# Geometric Problems in High Dimensions: Sketching

Piotr Indyk

# High Dimensions

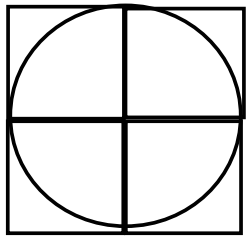
- We have seen several algorithms for low-dimensional problems (d=2, to be specific):
  - data structure for orthogonal range queries (kd-tree)
  - data structure for approximate nearest neighbor (kd-tree)
  - algorithms for reporting line intersections
- Many more interesting algorithms exist (see Computational Geometry course next year)
- Time to move on to high dimensions
  - Many (not all) low-dimensional problems make sense in high d:
    * nearest neighbor: YES (multimedia databases, data mining, vector quantization, etc..)
    * line intersection: probably NO
  - Techniques are very different

# What's the Big Deal About High Dimensions ?

- Let's see how kd-tree performs in $R^d$…

# Déjà vu I: Approximate Nearest Neighbor

- Packing argument:
  - All cells C seen so far have diameter > eps*r
  - The number of cells with diameter eps*r, bounded aspect ratio, and touching a ball of radius r is at most $O(1/eps^2)$
- In $R^d$, this gives $O(1/eps^d)$. E.g., take eps=1, r=1. There are $2^d$ unit cubes touching the origin, and thus intersecting the unit ball:



# Déjà vu II: Orthogonal Range Search

- What is the max number Q(n) of regions in an n-point kd-tree intersecting a vertical line ?
  - If we split on x, $Q(n)=1+Q(n/2)$
  - If we split on y, $Q(n)=2*Q(n/2)+2$
  - Since we alternate, we can write $Q(n)=3+2Q(n/4)$, which solves $O(\sqrt{n})$
- In $R^d$ we need to take Q(n) to be the number of regions intersecting a (d-1)-dimensional hyperplane orthogonal to one of the directions
- We get $Q(n)=2^{d-1} Q(n/2^d)+stuff$
- For constant d, this solves to $O(n^{(d-1)/d})=O(n^{1-1/d})$

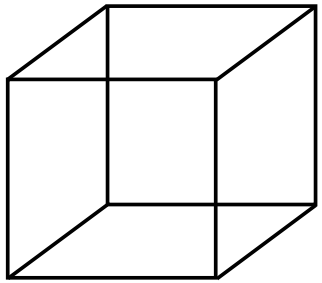
# High Dimensions

- Problem: when d > log n, query time is essentially O(dn)
- Need to use different techniques:
  - Dimensionality reduction, a.k.a. sketching:
    * Since d is high, let's reduce it while preserving the important data set properties
  - Algorithms with "moderate" dependence on d

    (e.g., $2^d$ but not $n^d$)

# Hamming Metric

- Points: from $\{0,1\}^d$ (or $\{0,1,2,\ldots,q\}^d$ )
- Metric: D(p,q) equals to the number of positions on which p,q differ
- Simplest high-dimensional setting
- Still useful in practice
- In theory, as hard (or easy) as Euclidean space
- Trivial in low d



Example (d=3):

{000, 001, 010, 011, 100, 101, 110, 111}

# Dimensionality Reduction in Hamming Metric

Theorem: For any r and eps>0 (small enough), there is a distribution of mappings G: $\{0,1\}^d \rightarrow \{0,1\}^t$, such that for any two points p, q the probability that:

- If D(p,q)< r then D(G(p), G(q)) <(c +eps/20)t
- If D(p,q)>(1+eps)r then D(G(p), G(q)) >(c+eps/10)t

is at least 1-P, as long as $t=O(\log(1/P)/eps^2)$.

- Given n points, we can reduce the dimension to O(log n), and still approximately preserve the distances between them
- The mapping works (with high probability) even if you don't know the points in advance

# Proof

- Mapping: $G(p) = (g_1(p), g_2(p),\ldots,g_t(p))$, where

$$g(p)=f(p_{|I})$$

  - I: a multiset of s indices taken independently uniformly at random from {1…d}
  - $p_{|I}$: projection of p
  - f: a random function into {0,1}

- Example: p=01101, s=3, I={2,2,4} $\rightarrow p_{|I} = 110$

# Analysis

- What is $Pr[p_{|I} = q_{|I}]$ ?
- It is equal to $(1-D(p,q)/d)^s$
- We set $s=d/r$. Then $Pr[p_{|I} = q_{|I}] = e^{-D(p,q)/r}$, which looks more or less like this:
- Thus
  - If $D(p,q) < r$ then $Pr[p_{|I} = q_{|I}] > 1/e$
  - If $D(p,q) > (1+eps)r$ then $Pr[p_{|I} = q_{|I}] < 1/e - eps/3$

# Analysis II

- What is Pr[g(p) <> g(q)] ?
- It is equal to $Pr[p_{|I} = q_{|I}]*0 + (1- Pr[p_{|I} = q_{|I}]) *1/2 = (1- Pr[p_{|I} = q_{|I}])/2$
- Thus
  - If D(p,q)< r then Pr[g(p) <> g(q)] < (1-1/e)/2 = c
  - If D(p,q)>(1+eps)r then Pr[g(p) <> g(q)] > c+eps/6
- By linearity of expectations

$$E[D(G(p),G(q))] = Pr[g(p) <> g(q)]\ t$$

- To get the high probability bound, use Chernoff inequality

# Algorithmic Implications

- Approximate Near Neighbor:
  - Given: A set of n points in $\{0,1\}^d$, eps>0, r>0
  - Goal: A data structure that for any query q:
    * if there is a point p within distance r from q, then report p' within distance (1+eps)r from q
- Can solve Approximate Near*est* Neighbor by taking r=1,(1+eps),…

# Algorithm I - Practical

- Set probability of error to 1/poly(n) → $t=O(\log n/\mathrm{eps}^2)$
- Map all points p to G(p)
- To answer a query q:
  - Compute G(q)
  - Find the nearest neighbor of G(q) among all points G(p)
  - Check the distance; if less than r(1+eps), report
- Query time: $O(n \log n/\mathrm{eps}^2)$

# Algorithm II - Theoretical

- The exact nearest neighbor problem in $\{0,1\}^t$ can be solved with
  - $2^t$ space
  - O(t) query time

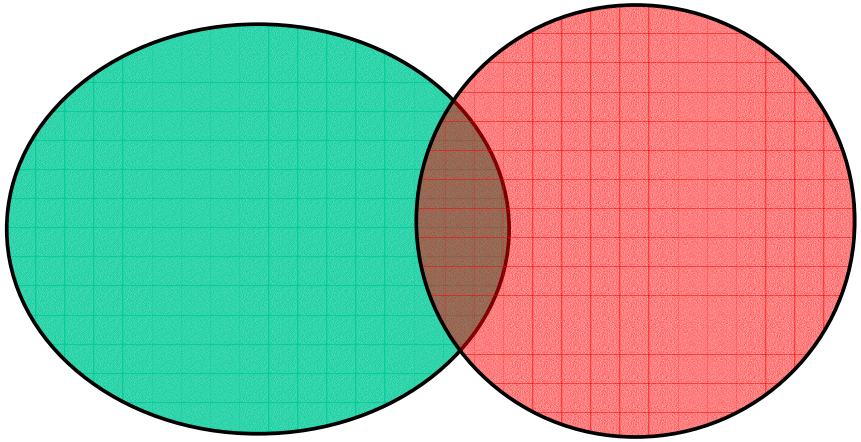
  (just store pre-computed answers to all queries)
- By applying mapping G(.), we solve approximate near neighbor with:
  - $n^{O(1/eps^2)}$ space
  - $O(d \log n/eps^2)$ time

# Another Sketching Method

- In many applications, the points tend to be quite sparse
  - Large dimension
  - Very few 1's
- Easier to think about them as sets. E.g., consider a set of words in a document.
- The previous method would require very large s
- For two sets A,B, define Sim(A,B)=$|A \cap B|/|A \cup B|$
  - If A=B, Sim(A,B)=1
  - If A,B disjoint, Sim(A,B)=0
- How to compute short sketches of sets that preserve Sim(.) ?

# “Min Approach”

- Mapping: $G(A)=\min_{a \text{ in } A} g(a)$, where g is a random permutation of the elements
- Fact:

$$\Pr[G(A)=G(B)]=Sim(A,B)$$

- Proof: Where is min( g(A) U g(B) ) ?