COMS E6998-9: Algorithms for Massive Data (Spring'19) Oct 1, 2025

Lecture 9: c-ANNS via Locality Sensitive Hashing

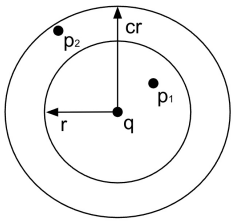
Instructor: *Alex Andoni* Scribes: *Alessandro Castillo*

1 Review of c-ANNS

1.1 Definition

Given a point set $P \subset \mathbb{R}^d$, with $n = |P|$, and a query $q \in \mathbb{R}^d$, the (c, r)-*Approximate Nearest Neighbor Search* problem is:

> If there exists $p^* \in P$ such that $\|q - p^*\| \leq r$, then we must return (with probability at least 90%) some $p \in P$ with $\|q - p\| \leq c\,r$, where $c > 1$.



1.jpeg

Figure 1: Geometric picture of a c-approximate neighbor: if p^* lies inside the r-ball around q, we can return any p in the larger cr-ball.

2 Discussion of the 90% success guarantee

2.1 Algorithm 1: Naïve scan with dimension reduction

A naïve query needs $O(nd)$ time: for each of the n points we compute a d-dimensional distance.

When n is huge (e.g. $n \approx 10^9$) and $d \approx 10^3$, this is too expensive.

A common speed-up is to apply a random **Johnson–Lindenstrauss (JL)** projection $\rho : \mathbb{R}^d \to \mathbb{R}^k$ with

$$k = O\left(\frac{\log(1/\delta)}{\varepsilon^2}\right),$$

to obtain a $(1 + \varepsilon)$-approximate embedding with failure probability δ.

We preprocess $P' = \{\rho(p) \mid p \in P\}$ and for a query q we compare $\rho(q)$ to all $\rho(p)$.

Space:

$$O(nd) + O(dk),$$

where the second term stores the projection matrix (or sketch).

Query time:

$$O(dk) + O(nk) = O\left(n \frac{\log(1/\varepsilon)}{\varepsilon^2}\right) \quad (\text{for } d \gg k).$$

The lecture emphasized that dimension reduction lowers the *per-distance cost*, but does not reduce the n-dependence of query time.

3 Towards Faster Query Time

We want to reduce the dependence on n, even if that means increasing preprocessing.

Two key ideas:

1. Allow the projection ρ to depend on the data P (e.g. data-dependent quantization).
2. Use hashing-based methods (esp. **Locality-Sensitive Hashing (LSH)**).

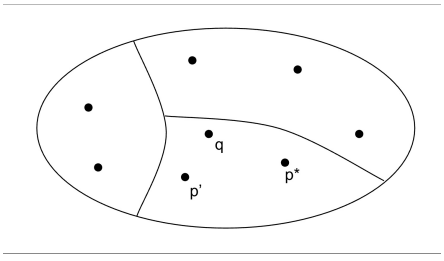
3.1 Hashing-based Approach

A hash function $h : \mathbb{R}^d \to U$ maps points to "buckets" so that nearby points collide with higher probability.

Definition 1 (LSH family)**.** *A family $\mathcal{H}$ of hash functions is (r, cr, p_1, p_2)-LSH if for all points p, q:*

$$\Pr_{h \sim \mathcal{H}}[h(p) = h(q)] = \begin{cases} \geq p_1, & \text{if } \|p - q\| \leq r, \\ \leq p_2, & \text{if } \|p - q\| \geq c\, r. \end{cases}$$

We require $p_1 > p_2$ but can never achieve $p_1 = 1$, $p_2 = 0$.



2.jpeg

Figure 2: Nearby points tend to hash into the same bucket more often than distant points.

3.2 Classic Indyk–Motwani (1998) Theorem

If such an LSH family exists, one can solve (c, r)-ANNS with

$$\text{space } = O(nd) + O(n^{1+\rho}), \qquad \text{query time } = O(n^{\rho}),$$

where

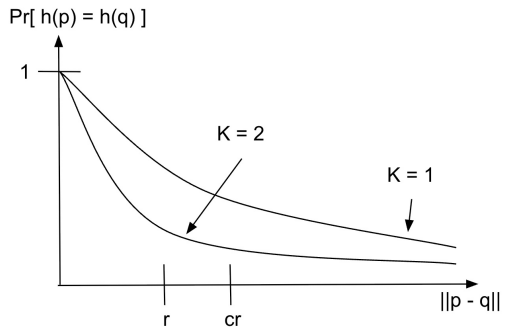
$$\rho = \frac{\log(1/p_1)}{\log(1/p_2)}.$$

3.3 Proof Sketch

Pick an integer $k > 0$ and define

$$g(p) = (h_1(p), h_2(p), \ldots, h_k(p)), \quad h_i \in \mathcal{H} \text{ i.i.d.}$$

Then g behaves like an LSH family with parameters (r, cr, p_1^k, p_2^k).



4.jpeg

Figure 3: Increasing k lowers collision probability for far points (curve drops) but also for near points.

Base algorithm:

1. Build a single hash table: store each data point p in the bucket indexed by $g(p)$.
2. Given a query q, compute $g(q)$ and examine only the bucket $g(q)$, checking all its occupants for a valid neighbor.

Analysis:

$$\mathbb{E}[\#\text{far points in } g(q)] \leq n\, p_2^k.$$

Success probability with a single table is at least p_1^k.

Boosting success: Use $L = \Theta(1/p_1^k)$ independent tables. Success probability ≥ 0.9.

Choosing k: Balance the bucket size $n\, p_2^k$ against success probability. Optimal choice (up to constants):

$$p_2^k \approx \frac{1}{n} \quad \Longrightarrow \quad k \approx \frac{\log n}{\log(1/p_2)}.$$

Query time:

$$T_q = L\big(k + \mathbb{E}[\#\text{candidates}]\big) = \tilde{O}(n^\rho), \quad \rho = \frac{\log(1/p_1)}{\log(1/p_2)}.$$

Your original derivation around the last few lines had algebraic slips; I have simplified to the standard textbook result.

4 Remarks

- The key idea is that hashing eliminates the need to scan all n points, leaving only about n^{ρ} candidates.
- In practice, many refinements exist (data-dependent LSH, product quantization, multi-probe, etc.) that improve constants or adapt to real distributions.
- The $(1+\varepsilon)$ vs. c approximation factors, the choice of distance metric, and how to implement h (e.g. random hyperplanes for cosine) are all practical concerns.