \inf_{y \in Y} d(x, y). \quad (5)$$

Note that the one-sided distance is not symmetric, as seen on Fig. 1.

▷ Rmk. For closed set, the min distance is realized: inf becomes min; sup becomes max.

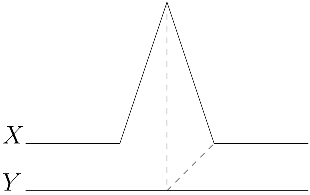


Figure: **The one-sided Hausdorff distance is not symmetric**

Nearest Neighbors Algorithms in Euclidean and Metric Spaces: Algorithms and Data Structures

Comparing Histograms

▷ Bin-to-bin methods:

$$d(H,K) = \sum_i \mid h_k - k_k \mid \tag{6}$$

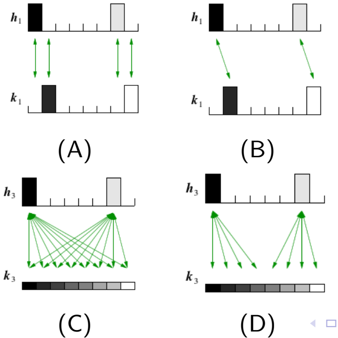
→ overestimates the distance since neighboring bins are not considered.

▷ Mixing (e.g. quadratic) methods:

$$d^2(H,K) = (\mathbf{h} - \mathbf{k})^t \mathbf{A} (\mathbf{h} - \mathbf{k}) \tag{7}$$

→ underestimates distances : tends to accentuate the similarity of color distributions without a pronounced mode.

▷ Illustrations:



Transport Plan Between Two Weighted Point Sets

▷ Weighted point sets:

$$P = \{(p_1, w_{p_1}) \ldots, (p_m, w_{p_m})\} \text{ and } Q = \{(q_1, w_{q_1}, \ldots, (q_n, w_{q_n})\}. \quad (8)$$

NB: nodes from P (resp. Q): production (resp. demand) nodes
Shorthand for the sum of masses: $W_P = \sum_i w_{p_i}$, $W_Q = \sum_j w_{q_j}$.
▷ A metric $d(\cdot, \cdot)$: distance between two points $d_{ij} = d(p_i, q_j)$.
▷ A transport plan: is a set of non-negative *flows* f_{ij} circulating on the edges of the bipartite graph $P \times Q$.

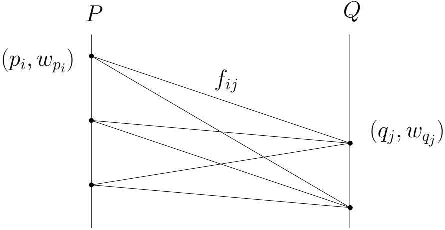


Figure: **Transport plan between two weighted point sets**

The Earth Mover Distance: Definition

▷ Optimization problem:

$$\text{Minimize}: C_{\text{EMD-LP}} = \sum_{ij} f_{ij} d_{ij} \quad \text{under the constraints:} \tag{9}$$

$$\begin{cases} (C1) f_{ij} \geq 0 \\ (C2) \sum_j f_{ij} \leq w_{p_i}, \forall i \\ (C3) \sum_i f_{ij} \leq w_{q_j}, \forall j \\ (C4) \sum_i \sum_j f_{ij} = \min(W_P, W_Q). \end{cases} \tag{10}$$

These constraints read as follows:

- (C1) Flows are positive
- (C2,C3) A node cannot export (resp. receive) more than its weight.
- (C4) The total flow neither exceeds the production nor the demand.

▷ Earth mover distance: defined from the cost by

$$d_{\text{EMD-LP}} = \frac{C_{\text{EMD-LP}}}{\sum_{ij} f_{ij}} = \frac{C_{\text{EMD-LP}}}{\min(W_P, W_Q)} \tag{11}$$

▷ Advantages:

- Applies to signatures in general, the histograms being a particular case.
- Embeds the notion of nearness, via the metric in the ground space.
- Allows for partial matches. See however, the comment in section **??**.
- Easy to compute: linear program.

▷Ref: Rubner, Tomasi, Guibas, IJCV, 2000

The Earth Mover Distance: Main Properties

▷ Theorems:

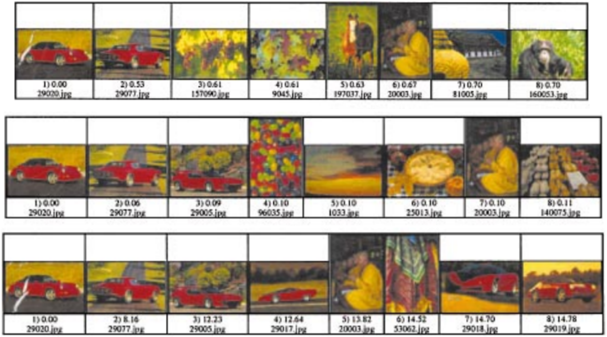
- Computed in polynomial time in the # of variables
- Number of edges carrying flow is $\leq n + m - 1$
- If $W_P = W_Q$ and $d(\cdot, \cdot)$ is a metric: EMD is also a metric

▷ Rmk: entropy regularized EMD distances, aka Sinkhorn distances, yield iterative algorithms – fater than LP solving. See refs. by M. Cuturi et al.

Application to image retrieval

- ▷ Image coding, two options:
  - ▶ convert image to histogram using a fixed binning of the color space; mass of bin: num. of pixel within it.
  - ▶ cluster pixels (say with k-means): mass of cluster is the fraction of pixels assigned to it

- ▷ Search on DB of 20,000 images: (a) L_1 (d) Quadratic form (e) EMD



Mallow's Distance – p-th Wasswerstein metric

▷ Consider: two RV in $\mathbb{R}^d$: $X \sim P$, $Y \sim Q$.

▷ Mallows distance between X and Y: minimum of expected difference between X and Y over all joint distributions F for (X, Y), such that the marginal of $F(X, \cdot)$ is P and that of $F(\cdot, Y)$ is Q (aka coupling):

$$M_p(X, Y)^p = \min_{F \in \mathcal{F}} \mathbb{E}_F[\|X - Y\|^p] : (X, Y) \sim F, \; X \sim P, Y \sim Q\}. \tag{12}$$

▷ Discrete setting: P and Q

$$P = \{(x_1, w_{p_1}) \ldots, (x_m, w_{p_m})\} \tag{13}$$
$$Q = \{(y_1, w_{q_1}, \ldots, (y_n, w_{q_n})\}. \tag{14}$$

▷ Joint distribution is specified by probabilities on all pairs i.e. $F = \{f_{ij}\}$, and the fact that it respects the marginals yields:

$$\sum_j f_{ij} = p_i, \; \sum_i f_{ij} = q_j, \; \sum_{ij} f_{ij} = 1. \tag{15}$$

▷ Functional to be minimized becomes:

$$M_p(X, Y)^p = \mathbb{E}_F[\|X - Y\|^p] = \sum_{ij} f_{ij} \|x_i - y_j\|^p . \tag{16}$$

Mallow's Distance versus EMD – Example with $p = 1$

▷ Mallows' distance ($W_P = W_Q = 1$):
$M_p = 1/4 \times 0 + 1/4 \times 1 + 1/4 \times 1 + 1/4 \times 0$

▷ EMD, assuming uniform weights on all points, ie $W_P = 2$ and $W_Q = 4$:
EMD $= 0$ since a flow of 2 units satisfies all constraints.

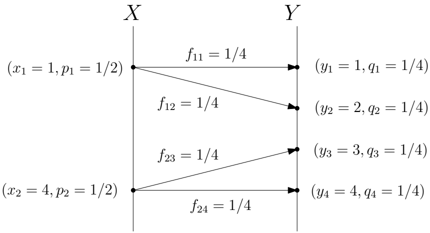


Figure: **Mallows's distance**

References

LB01 Elizaveta Levina and Peter Bickel. The earth mover's distance is the mallows distance: Some insights from statistics. In Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on, volume 2, pages 251–256. IEEE, 2001.

RTG00 Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. International Journal of Computer Vision, 40(2):99–121, 2000.

ZLZ05 Ding Zhou, Jia Li, and Hongyuan Zha. A new mallows distance based metric for comparing clusterings. In Proceedings of the 22nd international conference on Machine learning, pages 1028–1035. ACM, 2005.

CM14 F. Cazals and D. Mazauric. Mass transportation problems with connectivity constraints, with applications to energy landscape comparison. Submitted, 2014. Preprint: Inria tech report 8611.

MC13 M. Cuturi. Sinkhorn Distances: Lightspeed Computation of Optimal Transport. NIPS 2013.

Nearest Neighbors Algorithms in Euclidean and Metric Spaces: Algorithms and Data Structures

Metric spaces

Definition 1. A *metric space* is a pair (M, d), with $d : M \times M \to \mathbb{R}^+$, such that:

- (1) Positivity: $d(x, y) \geq 0$
- (1a) Self-distance: $d(x, x) = 0$
- (1b) Isolation: $x \neq y \Rightarrow d(x, y) > 0$
- (2) Symmetry: $d(x, y) = d(y, x)$
- (3) Triangle inequality: $d(x, y) \leq d(x, z) + d(y, z)$

▷ Product metric. Assume that for some $k > 1$:

$$M = M_1 \times \cdots \times M_k. \tag{17}$$

and that each (M_i, d_i) is a metric space. For $p \geq 1$, the *product metric* is:

$$d(x, y) = (\sum_{k=1}^{k} d_i(x_i, y_i)^p)^{1/p} \tag{18}$$

Some particular cases are:

- $(M_i = \mathbb{R}, d_i = |\cdot|)$: L_p metrics.
- $p = 1, d_i =$ uniform metric: Hamming distance.

Using the triangle inequality

Lemma 2. For any three points $p, q, s \in M$, for any $r > 0$, and for any point set $P \subset M$, one has:

$$\mid d(q,p) - d(p,s) \mid \leq d(q,s) \leq d(q,p) + d(p,s) \tag{19}$$

$$d(q,s) \geq d_P(q,s) := \max_{p \in P} \mid d(q,p) - d(p,s) \mid \tag{20}$$

$$\begin{cases} d(p,s) > d(p,q) + r \Rightarrow d(q,s) > r \\ d(p,s) < d(p,q) - r \Rightarrow d(q,s) > r. \end{cases} \tag{21}$$

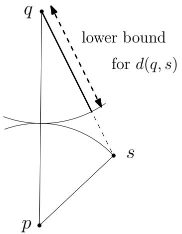
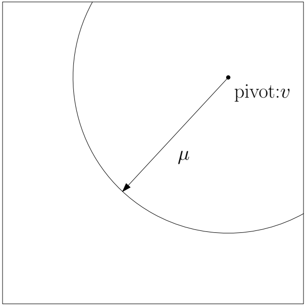


Figure: **Lower bound from the triangle inequality, see Lemma 2**

Metric tree: definition

▷ Definition:

- A binary tree
- Any internal node implements a spherical cut defined by the distance μ to a pivot v
  - right subtree: points p such that $d(pivot, p) \geq \mu$
  - left subtree: points p such that $d(pivot, p) < \mu$



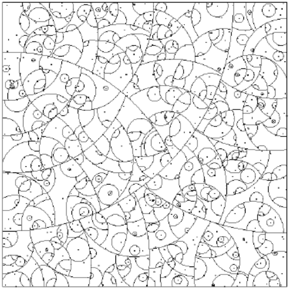


Figure: **Metric tree for a square domain (A)** One step **(B)** Full tree

Metric tree: construction

▷ Recursively construction:

- Choose a pivot, ideally inducing a partition into subsets of the same size
- Assign points to subtrees and recurse
- Complexity under the balanced subtrees assumption: $O(n \log n)$.

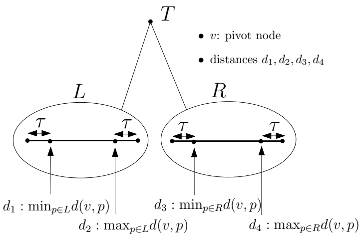
Algorithm 1 Algorithm build_MetricTree(S)

{build_MetricTree(S)}
if $S = \emptyset$ **then**
  return NIL
$n \leftarrow newNode$
Draw at random $Q \subset S$ and $v \in Q$
$n.pivot \leftarrow v$
$\mu \leftarrow \text{median}(\{d(v,p), p \in Q \backslash \{v\}\})$
{The pivot splits points into two subsets}
$L \leftarrow \{s \in S \backslash \{p\} | d(s,v) < \mu\}$
$R \leftarrow \{s \in S \backslash \{p\} | d(s,v) \geq \mu\}$
{For each subtree: min/max distances to points in that subtree}
$n.(d_1, d_2) \leftarrow$ (min,max) of distances $d(v,p), p \in L$
$n.(d_3, d_4) \leftarrow$ (min,max) of distances $d(v,p), p \in R$
{Recursion}
$n.L \leftarrow$ build_MetricTree(L)
$n.R \leftarrow$ build_MetricTree(R)

Searching a metric tree: algorithm

Algorithm 2 Algorithm search_MetricTree(T, q)

{Note of T is denoted n}
$nn(q) \leftarrow \emptyset$
$\tau \leftarrow \infty$
if $n = NIL$ **then**
  return
{Check whether the pivot is the nn}
$l \leftarrow d(q, n.pivot)$
if $l < \tau$ **then**
  $nn(q) \leftarrow n.pivot$
  $\tau \leftarrow l$
{Dilate the distance intervals for left and right subtrees}
$I_l \leftarrow [n.d_1 - \tau, n.d_2 + \tau]$
$I_r \leftarrow [n.d_3 - \tau, n.d_4 + \tau]$
if $l \in I_l$ **then**
  search_MetricTree($n.L, q$)
if $l \in I_r$ **then**
  search_MetricTree($n.R, q$)



Searching a metric tree: correctness – pruning lemma

Lemma 3. Consider the intervals associated with a node, as defined in Algorithm 1, that is $I_l \leftarrow [n.d_1 - \tau, n.d_2 + \tau]$ $I_r \leftarrow [n.d_3 - \tau, n.d_4 + \tau]$. Then:
(1) If $I \notin I_l$, the left subtree can be pruned.
(2) If $I \notin I_r$, the left subtree can be pruned.

Proof.
We prove (1), as condition (2) is equivalent. Let us denote $I_L = [d_1, d_2]$. Since $I = d(v, q) \notin I_l$, we have $d(v, q) < d_1 - \tau$ and $d(v, q) > d_2 + \tau$. We analyze these two conditions in turn.

▷ Condition on the right hand side. By definition of d_2, with v the pivot, we have:

$$\forall p \in L : d(v, q) > d(v, p) + \tau.$$

Using the triangle inequality for $d(v, q)$ yields

$$d(v, p) + d(p, q) \geq d(v, q) > d(v, p) + \tau \Rightarrow d(q, p) > \tau.$$

▷ Mutatis mutandis. □

Metric tree: choosing the pivot

▷ By the pruning lemma: for small τ and if q is picked uniformly at random, the measure of the boundary of the spheres of radius $d_1, \ldots, d_4$ determines the probability that no pruning takes place.
$\Rightarrow$ pick the pivot so as to minimize this measure.

▷ Example in 2D: 3 choices for the pivot, so as to split the unit square (mass: 1) into two regions of equal size (mass: 1/2)

▷ Choice of pivots (illustrated using μ (rather than the d_is):

- Best pivot: p_c
- Worst pivot: p_m

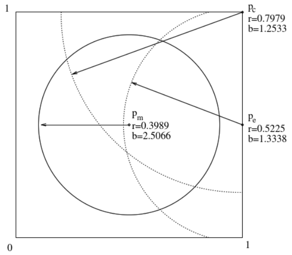


Figure: **Metric trees: minimizing the measure of boundaries.**

From metric trees to metric forests

- Search options:
  - (I) The exact search, based on the pruning lemma.
  - (II)The defeatist style search: visit one subtree only
- Compromising speed versus accurary
  - (I) Exact, but possibly costly if little/no pruning occurs. Worst-case: linear time.
  - (II) Faster, but error prone.
  - Compromise: using a forest of trees *rescues* erroneous branching decisions in the course of the defeatist search.

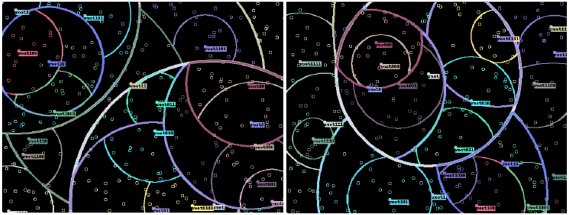


Figure: **Metric forest**

References

AMN+98 S. Arya et al. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. Journal of the ACM (JACM), 45(6):891–923, 1998.

ML09 Marius Muja and David G Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In VISAPP (1), pages 331–340, 2009.

MSMO03 Francisco Moreno-Seco, Luisa Mico, and Jose Oncina. A modification of the laesa algorithm for approximated k-nn classification. Pattern Recognition Letters, 24(1):47–53, 2003.

OD13 S. O'Hara and B.A. Draper. Are you using the right approximate nearest neighbor algorithm? In Applications of Computer Vision (WACV), 2013 IEEE Workshop on, pages 9–14. IEEE, 2013.

Yia93 Peter N Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In SODA, volume 93, pages 311-321, 1993.