COMS 4995-3: Advanced Algorithms — Feb 20, 2017

Lecture 10 – Advanced Nearest Neighbor Search

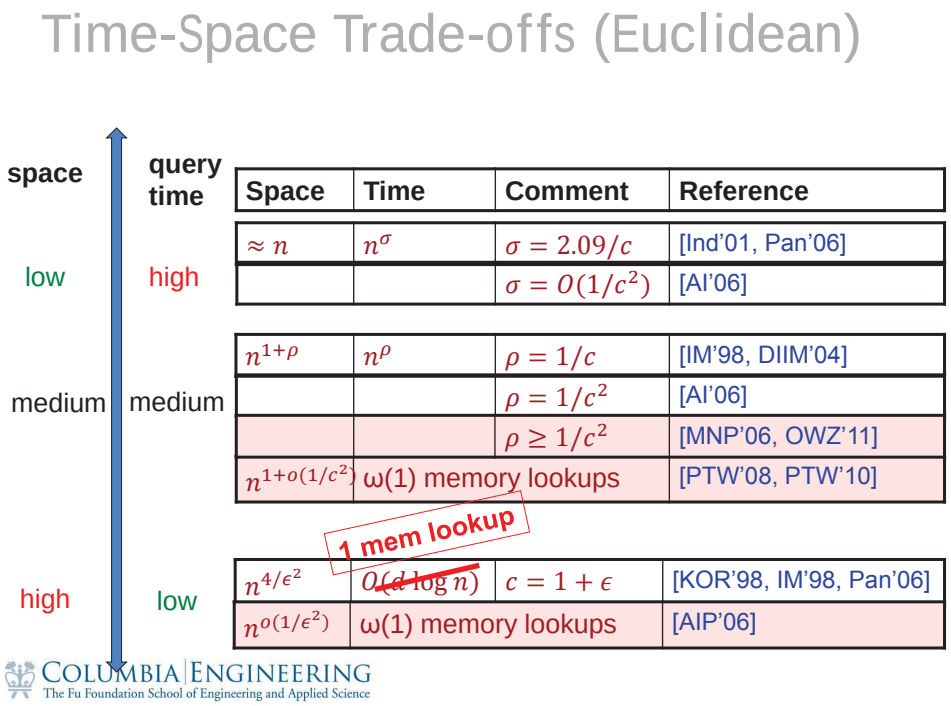
Instructor: *Alex Andoni* — Scribe: *Rex Lei*

1 Introduction

Today's lecture continues the discussion on Locality-Sensitive Hashing from the last lecture and briefly introduces Data-Dependent Hashing.

2 Overview of LSH Results

The following table (Figure 1) shows some of the known algorithms for the Approximate **Near** Neighbor Problem, sorted by their space/query time tradeoffs.



Space	Time	Comment	Reference
$\approx n$	n^{σ}	$\sigma = 2.09/c$	[Ind'01, Pan'06]
		$\sigma = O(1/c^2)$	[AI'06]
$n^{1+\rho}$	n^{ρ}	$\rho = 1/c$	[IM'98, DIIM'04]
		$\rho = 1/c^2$	[AI'06]
		$\rho \geq 1/c^2$	[MNP'06, OWZ'11]
$n^{1+o(1/c^2)}$	ω(1) memory lookups		[PTW'08, PTW'10]
n^{4/ϵ^2}	~~$O(d \log n)$~~	$c = 1 + \epsilon$	[KOR'98, IM'98, Pan'06]
$n^{o(1/\epsilon^2)}$	ω(1) memory lookups		[AIP'06]

Figure 1: Recent Results for the Approximate Near Neighbor Problem

Some quick observations:

Observation 1. *These algorithms are for Euclidean space.*

Observation 2. *We ignore the $O(d \lg n)$ factors for the query time (needed to do a lookup) and $O(dn)$ factors in space (needed to store the data set).*

Observation 3. *Recall that c is the approximation factor in the approximate near neighbor problem.*

2.1 Low Space, High Query Time

These results tend to have roughly near linear space and large query time.

The first result [Indyk'01, Panigrahy'06] was discussed in class (next section).

The second result [Andoni, Indyk'06] was mentioned as an upper bound to the query time.

2.2 Medium Space, Medium Query Time

These results tend to have slightly larger than n space and smaller than n query time.

The first result was discussed last lecture (Lecture 9, pg 3-5). In class, we constructed such a scheme for Hamming space, but we will see how to extend it to Euclidean space on the homework.

The second result [Andoni, Indyk'06] showed an LSH scheme with an improved space and run time.

The third result [Motwani, Naor, Panigrahi'06, O'Donnell, Wu, Zhou] proved a lower bound showing that $\rho \leq 1/c^2$ was the best we can achieve. This lower bound is related to the isoperimetric inequality and proved using functional analysis.

The fourth result was not covered in class.

2.3 High Space, Low Query Time

These results tend to precompute and store lots of data so the query time is very fast.

The first result [Kushilevitz, Ostrovsky, Rabani98] was covered in the previous lecture (Lecture 9, top of page 2). In this solution, we precomputed all possible k-bit strings $\phi(q)$ and stored them, using n^{4/ϵ^2} space. The positive side was that each query only required one memory lookup (d dimensions, n elements total, so we read d bits after finding the element q in our data structure).

The second result [Andoni, Indyk, Patrascu] shows a relationship between the size of the exponent for the space and the number of memory lookups needed.

3 Near-Linear Space for $\{0,1\}^d$ [Indyk'01, Panigrahy'06]

This algorithm takes a low amount of space in exchange for a high query time. It is not the currently best-known algorithm, but we discussed it in class to get a feel for what low/high algorithms are like.

Previously, we have seen algorithms that sample a single bucket from multiple hash tables. The main idea of the algorithm is to **sample a few buckets in the same hash table**.

3.1 Setting

We'll think about close points being at a distance at most $r = \frac{d}{2c} \to P_1 = (1 - \frac{1}{2c})$ and far points being at a distance at least $cr = \frac{d}{2} \to P_2 = 1/2$.

3.2 Algorithm

We will use one hash table with $k = \frac{logn}{log\frac{1}{p_2}} = \alpha \ln n$ for some constant α. Our hash function is

$$g(x) = x|_{i_1,i_2,\ldots,i_k}$$

where each i_j is a random integer in $[d]$, representing some coordinate of x.

On query q, we compute $w := g(q) \in \{0,1\}^k$. Then, we repeat the following $R = n^\sigma$ times:
Let w' : be defined as w but we flip each w_j (the j'th coordinate) with probability $1 - P_1$
Look up bucket w' and compute the distance from q to all points there
If found an approximate near neighbor, then stop.

3.3 Correctness and Runtime

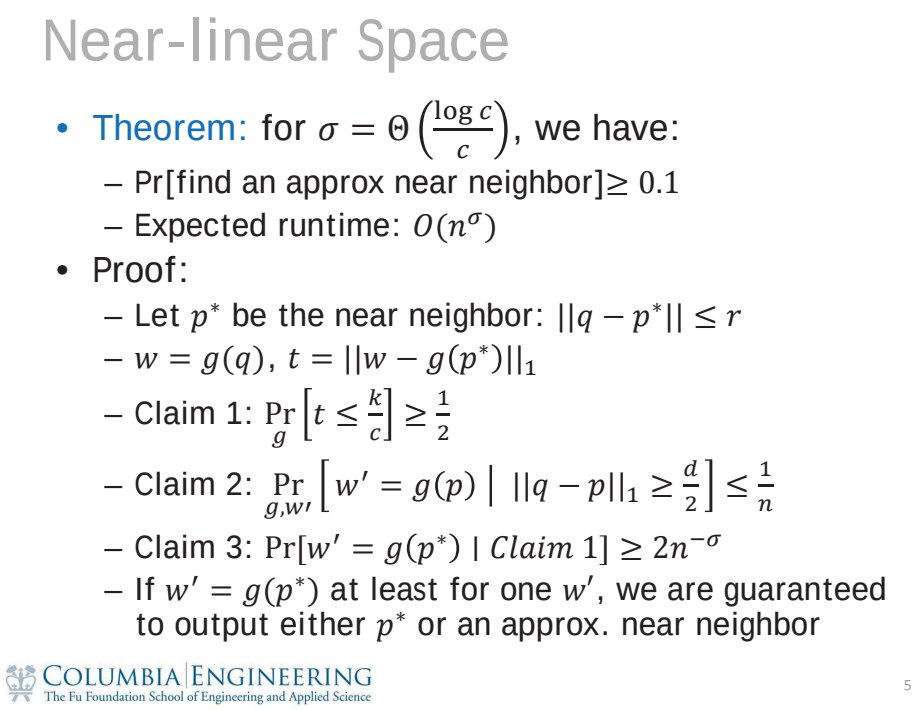


Figure 2: Analysis of [Indyk'01, Panigrahy'06]

The theorem and proof sketch analyzing this algorithm are given in Figure 2. We briefly discussed the claims in class, but did not prove them. Note that $r = d/2c$ in this algorithm.

Proof. Assuming that a near neighbor p^* exists for query q, we define $w = g(q)$ and random variable $t = ||w - g(p^*)||_1$, the Hamming distance between w and the hashed value of p^*.

Intuitively, if p^* and q are near each other, then they should differ only on a few bits in the hash function (which is just a projection of the coordinates). This is described by the first claim (k is the

maximum number of bits any two hashed values can differ on).

The second claim says that the probability that our randomly generated w' is the same value as the hash of some far point (Hamming distance $\geq d/2$) is at most $1/n$. Therefore, we'd expect a constant number of collisions (1) in each w' bucket we check. Stated differently, this means that the runtime of the algorithm is bounded by the number of different w' (buckets) we check.

Assuming the condition of Claim 1 holds ($t \leq k/c$), Claim 3 states that the probability that our $w' = g(p^*)$ (which is the probability we find the hash of p^* and check it) $\geq 2n^{-\sigma}$. This is useful to bound the total number of times we need to generate w'.

Combining claims 1 and 3, we have that after $O(n^\sigma)$ iterations, we will have checked roughly $O(n^\sigma)$ many points to see if they are near to q (claim 2), and we will find p^* or an approximate near neighbor with constant probability.

□

4 Beyond LSH

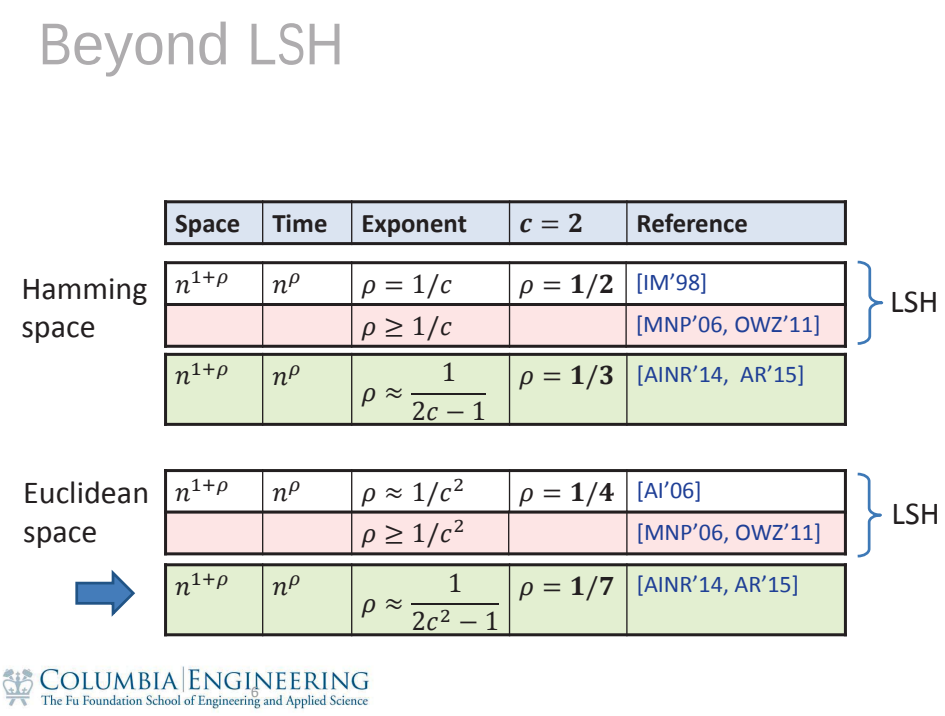


Figure 3: LSH-Algorithms which also take advantage of the dataset

While all the previous algorithms for solving the approximate near neighbor problem have used locality-sensitive hashing, this approach is constrained by the fact that hash functions must be chosen independent of the data set. However, if we relax this restriction, we can get more powerful algorithms.

5 Data-Dependent Hashing

We want to decide on a random hash function after seeing the given dataset.

Definition 4. *Data Dependent Locality-Sensitive Hashing:*
Given $P \subset \mathbb{R}^d, |P| = n$, *find a hash function* $\mathcal{H}_D$ *such that* $\forall p \in D, q \in \mathbb{R}^d$,
if p, q *are far, then* $Pr_{h \in H}[h(p) = h(q)] \leq P_2$ *(low probability) and*
if p, q *are close, then* $Pr_{h \in H}[h(p) = h(q)] > P_1$ *(high probability).*

(Recall that $\rho := \frac{\lg(1/p_1)}{\lg(1/p_2)}$.)

Essentially we only need our hash function to provide guarantees on $p \in D$, whereas for non-data-dependent hash functions we would like guarantees on all $p \in \mathbb{R}^d$.

Since we know the individual points $p \in D$, we can tailor our hash function to these individual points. Recall that a hash function can be viewed as a partition of $\mathbb{R}^d$. Therefore, an ideal solution to the D.D. LSH problem is the Voronoi diagram!

5.1 Voronoi diagram as a solution to D.D. LSH problem

The Voronoi diagram, first discussed in Lecture 8, pg 3, would be an ideal candidate for D.D. LSH, since we can just use the points of D as the points of the diagram.

However, the Voronoi diagram is not efficiently computable, especially at higher dimensions, so it would be inefficent to use it.

Observation 5. *One possible project is to assume your dataset has specific "nice" properties, and then create an efficient data-dependent hash to solve the approximate near neighbor problem for that dataset.*

6 Construction of hash function [Andoni-Indyk-Nguyen-Razenshteyn'14], [Andoni-Razenshteyn'15]

The two components of a data-dependent hash function are

- Nice geometric structure (has better LSH)
- Reduction to such structure (data dependent)

This paper deals with points on a unit sphere, where $cr = \sqrt{2}$, so far pairs are near orthogonal.

Claim 6. $cr = \sqrt{2}$ *would be the distance between two points if the dataset were generated randomly on the sphere.*

Proof. Imagine q is a fixed point $(1, 0, 0, \ldots, 0)$ on the sphere. Note that we can rotate the sphere so this is the case.

Let a far point p be a point randomly generated on the sphere. One way to do this is to let $p' = (g_1, g_2, \ldots, g_d)$ where each g_i is a Gaussian. Then, we normalize by the length of the Gaussian, giving $p = \frac{p'}{||p'||_2}$. The size of the denominator is roughly d, since it follows a χ^2 distribution.

Taking the dot product of p and q, we get $p \cdot q = \frac{g_1 \cdot 1}{||p'||_2} \approx \frac{g_1}{\sqrt{d}} \in \left(\frac{-3}{\sqrt{d}}, \frac{3}{\sqrt{d}}\right)$ with constant probability.

Therefore, p and q have dot product close to 0 with constant probability, so they tend to be orthogonal (for large d).

□

Now consider close points of distance $r = \sqrt{2}/c$. Let our query be a 45 degree angle from a near neighbor. The following two algorithms were compared in class:

6.1 Hyperplanes [Charikar'02]

In this algorithm, we sample a unit vector r uniformly at random, and hash each point p into $sgn\langle r, p\rangle$, which is the sign of the dot product of r and p. This function divides the sphere in half along the hyperplane orthogonal to vector r. Therefore, the probability that p and q has to the same sign is $1 - \alpha/\pi$, where α is the angle between p and q in radians.

This algorithm yields $P_1 = 3/4$ (close points at 45 degrees), $P_2 = 1/2$ (far points at roughly 90 degrees), and $\rho \approx .42$.

6.2 Voronoi [Andoni-Indyk-Nguyen-Razenshteyn'14] based on [Karger-Motwani-Sudan'94]

Sample T i.i.d. standard d-dimensional Gaussians $g_1, g_2, \ldots, g_T$. Hash p into $h(p) = \text{argmax}_{1 \le i \le T}\langle p, g_i\rangle$.

(Note that $T = 2$ is simply hyperplane LSH.)

6.3 Hyperplane vs. Voronoi

The comparison between the two algorithms for $k = 6$ hyperplanes and (equivalently) 2^6 Gaussian vectors is done in the following slide (Figure 4). The graph plots the probability that the two algorithms return a point (if it is a near neighbor or if it is a far neighbor).

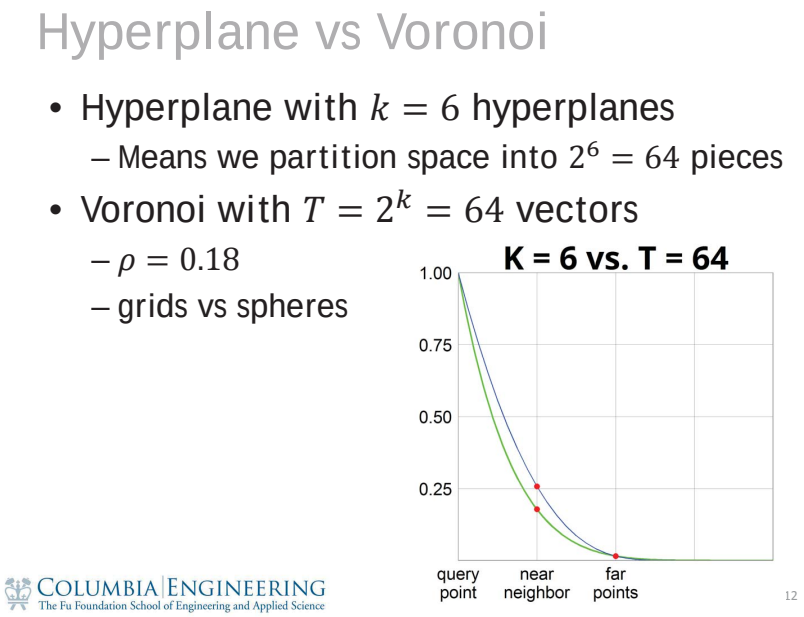


Figure 4: Comparison between Hyperplane and Voronoi Algorithms

7 Additional Resources

- Notes from Algorithmic Techniques for Massive Data, COMS 6998-9 (Fall'15)
- Resources under Class Page's Feb 20th section
- References to papers correspond with references in this paper