# Mapping-based Techniques

## Algorithms for Nearest Neighbor Search: Lecture 3

Yury Lifshits

http://yury.name

Steklov Institute of Mathematics at St.Petersburg
California Institute of Technology

RuSSIR
Russian Summer School
in Information Retrieval
2007

# Outline

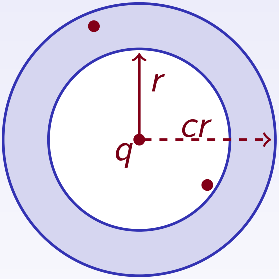
1. Locality-Sensitive Hashing (LSH)
   - General Scheme
   - Ball Grids Hashing

# Outline

1. Locality-Sensitive Hashing (LSH)
   - General Scheme
   - Ball Grids Hashing
2. Random Projections

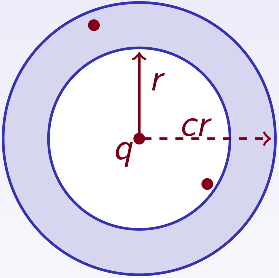
# Approximate Algorithms

***c*-Approximate *r*-range query:** if there at least one $p \in S: \ d(q, p) \leq r$ return some $p' : \ d(q, p') \leq cr$



# Approximate Algorithms

$c$-**Approximate** $r$-**range query:** if there at least one $p \in S: \; d(q,p) \leq r$ return some $p' : \; d(q,p') \leq cr$




$c$-**Approximate nearest neighbor query:** return some $p' \in S: \; d(p',q) \leq cr_{NN}$, where $r_{NN} = \min_{p \in S} d(p,q)$

Today we consider only range queries

# Today's Focus

## Data models:

- $d$-dimensional Euclidean space: $\mathbb{R}^d$
- Hamming cube: $\{0,1\}^d$ with Hamming distance

# Today's Focus

## Data models:

- $d$-dimensional Euclidean space: $\mathbb{R}^d$
- Hamming cube: $\{0, 1\}^d$ with Hamming distance

## Our goal: provable performance bounds

- Sublinear search time, near-linear preprocessing space
- Logarithmic search time, polynomial preprocessing space

# Today's Focus

**Data models:**

- $d$-dimensional Euclidean space: $\mathbb{R}^d$
- Hamming cube: $\{0, 1\}^d$ with Hamming distance

**Our goal:** provable performance bounds

- Sublinear search time, near-linear preprocessing space
- Logarithmic search time, polynomial preprocessing space

**Still an open problem:** approximate nearest neighbor search with logarithmic search and linear preprocessing

# Chapter IX

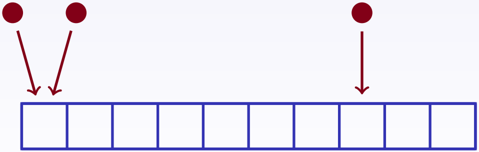
# Locality-Sensitive Hashing

# Definition of LSH

Indyk&Motwani'98

**Locality-sensitive hash family** $\mathcal{H}$ with parameters $(c, r, P_1, P_2)$:

- If $\|p - q\| \leq r$ then $\mathcal{Pr}_{\mathcal{H}}[h(p) = h(q)] \geq P_1$
- If $\|p - q\| \geq cr$ then $\mathcal{Pr}_{\mathcal{H}}[h(p) = h(q)] \leq P_2$



# The Power of LSH

Notation: $\rho = \frac{\log(1/P_1)}{\log(1/P_2)} < 1$

### Theorem

*Any $(c, r, P_1, P_2)$-locality-sensitive hashing leads to an algorithm for $c$-approximate $r$-range search with (roughly) $n^\rho$ query time and $n^{1+\rho}$ preprocessing space*

# The Power of LSH

Notation: $\rho = \frac{\log(1/P_1)}{\log(1/P_2)} < 1$

**Theorem**

*Any $(c, r, P_1, P_2)$-locality-sensitive hashing leads to an algorithm for $c$-approximate $r$-range search with (roughly) $n^\rho$ query time and $n^{1+\rho}$ preprocessing space*

Proof in the next four slides

# LSH: Preprocessing

Composite hash function: $g(p) =< h_1(p), \ldots, h_k(p) >$

# LSH: Preprocessing

Composite hash function: $g(p) =< h_1(p), \ldots, h_k(p) >$

Preprocessing with parameters $L, k$:

1. Choose at random $L$ composite hash functions of $k$ components each
2. Hash every $p \in S$ into buckets $g_1(p), \ldots, g_L(p)$

Preprocessing space: $\mathcal{O}(Ln)$

# LSH: Search

1. Compute $g_1(q), \ldots, g_L(q)$
2. Go to corresponding buckets and explicitly check $d(p, q) \leq ? cr$ for every point there
3. Stopping conditions: (1) we found a satisfying object or (2) we tried at least $3L$ objects

Search time is $\mathcal{O}(L)$

# LSH: Analysis (1/2)

In order to have probability of error at most $\delta$ we set $k, L$ such that

$$P_2^k n \approx 1 \qquad\qquad L \approx (1/P_1)^k \log(1/\delta)$$

# LSH: Analysis (1/2)

In order to have probability of error at most $\delta$ we set $k, L$ such that

$$P_2^k n \approx 1 \qquad\qquad L \approx (1/P_1)^k \log(1/\delta)$$

Solving these constraints:

$$k = \frac{\log n}{\log(1/P_2)}$$

# LSH: Analysis (1/2)

In order to have probability of error at most $\delta$ we set $k, L$ such that

$$P_2^k n \approx 1 \qquad\qquad L \approx (1/P_1)^k \log(1/\delta)$$

Solving these constraints:

$$k = \frac{\log n}{\log(1/P_2)}$$

$$L = (1/P_1)^{\frac{\log n}{\log(1/P_2)}} \log(1/\delta) = n^{\frac{\log(1/P_1)}{\log(1/P_2)}} \log(1/\delta) = n^{\rho} \log(1/\delta)$$

# LSH: Analysis (2/2)

The expected number of $cr$-far objects to be tried is $P_2^k L n \approx L$

For true $r$-neighbor the chance to be hashed to the same bucket as $q$ is at least
$1 - (1 - (1/P_1)^k)^L \geq 1 - (1/e)^{\frac{L}{(1/P_1)^k}} \geq 1 - \delta$

# LSH: Analysis (2/2)

The expected number of $cr$-far objects to be tried is $P_2^k Ln \approx L$

For true $r$-neighbor the chance to be hashed to the same bucket as $q$ is at least
$1 - (1 - (1/P_1)^k)^L \geq 1 - (1/e)^{\frac{L}{(1/P_1)^k}} \geq 1 - \delta$

Preprocessing space $\mathcal{O}(Ln) \approx n^{1+\rho+o(1)}$
Search $\mathcal{O}(L) \approx n^{\rho+o(1)}$

# Ball Grids Hashing: Idea

1. Apply low distortion embedding $A$ into $t$-dimensional Euclidean space

# Ball Grids Hashing: Idea

1. Apply low distortion embedding $A$ into $t$-dimensional Euclidean space
2. Set up $U$ $4w$-step grids of $w$-radius balls that all together cover $t$-dimensional space

# Ball Grids Hashing: Idea

1. Apply low distortion embedding $A$ into $t$-dimensional Euclidean space
2. Set up $U$ $4w$-step grids of $w$-radius balls that all together cover $t$-dimensional space
3. Hash object $p$ to the id of the first ball covering $A(p)$

# BG Hashing: Initialization

Parameters: $t = \log^{2/3} n, w = r \log^{1/6} n, U = 2^{t \log t} \log n$

- Construct $d \times t$ matrix $A$ taking every element at random from normal distribution $N(0, \frac{1}{\sqrt{t}})$
- For every $1 \le i \le U$ choose a random shift $\bar{v}_i \in [0, 4w]^t$

# BG Hashing: Computing

1. Compute $p' = A(p)$
2. From $i = 1$ to $U$ check whether $p'$ is covered by $i$-th grid of balls. If so return $i$ and ball's center and stop.
3. If no such ball found return FAIL

# BG Hashing: Analysis

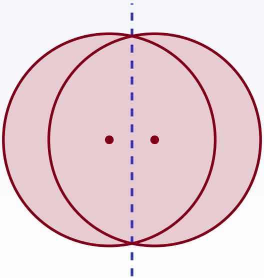
**Fact:** Probability of $\frac{\|Ap - Ap'\|}{\|p - p'\|} \notin [1 - \varepsilon, 1 + \varepsilon]$ is at most $\exp(-\varepsilon^2 t)$

# BG Hashing: Analysis

**Fact:** Probability of $\frac{\|Ap - Ap'\|}{\|p - p'\|} \notin [1 - \varepsilon, 1 + \varepsilon]$ is at most $\exp(-\varepsilon^2 t)$

Given two points $p, s \in \mathbb{R}^t : \|p - s\| = \Delta$:

$$Pr[h(p) = h(s)] = \frac{B(p, w) \cap B(s, w)}{B(p, w) \cup B(s, w)}$$



# BG Hashing: Final Result

3-pages computational proof:

$$\rho = \frac{\log(1/P_1)}{\log(1/P_2)} = 1/c^2 + o(1)$$

# BG Hashing: Final Result

3-pages computational proof:

$$\rho = \frac{\log(1/P_1)}{\log(1/P_2)} = 1/c^2 + o(1)$$

**Theorem (Andoni & Indyk 2006)**

*Consider $c$-approximate $r$-range search in $d$-dimensional space. Then for every $\delta$ there is a randomized algorithm with (roughly) $n^{1/c^2+o(1)}$ query time and $n^{1+1/c^2+o(1)}$ preprocessing space. For every query this algorithm answers correctly with probability at least $1-\delta$*

# Future of LSH

**Achievements:**

- Provably sublinear search time
- Utilization of low-distortion embedding

# Future of LSH

**Achievements:**

- Provably sublinear search time
- Utilization of low-distortion embedding

**Current drawbacks**:

- Probability of error can not be amplified only in preprocessing stage, it can not be decreased to $1/n$
- Asymptotic analysis of power degree: from what place $n^{1/c^2+o(1)}$ is really sublinear?
- For nearest neighbor search $c = \max \frac{r_{NN}(q)}{r_{FN}(q)}$, where $r_{FN}(q)$ is the farthest neighbor. This might be pretty close to 1

# Chapter X

# Random Projections

# Self-Reduction in a Nutshell

**Problem:** $(1+\varepsilon)$-approximate $l$-range queries in $d$-dimensional Hamming cube

# Self-Reduction in a Nutshell

**Problem:** $(1+\varepsilon)$-approximate $l$-range queries in $d$-dimensional Hamming cube

- Apply embedding $\{0,1\}^d$ into $\{0,1\}^k$ such that $l$-neighbors usually fall within $\delta_1 k$ from each other, while $(1+\varepsilon)l$-far objects are embedded at least $\delta_2 k$ from each other
- Precompute all $(\frac{\delta_1+\delta_2}{2})k$-neighbors for every point in $\{0,1\}^k$
- In search step, embed $q$ and explicitly check all precomputed $(\frac{\delta_1+\delta_2}{2})k$-neighbors

# RP: Inner product test

**Single test:**

- Choose random subset of positions of size $\frac{1}{2l}$
- Randomly assign 0 or 1 to every of them, the rest assign to 0, call the resulting vector $r$
- $h_r(p) = r \cdot p$

# RP: Inner product test

**Single test:**

- Choose random subset of positions of size $\frac{1}{2l}$
- Randomly assign 0 or 1 to every of them, the rest assign to 0, call the resulting vector $r$
- $h_r(p) = r \cdot p$

**Claim:** there exist constants $\delta_1 > \delta_2$

- $H_d(p, s) \leq l \Rightarrow Pr[h(p) = h(q)] \geq \delta_1$
- $H_d(p, s) \geq (1 + \varepsilon)l \Rightarrow Pr[h(p) = h(q)] \leq \delta_2$

# RP: Preprocessing

**Inner product mapping:**

- Choose $k$ random tests $r_1, \ldots, r_k$
- Map every $p$ into $A(p) = h_{r_1}(p) \ldots h_{r_k}(p)$

# RP: Preprocessing

**Inner product mapping:**

- Choose $k$ random tests $r_1, \ldots, r_k$
- Map every $p$ into $A(p) = h_{r_1}(p) \ldots h_{r_k}(p)$

Data Structure

- Apply inner product mapping to all strings in database
- For every $v \in \{0, 1\}^k$ precompute all $(\frac{\delta_1 + \delta_2}{2})k$-neighbors

# RP: Search

- Compute $A(q) = h_{r_1}(q) \ldots h_{r_k}(q)$
- Retrieve and explicitly check all $(\frac{\delta_1+\delta_2}{2})k$-neighbors of $A(q)$

# RP: Search

- Compute $A(q) = h_{r_1}(q) \ldots h_{r_k}(q)$
- Retrieve and explicitly check all $(\frac{\delta_1+\delta_2}{2})k$-neighbors of $A(q)$

**Analysis:**

- Chances to miss true $l$-neighbor: $\exp(-\frac{\delta_1-\delta_2}{2\delta_1}k)$
- Chances to waste time on $(1+\varepsilon)l$-far neighbor: $\exp(-\frac{\delta_1-\delta_2}{2\delta_1}k)$

# RP: Search

- Compute $A(q) = h_{r_1}(q) \dots h_{r_k}(q)$
- Retrieve and explicitly check all $(\frac{\delta_1+\delta_2}{2})k$-neighbors of $A(q)$

**Analysis:**

- Chances to miss true $l$-neighbor: $\exp(-\frac{\delta_1-\delta_2}{2\delta_1}k)$
- Chances to waste time on $(1+\varepsilon)l$-far neighbor: $\exp(-\frac{\delta_1-\delta_2}{2\delta_1}k)$

Thus we should take near-logarithmic $k$ which lead to polynomial size of $\{0,1\}^k$ to be NN-precomputed

# RP: Formal Claim

**Theorem (Kushilevetz, Ostrovsky, Rabani, 1998)**

*Consider $(1+\varepsilon)$-approximate $l$-range search in $d$-dimensional Hamming cube. Then for every $\mu$ there is a randomized algorithm with (roughly) $d^2 polylog(d, n)$ query time and $n^{\mathcal{O}(\varepsilon^{-2})}$ preprocessing space. For every query this algorithm answers correctly with probability at least $1-\mu$*

# Exercise

Prove that $2^{\mathcal{O}(t)}$ number of randomly chosen $(w, 4w)$ ball grids is enough to cover $t$-dimensional space with probability $1/2$

# Highlights

- Locality-sensitive hashing: use random projection for defining a candidate list, check its members explicitly

# Highlights

- Locality-sensitive hashing: use random projection for defining a candidate list, check its members explicitly
- Random projections: low-distortion embedding into finite sets + fully precomputed nearest neighbors

# Highlights

- Locality-sensitive hashing: use random projection for defining a candidate list, check its members explicitly
- Random projections: low-distortion embedding into finite sets + fully precomputed nearest neighbors

Thanks for your attention! Questions?

# References

**Course homepage** http://simsearch.yury.name/tutorial.html


Y. Lifshits
The Homepage of Nearest Neighbors and Similarity Search
http://simsearch.yury.name

A. Andoni, P. Indyk
Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. FOCS'06
http://web.mit.edu/andoni/www/papers/cSquared.pdf

E. Kushilevitz, R. Ostrovsky, Y. Rabani
Efficient Search for Approximate Nearest Neighbor in High Dimensional Spaces. STOC'98
http://www.cs.technion.ac.il/~rabani/pss/Publications/KushilevitzOR98.ps.gz