

An Algorithm and Hardware Design for Very Fast Similarity Search in High Dimension

Vishwakarma Singh
Department of Computer Science, UCSB
vsingh@cs.ucsb.edu

Wenyu Jiang
Dolby Laboratories Inc.
wzj@dolby.com

Abstract—Similarity search in very high dimensions is vital for many scientific research activities as well as real applications. A high performance, scalable, and optimal solution to the problem still remains challenging. We¹ propose a vote count based algorithm using p -stable distribution for approximate similarity search. Approximate similarity search effectively serves purpose for many real applications. Our algorithm is efficient and scalable with both dimension and database size. We also propose a novel hardware implementation of the algorithm using simple modification to Random Access Memory(RAM). The hardware design gives real time search for millions of points at practical cost. We empirically achieve high accuracy for query results using our algorithm on both synthetic and real datasets.

I. INTRODUCTION

Similarity search is the process of finding the most similar object (image, document, or graph) for a given query object in a collection of objects (called database objects). The most commonly used model for similarity search is to characterize the objects by a collection of features having numerically measurable values. All the objects including the query object are represented as points in a high-dimensional attribute space \mathbf{R}^d . Similarity between the query and a given object is defined as a distance function of their feature vectors. L_p -norm metric particularly L_2 is one of the most commonly used function. The most similar object is defined as the object having the least distance from the query. This process is called Nearest Neighbor search and works as a model to find the most similar object. It should be noted that the nearest neighbor may not always be a meaningful similar result, e.g. nearest neighbor of a query image in a dataset of images may look visually different, if such nearest distance is too far. The features should also be perceptually relevant in order for the nearest neighbor match to be perceptually similar. Here we also adopt the nearest neighbor search model for solving similarity search.

Nearest neighbor (NN) search problem has come to be one of the most fundamental problem since it was proposed by Donald Knuth as Nearest Post-office Problem [1]. Formally, NN of a query q in a dataset D of points in d -dimensional space is the point o' such that there is no $o \in D$ satisfying $\|o, q\| < \|o', q\|$, where $\|\cdot, \cdot\|$ denotes the distance between the points. NN search is indispensable to a variety of applications: databases and data mining [2], [3], [4], content based image and video retrieval [5], [6], machine learning [7], and pattern

recognition [8]. Therefore, there is always the need for an efficient, scalable, and real time solution that produces high quality results for the NN search problem.

In order to achieve high performance in NN search, multi-dimensional index structures are applied to the feature vectors. An indexing scheme should provide accurate solution, should be space, and time efficient, and should scale with dimension of the feature vectors as well as the database size. There are many well known index structures for effectively solving the NN search in low dimensions (less than 10). These indices are either space partitioning (e.g. Voronoi diagram [9], KD-tree [10]), data clustering in vector space (e.g. R-Tree [11]), or data clustering in metric space (e.g. M-Tree [12]). Weber et al. [13] showed that these methods degrade to linear search for sufficiently large dimensions making them unsuitable for large scale data. Many applications have features with number of dimensions ranging from tens to thousands. For example, SIFT [14] feature vector which is commonly used in content based image and video retrieval has 128 dimensions. Dimensionality reduction techniques [15], e.g. principal component analysis, fourier transform, wavelet transform, locally linear embedding [16], and locality preserving projections [17] are commonly applied to reduce the dimensions of features and then, existing indices are used to extract performance. But, there are many applications which inherently have very high intrinsic dimensionality, e.g. similar document retrieval and chemical molecular structure, leaving dimensionality reduction method insufficient.

The state of the art technique for high dimensional NN search is an approximate technique called Locality Sensitive Hashing (LSH) [18]. This method provides time and space efficiency and scalability with dimension and dataset size at the loss of result quality. It is widely argued that approximate NN already fulfills need of many application [18]. LSH is designed to solve (R, ϵ) -neighbor problem. It determines whether there exists a point o within a fixed distance R of query q , or whether all points in the database are at least a distance $(1 + \epsilon)R$ away from q . In the first case, the algorithm is required to return a point o within distance at most $(1 + \epsilon)R$ from q . This solution is then extended to solve ϵ -approximate NN search. Formally, a point o is a $(1 + \epsilon)$ -approximate NN of q if its distance to q is at most $(1 + \epsilon)$ times the distance from q to its exact NN o' where $\epsilon > 0$ and $(1 + \epsilon)$ is the approximation ratio.

The key idea in LSH is to use a family of hash functions

¹Both authors have contributed equally to the work.

to hash points in the database. These locality sensitive hash functions has the property that the nearby objects have higher probability of hashing in the same bucket than those which are far apart. One can determine near neighbors by hashing the query point and retrieving elements stored in the bucket containing it. Many families of hash functions have been developed [19], [18], [20], [21], [22] for NN search. The most commonly used family is the p -stable distribution based hash functions proposed by Datar et al. [21].

In this paper, we also use p -stable distributions discussed in Section III to design an algorithm for approximate NN search specifically for L_2 metric. Our NN search algorithm is different from the one proposed by Datar et al. [21] using the same p -stable distributions. We exploit the fundamental property of p -stable distribution to discriminate a nearest neighbor from a farther neighbor. We use frequency count of the co-occurrence of a query point with the database points as described in Section IV to find probable nearest candidates, where only database points with a frequency count higher than a certain threshold are listed as candidates. Then, we compute actual distance from query to each candidate to find the nearest neighbor. Simplicity of our method helps us to design a novel hardware for performing real time NN search which is discussed in Section V. If the configuration parameters are carefully chosen, the search time can be made roughly independent of database size. We use a Random Access Memory (RAM) like architecture with Counters and Comparators for hardware implementation of algorithm at very low production cost. Currently, we are unable to fabricate the hardware and report empirical results on it because of lack of resource. However, to corroborate our algorithm, we implement our algorithm in software to measure its accuracy in finding the true nearest neighbors as described in Section VI. We achieve high accuracy for both real and synthetic datasets. We also discuss the space requirement and query time performance of our algorithm.

We foresee far reaching impact of our work in various scientific domains, e.g. information retrieval, chemoinformatics, and media retrieval. Availability of truly real time technique for NN search will revolutionize research in many of the scientific domains. Our main contributions are as follows:

- A new vote count based algorithm using p -stable distribution for approximate NN search.
- A novel hardware design of the algorithm for real time search.

II. RELATED WORK

Nearest Neighbor search is well solved for low dimensional (less than 10) data. A good survey of the methods can be found in Gaede et al. [23] and Hanan et al. [24]. All the search methods proposed in the literature can be categorized into two major categories: Space partitioning and Data Clustering. Berchtold et al. [25] proposed partitioning the space using Voronoi diagram method and answering the query by searching the cell in which the query lies. Space partitioning trees like KD-Tree [10] recursively partitions the space on different

dimensions and provides algorithm for search on the tree. Data clustering technique like R-Tree [11] and M-Tree [12] encloses relatively near points in Minimum Bounding Rectangle or Spheres and recursively builds a tree structure. Weber et al. [13] proposed VA-file based on quantization of each dimensions to compress the dataset and minimize sequential search cost. Jagadish proposed iDistance [26] which projects high-dimensional data into one dimension and constructs B-Tree for NN search. Techniques like space filling curve [27], principal component analysis, fourier transform, wavelet transform, and locally linear embedding [16] are used to project data into low dimensions and existing techniques are used for multidimensional access.

NN search in high dimensional data are solved efficiently only for approximate cases [18], [28]. Locality sensitive hashing (LSH) proposed by Indyk et al. [19] provides sub-linear search time and a probabilistic bound on the quality of the results for approximate NN search. Various LSH families [19], [18], [20], [21], [22] have been proposed in the literature. Improvements were proposed on the basic LSH algorithm by Lv et al. [29] and Dong et al. [30].

III. p -STABLE DISTRIBUTION

In this section, we describe p -stable distribution which is also discussed by Datar et al. [21]. Stable distributions are defined as the limits of normalized sums of iid variables. Formally, a distribution \mathcal{D} of a random variable \mathcal{X} over \mathcal{R} is called p -stable if there exists $p \geq 0$ such that for any n real numbers $v_1 \cdots v_n$ and iid variables $\mathcal{X}_1 \cdots \mathcal{X}_n$ with distribution \mathcal{D} , the random variable $\sum_i v_i \mathcal{X}_i$ has the same distribution as the variable $(\sum_i |v_i|^p)^{\frac{1}{p}} \mathcal{X}$, where \mathcal{X} is a random variable with distribution \mathcal{D} . Stable distribution exists for any $p \in (0, 2]$. A gaussian distribution $g(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$ is 2-stable.

IV. VOTE COUNT ALGORITHM

p -stable distribution property can be used to design an effective and efficient solution for NN search. If we take a vector \mathbf{a} of dimension d whose each entry is taken independently from a p -stable distribution \mathcal{D} . Then, given a vector \mathbf{v} of dimension d , the dot product $\mathbf{a} \cdot \mathbf{v}$ is a random variable which is distributed as $(\sum_i |v_i|^p)^{\frac{1}{p}} \mathcal{X}$ where \mathcal{X} is random variable having distribution \mathcal{D} . If we are given two vectors $(\mathbf{v}_1, \mathbf{v}_2)$, then from p -stability it follows that the distance between their projections $\mathbf{a} \cdot \mathbf{v}_1 - \mathbf{a} \cdot \mathbf{v}_2 = (\mathbf{v}_1 - \mathbf{v}_2) \cdot \mathbf{a}$ is distributed as $\|\mathbf{v}_1 - \mathbf{v}_2\|_p \mathcal{X}$.

$$\mathbf{a} \cdot \mathbf{v}_1 - \mathbf{a} \cdot \mathbf{v}_2 = \|\mathbf{v}_1 - \mathbf{v}_2\|_p \mathcal{X} \quad (1)$$

In this paper we specifically deal with L_2 norm as measure of distance and therefor, \mathcal{X} is a normal distribution and $\|\mathbf{v}_1 - \mathbf{v}_2\|$ represents the L_2 Euclidean distance between the two vectors \mathbf{v}_1 and \mathbf{v}_2 .

If \mathcal{X} follows a gaussian distribution $\mathcal{D} = \mathcal{N}(\mu, \sigma)$, it is normally distributed, and $a\mathcal{X}$ would have distribution $\mathcal{D}' = \mathcal{N}(a\mu, a^2\sigma^2)$. We take $\mu=0$ and $\sigma=1$ for the purpose of simplicity. Then $a\mathcal{X}$ has distribution $\mathcal{N}(0, a^2)$ where a

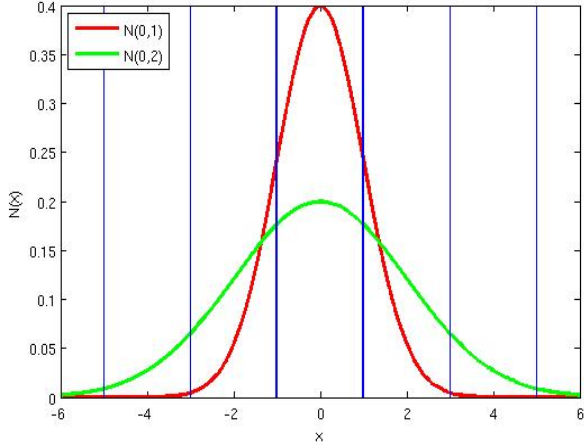


Fig. 1. Two Gaussian distribution with mean $\mu=0$ and standard deviation $r_1=1$ and $r_2=2$ respectively. We can see that distribution with variance 4 is flatter than distribution with variance 1 and has less mass in the bin around mean.

is the standard deviation, which describes how concentrated the distribution is around its mean. This is also called scale characteristic of the distribution and determines the shape of the curve. A distribution curve having large scale is flatter than the curve having small scale. We exploit this property to design our algorithm.

$$a\mathcal{X} = a\mathcal{N}(0, 1) = \mathcal{N}(0, a^2) \quad (2)$$

If we have a query vector \mathbf{q} and its neighbors \mathbf{o}_1 and \mathbf{o}_2 in a d -dimensional space such that $\|\mathbf{q} - \mathbf{o}_1\| = r_1 < \|\mathbf{q} - \mathbf{o}_2\| = r_2$. Then, for random vector \mathbf{a} whose each component is generated from a normal distribution $\mathcal{N}(0, 1)$ we get $\mathbf{a} \cdot \mathbf{q} - \mathbf{a} \cdot \mathbf{o}_1 = (\mathbf{q} - \mathbf{o}_1) \cdot \mathbf{a} = \|\mathbf{q} - \mathbf{o}_1\| \mathcal{N}(0, 1) = \mathcal{N}(0, r_1^2)$ using p -stable Equation 1 and gaussian distribution Equation 2. Similarly, $\mathbf{a} \cdot \mathbf{q} - \mathbf{a} \cdot \mathbf{o}_2 = \mathcal{N}(0, r_2^2)$. Since $r_1 < r_2$, the shape of distribution $\mathbf{a} \cdot \mathbf{q} - \mathbf{a} \cdot \mathbf{o}_2$ is flatter than the shape of distribution $\mathbf{a} \cdot \mathbf{q} - \mathbf{a} \cdot \mathbf{o}_1$ as shown in Figure 1. If we divide each distribution into equal number of bins, its obvious that the probability mass around mean of $\mathbf{a} \cdot \mathbf{q} - \mathbf{a} \cdot \mathbf{o}_1$ is greater than $\mathbf{a} \cdot \mathbf{q} - \mathbf{a} \cdot \mathbf{o}_2$. This difference is useful for distinguishing a nearer neighbor \mathbf{o}_1 of \mathbf{q} from a farther neighbor \mathbf{o}_2 .

Now we describe our search algorithm called **vote count**. We generate a random vector \mathbf{a} . Each component of \mathbf{a} is randomly picked from normal distribution $\mathcal{N}(0, 1)$. We compute dot product $\mathbf{a} \cdot \mathbf{o}$ of each point \mathbf{o} with the random vector \mathbf{a} . We divide the dot product values of all the points in the dataset into B bins. Bin id of a point \mathbf{o} is given by $\lfloor (\mathbf{a} \cdot \mathbf{o} - \min(\mathbf{a} \cdot \mathbf{o}) / W) \rfloor$ where W is bin-width. We get W using $(\max(\mathbf{a} \cdot \mathbf{o}) - \min(\mathbf{a} \cdot \mathbf{o})) / B$. We repeat the same process for L random vectors. This generates L different bin ids for each \mathbf{o} . We process a query by computing its bin id on each \mathbf{a}_i where $1 \leq i \leq L$. We maintain a counter for each \mathbf{o} . For each \mathbf{a}_i , we increment the counter of a point if it has the same bin id as the query. Here, we are counting the number of times a database point is hashed in the same bin as a query on L random vectors. A database point can have a maximum of L votes. Database point having the k th highest vote is

reported as the k th probable nearest neighbor. For many types of high dimensional real data, such as in video fingerprint identification, a query usually has only one point very near to it and the remaining points are far. Therefore, we are interested in finding only the top-1 neighbor.

We measure the quality of search by the percentage of queries for which our algorithm retrieves the true top-1 NN. This is called accuracy. Obtaining high search accuracy just by counting votes requires large number of random vectors. This increases space cost. Our goal is to achieve high accuracy for NN search with minimal number of random vectors. Our algorithm may yield large false positives for small number of random vectors. Therefore, our vote ranking alone does not always find the true NN. We adopt the following strategy to obtain high result accuracy. We choose all the database points as candidates whose vote count is greater than or equal to a pre-learned threshold for a given L . Threshold is specified as a percentage T votes of L . Finally, we perform a linear search on all the candidates to obtain the top-1 NN.

We apply a heuristic called *partial distance pruning* to speed up the linear search over candidates. For L_2 distance metric, we avoid computation of the square root by comparing distances of points o and o' from query q by using distance $l = \sum_{1 \leq i \leq d} (q_i - o_i)^2$ where d is the number of dimensions. Further, if l^* is the distance of the nearest point found till now during linear search then the full distance computation of the next point o' can be avoided if the partial distance $l' = \sum_{1 \leq i \leq j} (q_i - o'_i)^2 \geq l^*$ where $j < d$. This helps to obtain sub-linear search time over candidates.

Due to limited space, it suffices to note that a key distinction of the vote count algorithm from LSH is that it exploits the difference in binomial distributions of the vote count between true NN and other points, whereas LSH utilizes concatenation of hash values to increase precision and subsequently multiple hash tables to compensate for the decrease in retrieval rate due to concatenation.

Space Analysis: We assume that all the vectors are stored in memory for fast linear search over candidates. Size of the data space is $N * d * b$ bits where N is the number of points in the database, d is the number of dimension, and b is the number of bits required to store the data type in each dimension. Size of the index space is $L * N * \lceil \log_2(B) \rceil$ bits. $\lceil \log_2(B) \rceil$ is the number of bits required for storing bin id of each point where B is the number of bins.

V. HARDWARE DESIGN

With the vote count algorithm, if $N = 1$ million, and $B = 2$, then on average 0.5 million points will hash into the same bin as the query point on one random vector. Updating 0.5 million counters in software is clearly neither scalable nor desirable. Fortunately, this can be efficiently implemented in hardware with moderate additional cost, by modifying Random Access Memory (RAM) circuit, in particular Dynamic RAM (DRAM). DRAM generally has a 1-transistor 1-capacitor (1T1C) structure. The charge and the voltage of

the capacitor depend on the stored value (0 or 1), and thus controls the read out during a DRAM read access.

An interesting property of DRAM (and also true for many other RAM circuits) is that when a single cell is being read, the values of all cells in that row on the same *matrix grid* (often referred to as a *DRAM macro*) become available simultaneously. In DRAM, each column has its own sense-amplifier for sensing and reading the bit value stored in the cell belonging to that column. The output of the sense-amplifier is further sent to a latch or register for temporary storage. The sense-amplifier and latch/register are shared across rows to reduce the overhead of such circuitry.

If $B=2$, then bin id can be represented by 1 bit, or 1 cell in DRAM. If we store the bin id of each database point in a separate column, and place them on the same row for the same random vector, then we can read one entire row, compare each column's output value to the bin id of the query point, and if equal, we increment a counter associated with that database point (i.e., that column). This forms our basic hardware architecture implementing the vote count algorithm.

Therefore, when $B=2$, for a database of N points, we would create a DRAM circuit with L rows and N columns, with each row corresponding to one random vector. Since the sense-amplifier and its associated latch or register at the end of each column holds the cell value, we can aptly place a (equality) comparator right after the latch/register, and then use the comparator's logic output to control whether to increment the counter associated with that column. This architecture is illustrated in Figure 2.

In practice, a DRAM circuit with N columns (where N is very large) will cause the DRAM row access time to significantly increase due to intrinsic capacitance in the wordline (row) circuit (capacitance $\propto N$). Instead, observing that a DRAM memory card consists of multiple DRAM chips, which in turn consists of multiple DRAM macros, we simply need to read the same row on all DRAM memory cards responsible for the vote count procedure, which then read the same row on all corresponding DRAM chips and corresponding DRAM macros, respectively, all simultaneously.

After reading all L rows, the counters in each column will be ready for comparison with the threshold value $T * L$,² and if $\geq T * L$, the point will be reported. If $>$ semantics instead of \geq is desired, the threshold can be decreased by 1 and we can keep using the same \geq logic circuit. When multiple columns satisfy this test, instead of reporting all of them simultaneously, one may simplify circuit design by using a priority encoder circuit [31] to select the column with the highest vote (also with some tie-breaking logic), and report the vote as well as the column number. It should then clear the selected column so as to select the column with second highest vote. The process repeats until all satisfied columns are reported. This is illustrated at the bottom of Figure 2.

If $B > 2$, a bin id requires $\lceil \log_2(B) \rceil > 1$ bits, and one may either spread the $\lceil \log_2(B) \rceil$ bits into $\lceil \log_2(B) \rceil$ columns,

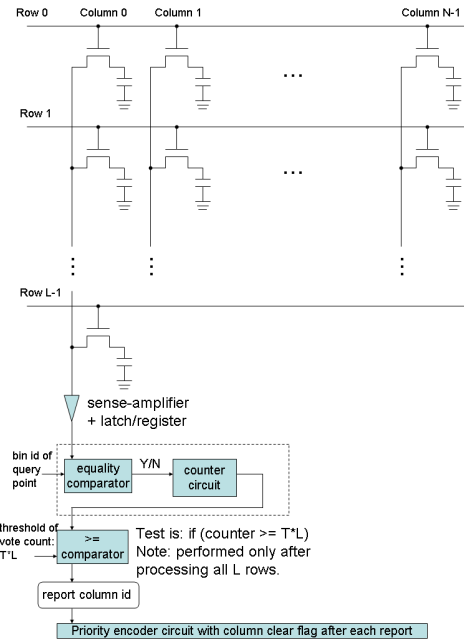


Fig. 2. Modifying DRAM circuit to implement vote count

or $\lceil \log_2(B) \rceil$ rows, or a mixture of both. The first option is faster in running time because only L rows need to be read, whereas the second option requires reading $L * \lceil \log_2(B) \rceil$ rows. However, the second option is more space and cost efficient because the equality comparator, counter, and \geq comparator circuits are shared across more rows in a column.

A. Hardware Cost Estimation and Practical Issues

The additional cost of the proposed hardware (compared to DRAM) lies mainly in the extra transistors for implementing the vote count logic. If $B=2$, the equality comparator is simply an XOR gate (6 transistors or 6T), a counter of $C = \lceil \log_2(L) \rceil$ bits requires $14C$ transistors, and a \geq comparator takes about $20C$ transistors, and a 1-bit register to hold the threshold comparison output (11T), and priority encoder takes about 10T per input. These counts are based on Leonides [32]. For $L=100$, $C = \lceil \log_2(L) \rceil = 7$, column-wise vote count overhead is about $6 + 14 * 7 + 20 * 7 + 11 + 10 = 265$ transistors. If we add sense-amplifier (20T) and latch (10T), total column-wise overhead is about 295T, or almost 3 times the transistor count of DRAM cells (since $L=100$). This implies such hardware to be almost 4 times as expensive as conventional DRAM.

Fortunately, we can reduce the overhead significantly, by storing the counter bits in DRAM cells and by implementing a serial \geq comparator requiring C clock cycles instead of a parallel comparator requiring 1 clock cycle. This is illustrated in Figure 3 for $B = 2$, with detailed logic circuit design and transistor count (totally 57T). Adding priority encoder and sense-amplifier and latch/register, column-wise overhead now becomes $57 + 10 + 30 = 97T$ compared to the previous 295T. Although still non-negligible, such a circuit would now likely cost only twice as much as conventional DRAM for $L=100$. Similarly, for $B > 2$, a serial equality comparator requiring $\lceil \log_2(B) \rceil$ cycles can be used to save on transistors.

²One may use ceiling, floor, or round function to convert $T * L$ to an integer

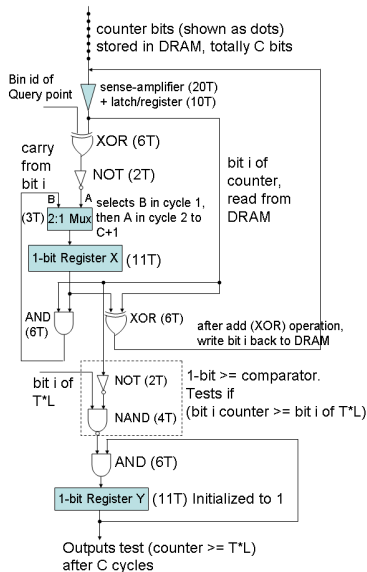


Fig. 3. Reducing vote count logic overhead by storing counter bits on DRAM circuit and using serial comparator

With the reduced overhead implementation, the time complexity of vote count procedure is $(1+C)*L+C$ cycles, where 1 clock cycle is the random access cycle in DRAM, which is typically around 45ns in today’s implementation. For $B > 2$, the time becomes $(\lceil \log_2(B) \rceil + C) * L + C$ cycles, assuming the bin id is spread to $\lceil \log_2(B) \rceil$ rows but kept in the same column. So for $L=100$, $B=2$, the vote count procedure will have all results ready after about $(1 + 7) * 100 + 8$ cycles = $808 * 45\text{ns} = 36.36\mu\text{s}$. Then, the priority encoder circuit will output the matches one at a time. Each reported match can be further verified by computing their actual Euclidean distance to the query point, to filter out false matches.

VI. EMPIRICAL EVALUATION

A good search algorithm should be space and time efficient and should yield high quality results. Hardware design of our algorithm addresses time efficiency. We performed experiments in software on a synthetic and a real dataset to show that our method yields high quality results at manageable space cost. Important parameters for our algorithm is the number of random vectors L , number of bins B , and threshold T . An exhaustive behavior of our algorithm can be studied with a dataset that has queries which have their first nearest neighbor at varying distance R and the second nearest neighbor is at distance $R+\Delta r$ for varying Δr . We generated 128 dimensional synthetic datasets in a controlled way to perform the study. We randomly chose a query point and generated its two nearest neighbors at distance R and $R+\Delta r$ respectively. For a given value of R and Δr , we generated 10,000 random queries which gave us a dataset of 20,000 points. We also made sure that any pair of queries are at least $2 * (R+\Delta r)$ far apart to maintain the correctness of the order of the nearest neighbors for each query. We used 5 different values for both R and Δr . Thus, we had 25 different synthetic datasets each of size 20,000. We performed experiments for each dataset separately for varying L , B , and T . We averaged the result

over all the 10,000 queries for each dataset. Final result is reported by averaging the results over all the 25 datasets for a set of (L, B, T) . Our real dataset consists of 1 million SIFT feature vectors of dimension 128 extracted from random images which are downloaded from the web. We randomly pick 1,000 queries and report results which are averaged over all the queries.

We report following measurements in the paper: (1) Result accuracy, (2) Max Vote : This is the maximum vote among all database points with respect to a query point, (3) True NN Vote: This is the vote obtained by top-1 NN, and (4) Candidates: This the percentage of database points selected after applying the threshold T for a given B . Accuracy is computed by comparing against ground truth top-1 NN (as opposed to the $1 + \epsilon$ approximation criterion in LSH) obtained by linear search over all the points. All measurements above are averaged over all experiments (i.e., all query points and for synthetic datasets all combinations of R and Δr).

Results for real data is shown in Table I for different parameters. We can see from Table I(a) that for $(T=70\%, B=2, L=100)$ we obtained accuracy of 95% with 1.14% candidates. Accuracy increased with larger L but the size of the candidates decreased. Decrease in the candidate size is because as L increases false positives decrease continuously which is the expected behavior. Results for $(T=65\%, B=2)$ in Table I(b) show that we achieved 99.1% accuracy with 5.4% candidates for $L=75$, and 97.9% accuracy with 6.4% candidates for $L=50$ respectively. Increasing the number of bins to $B = 4$ produced 97% accuracy for $L=100$ and $T=40\%$ as seen in Table I(c). Here candidate size decreases with increase in B but it leads to increase in space. Increase in B decreases false positives but the max vote count also decreases. Therefore, we have to decrease T to pick sufficient number of candidates and get higher accuracy. A hardware implementation may use the

L	% Accuracy	% Candidates	Max Vote	True NN Vote
25	81.61	3.89	24	20
50	91.10	1.73	45	39
75	91.10	1.55	65	59
100	95.00	1.14	86	79
150	96.60	1.07	126	119

(a) $B=2, T=70\%$

L	% Accuracy	% Candidates	Max Vote	True NN Vote
25	90.6	8.16	24	20
50	97.90	6.37	45	39
75	99.10	5.4	65	59
100	99.70	4.0	86	79
150	100.0	4.28	126	119

(b) $B=2, T=65\%$

L	% Accuracy	% Candidates	Max Vote	True NN Vote
25	90.40	4.56	19	13
50	94.90	2.90	33	26
75	96.00	2.15	47	39
100	97.00	1.93	61	53
150	98.50	1.71	89	81

(c) $B=4, T=40\%$

TABLE I
RESULTS OF VOTE COUNT METHOD FOR REAL DATASET ON 1 MILLION POINTS FOR VARIOUS L , NUMBER OF BINS B , AND THRESHOLD PERCENTAGE T

L	% Accuracy	% Candidates	Max Vote	True NN Vote
25	99.38	0.07	24	24
50	99.87	0.01	48	48
75	99.98	0.01	72	71
100	99.99	0.01	95	95

(a) $B=2, T=80\%$

L	% Accuracy	% Candidates	Max Vote	True NN Vote
25	99.96	2.77	24	24
50	100	0.26	48	48
75	100	0.08	72	71
100	100	0.02	95	95

(b) $B=2, T=70\%$

Number of Bins $B=4$				
Vote Count Threshold $T=60\%$				
L	% Accuracy	% Candidates	Max Vote	True NN Vote
25	99.58	0.04	22	22
50	99.93	0.01	44	44
75	99.99	0.01	66	66
100	100	0.01	88	88

(c) $B=4, T=60\%$

TABLE II

RESULTS OF VOTE COUNT METHOD FOR SYNTHETIC DATASET FOR VARIOUS L , NUMBER OF BINS B , AND THRESHOLD PERCENTAGE T .

value ($T=65\%$, $B=2$, $L=75$) which ensures high accuracy of 99.1% with only 5.4% candidates. These parameters ensure low space cost with real time search performance. We would just need 7.5 giga bit of RAM (and 17.2 giga transistors with the reduced overhead implementation of Figure 3) in indexing space to search through 100 million points in real time.

We present the result of synthetic dataset in Table II for different parameters. We obtained accuracy of 99.38% with ($T=80\%$, $B=2$, $L=25$) as shown in Table II(a). Accuracy improved with increase in L and the number of candidate size decreased. Accuracy also increased with decrease in threshold from $T=80\%$ in Table II(a) to $T=70\%$ in Table II(b) at the cost of increase in the candidate size. We also performed experiment with number of bins $B=4$ as shown in Table II(c). Impact of increase in B is decrease in the number of candidates.

Our empirical results show that the vote count method produces high quality results for both real and synthetic dataset for small values of L and B making it feasible to be implemented in hardware at low cost.

VII. CONCLUSION

In this paper we proposed a new algorithm for a fast and scalable search with high accuracy in a very high dimensional space using the fundamental property of p -stable distribution. We proposed a simple design to implement the algorithm in hardware to obtain real time performance. Empirically we obtained high quality results on both real and synthetic datasets with as few as 25 random vectors and 2 bins. These parameters make it feasible for the algorithm to be implemented in hardware at minimal production cost. In the future, we want to test the algorithm on datasets from various domains. We would also like to actually produce the hardware and promote it for real use.

REFERENCES

[1] D. KNUTH, *The Art of Computer Programming, Vol. 3: Sorting and Searching*. Addison-Wesley, 1973, vol. 3.

[2] S. Belongie, J. Malik, and J. Puzicha, "Shape matching and object recognition using shape contexts," *PAMI*, vol. 24, pp. 509–522, 2001.

[3] K. Grauman and T. Darrell, Eds., *Fast Contour Matching Using Approximate Earth Mover's Distance*. CVPR, 2004.

[4] T. Hastie and R. Tibshirani, "Discriminant adaptive nearest neighbor classification," pp. 607–616, 1996.

[5] F. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz, "Efficient and effective querying by image content," *J. Intell. Inf. Syst.*, vol. 3, no. 3-4, pp. 231–262, 1994.

[6] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker, "Query by image and video content: The qbic system," *Computer*, vol. 28, no. 9, pp. 23–32, 1995.

[7] S. Cost and S. Salzberg, "A weighted nearest neighbor algorithm for learning with symbolic features," *Mach. Learn.*, vol. 10, no. 1, pp. 57–78, 1993.

[8] T. Cover and P. Hart, "Nearest neighbor pattern classification," vol. 13, pp. 21–27, 1967.

[9] F. Aurenhammer, "Voronoi diagrams - a survey of a fundamental geometric data structure," vol. 23, no. 3, pp. 345–405, 1991.

[10] J. L. Bentley, "Multidimensional binary search trees used for associative searching," vol. 18, no. 9, pp. 509–517, 1975.

[11] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *SIGMOD*, 1984, pp. 47–57.

[12] P. Ciaccia, M. Patella, and P. Zezula, "M-tree: An efficient access method for similarity search in metric spaces," in *VLDB*, 1997, pp. 426–435.

[13] R. Weber, H.-J. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces," in *VLDB*, 1998, pp. 194–205.

[14] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," pp. 91–110, 2004.

[15] I. Fodor, "A survey of dimension reduction techniques," Tech. Rep., 2002.

[16] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *SCIENCE*, vol. 290, pp. 2323–2326, 2000.

[17] X. He and P. Niyogi, "Locality preserving projections," in *NIPS*, 2003.

[18] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *VLDB*. Morgan Kaufmann Publishers Inc., 1999, pp. 518–529.

[19] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *STOC*, 1998, pp. 604–613.

[20] M. S. Charikar, "Similarity estimation techniques from rounding algorithms," in *STOC*, 2002, pp. 380–388.

[21] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p -stable distributions," in *Symposium on Computational Geometry*, 2004, pp. 253–262.

[22] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," *Commun. ACM*, vol. 51, no. 1, pp. 117–122, 2008.

[23] V. Gaede and O. Günther, "Multidimensional access methods," *ACM Comput. Surv.*, vol. 30, no. 2, pp. 170–231, 1998.

[24] H. Samet, *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann Publishers Inc., 2005.

[25] S. Berchtold, D. A. Keim, H.-P. Kriegel, and T. Seidl, "Indexing the solution space: A new technique for nearest neighbor search in high-dimensional space," *TKDE*, vol. 12, no. 1, pp. 45–57, 2000.

[26] H. V. Jagadish, B. C. Ooi, K.-L. Tan, C. Yu, and R. Zhang, "idistance: An adaptive b+-tree based indexing method for nearest neighbor search," *ACM Trans. Database Syst.*, vol. 30, no. 2, pp. 364–397, 2005.

[27] J. K. Lawder and P. J. H. King, "Using space-filling curves for multi-dimensional indexing," in *BNCOD*, 2000, pp. 20–35.

[28] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu, "An optimal algorithm for approximate nearest neighbor searching," in *SODA*, 1994, pp. 573–582.

[29] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe lsh: efficient indexing for high-dimensional similarity search," in *VLDB*, 2007, pp. 950–961.

[30] W. Dong, Z. Wang, W. Josephson, M. Charikar, and K. Li, "Modeling lsh for performance tuning," in *CIKM*, 2008, pp. 669–678.

[31] C. KUN, S. Quan, and A. Mason, "A power-optimized 64-bit priority encoder utilizing parallel priority look-ahead," in *ISCAS*, 2004.

[32] C. T. Leondes, *Digital signal processing systems: implementation techniques*, 1995.