Near-Optimal Hashing Algorithms for Approximate Near(est) Neighbor Problem

Piotr Indyk
MIT

Joint work with: Alex Andoni, Mayur Datar, Nicole Immorlica, Vahab Mirrokni
Definition

• Given: a set $P$ of points in $\mathbb{R}^d$

• Nearest Neighbor: for any query $q$, returns a point $p \in P$ minimizing $||p-q||$

• $r$-Near Neighbor: for any query $q$, returns a point $p \in P$ s.t. $||p-q|| \leq r$ (if it exists)
Nearest Neighbor: Motivation

• Learning: nearest neighbor rule
• Database retrieval
• Vector quantization, a.k.a. compression
Brief History of NN
The case of $d=2$

- Compute Voronoi diagram
- Given $q$, perform point location
- Performance:
  - Space: $O(n)$
  - Query time: $O(\log n)$
The case of $d \geq 2$

- Voronoi diagram has size $n^{O(d)}$
- We can also perform a linear scan: $O(dn)$ time
- That is pretty much all what known for exact algorithms with theoretical guarantees
- In practice:
  - kd-trees work “well” in “low-medium” dimensions
  - Near-linear query time for high dimensions
Approximate Near Neighbor

• c-Approximate r-Near Neighbor: build data structure which, for any query q:
  – If there is a point \( p \in P \), \( ||p-q|| \leq r \)
  – it returns \( p' \in P, \ ||p-q|| \leq cr \)

• Reductions:
  – c-Approx Nearest Neighbor reduces to c-Approx Near Neighbor
    (log overhead)
  – One can enumerate all approx near neighbors
    → can solve exact near neighbor problem
  – Other apps: c-approximate Minimum Spanning Tree, clustering, etc.
### Approximate algorithms

- **Space/time exponential in d** [Arya-Mount-et al], [Kleinberg’97], [Har-Peled’02], [Arya-Mount-…]

- **Space/time polynomial in d** [Kushilevitz-Ostrovsky-Rabani’98], [Indyk-Motwani’98], [Indyk’98], [Gionis-Indyk-Motwani’99], [Charikar’02], [Datar-Immorlica-Indyk-Mirrokni’04], [Chakrabarti-Regev’04], [Panigrahy’06], [Ailon-Chazelle’06]…

<table>
<thead>
<tr>
<th>Space</th>
<th>Time</th>
<th>Comment</th>
<th>Norm</th>
<th>Ref</th>
</tr>
</thead>
<tbody>
<tr>
<td>$dn+n^4/\varepsilon^2$</td>
<td>$d \cdot \log \varepsilon^2$ or 1</td>
<td>$c = 1 + \varepsilon$</td>
<td>$l_2$</td>
<td>[KOR’98, IM’98]</td>
</tr>
<tr>
<td>$n^{\Omega(1/\varepsilon^2)}$</td>
<td>$O(1)$</td>
<td></td>
<td></td>
<td>[AIP’0?]</td>
</tr>
<tr>
<td>$dn+n^{1+p(c)}$</td>
<td>$dn^{p(c)}$</td>
<td>$p(c) = 1/c$</td>
<td>$l_2$</td>
<td>[IM’98], [Cha’02]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$p(c) &lt; 1/c$</td>
<td></td>
<td>[DIIM’04]</td>
</tr>
<tr>
<td>$dn \cdot \log{s}$</td>
<td>$dn^{\sigma(c)}$</td>
<td>$\sigma(c) = O(\log c/c)$</td>
<td>$l_2$</td>
<td>[Ind’01]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\sigma(c) = O(1/c)$</td>
<td></td>
<td>[Pan’06]</td>
</tr>
<tr>
<td>$dn+n^{1+p(c)}$</td>
<td>$dn^{p(c)}$</td>
<td>$p(c) = 1/c^2 + o(1)$</td>
<td>$l_2$</td>
<td>[Al’06]</td>
</tr>
<tr>
<td>$dn \cdot \log{s}$</td>
<td>$dn^{\sigma(c)}$</td>
<td>$\sigma(c) = O(1/c^2)$</td>
<td></td>
<td>[Al’06]</td>
</tr>
</tbody>
</table>
Locality-Sensitive Hashing

• Idea: construct hash functions $g: \mathbb{R}^d \rightarrow U$ such that for any points $p, q$:
  
  - If $||p-q|| \leq r$, then $\Pr[g(p)=g(q)]$ is “high” “not-so-small”
  
  - If $||p-q|| > cr$, then $\Pr[g(p)=g(q)]$ is “small”

• Then we can solve the problem by hashing
LSH [Indyk-Motwani’98]

• A family $H$ of functions $h: \mathbb{R}^d \rightarrow U$ is called $(P_1, P_2, r, cr)$-sensitive, if for any $p, q$:
  – if $||p-q|| < r$ then $\Pr[h(p)=h(q)] > P_1$
  – if $||p-q|| > cr$ then $\Pr[h(p)=h(q)] < P_2$

• Example: Hamming distance
  – LSH functions: $h(p)=p_i$, i.e., the $i$-th bit of $p$
  – Probabilities: $\Pr[h(p)=h(q)] = 1-D(p,q)/d$

$p=10010010$
$q=11010110$
LSH Algorithm

• We use functions of the form
  \[ g(p)=<h_1(p), h_2(p), \ldots, h_k(p)> \]

• Preprocessing:
  – Select \( g_1 \ldots g_L \)
  – For all \( p \in P \), hash \( p \) to buckets \( g_1(p) \ldots g_L(p) \)

• Query:
  – Retrieve the points from buckets \( g_1(q), g_2(q), \ldots \), until
    • Either the points from all \( L \) buckets have been retrieved, or
    • Total number of points retrieved exceeds \( 2L \)
  – Answer the query based on the retrieved points
  – Total time: \( O(dL) \)
Analysis

• LSH solves \( c \)-approximate NN with:
  – Number of hash fun: \( L=n^\rho \), \( \rho=\log(1/P_1)/\log(1/P_2) \)
  – E.g., for the Hamming distance we have \( \rho=1/c \)
  – Constant success probability per query \( q \)

• Questions:
  – Can we extend this beyond Hamming distance?
    • Yes:
      – embed \( l_2 \) into \( l_1 \) (random projections)
      – \( l_1 \) into Hamming (discretization)
  – Can we reduce the exponent \( \rho \)?
Projection-based LSH

[Datar-Immorlica-Indyk-Mirrokni’04]

- Define \( h_{X,b}(p) = \lfloor (p^*X + b)/w \rfloor \):
  - \( w \approx r \)
  - \( X=(X_1 \ldots X_d) \), where \( X_i \) is chosen from:
    - Gaussian distribution (for \( l_2 \) norm)
    - “s-stable” distribution* (for \( l_s \) norm)
  - \( b \) is a scalar
- Similar to the \( l_2 \rightarrow l_1 \rightarrow \text{Hamming} \) route

* I.e., \( p^*X \) has same distribution as \( ||p||_s \) Z, where Z is s-stable
Analysis

• Need to:
  – Compute $Pr[h(p)=h(q)]$ as a function of $||p-q||$ and $w$; this defines $P_1$ and $P_2$
  – For each $c$ choose $w$ that minimizes $\rho=\log_{1/P_2}(1/P_1)$

• Method:
  – For $l_2$: computational
  – For general $l_s$: analytic
$\rho(w)$ for various $c$’s: $l_1$
\( \rho(w) \) for various c's: \( l_2 \)
$\rho(c)$ for $l_2$
New LSH scheme
[Andoni-Indyk’06]

• Instead of projecting onto $\mathbb{R}^1$, project onto $\mathbb{R}^t$, for constant $t$
• Intervals $\rightarrow$ lattice of balls
  – Can hit empty space, so hash until a ball is hit
• Analysis:
  – $\rho = 1/c^2 + O(\log t / t^{1/2})$
  – Time to hash is $t^{O(t)}$
  – Total query time: $dn^{1/c^2+o(1)}$
• [Motwani-Naor-Panigrahy’06]: LSH in $l_2$ must have $\rho \geq 0.45/c^2$
Connections to

- [Charikar-Chekuri-Goel-Guha-Plotkin’98]
  - Consider partitioning of $\mathbb{R}^d$ using balls of radius $R$
  - Show that $\Pr[\text{Ball}(p) \neq \text{Ball}(q)] \leq ||p-q||/R \times d^{1/2}$
    - Linear dependence on the distance – same as Hamming
    - Need to analyze $R \approx ||p-q||$ to achieve non-linear behavior!
      (as for the projection on the line)

- [Karger-Motwani-Sudan’94]
  - Consider partitioning of the sphere via random vectors $u$ from $\mathbb{N}^d(0,1)$:
    - $p$ is in $\text{Cap}(u)$ if $u^*p \geq T$
  - Showed $\Pr[\text{Cap}(p) = \text{Cap}(q)] \leq \exp[-(2T/||p+q||)^2/2]$  
    - Large relative changes to $||p-q||$ can yield only small relative changes to $||p+q||$
Proof idea

Claim: $\rho = \log(P1)/\log(P2) \to 1/c^2$

- $P1 = Pr(1), P2 = Pr(c)$
- $Pr(z)$ = prob. of collision when distance $z$

Proof idea:
- **Assumption:** ignore effects of mapping into $R^t$
- $Pr(z)$ is proportional to the volume of the cap
- Fraction of mass in a cap is proportional to the probability that the x-coordinate of a random point $u$ from a ball exceeds $x$
- **Approximation:** the x-coordinate of $u$ has approximately normal distribution
  $\to Pr(x) \approx \exp(-A x^2)$

- $\rho = \log[ \exp(-A1^2) ] / \log[ \exp(-Ac^2) ] = 1/c^2$
New LSH scheme, ctd.

• How does it work in practice?
  • The time $t^O(t)dn^{1/c^2+f(t)}$ is not very practical
    – Need $t \approx 30$ to see some improvement
• Idea: a different decomposition of $\mathbb{R}^t$
  – Replace random balls by Voronoi diagram of a lattice
  – For specific lattices, finding a cell containing a point can be very fast
    → fast hashing
Leech Lattice LSH

• Use Leech lattice in $\mathbb{R}^{24}$, t=24
  – Largest kissing number in 24D: 196560
  – Conjectured largest packing density in 24D
  – 24 is 42 in reverse…

• Very fast (bounded) decoder: about 519 operations [Amrani-Beery’94]

• Performance of that decoder for $c=2$:
  – $1/c^2$ 0.25
  – $1/c$ 0.50
  – Leech LSH, any dimension: $\rho \approx 0.36$
  – Leech LSH, 24D (no projection): $\rho \approx 0.26$
Conclusions

• We have seen:
  – Algorithm for c-NN with $dn^{1/c^2+o(1)}$ query time (and reasonable space)
    • Exponent tight up to a constant
  – (More) practical algorithms based on Leech lattice
• We haven’t seen:
  – Algorithm for c-NN with $dn^{O(1/c^2)}$ query time and $dn \log n$ space
• Immediate questions:
  – Get rid of the $o(1)$
  – …or came up with a really neat lattice…
  – Tight lower bound
• Non-immediate questions:
  – Other ways of solving proximity problems
Advertisement

• See LSH web page (linked from my web page for):
  – Experimental results (for the ’04 version)
  – Pointers to code
Experiments
Experiments (with ’04 version)

• E^2LSH: Exact Euclidean LSH (with Alex Andoni)
  – Near Neighbor
  – User sets r and P = probability of NOT reporting a point within distance r (=10%)
  – Program finds parameters k,L,w so that:
    • Probability of failure is at most P
    • Expected query time is minimized

• Nearest neighbor: set radius (radiae) to accommodate 90% queries (results for 98% are similar)
  – 1 radius: 90%
  – 2 radiae: 40%, 90%
  – 3 radiae: 40%, 65%, 90%
  – 4 radiae: 25%, 50%, 75%, 90%
Data sets

- MNIST OCR data, normalized (LeCun et al)
  - $d=784$
  - $n=60,000$
- Corel_hist
  - $d=64$
  - $n=20,000$
- Corel_uci
  - $d=64$
  - $n=68,040$
- Aerial data (Manjunath)
  - $d=60$
  - $n=275,476$
Other NN packages

- **ANN (by Arya & Mount):**
  - Based on kd-tree
  - Supports exact and approximate NN
- **Metric trees (by Moore et al):**
  - Splits along arbitrary directions (not just x,y,..)
  - Further optimizations
## Running times

<table>
<thead>
<tr>
<th></th>
<th>MNIST</th>
<th>Speedup</th>
<th>Corel_hist</th>
<th>Speedup</th>
<th>Corel_uci</th>
<th>Speedup</th>
<th>Aerial</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>E2LSH-1</td>
<td>0.00960</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E2LSH-2</td>
<td>0.00851</td>
<td>0.00024</td>
<td>0.00070</td>
<td>0.07400</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E2LSH-3</td>
<td>0.00018</td>
<td>0.00055</td>
<td>0.00833</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E2LSH-4</td>
<td>0.00668</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ANN</td>
<td>0.25300</td>
<td>29.72274</td>
<td>0.00018</td>
<td>1.011236</td>
<td>0.00274</td>
<td>4.954792</td>
<td>0.00741</td>
<td>1.109281</td>
</tr>
<tr>
<td>MT</td>
<td>0.20900</td>
<td>24.55357</td>
<td>0.00130</td>
<td>7.303371</td>
<td>0.00650</td>
<td>11.75407</td>
<td>0.01700</td>
<td>2.54491</td>
</tr>
</tbody>
</table>
LSH vs kd-tree (MNIST)
Caveats

• For ANN (MNIST), setting $\varepsilon=1000\%$ results in:
  – Query time comparable to LSH
  – Correct NN in about 65% cases, small error otherwise

• However, no guarantees

• LSH eats much more space (for optimal performance):
  – LSH: 1.2 GB
  – Kd-tree: 360 MB