# Ανάπτυξη Λογισμικού για Δυσεπίλυτα Αλγοριθμικά Προβλήματα

## Ενότητα 1: Nearest neighbors

Γιάννης Εμίρης

Τμήμα Πληροφορικής & Τηλεπικοινωνιών
Πανεπιστήμιο Αθηνών

Χειμερινό 2016

# Outline

# Nearest Neighbor

## Exact NN

Let us consider $d$-dimensional space $D$. Given set $P \subset D$, and query point $q \in D$, its NN is point $p_0 \in P$:

$$\text{dist}(p_0, q) \leq \ \text{dist}(p, q), \quad \forall p \in P.$$

## Approximate NN

Given set $P \subset D$, approximation factor $1 > \epsilon > 0$, and query point $q$, an $\epsilon$-NN, or ANN, is any point $p_0 \in P$:

$$\text{dist}(p_0, q) \leq (1 + \epsilon)\text{dist}(p, q), \quad \forall p \in P.$$

# Two $(r, c)$-neighbor problems

## Definition

*Input: finite set of strings/points $P$, approximation factor $c > 1$, radius $r$, query $q$.*

*Output of $(r, c)$-Neighbor problem:*

*-- Range search: Report all $p \in P$ s.t. $dist(q, p) \leq c \cdot r$.*
*In practice, if $c$ is not given, take $c = 1$.*

*-- Decision problem. If $\exists\, p_0$ within radius $r$, output any $p : dist(q, p) \leq c \cdot r$; otherwise: output a point $p$ within $cr$ or NULL.*

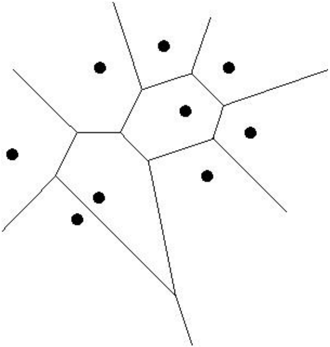
# NN in $\mathbb{R}$

Sort/store the points, use binary search for queries, then:

- Prepreprocessing in $O(n \log n)$ time
- Data structure requiring $O(n)$ space
- Answer the query in $O(\log n)$ time

# NN in $\mathbb{R}^2$

- Preprocessing: Voronoi Diagram in $O(n \log n)$.
- Storage $= O(n)$.
- Given query $q$, find the cell it belongs (point location) in $O(\log n)$.
  NN = site of cell containing $q$.



# NN in $\mathbb{R}^d$

Exact NN:

- Voronoi diagram $= O(n^{\lceil d/2 \rceil})$;
- State of the art: kd-trees: Sp $= O(dn)$, Query $\simeq O(d \cdot n^{1-1/d})$.

Hence the Curse of Dimensionality: Can we solve NN in poly-time in $d$ and faster than linear-time in $n$?

Approximate $\epsilon$-NN:

- BBD-tree (Arya,Mount et al.'94'98) and kd-tree yield optimal space $= O(dn)$ and query $(\log n)$ for $d = O(1)$. Similar for AVD's.
- Locality sensitive hashing (LSH): Both space, query-time bounded by poly$(n, d)$, exp$(1/\epsilon)$: Sp $\simeq dn^{1+1/(1+\epsilon)}$, Q $\simeq dn^{1/(1+\epsilon)}$.
- New projection-based method: Optimal Sp $= O(dn)$, Q $\simeq dn^{\rho}$, $\rho$ somewhat larger than in LSH (Anagnostopoulos,Emiris,Psarros'15).

# Outline

# LSH idea

Map similar strings to same bucket: increase collisions of similar strings

## LSH Family

Let $r_1 < r_2$, probabilities $P_1 > P_2$. A family $H$ of functions is $(r_1, r_2, P_1, P_2)$-sensitive if, for any points $p \neq q$ and any randomly selected function $h \in_R H$,

- if $\text{dist}(p, q) \leq r_1$, then $\text{prob}[h(q) = h(p)] \geq P_1$,
- if $\text{dist}(p, q) \geq r_2$, then $\text{prob}[h(q) = h(p)] \leq P_2$.

Notation: $h \in_R H$ means $h$ is randomly chosen (following the uniform distribution) from $H$.

(Wikipedia)

# Amplification

## Hash-table

LSH creates hash-table using (amplified) hash functions by concatenation:

$$g(p) = [h_1(p), h_2(p), \cdots, h_k(p)],$$

where every $h_i \in_R H$ is chosen uniformly at random (with repetition) from $H$.

This implies some $h_i$ may be chosen more than once for a given $g$ or for different $g$'s.

# Construction

## Preprocess

- Having defined $H$ and hash-function $g$:
- Select $L$ hashing functions $g_1, \ldots, g_L$.
- Initialize $L$ (sparse) hashtables, hash all points to all tables using $g$ (or $\phi$).

Large $k \Rightarrow$ larger gap between $P_1, P_2$. Practical choices are $k = 4$ to 6, $L = 5$ (or 6), and HashTable size $n/4$ (alternatively $n/2$ or $n/8$).

# Range Search

## Range $(r, c)$-Neighbor search

```
Input: $r$, $c$, query $q$
**for** $i$ from 1 to $L$ **do**
    **for** each item $p$ in bucket $g_i(q)$ **do**
        **if** $\mathrm{dist}(q, p) < cr$ **then output** $p$
        **end if**
    **end for**
**end for**
```

In practice, if $c$ not given assume $c = 1$.

Decision problem: "**return** $p$" instead of "**output** $p$".
At end "**return** FAIL"; may also FAIL if many examined points.

# NN search

## Approximate NN

```
Input: query $q$
Let $b \leftarrow$Null; $d_b \leftarrow \infty$
for $i$ from 1 to $L$ do
    for each item $p$ in bucket $g_i(q)$ do
        if large number of retrieved items (e.g. $> 3L$) then return $b$        // trick
        end if
        if $\mathrm{dist}(q,p) < d_b$ then $b \leftarrow p$; $d_b \leftarrow \mathrm{dist}(q,p)$
        end if
    end for
    return $b$
end for
```

Theoretical bounds for $c(1+\epsilon)$-NN by reduction to $((1+\epsilon)^i, c)$-Neighbor decision problems, $i = 1, 2, \ldots, \log_{1+\epsilon} d$.

# Known LSH-able metrics

- Hamming distance,
- $\ell_2$ (Euclidean) distance,
- $\ell_1$ (Manhattan) distance,
- $\ell_k$ distance for any $k \in [0, 2)$,
- $\ell_2$ distance on a sphere,
- Cosine similarity,
- Jaccard coefficient.

Recall $\ell_k$ norm: $$\mathrm{dist}_{\ell_k}(x, y) = \sqrt[k]{\sum_{i=1}^{d} |x_i - y_i|^k}.$$

(Andoni-Indyk:J.ACM'08)

# Outline

# Hamming distance

## Definition

*Given strings $x, y$ of length $d$, their Hamming distance $\mathrm{d}_H(x, y)$ is the number of positions at which $x$ and $y$ differ.*

## Example

*Let $x = 10010$ and $y = 10100$. Then, $\mathrm{d}_H(x, y) = 2$.*

# Definition of hash functions

Recall ▸ idea. Given $x = (x_1, \ldots, x_d) \in \{0,1\}^d$:

$$H = \{h_i(x) = x_i \ : \ i = 1, \ldots, d\}.$$

Obviously, $|H| = d$.
Pick uniformly at random $h \in_R H$: Then $\mathrm{prob}[h(x) \neq h(y)] = \mathrm{d}_H(x,y)/d$,

$$\mathrm{prob}[h(x) = h(y)] = 1 - \mathrm{d}_H(x,y)/d.$$

**Corollary**

*The family $H$ is $(r_1, r_2, 1 - r_1/d, 1 - r_2/d)$-sensitive, for $r_1 < r_2$.*

# LSH in Hamming Space

However probabilities $1 - r_1/d$, $1 - r_2/d$ can be close to each other.

## Amplification

Given parameter $k$, define new family $G$ by concatenation. $G$ is the set of all functions

$$g : \{0,1\}^d \to \{0,1\}^k \ : \ g(x) = [h_{i_1}(x), \cdots, h_{i_k}(x)],$$

where $h_{i_j} \in_R H$ is uniformly chosen for $j = 1, \ldots, k$.

-- We must have $L < |G| = d^k$, so as to pick $L$ different $g$'s.
-- The range of each $g$ is $[0, 2^k)$, so $k < \lg n$.
-- So $k$ should be close to $\lg n - 1$ unlike later cases where $k = 4, 5$.

# Build Hash-tables

## Build

```
Pick uniformly at random $L$ functions $g_1, \dots, g_L \in_R G$, assuming $L < d^k$, by
using random functions $h_i \in_R H$ (chosen uniformly with repetition).
for $i$ from 1 to $L$ do
    Initialize (one-dim) hash-table $T_i$ of size $2^k$:
    for each $p \in P$, store $p$ in bucket $g_i(p)$.
end for
```

## Complexity

Time to build: $O(Lnk)$ $H$-function calls.
Space: $L$ hashtables and $n$ pointers to strings per table $= O(Ln)$ pointers.
Also store $n$ strings $= O(dn)$ bits.
$(r, c)$-Neighbors: Query $= O(L(k + d))$, assuming $O(1)$ strings per bucket.

# Outline

# Euclidean Space

Recall: $\text{dist}_{\ell_2}(x, y)^2 = \sum_{i=1}^{d}(x_i - y_i)^2$.

## Definition

*Let d-vector $v \sim \mathcal{N}(0, 1)^d$ have coordinates identically independently distributed (i.i.d.) by the standard normal (next slide).*
*Set "window" $w \in \mathbb{N}^*$ for the entire algorithm, pick single-precision* `real` *$t$ uniformly $\in_R [0, w)$. For point $p \in \mathbb{R}^d$, define:*

$$h(p) = \lfloor \frac{p \cdot v + t}{w} \rfloor \in \mathbb{Z}.$$

-- Essentially project $p$ on the line of $v$, shift by $t$, partition into cells of length $w$.
-- In general, $w = 4$ is good but should increase for range queries with large $r$.
-- Also $k = 4$ (but can go up to 10), and $L$ may be 5 (up to 30).

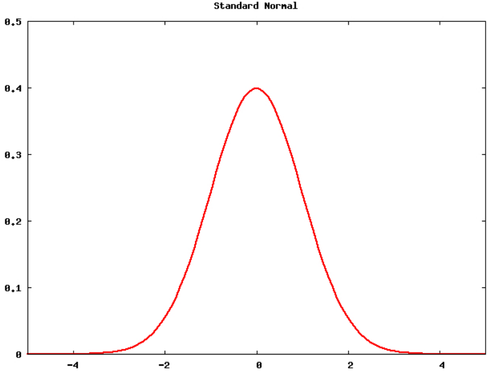
# Normal distribution

Vector $v \sim \mathcal{N}(0,1)^d$ has single-precision `real` coordinates distributed according to the standard normal (Gaussian) distribution:

$$v_i \sim \mathcal{N}(0,1), \quad i = 1, 2, \ldots, d,$$

with mean $\mu = 0$, variance $\sigma^2 = 1$ ($\sigma$ is the standard deviation).

The bell curve:



# Normal from Uniform

Given uniform generator (Wikipedia):

- Marsaglia: Use independent uniform $U, V \in_R (-1, 1)$, $S = U^2 + V^2$. If $S \geq 1$ then start over, otherwise:

  $$X = U\sqrt{\frac{-2\ln S}{S}}, \qquad Y = V\sqrt{\frac{-2\ln S}{S}}$$

  are independent and standard normally distributed.

The $U, V, X, Y$ are single-precision `real`s.

# Hash-table

Indexing function $g(p) = [h_1(p), h_2(p), \ldots, h_k(p)]$ would yield a $k$-dimensional hashtable with many empty buckets. So we define $\phi$ instead:

## 1-dimensional hash-function

We build a 1-dim hash-table with classic index:

$$\phi(p) = (r_1 h_1(p) + r_2 h_2(p) + \cdots + r_k h_k(p) \bmod M) \bmod \text{TableSize},$$

s.t. `int` $r_i \in_R \mathbb{Z}$, prime $M = 2^{32} - 5$ if $h_i(p)$ are `int`, TableSize$= n/2$ (or $n/4$)

Notice $\phi$ computed in `int` arithmetic, if all $h_i(p), r_i$ are `int` ($\leq$ 32 bits).
Can have smaller TableSize$= n/8$ or $n/16$ (heuristic choice).
Recall $(a + b) \bmod m = ((a \bmod m) + (b \bmod m)) \bmod m$.

# Querying trick

Store object ID along with pointer to object, for all bucket elements.

## Object ID

For every $p$, store

$$\mathrm{ID}(p) = r_1 h_1(p) + r_2 h_2(p) + \cdots + r_k h_k(p) \bmod M.$$

Then indexing hash-function is $\phi(p) = \mathrm{ID}(p)$ mod TableSize.

ID is locality sensitive: depends on $w$-length cells on the $v$-lines.

To avoid computing Euclidean distance to all elements in the bucket, do it only for $p$: $\mathrm{ID}(p) = \mathrm{ID}(q)$, assuming such $p$ exists.

# Outline

# Hash-function

$$\text{Recall} \quad \text{dist}_{\ell_1}(x,y) = \sum_{i=1}^{d} |x_i - y_i|.$$

Consider $\mathbb{R}^d$, $r$ is the radius of the range search.
Pick reals: $w \gg r$, uniformly distributed $s_i \in_R [0, w)$, $i = 0, 1, \ldots, d-1$.
Consider $d$-dimensional hashtable, corresponding to grid shifted by the $s_i$'s, where every cell is a bucket; the cell size is determined by $w$.

## LSH function for 1-dim hashtable

$$\text{Let } a_i = \lfloor \frac{x_i - s_i}{w} \rfloor \in \mathbb{Z}, \quad i = 0, 1, \ldots, d-1, \text{ then:}$$

$$h(x) = a_{d-1} + m \cdot a_{d-2} + \cdots + m^{d-1} \cdot a_0, \quad m > \max_i a_i.$$

By concatenation, hash-function $g(x) = [h_1(x), h_2(x), \cdots, h_k(x)]$.

# Outline

# LSH for Cosine similarity

Consider $\mathbb{R}^d$, equipped with cosine similarity of two vectors:

$$\cos(x, y) = \frac{x \cdot y}{\|x\| \cdot \|y\|},$$

which expresses the angle between vectors $x, y$. Notice similarity is opposite of distance: $\text{dist}(x, y) = 1 - \cos(x, y)$.

For comparing documents or, generally, long vectors based on direction only, not length.

Shall be approximated by random projections (next slide).

# Random projection

## Definition

*Let $r_i \sim \mathcal{N}(0,1)^d$, with each* `real` *coordinate iid $\mathcal{N}(0,1)^d$. Define*

$$h_i(x) = \begin{cases} 1, & \textit{if } r_i \cdot x \geq 0 \\ 0, & \textit{if } r_i \cdot x < 0 \end{cases}.$$

*Then $H = \{h_i(x) \mid \text{ for every } r_i\ \}$ is a locality sensitive family.*

## Intuition

Each $r_i$ is normal to a hyperplane. Two vectors lying on same side of many hyperplanes are very likely similar.

Lemma: Two vectors match with probability proportional to their cosine.

Amplification: Given parameter $k$, define new family $G$ by concatenation:

$$G = \{g : \mathbb{R}^d \to \{0,1\}^k \mid g(x) = [h_1(x), h_2(x), \cdots, h_k(x)]\}.$$

# Outline

# Distance Measure

## Definition (Distance Measure)

*A distance measure* $d : D^2 \to \mathbb{R}$ *is a function that satisfies:*

- *Non-negativity:* $d(x, y) \geq 0$
- *Isolation:* $x \neq y \Leftrightarrow d(x, y) > 0$
- *Symmetry:* $d(x, y) = d(y, x)$
- *Triangle inequality:* $d(x, y) \leq d(x, z) + d(z, y)$

It follows that $d(x, x) = 0$, and $|d(x, z) - d(z, y)| \leq d(x, y)$.

## Example

*Distances in vector spaces (e.g. Hamming, Euclidean, Manhattan, any* $\ell_k$ *metric) are all distance measures. They moreover have compact representation, could compute e.g. the mean or a total order.*

# NN in Metric Spaces

## Definition (Metric Space)

*$(D, d)$ is a metric space if $D$ is a domain and $d(\cdot,\cdot)$ a distance measure on $D$ computing the dissimilarity of two objects in the "database". Typically the "database" consists of $P \subset D$.*

## Definition (NN Search)

*Consider metric space $(D, d)$, and $P \subset D$, $|P| = n$, where $d(\cdot,\cdot)$ is given as function (oracle), or by a distance matrix $|P| \times |P|$. Query points lie in $D$ or in $P$, respectively.*

*Problem: Build data-structure so that, for any query $q$, reports $p_0 \in P$: $\forall\, p \in P$,*

*(exact)*: $\mathrm{d}(q, p_0) \leq \mathrm{d}(q, p)$.

*(approximate)*: $\mathrm{d}(q, p_0) \leq (1+\epsilon)\mathrm{d}(q, p)$, *given approximation ratio* $\epsilon > 0$.

# Motivation

Data cannot be handled by traditional techniques:

- Complex models do not have vector representation.
- No intuitive linear sorting, e.g. sort colours wrt hue, but not in a way that for every hue its immediate neighbour is the hue most similar to it.
- Even with vector representation, the dimensionality of feature space is so large that only distances can be computed.

Benefits of NN in metric spaces:

- Sole option.
- Comparable *efficiency* of solutions for specific domains and metric spaces
- High extensibility: a good algorithm for metric spaces has potential in many different domains and types of queries.

# Distance Based Hashing (DBH)

- LSH needs specific families of locality sensitive functions, so it is not applicable to novel, or not studied, metrics.
- DBH produces a family of hash functions tailored to the metric space (and the dataset) by considering only calls to the distance measure and by making no assumptions about the domain (in contrast to LSH).
- Due to the generality of the method there are no theoretical guarantees about the performance of the queries, so the method is optimized through sampling of the dataset.

# DBH family of functions

Consider metric space $(D, d)$ and data $P \subset D$.
Construct family of functions $H$ that behaves like the family $H$ of LSH, but with no theoretical guarantee.

## Definition (Line projection)

*Given $x_1, x_2 \in P \subset D$ define the line projection function*

$$h^{x_1,x_2} : D \to \mathbb{R} : x \mapsto \frac{d(x,x_1)^2 + d(x,x_2)^2 - d(x_1,x_2)^2}{2d(x_1,x_2)}.$$

If $D$ is Euclidean, this is the projection of $x$ on the line through $x_1, x_2$.

# Discretization and balancing

## Definition (Discretization)

*For hashing, discretize* $h^{x_1,x_2}$ *by using thresholds* $t_1, t_2 \in \mathbb{R} \cup \{\pm\infty\}$*:*

$$h^{x_1,x_2}_{t_1,t_2} : D \to \{0,1\} : x \mapsto \begin{cases} 1, & \text{if } h^{x_1,x_2}(x) \in [t_1, t_2] \\ 0, & \text{otherwise} \end{cases}$$

The $t_1, t_2$ should map half the objects of $P$ to 0 and the other half to 1, i.e.:

## Definition (Set of valid thresholds $V$)

*For* $x_1, x_2 \in P$*, the set of thresholds yielding "balanced" h is*

$$V(x_1,x_2) = \{[t_1,t_2] : prob_{x \in P}[h^{x_1,x_2}_{t_1,t_2}(x) = 0] = 1/2\}.$$

# Hash functions

## Definition

*Consider the "balanced" functions*

$$H = \{h^{x_1,x_2}_{t_1,t_2} \; : \; x_1, x_2 \in P \text{ and } [t_1, t_2] \in V(x_1, x_2)\ \}.$$

*Using random $h_i \in_R H$ we define $L$ hash functions by concatenation*

$$g_i(x) = [h_{i1}(x), h_{i2}(x), \cdots, h_{ik}(x)], \; i = 1, \ldots, L.$$

Implement:

-- Pick $x_1, x_2 \in_R P$ uniformly among points where oracle/distance matrix defined

-- Evaluate $h^{x_1,x_2}(x) \in \mathbb{R}$ for all $x \in P$.

-- Set $t_1 =$ median of $\{h^{x_1,x_2}(x) : x \in P\}$, $t_2 = \infty$.

Indexing and search are performed as in LSH using $g_1, \ldots, g_L$.