Nearest Neighbor Search using Kd-trees

Rina Panigrahy *

December 4, 2006

Abstract

We suggest a simple modification to the kd-tree search algorithm for nearest neighbor search resulting in an improved performance. The Kd-tree data structure seems to work well in finding nearest neighbors in low dimensions but its performance degrades even if the number of dimensions increases to more than three. Since the exact nearest neighbor search problem suffers from the curse of dimensionality we focus on approximate solutions; a c-approximate nearest neighbor is any neighbor within distance at most c times the distance to the nearest neighbor. We show that for a randomly constructed database of points the traditional kd-tree search algorithm has a very low probability of finding an approximate nearest neighbor; the probability of success drops exponentially in the number of dimensions d as $e^{-\Omega(d/c)}$. However, a simple change to the search algorithm results in a much higher chance of success. Instead of searching for the query point in the kd-tree the search for a random set of points in the neighborhood of the query point. It turns out that searching for $e^{\Omega(d/c)}$ such points can find the c-approximate nearest neighbor with a much higher chance of success.

1 Introduction

In this chapter we study the problem of finding the nearest neighbor of a query point in a high dimensional (at least three) space focusing mainly on the Euclidean space: given a database of n points in a d dimensional space, find the nearest neighbor of a query point. This fundamental problem arises in several applications including data mining, information retrieval, and image search where distinctive features of the objects are represented as points in $\mathbb{R}^d$ [28, 30, 6, 7, 12, 11, 27, 10].

One of the earliest data structures proposed for this problem that is still the most commonly used is the kd-tree [3] that is essentially a hierarchical decomposition of space a long different dimensions. For low dimensions this structure can be used for answering nearest neighbor queries in logarithmic time and linear space. However the performance seems to degrade as a number of dimensions becomes larger than two. For high dimensions, the exact problem of nearest neighbor search seems to suffer from the curse of dimensionality that is either the running time or the space requirement grows exponentially in d. For instance Clarkson [5] makes use of $O(n^{\lceil d/2 \rceil(1+\delta)})$ space and achieves $O(2^{O(d \log d)} \log n)$ time. Meiser [24] obtains a query time of $O(d^5 \log n)$ but with $O(n^{d+\delta})$ space.

The situation is much of a better for finding an approximate solution whose distance from the query point is at most $1+\epsilon$ times its distance from the nearest neighbor. [2, 21, 18, 22]. Arya et. al. [2] use a variant of kd-trees that they call BDD-trees ((Balanced Box-Decomposition trees) that performs $(1+\epsilon)$-approximate nearest neighbor queries in time $O(d\lceil 1 + 6d/\epsilon \rceil^d \log n)$ and linear space.

For arbitrarily high dimensions, [22] provide an algorithm for finding an $(1+\epsilon)$-approximate nearest neighbor of a query point in time $\tilde{O}(d \log n)$ using a data structure of size $(nd)^{O(1/\epsilon^2)}$. Since the exponent of the space requirement grows as $1/\epsilon^2$, in practice this may be prohibitively expensive for small ϵ. Indeed,

*. E-mail: `rinap@cs.stanford.edu`.

since even a space complexity of $(nd)^2$ may be too large,perhaps it makes more sense to interpret these results as efficient, practical algorithms for c-approximate nearest neighbor where c is a constant greater than one.

Indyk and Motwani [18] provide results similar to those in [22] but use hashing to perform approximate nearest neighbor search. They provide an algorithm for finding the c-approximate nearest neighbor in time $\tilde{O}(d + n^{1/c})$ using an index of size $\tilde{O}(n^{1+1/c})$ (while their paper states a query time of $\tilde{O}(dn^{1/c})$, if d is large this can easily be converted to $\tilde{O}(d + n^{1/c})$ by dimension reduction). In their formulation, they use a locality sensitive hash function that maps points in the space to a discrete space where nearby points out likely to get hashed to the same value and far off points out likely to get hashed to different values. Precisely, given parameter m that denotes the probability that two points at most r apart hash to the same bucket and g the probability that two points more than cr apart hash to the same bucket, they show that such a hash function can find a c-approximate nearest neighbor in $\tilde{O}(d + n^{\rho})$ time using a data structure of size $\tilde{O}(n^{1+\rho})$ where $\rho = log(1/m)/log(1/g)$. Recently, Andoni and Indyk [1] obtained and improved locality sensitive hash function for the Euclidean norm resulting in a ρ value of $O(1/c^2)$. Lower bounds on the performance of locality sensitive hashing method were studied in [25].

However, to the best of our knowledge the most commonly used method in practice is the old kd-tree. We show that a simple modification to the search algorithm on a kd-tree can be used to find the nearest neighbor in high dimensions. The modification consists of simply perturbing the query point before traversing the tree, and repeating this for a few iterations. For a certain database of random points, we show that this modified algorithm can find the c-approximate nearest neighbor in $e^{O(d/c)}$ iterations with reasonable probability, while the traditional kd-tree succeeds only with probability $e^{-\Omega(d/c)}$. We then provide empirical evidence through simulations to show that the simple modification results in high probabilities of success in finding nearest neighbor search in high dimensions.

2 Review

2.1 Problem statement

Given a set S of n points in d-dimensions, construct a data structure that given a query finds the nearest (or a c-approximate) neighbor. A c approximate near neighbor is a point at distance at most c times the distance to the nearest neighbor. Alternatively it can be viewed as finding the exact nearest neighbor when the second nearest neighbor is more than c times the distance to the nearest neighbor.

We also work with the following decision version of the c-approximate nearest neighbor problem: given a query point and a parameter r indicating the distance to its nearest neighbor, find any neighbor of the query point that is that distance at most cr. We will refer to this decision version as the (r, cr)-nearest neighbor problem and a solution to this as a (r, cr)-nearest neighbor. It is well known that the reduction to the decision version adds only a logarithmic factor in the time and space complexity [18, 13]. Unless otherwise stated, we will use the l_2 norm in R^d.

Notations:

- $N(\mu, r), \eta(x)$: Let $N(\mu, r)$ denote the normal distribution with mean μ and variance r^2 with probability density function given by $\frac{1}{r\sqrt{2\pi}} e^{-(x-\mu)^2/(2r^2)}$. Let $\eta(x)$ denote the function $\frac{1}{\sqrt{2\pi}} e^{-x^2/2}$.

- $N^d(p, r)$: For the d-dimensional Euclidean space, for a point $p = (p_1, p_2, ..., p_d) \in \mathbb{R}^d$ let $N^d(p, r)$ denote the normal distribution in $\mathbb{R}^d$ around the point p where the ith coordinate of a random point has the normal distribution $N(p_i, r/\sqrt{d})$ with mean p_i and variance r^2/d. It is well known that this distribution is spherically symmetric around p. A point from this distribution is expected to be at root-mean squared distance r from p; in fact, for large d its distance from p is close to r with high probability (see for example lemma 6 in [18])

- $erf(x), \Phi(x)$: The well-known error function $erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-x^2} dx$, is equal to the probability that a random variable from $N(0, 1/\sqrt{2})$ lies between $-x$ and x. Let $\Phi(x) = \frac{1-erf(x/\sqrt{2})}{2}$. For $x \geq 0$, $\Phi(x)$ is the probability that a random variable from the distribution $N(0,1)$ is greater than x.

2.2 kd-Trees

Although many different flavors of kd-trees have been devised, their purpose is always to hierarchically decompose space into a relatively small number of cells such that no cell contains too many input objects. This provides a fast way to access any input object by position. We traverse down the hierarchy until we find the cell containing the object.

Typical algorithms construct kd-trees by partitioning point sets recursively along with different dimensions. Each node in the tree is defined by a plane through one of the dimensions that partitions the set of points into left/right (or up/down) sets, each with half the points of the parent node. These children are again partitioned into equal halves, using planes through a different dimension. Partitioning stops after $\log n$ levels, with each point in its own leaf cell. The partitioning loops through the different dimensions for the different levels of the tree, using the median point for the partition. kd-trees are known to work well in low dimensions but seem to fail as the number of dimensions increase beyond three.

2.3 BDD Trees

BDD-trees (Balanced Box-Decomposition trees) [2] are extensions of kd Trees with additional auxiliary data structures with the following guarantees on computing an $(1+\epsilon)$-approximate nearest neighbor.

Theorem 1 *The $(1+\epsilon)$ -approximate nearest neighbor among n points in R^d with l_p norm can be computed using a BDD-tree in time $O(d\lceil 1 + 6d/\epsilon \rceil^d \log n)$*

Just like k-d trees, BDD trees hierarchically decompose space into regions of O(d) complexity defined by axis aligned hyper rectangles with the following differences:

1. The rectangles are fat – that is, the ratio between the longest and shortest sides is bounded.
2. Space is recursively subdivided it into a collection of cells, each of which is either a d-dimensional rectangle or the set theoretic difference of the rectangles, one enclosed within the other.

Search Algorithm: The search algorithm starts just as in a k-d tree by a simple descent down the tree. Upon reaching a leaf node, we enumerate the leaf cells in increasing order of distance from the query point, where the distance to a cell is measured as the distance to the nearest point in the cell from the query point. Let p denote the closest points seen so far. As soon as the distance from the query point to the current cell exceeds $d(p,q)/(1+\epsilon)$ it follows that the search can be terminated and p can be reported as an approximate nearest neighbor to the query point. The enumeration of the cells in increasing distance from the query point can be performed by using an auxiliary heap.

Proof Intuition: They show that the number of cells visited in the search depends on d and ϵ, but not on n. To see the intuition, considered the last leaf cell to be visited that did not cause the algorithm to terminate. If we let r denote the distance from the query point q to the cell, and p denote the closed data point encountered so far, then because we do not terminate, we know that the distance from q to p is at least $r(1+\epsilon)$. We could not have seen a leaf cell of diameter less than $r\epsilon$ up to now, since the associated data point would necessarily be closer to q that p. This provides a lower bound on the sizes of these cells seen. The fact that cells are fat and a simple packing argument provide an upper bound on the number of cells encountered.

2.4 Locality Sensitive Hashing

Indyk and Motwani [18] use locality sensitive hashing to provide an algorithm for finding the c-approximate nearest neighbor in time $\tilde{O}(d+n^{1/c})$ using an index of size $\tilde{O}(n^{1+1/c})$ (while their paper states a query time of $\tilde{O}(dn^{1/c})$, if d is large this can easily be converted to $\tilde{O}(d+n^{1/c})$ by dimension reduction)

A locality sensitive hash function is a random function from a hash family that it maps points in the given space to a discrete space where nearby points are likely to get hashed to the same value and far off points out likely to get hashed to different values.

Definition 1 *A family $\mathcal{H}$ of hash functions from a metric space to a discrete range is called (r_1, r_2, m, g)-sensitive if for any two points p, q that are at most distance r_1 apart, $Pr_{h\in\mathcal{H}}[h(q) = h(p)] \leq m$, and for any two points p, q that are at least distance r_2 apart, $Pr_{h\in\mathcal{H}}[h(q) = h(p)] \geq g$,*

An (r, cr, m, g)-sensitive hash function can be used to compute the c-approximate nearest neighbor in time $\tilde{O}(d+n^{\rho})$ time using a data structure of size $\tilde{O}(n^{1+\rho})$ where $\rho = \log(1/m)/\log(1/g)$. We also need to assume that $\log n > \log(1/g)$.

The (r, cr)-nearest neighbor is obtained as follows: Pick $k = \log n/\log(1/g)$ random hash functions $h_1, h_2, .., h_k$. For each point p in the database compute $H(p) = (h_1(p), h_2(p), .., h_k(p))$ Store these values $H(p)$ for all points in a hash table so as to save space for the values that are absent. Construct n^{ρ} such hash tables independently. To search for the nearest neighbor of a query point q, for each hash table, compute $H(q)$ and examine all points that hashed to the same bucket limiting the search to at most 2 points.

Theorem 2 *[18] Given a (r, cr, m, g)-sensitive hash family with $\log n > \log(1/g)$, with probability at least $n^{-\rho}$ a query point q at distance at most r from its nearest neighbor p hashes to the same value as q; that is, $H(q) = H(p)$. The expected number of points at distance more than cr from the query point that hash to the same value as q is atmost* 1

Proof: Probability that the query point q hashes to the same value as p is at least $m^k = m^{\log n/\log(1/g)} = n^{-\log(1/m)/\log(1/g)} = n^{-\rho}$.

Probability that one point at distance more than cr from q hashes to $H(q)$ is at most $g^k = 1/n$. So out of n possible such points, number of points that hash to the same value as q is atmost 1 . □

The above theorem implies that with constant probability, the algorithm will succeed in finding the (r, cr)-nearest neighbor over the n^{ρ} hash tables. By using $O(\log n)$ instances of such a data structure the algorithm can be made to succeed with high probability.

The locality sensitive hash function [18, 9] is obtained by dividing the real line into equal sized intervals of a certain size D and adding a random shift. Precisely, the point p is hashed to an integer $h(p) = \lfloor f(p)/D \rfloor = \lfloor (p.v + \beta)/D \rfloor$ where v is a random variable from the distribution $N^d(0, 1)$ and β is a random number in $[0, D]$. $H(p) = (h_1(p), h_2(p), ..., h_k(p))$. Essentially $H(p)$ is obtained by dividing R^k into a grid of cubes of side length D after a random rotation and shift.

2.5 Hamming metric and L1 Norm

Kushilevitz et al [22] prove results very similar to those in [18] but with simplerdirect algorithms for the hamming metric that work well even if c is close to 1. Their result for the hamming metric easily extends to the l_1 or l_2 norms by distance preserving embeddings. Their main theorem for the Hamming metric can be summarized as follows.

Theorem 3 *There's a data structure of size $(nlogd)^{O(1/\epsilon^2)}$ can be used to compute $1+\epsilon$-nearest neighbor queries in a d-dimensional hypercube in time $\tilde{O}(d)$ with high probability.*

The main idea behind their algorithm for a hypercube is to design a separation test that distinguishes between distances smaller than l and larger than $l(1+\epsilon)$. The test is simple: for a d-dimensional hypercube, the test consists of a subset C of about sd coordinates where each coordinate is chosen with probability $s = \frac{1}{2l}$ and generating for each vector $v = (v_1, ..., v_d)$ a single bit $P(v) = \sum_{i \in C} r_i v_i (mod 2)$, where each r_i is a random bit.

Let p_1 the probability that to far-off points (distance greater than $l(1+\epsilon)$) hash to the same bit value) and p_2 denote the same probability for close by points (distance less than l). Then they prove that $p_2 - p_1 \geq O(\epsilon)$. The data structure is obtained by using $t = \tilde{O}(1/\epsilon^2)$ such bits producing a t-bit vector $T(v)$ for every point in the database. They then construct a table where each point v is recorded against $T(v)$ and all t-bit vectors close to $T(v)$; that is, at hamming distance less than $p_1 t$ from v.

3 Results

3.1 Trees vs Hashing for NNS

Here are some differences between tree based and hash based metods for nearest neighbor search.

1 Hashing uses one memory access per table vs trees that take $\log n$ memory accesses.

2 Trees have adaptive granularity depending on the density of the points. A locality sensitive hash function typically breaks up space into equal sized regions. So multiple hash functions need to be used for different granularities.

3 In trees the partitioning planes are chosen in a special way instead of randomly.

4 kd-trees are in common use (as far as I have heard).

3.2 New Search Algorithm on Kd trees

Although the kd tree is the most widely used data structure for nearest neighbor search in high dimension, as the number of dimensions increase the efficacy of finding the nearest neighbor drops. Consider for instance a random database of points chosen randomly and uniformly from $[0,1]^d$. Let r denote the distance from a random point in the database to its nearest point in the database. We first show that if the query point is chosen to be a random point at distance about r/c then the probability that a kdtree data structure results in the nearest neighbor is about $e^{-\Omega(d/c)}$.

We then propose a minor modification to the search algorithm so as to boost this probability to a large value. The modification is simple. Instead of searching for the query point in kdtree, perturb it randomly by distance r/c (precisely, from $N^d(p, r/c)$) and search for this perturbed point. Repeat this for $e^{O(d/c)}$ random perturbations and report the nearest neighbor found using the kd tree.

3.3 Analysis

Lemma 4 *Expected distance of any point to the nearest point in the database r is $\Theta(\sqrt{d}/n^{1/d})$*

Proof: The volume of a sphere of radius r in d dimensions is $\frac{2\pi^{d/2} r^d}{d\Gamma(d/2)} = \Theta(1)\frac{1}{d^{3/2}}(\frac{2e\pi}{d})^{d/2} r^d$. The radius at which the expected number of points that fall within the ball becomes one this fraction is about $1/n$, giving $r = \Theta(\frac{\sqrt{d}}{n^{1/d}})$ Note that even if the point is near the boundary of the cube then at least $1/2^d$ fraction of the sphere is inside the cube which does not change the value of r by more than a constant factor. The value of r is sharply concentrated around this value as the volume of the sphere changes dramatically with a change in the radius. High concentration bounds can be used to show that there must be at least one point for larger radii and almost no point for smaller radii. □

Distribution of kd-tree partition sizes: Consider a random point p in the database and the kd-tree traversal while searching p. The cell of p is specified by $\log n$ choices of left or right while traversing the kd tree. The number of decisions along each dimension is $\log n/d$. Focusing on the first dimension, let l_i denote length of the cell in the i^{th} branch along the first dimension. $l_0 = 1$. l_i can be thought of as to be obtained from the following process: Pick n random points in $[0, 1]$. Take the median point as a partition point and consider the left interval. Randomly sample $1/2^{d-1}$ fraction of the points. Now take the median again and repeat this process i times. The final interval size corresponds to l_i which is sharply concentrated around $1/2^i$ – more precisely, l_i is distributed as the inverse poisson distribution at rate $2^{(d-1)i}/n$ for $n/2^{di}$ arrivals. The distance x_i been p and the dividing plane at the i^{th} level is a random value between 0 and l_i. At the leaf level x_i is distributed by the exponential distribution with rate $2^{(d-1)\log n/d}/n = 1/n^{1/d}$

Lemma 5 *If we pick a random query point q at distance r/c from a random point in the database then for large dimensions the probability that a kd search returns the nearest neighbor is about $e^{-\Omega(d/c)}$*

Proof: Let us focus on a leaf cell containing the point p . What is the probability that a random point q chosen from $N(p, r/c)$ lies within the same cell? The cell has $2d$ faces. The distance of each face from the point p is approximately given by the exponential distribution with rate $1/n^{1/d}$. The probability that the query point q , lies on the p side of a given face is about $1 - \Omega(1/c)$. The probability that this happens for all the d directions is about $(1 - \Omega(1/c))^d = e^{-\Omega(d/c)}$. □

Theorem 6 *With probability at least $\Omega(c/d)$, the new search algorithm finds the nearest neighbor in $e^{O(d/c)}$ iterations for the random database and the random query point.*

The proof of this theorem follows from the following two lemmas.

Guessing the value of a random variable: If a random variable takes one of N discrete values with equal probability then a simple coupon collection based argument shows that if we guess N random values at least one of them should hit the correct value with constant probability. The following lemma taken from [26] states the required number of samples for arbitrary random variables so as to 'hit' a given random value of the variable. It essentially states how many guesses are required to guess the value of a random variable.

Lemma 7 *[26] Given an random instance x of a discrete random variable with a certain distribution D with entropy I, if $O(2^I)$ random samples are chosen from this distribution at least one of them is equal to x with probability at least $\Omega(1/I)$.*

Proof: Let $w_1, w_2, ..., w_N$ denote the probability distribution D of the discrete space.

After $s = 4.(2^I + 1)$ samples the probability that x is chosen is $\sum_i w_i[1 - (1 - w_i)^s]$.

If $w_i \geq 1/s$ then the term is at least $w_i(1 - 1/e)$. So if all the $w_i's$ that are at least $1/s$ add up to at least $1/I$ then the above sum is at least $\Omega(1/I)$. Otherwise we have a collection of $w_i's$ each of which is at most $1/s$ and they together add up to more than $1 - 1/I$.

But then by paying attention to these probabilities we see that the entropy $I = \sum_i w_i \log(1/w_i) \geq \sum_i w_i \log s \geq (1 - 1/I) \log s \geq (1 - 1/I)(I + 2) = I + 1 - 2/I$. For $I \geq 4$, this is strictly greater than I, which is a contradiction. If $I < 4$ then the largest w_i must be at least $1/16$ as otherwise a similar argument shows that $I = \sum_i w_i \log(1/w_i) > w_i \log 16 = 4$, a contradiction; so in this case even one sample guesses x with constant probability.

Table 1: Simulation Results: The entries indicate the percentage of times the nearest neighbor os found. As the number of iterations k is increased the success rate increases.

d	c	Kd tree	5 iter	15 iter	20 iter	25 iter	30 iter
3	4	84	96.1	98.8	99.3	99.3	99.8
3	2	73.9	89.5	97.4	98.4	99.0	98.7
3	4/3	73	88.5	96	96.6	98.7	98.7
5	4	73.6	91	97.5	98.1	98.5	99.3
5	2	54	78	92.1	94.9	94.4	96.2
5	4/3	50.7	71.3	87	91.2	92.3	94
10	4	60.7	80.5	94.8	96.6	96.7	96.8
10	2	36	56.4	77.6	84.3	86.6	88.4
10	4/3	25	43.7	61	70	73.4	75.6

$\square$

Let p be a random datebase point and q is a random point with distribution $N^d(p, r/c)$. Let $L(p)$ denote the leaf of the kd-tree where p resides. We will estimate $I[L(p)|q,T]$.

Lemma 8 $I[L(p)|q,T] = O(d/c)$

Proof:

The cell of p is specified by $\log n$ choices of left or right while traversing the kd tree. The number of decisions along each dimension is $\log n/d$. Let b_{ij} $(i \in 1..\log n/d, j \in 0..d-1)$ denote this choice in the i^{th} branch along dimension j.

Then $I[L(p)|q,T] \leq \sum_{i,j} I[b_{ij}|q,T]$

Focusing on the first dimension for simplicty, for the i^{th} choice in the first dimension, the distance x_i between p and the deciding plane is close to uniform in the range $[0, 1/2^i]$.

The distance between q and p along the dimension is given by $N(0, O(\frac{1}{cn^{1/d}})$.

Since, $I[b_{i0}|x_i = x] = I(\Phi(x), 1 - \Phi(x))$, $I[b_{i0}|d_i(p) = x]$ is negligible unless x is $O(1/c)$. This happens with probability $O(\frac{1}{c2^i})$. So $I[b_{i0}|q,T] = O(\frac{1}{c2^i})$.

Similarly bounding and summing over all dimensions, $I[L(p)|q,T] \leq \sum_{i,j} I[b_{ij}|q,T] = O(d/c)$ $\square$

3.4 Experiments

We perform experiments with our modified algorithm for different dimensions d, with different number of iterations k for different values of c. The results are summarized in table 3.4. We use of the GNU implementation of kd trees. We vary the number of dimensions from 3 to 10 and the approximation constant c from 4/3 to 4. In 3 dimensions for $c = 4$ kd-trees report the nearest neighbor 84% of the time whereas even with 5 iterations of the new algorithm this goes up to 96%. In 10 dimensions for $c = 4$, 15 iterations raises the success rate from 60% to about 95%.

References

[1] Alexandr Andoni, Piotr Indyk, Near-Optimal Hashing Algorithms for Near Neighbor Problem in High Dimensions. In Proceedings of the Symposium on Foundations of Computer Science (FOCS'06), 2006.

[2] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. In Proc. 5th ACM-SIAM Sympos. Discrete Algorithms, pages 573–582, 1994.

[3] J. L. Bentley, J. H. Friedman, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. ACM Transactions on Mathematical Software, 3(3):209–226, 1977.

[4] A. Borodin, R. Ostrovsky, and Y. Rabani. Lower bounds for high dimensional nearest neighbor search and related problems. In Proceedings of the 31st ACM Symposium on Theory of Computing, pages 312–321, 1999.

[5] K. L. Clarkson. Nearest neighbor queries in metric spaces. In Proceedings of the 29th Annual ACM Symposium on Theory of Computing (May 1997), pp. 609–617.

[6] T. Cover, P. Hart, "Nearest neighbor pattern classification," IEEE Trans. Information Theory IT-13(1967), pp. 21-27.

[7] S. Deerwester, S. T. Dumais, T.K. Landauer, G.W. Furnas, and R.A. Harshman. Indexing by latent semantic analysis, Journal of the Society for Information Science, 41(6), 391-407, 1990.

[8] Danny Dolev, Yuval Harari, and Michael Parnas. Finding the neighborhood of a query in a dictionary. In Proc. 2nd Israel Symposium on Theory of Computing and Systems, pages 33–42, 1993.

[9] M. Datar, N. Immorlica, P. Indyk and V. Mirrokni. Locality-Sensitive Hashing Scheme Based on p-Stable Distributions. In Proceedings of the Symposium on Computational Geometry, 2004. Talk is available at http://theory.lcs.mit.edu/ indyk/brown.ps

[10] L. Devroye and T.J. Wagner. Nearest neighbor methods in discrimination. Handbook of Statistics, volume 2, P.R. Krishnaiah and L.N. Kanal, editors, North-Holland, 1982.

[11] R. Fagin, Fuzzy Queries in Multimedia Database Systems, Proc. ACM Symposium on Principles of Database Systems (1998), pp. 1–10

[12] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, P. Yanker. Query by image and video content: The QBIC system. Computer 28 (1995) 23–32

[13] S. Har-Peled. A replacement for voronoi diagrams of near linear size. Proceedings of the Symposium on Foundations of Computer Science, 2001.

[14] P. Indyk. High-dimensional computational geometry. Dept. of Comput. Sci., Stanford Univ., 2001.

[15] Piotr Indyk, Approximate Nearest Neighbor under Frechet Distance via Product Metrics, ACM Symposium on Computational Geometry, 2002

[16] Piotr Indyk. Nearest neighbors in high-dimensional spaces. In Jacob E. Goodman and Joseph O'Rourke, editors, Handbook of Discrete and Computational Geometry, chapter 39. CRC Press, 2rd edition, 2004.

[17] P. Indyk, R. Motwani, P. Raghavan, and S. Vempala. Locality-preserving hashing in multidimensional spaces. In Proceedings of the 29th ACM Symposium on Theory of Computing, pages 618–625, 1997.

[18] P. Indyk and R. Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. Proc. 30th Symposium on Theory of Computing, 1998, pp. 604–613.

[19] Piotr Indyk and Nitin Thaper, Embedding Earth-Mover Distance into the Euclidean space, manuscript, 2001.

[20] T. S. Jayram, S. Khot, R. Kumar, and Y. Rabani. Cell-probe lower bounds for the partial match problem. In Proc. 35th Annu. ACM Symp. Theory Comput., pages 667–672, 2003.

[21] J. Kleinberg. Two algorithms for nearest-neighbor search in high dimension. Proc. 29th Annu. ACM Sympos. Theory Comput., pp. 599–608. 1997.

[22] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. In Proc. of 30th STOC, pp. 614–623, 1998.

[23] N. Linial and O. Sasson, Non-Expansive Hashing, In Proc. 28th STOC (1996), pp. 509–517.

[24] S. Meiser. Point location in arrangements of hyperplanes. Information and Computation, 106(2):286–303, 1993.

[25] Rajeev Motwani, Assaf Naor, and Rina Panigrahy. Lower Bounds on Locality Sensitive Hashing. In Proceedings of the 22nd Annual ACM Symposium on Computational Geometry 2006

[26] R. Panigrahy. Entropy based nearest neighbor search in high dimensions. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm (Miami, FL, 2006)*, pages 1186–1195. ACM Press, New York, NY, 2006.

[27] A. Pentland, R. W. Picard, and S. Sclaroff. Photobook: Tools for content-based manipulation of image databases. In Proceedings of the SPIE Conference On Storage and Retrieval of Video and Image Databases (February 1994), vol. 2185, pp. 34–47.

[28] C. J. van Rijsbergen. Information Retrieval. Butterworths, London, United Kingdom, 1990.

[29] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. Theory Probab. Appl., 16:264–280, 1971.

[30] G. Salton. Automatic Text Processing. Reading, MA: Addison-Wesley, 1989