A Time-Space Efficient Locality Sensitive Hashing Method for Similarity Search in High Dimensions

Qin Lv William Josephson Zhe Wang Moses Charikar Kai Li

Department of Computer Science
Princeton University, Princeton, NJ 08544

{*qlv, wkj, zhewang, moses, li*}*@CS.Princeton.EDU*

ABSTRACT

Locality Sensitive Hashing (LSH) by Indyk and Motwani is a popular technique for designing indexing data structures for approximate nearest neighbor search on high dimensional data. However, its drawback is that a very large number of hash tables are needed in order to achieve good approximation accuracy. As a result, the high space requirement makes the approach impractical for large datasets. A recent (theoretical) result showed that a point perturbation technique has the potential to reduce the space requirement of the LSH approach at the cost of an increase in query time. This paper proposes a new LSH method called hash-perturbation LSH based on perturbing the hash values. Our evaluation with an image dataset and an audio dataset shows that the proposed approach is both time and space efficient. To achieve similar search quality, the hash-perturbation LSH approach has a similar time efficiency as the basic LSH approach while reducing the space requirement by a factor of five. For image dataset, its time efficiency is twice of the the point perturbation approach.

1. INTRODUCTION

Similarity search (or query) in a high-dimensional space has become increasingly important in database, data mining and information retrieval systems, typically in the context of content-based search of feature-rich data such as audio recordings, digital photos, digital videos, and sensor data. Although users may want to search some datasets such as digital photos based on human-perceived similarity, most search systems are designed to represent each object with a set of high-dimensional feature vectors and to search k-nearest neighbors (KNN) of a query object in a high- dimensional space. Feature-rich datasets are typically very large and the dimensions of their feature vectors are high. When the dimension is high and dataset size is large, the search problem is difficult and some call it *curse of dimensionality.* The challenge is to efficiently perform indexing or similarity search for query objects in a high-dimensional space. Feature-rich data typically represent their features as points in a high-dimensional space and use a distance metric (such as Euclidean distance) to measure similarity of objects. The dimensionality of such feature vectors ranges from tens to thousands. For example, the number of features of an image search system can sometimes be several hundreds [9, 11]. When the number of dimensions reaches certain level, indexing methods based on space-partition (such as K-D trees, R-trees and X-trees) will require substantial space and time [24, 4], often less efficient than the brute-force linear search approach. A recent study shows that in both theory and practice, if the number of dimensions exceeds around 10, linear scan outperforms all existing indexing data structures for nearest-neighbor search [31].

For high-dimensional data, the known general method that works faster than the linear scan approach in practice is to perform approximate nearest-neighbor search using *locality sensitive hashing* (LSH) [15, 6]. The main idea of the technique is to use a special family of hash functions (called locality sensitive hash functions) to hash objects into buckets, such that objects close to each other in their high-dimensional space have a higher probability to be hashed to the same bucket. To make this approach practical, multiple hash tables are typically used to create a candidate result set and then rank the results by a linear scan. An experimental study shows that the approach can approximate nearest neighbor search reasonably well but it require many hash tables to cover most nearest neighbors and that sometimes LSH may require over a hundred hash tables to achieve reasonable accurate approximations [13]. Such space requirements make this basic LSH approach impractical.

Recently, Panigrahy proposed a point-perturbation based (also called entropy-based) LSH approach to reduce the space requirement of the original LSH approach [21]. The approach generates some randomly "perturbed" objects in the neighborhood of the query object, query them as well as the query object, and return the union of all results as the candidate result set. Although no experimental study has been conducted to show the tradeoffs of this approach, the potential of this approach is to trade time for space—spend time on querying the perturbed objects in order to use fewer number of hash tables than the originally proposed LSH

approach to achieve similar search quality. The potential disadvantage of this approach is that it may not be time efficient.

This paper makes two main contributions. First, we have proposed a novel perturbation-based LSH approach that is efficient in both time and space. Instead of generating a large number of perturbed objects, this approach perturb the hash results to look up other buckets in the hash tables. We call this approach *hash-perturbation* LSH approach. The main rationale is that since the hash functions are locality sensitive, most near neighbors of the query object either have the same hash values as the query object or different by only one or two hash values. By perturbing hash values, we avoid looking at the overlapped buckets. The goal is to achieve similar space efficiency to the point-perturbation LSH approach and similar time efficiency to the basic LSH approach.

Second, we have implemented all three approaches and evaluate them with an image dataset and an audio dataset. The image dataset has 1.2 million images and the dimensionality of the feature vectors is 64. The audio dataset has over 2.6 million words and the dimensionality of the feature vector is 192. Our evaluation shows the proposed hash-perturbation LSH approach is indeed time and space efficient. To achieve a similar search speed and quality, the approach indeed achieves similar time efficiency to the basic LSH approach and the similar space efficiency to the entropy-based LSH approach. It typically reduces the space requirement of the basic LSH approach by a factor of five. For image similarity search, it reduces the search time of the entropy-based approach by a factor of two.

The rest of the paper is organized as follows: Section 2 introduces locality sentive hashing and two LSH methods that have been proposed previously. Section 3 presents the new LSH method we have have proposed, called *hash-perturbation based LSH*. Section 4 describes the implementation details of the three LSH methods we have implemented. Experimental results are presented in Section 5. Finally, Section 6 summarizes the related work and Section 7 concludes the paper.

2. PREVIOUS LSH METHODS

LSH stands for *locality sensitive hashing* (LSH). This is a technique that has been proposed for approximate nearest-neighbor search for high-dimensional data. We begin by introducing the notion of locality sensitive hashing, and a particular LSH family based on p-stable distributions. Next, we describe the basic LSH scheme for nearest neighbor search. We then describe point-perturbation based (also called entropy-based) LSH, a recent improvement on the basic LSH scheme.

Let S be the domain of objects and let each object be represented by a d-dimensional feature vector, *i.e.* a point in a d-dimensional real vector space $\mathbb{R}^d$. The distance function we consider between two objects p and q is the Euclidean distance:

$$d(p,q) = \Big(\sum_{i=1}^{d}(p_i - q_i)^2\Big)^{1/2} \quad (1)$$

2.1 Locality Sensitive Hashing (LSH)

DEFINITION 1. *A function family $\mathcal{H} = \{h : S \to U\}$ is called locality sensitive if for any two objects u and v, the probability $Pr_{\mathcal{H}}[h(u) = h(v)]$ is strictly decreasing in $d(u,v)$. That is, the probability that u and v collide (hash to the same value) decreases as $d(u,v)$ increases.*

The notion of locality sensitive hashing was first introduced in [15]. It provides a dimension reduction technique which projects objects in high-dimensional spaces to lower-dimensional spaces while still preserving the relative distances among objects. Different LSH families can be used for different distance functions. The LSH technique proposed by Datar *et al.* [6] uses p-stable distributions.

DEFINITION 2. *A distribution $\mathcal{D}$ over $\mathbb{R}$ is called p-stable, if there exists $p \geq 0$ such that for any n real numbers $v_1, \ldots, v_n$ and i.i.d. variables $X_1, \ldots, X_n$ with distribution $\mathcal{D}$, the random variable $\sum_i v_i X_i$ has the same distribution as the variable $(\sum_i |v_i|^p)^{1/p} X$, where X is a random variable with distribution $\mathcal{D}$.*

It is known that stable distributions exist for any $p \in (0, 2]$ [32]. In particular:

- A *Cauchy distribution* $\mathcal{D}_C$, defined by the density function $c(x) = \frac{1}{\pi}\frac{1}{1+x^2}$, is 1-stable.
- A *Gaussian (normal) distribution* $\mathcal{D}_G$, defined by the density function $g(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$, is 2-stable.

In a LSH family using p-stable distribution, each hash function is indexed by a choice of random a and b where a is a d-dimensional vector with entries chosen independently from a p-stable distribution and b is a real number chosen uniformly from the range $[0, W]$. Each hash function $h_{a,b} : \mathbb{R}^d \to \mathbb{Z}$ maps a d-dimensional vector v onto the set of integers. For a fixed a, b, the hash function $h_{a,b}$ is given by:

$$h_{a,b}(v) = \Big\lfloor \frac{a \cdot v + b}{w} \Big\rfloor \quad (2)$$

In our evaluations, we use the Gaussian distribution, which is 2-stable and works for the Euclidean distance.

2.2 The Basic LSH Scheme

The basic LSH scheme for nearest neighbor search has been used in [15, 13, 6]. Given a query object q, suppose we want to find its nearest neighbors within distance R. Based on the property of locality sensitive hashing, we know that for two objects u and v, if $d(q,u) \leq R$ and $d(q,v) > R$, then u is more likely to have the same hash value as q than v. To amplify the gap between the collision probabilities for the range $[0, R]$ (where the nearest neighbors lie) and the range $[R, \infty]$, we concatenate M such functions $h \in \mathcal{H}$. We define a function family $\mathcal{G} = \{g : S \to U^M\}$ such that $g(v) = (h_1(v), \ldots, h_M(v))$, where $h_i \in \mathcal{H}$. For an integer L we choose L functions $g_1, \ldots, g_L$ from $\mathcal{G}$, independently and

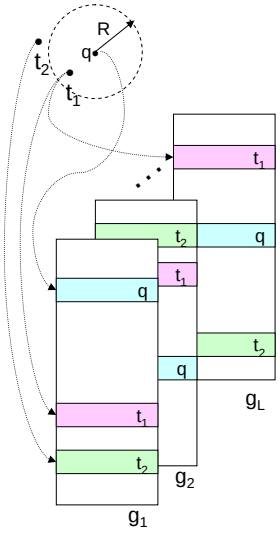

Figure 1: Locality Sensitive Hashing: Basic Scheme

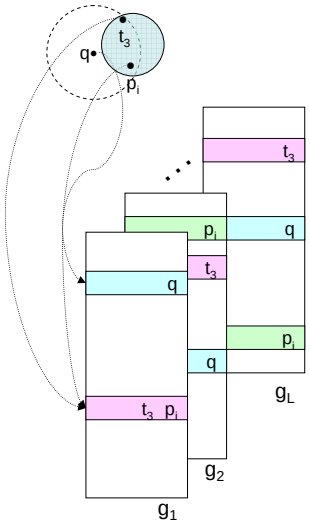

Figure 2: Point-Perturbation Based LSH

uniformly at random. Each of the L functions g_j is used to construct one hash table, resulting in L hash tables[1].

Figure 1 shows how the basic LSH scheme works. When a new object q is inserted into the system, for $j = 1, \ldots, L$, we compute its hash value $g_j(q)$ and store q in the bucket that $g_j(q)$ points to in the j-th hash table. Since the total number of buckets may be large, we retain only the non-empty buckets by resorting to regular hashing. To find the k nearest neighbors of a query object q, we search all buckets $g_1(q), \ldots, g_L(q)$. For each object p found in one of these buckets, we compute the distance $d(q, p)$, and the k objects whose distance are closest to q are returned[2].

An experimental study [13] shows that the basic LSH approach can approximate nearest neighbor search reasonably well but it requires many hash tables to cover most nearest neighbors and that sometimes LSH may require over a hundred hash tables to achieve reasonably accurate approximations. Such space requirements make this basic LSH approach impractical for large datasets.

2.3 Point-Perturbation Based LSH

Recently, Panigrahy proposed a *point-perturbation* (also called entropy-based) LSH approach [21], which is an extension of the basic LSH scheme designed to reduce the number of hash tables required for good search quality and performance. As shown in Figure 2, given a query object q, this new approach generates several randomly "perturbed" objects in the neighborhood of the query object and uses the perturbed objects (in addition to the query object itself) as input to the search process. The algorithm then takes the union of the search results of the original query point and the search results of the perturbed objects as the result of the query. Specifically, let q be the query object, let R be the distance between q and its nearest neighbor, the point-perturbation based LSH scheme "perturbs" q to generate random objects that are R distance away from q and hashes these perturbed objects to locate buckets. Let T be the number of perturbations, the new scheme checks $T + 1$ buckets per hash table, instead of just one bucket per hash table in the basic LSH scheme.

Intuitively, since the perturbed objects are close to the query object, their hash values are close to the hash values of the query object, based on the property of locality sensitive hashing. As a result, the buckets that these perturbed objects are hashed to are likely to contain some of the query object's nearest neighbors. This is helpful in finding objects that are nearest neighbors of the query object but are not hashed to the same bucket as the query object. The end goal of the point-perturbation based LSH scheme is to trade time for space. That is, to achieve the same search quality, it uses a smaller number of hash tables than the basic LSH scheme, but needs to check more hash buckets.

The point-perturbation based LSH scheme is a good improvement on the basic LSH scheme. But it has several issues:

- It is hard to choose the perturbation distance R for a given query object q.
- The optimal perturbation distance R for different query objects can be very different.
- It is hard to choose the number of perturbations needed. More perturbations may give us better search quality, but it also means more buckets to check, more objects to compare, and thus longer search time. The author

[1]To achieve good search quality, different M and L values are needed for different R values. In practice, multiple sets of hash tables are used in order to cover different R values (e.g., $r, 2r, 4r, \ldots$).

[2]A variation of this scheme is to find all objects that are within distance R from query object q. Then, instead of returning the k closest objects, each object p whose distance $d(q, p) \leq R$ is returned.

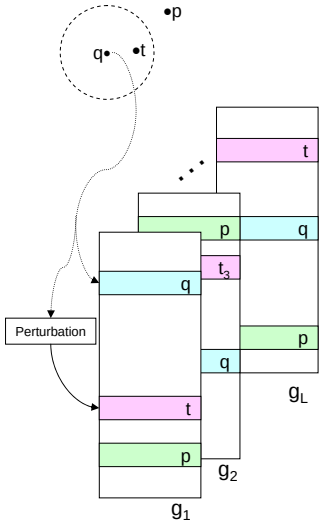

Figure 3: Hash-Perturbation Based LSH

of [21] gives some theoretical analysis based on entropy. But that is hard to compute in practice.

- Multiple perturbed objects may hash to the same bucket. It is wasteful to check the same bucket multiple times. When the number of perturbation is large, this kind of duplicate work unnecessarily slows down the query process.

- The operations of perturbing points and computing hash values (which involves computing dot products) are not very fast. To answer a query, T point perturbations and $T \times L$ hash computations are needed. This can slow down the query process quite a bit if T is large.

3. HASH-PERTURBATION BASED LSH

We propose a new perturbation-based LSH method, called *hash-perturbation* based LSH. Similar to the point perturbation method, our method also checks multiple buckets for each hash table. What is different is that our method perturbs directly on the computed hash values of the query object, rather than perturbing the query object. As a result, we avoid the overhead of perturbing objects and dot product computations associated with each perturbation and each hash function.

Given a query object q, let $(h_1, \ldots, h_M)$ be the hash values of q in a hash table. As in the basic LSH method, $(h_1, \ldots, h_M)$ identifies the bucket that the query object q is hashed to. Given the property of locality sensitive hashing, we know that if an object is close to q but not hashed to the same bucket as q, it is likely to be in a bucket that is "closeby". So our goal is to perturb the hash values of q in order to get to buckets that are "closeby", thus increasing our chance of finding the objects that are close to q. We first consider 1-step perturbation, *i.e.*, we perturb only one hash value. Let $i(1 \leq i \leq M)$ be the position we want to perturb and d_i be the "strength" with which we want to perturb, then the hash values after 1-step perturbation are:

$$(h_1, \ldots, h_i + d_i, \ldots, h_m)$$

Similarly, when considering 2-step perturbation, we pick two positions i and j $(1 \leq i < j \leq M))$ and d_i and d_j, then the hash values after 2-step perturbation are:

$$(h_1, \ldots, h_i + d_i, \ldots, h_j + d_j, \ldots, h_m)$$

Figure 3 shows how the hash-perturbation based LSH scheme works.

Again, based on the property of locality sensitive hashing, we know that buckets that are one step away (*i.e.*, only one hash value is different from the M hash values of the query object) are more likely to contain objects that are close to the query object. So, in our hash perturbation method, we always check buckets that are one step away before we check buckets that are two steps away.

Now the question is what values we should use for d_i and d_j. Remember that the LSH functions we use are of the format $h_{a,b}(v) = \lfloor \frac{a \cdot v + b}{W} \rfloor$. If we pick W to be reasonably large, with high probablilty, similar objects should fall into adjacent buckets, *i.e.* their hash values only differ by 1. It is then reasonable to pick d_i and d_j to be either -1 or 1.

Figure 4 and Figure 5 shows the distribution of bucket distance for the top $k(k = 20, 40, 60, 80, 100)$ nearest neighbors of a query object, for the image data and audio data respectively. In each figure, the plot on the left shows the difference for a single hash value (d_i and d_j values). The plot on the right shows the number of hash values (out of M per hash table) that differ from the hash values of the query object (*i.e.*, how many steps away). As we can see from the plots, almost all of the individual hash values of the top k nearest neighbors are either the same as the hash values of the query object or differ by just -1 or 1. Also, most of the top k nearest neighbors are hashed to buckets that are within 2 steps away from the bucket that the query object is hashed to.

Our hash-perturbation based LSH scheme improves on the point-perturbation based LSH scheme in several ways:

- *Faster perturbation* To perturb the hash values, only +1/-1 operations are needed, which is much faster than the operations needed to perturb an object for the point perturbation scheme.

- *Easier parameter tuning* Besides the parameters needed by any LSH scheme (W, M, L), our hash perturbation scheme introduces just one parameter: T, the number of perturbations, which is easy to decide based on the number of the buckets that are 1-step or 2-step away.

- *No duplicate buckets* Unlike the point perturbation scheme, each hash perturbation points to a different bucket, so no bucket is checked more than once.

4. IMPLEMENTATION DETAILS

We have implemented all the three LSH methods as specified in previous subsections: *LSH_basic, LSH_point, LSH_hash.* All methods are implemented in C. At startup time, each

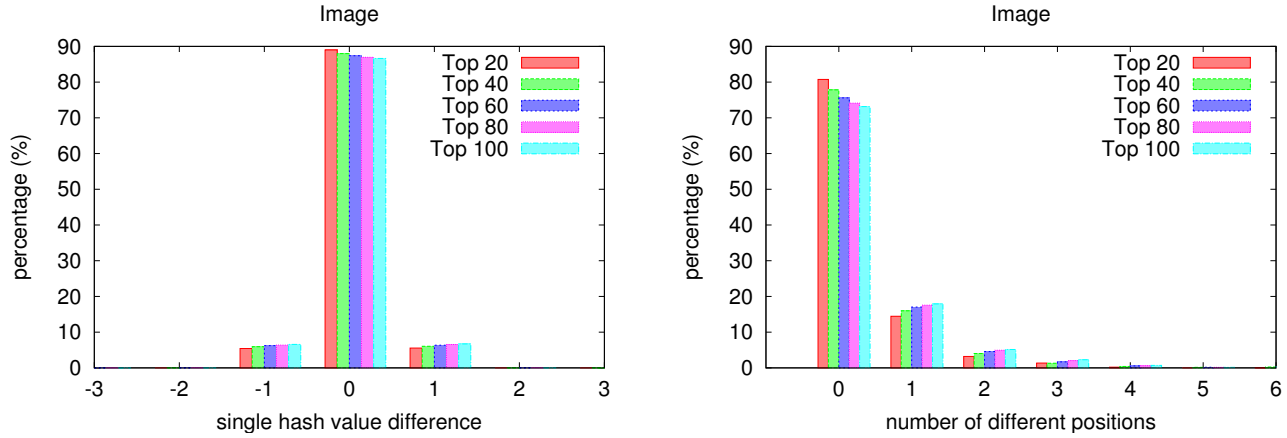

Figure 4: Distribution of bucket distance for the top $k(k = 20, 40, 60, 80, 100)$ nearest neighbors of a query object. The plot on the left shows the difference for a single hash value. The plot on the right shows the number of hash values (out of M per hash table) that differ from the hash values of the query object. The numbers are averaged over the 100 query objects in the image benchmark. The parameters used are $W = 0.6, M = 16, L = 15$.

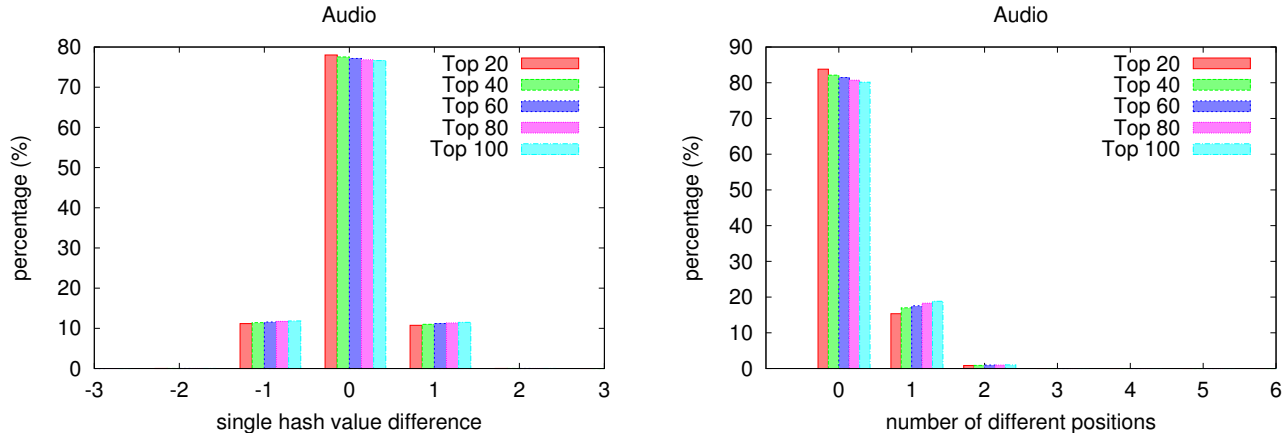

Figure 5: Distribution of bucket distance for the top $k(k = 20, 40, 60, 80, 100)$ nearest neighbors of a query object. The plot on the left shows the difference for a single hash value. The plot on the right shows the number of hash values (out of M per hash table) that differ from the hash values of the query object. The numbers are averaged over the 100 query objects in the audio benchmark. The parameters used are $W = 23, M = 8, L = 15$.

method reads in all the objects (each represented as a d-dimentional feature vectors) and constructs hash tables according to the parameters specified. At query time, each method checks its L hash tables to generate a candidate set and returns the k candidates that are closest to the query object. Unlike the *LSH_basic* method, which checks only one bucket in each table, *LSH_point* and *LSH_hash* use point perturbations and hash perturbations respectively to check multiple buckets in each hash table.

Specifically, there are mainly three procedures provides by the LSH schemes:

- $lsh\_init(D, W, M, L)$ initializes the hash tables, using the parameters specified: D is the dimentionality of the feature vectors, W is the hash width for each single locality sentive hash function, M is the number of hash functions to use for each hash table, and L is the total number of hash tables.
- $lsh\_insert(rgn\_id)$ inserts a region into the hash tables. rgn_id is a unique id for each region in the system. It can be used to access a region's feature vector (or region sketch, when sketches are used), as well as accessing the image that the region belongs to.
- $lsh\_search(rgn\_id, k, max\_d, T, R)$ searches for the k regions that are closest to the query region and their distances to the query region are within max_d. When T is 0, the basic LSH scheme is used (*i.e.*, checks only one bucket in each hash table). When $T > 0$ and $R > 0$, the point perturbation based LSH scheme is used, using T point perturbations and the perturbation distance is R. When $T > 0$ and $R = 0$, the hash perturbation based LSH scheme is used, using T hash perturbations.

The LSH functions we use are in the form $h_{a,b}(v) = \lfloor \frac{a \cdot v + b}{w} \rfloor$, where the entries of a is drawn independently from the Gaussian (normal) distribution $N(0, 1)$. As said previously, a Gaussian distribution is 2-stable, so the LSH functions we use approximate ℓ_2 distance. The a, b values are randomly picked at startup time and are then used to compute the hash values of each object.

In the *LSH_point* method, to generate a perturbed object $p = (p_1, \ldots, p_d)$ that is at distance R from query object $q = (q_1, \ldots, q_d)$, we pick $p_i (i \in [1, d])$ randomly from a normal distribution with mean $\mu = q_i$. See Algorithm 1, where function $Gaussian()$ returns a real number randomly picked from the Gaussian distribution $N(0, 1)$ (*i.e.* the distribution's mean is 0 and variance is 1). The original point perturbation paper [21] only considers the single nearest neighbor and sets R to be the exact distance of nearest neighbor. Since we consider k nearest neighbors, we pick a fixed R value that is slightly smaller than the average distance of the k-th nearest neighbor. This R value is 0.04 for the MIXED_WEB image data set we use in our evaluations.

In the *LSH_hash* method, let $(h_1, \ldots, h_i, \ldots, h_M)$ be the hash values of the query object, we first check the 1-step perturbations:

$$(h_1, \ldots, h_i + d_i, \ldots, h_M) \quad i = 1, \ldots, M, \ d_i \in [-1, 1]$$

Next, we check the 2-step perturbations:

$$(h_1, \ldots, h_i + d_i, \ldots, h_j + d_j, \ldots, h_M)i$$

for $1 \le i < j \le M, \ldots, \ d_i, d_j \in [-1, 1]$. We stop when we have done the specified number of perturbations.

As a baseline comparison, we also implemented the *brute force* method. At startup time, it reads in all the feature vectors. Then, for each query object q, it goes through each object p in the system, computes the ℓ_2 distance between q and p, and returns the k nearest neighbors. This method is also implemented in C.

Algorithm 1 Point Perturbation

Input query object $q = (q_1, \ldots, q_d)$
perturbation distance R
Output perturbed object $p = (p_1, \ldots, p_d)$

```
s = 0
for i ∈ [1, ..., d] do
  p_i = Gaussian()
  s = s + p_i × p_i
end for

s = R/sqrt(s)
for i ∈ [1, ..., d] do
  p_i = q_i + p_i × s
end for

return p = (p_1, ..., p_d)
```

5. PERFORMANCE EVALUATION

In this section, we evaluate the performance of the three locality sensitive hashing methods: LSH_basic, LSH_point and LSH_hash. We are interested in answering questions about the space requirements, search time and search quality tradeoffs for different LSH methods.

We experiment with the following parameters:

- W: hashing interval of an individual hash function
- M: number of hash functions used for each table
- L: number of hash tables to use
- T: number of perturbations, used by the point perturbation method (LSH_point) and the hash perturbation method (LSH_hash)
- R: perturbation distance, used by the point perturbation method (LSH_point)

For each method, we experiment with various parameter settings and pick the one(s) that achieve certain recall.

5.1 Benchmarks

We have used two data sets in our evaluations:

Image Data The image dataset is obtained from Stanford's WebBase project [26]. It contains images crawled from

Dataset	#Objects	Dimension	Total Size
Image	1,312,581	64	336 MB
Audio	2,663,040	192	2.0 GB

Table 1: Evaluation Datasets.

the web. We only pick images that are of the JPEG format and are larger than 64×64 in size. In total, this gives us 1.3 million images. For each image, we use the *extractcolorhistogram* tool from the FIRE image search engine [27] to extract a 64-dimensional color histogram.

Audio Data The audio dataset is extracted from the LDC SWITCHBOARD-1 [10] collection. It is a collection of about 2400 two-sided telephone conversations among 543 speaker from all areas of the United States. The conversations are split into individual words based on the human transcription. We have a total of about 2.6 million words. For each word segment, we then use the Marsyas library [28] to extract feature vectors by taking a 512-sample sliding window with variable stride to obtain 32 windows for each word. For each window, we extract the first six MFCC parameters to obtain a 192 dimensional feature vector. The MFCC feature vectors are known to be relatively insensitive to variation of different human voice and dialects.

Table 1 lists the number of objects in each dataset and the dimensionality of its feature vectors.

To evaluate the performance of similarity search, we have created two benchmarks, one for each dataset. Each benchmark contains 100 query objects randomly picked from the corresponding dataset. For each query object, we compute its k nearest neighbors (not including the query object itself) using Euclidean distance. This is the groundtruth for each query object.

5.2 Evaluation Metrics

Given the benchmarks above, for each query object, the goal is to find as many its k nearest neighbors as possible, while examining as fewer other objects as possible. For each query q_i, let s_i be the total number of objects examined (*i.e.* objects that are hashed to the same bucket as the query object or the perturbed objects in at least one of the hash tables), let m_i be the number of nearest neighbors found that are actually in the benchmark. To measure search quality, we use the following metric and report the average numbers over all 100 queries in a benchmark.

$$Recall \quad = \quad m_i/k$$

In the ideal case, the recall score is 1.0, which means all the k nearest neighbors are returned.

To measure the search speed of a similarity search system, we use the *average query time* of all the query objects in a benchmark. Given a query object, its query time is the time difference between when the query is submitted and when the search results are returned.

To evalaute the space requirement of the LSH methods, we use the number of hash tables needed in order to achieve certain search quality. We are also interested in how big the total metadata size is and whether the metadata can all fit into memory. This can be measured by the total amount of memory used by each LSH method.

Recall	Method	Query Time (s)	#Hash Tables	Memory Overhead
1.0	brute force	0.21		
0.904	LSH_basic	0.042	100	2.42 GB
0.900	LSH_point	0.139	20	0.50 GB
0.908	LSH_hash	0.036	15	0.34 GB
0.860	LSH_basic	0.033	90	2.23 GB
0.859	LSH_point	0.077	10	0.24 GB
0.861	LSH_hash	0.021	10	0.23 GB
0.808	LSH_basic	0.020	80	2.10 GB
0.807	LSH_point	0.053	10	0.26 GB
0.807	LSH_hash	0.019	10	0.24 GB

Table 2: Performance comparison on the image dataset.

Recall	Method	Query Time (s)	#Hash Tables	Memory Overhead
1.0	brute force	1.16		
0.976	LSH_basic	0.545	50	2.15 GB
0.975	LSH_point	0.699	15	0.65 GB
0.975	LSH_hash	0.681	15	0.64 GB
0.915	LSH_basic	0.212	50	2.17 GB
0.913	LSH_point	0.245	15	0.68 GB
0.911	LSH_hash	0.228	15	0.66 GB
0.869	LSH_basic	0.126	50	2.20 GB
0.868	LSH_point	0.220	10	0.45 GB
0.862	LSH_hash	0.212	10	0.44 GB

Table 3: Performance comparison on the audio dataset.

5.3 Experimental Results

The evaluation is done on a PC system with one dual-processor Intel Xeon 3.2GHz CPU with 1024KB L2 cache. The PC system has 6GB of DRAM and a 160GB 7,200RPM SATA disk. It runs Linux with a 2.6.9 kernel. Since the LSH methods use randomly picked hash functions, each experiment is repeated multiple times and the average is reported.

Our main observations are:

- The new LSH_hash method is both time and space efficient.
- With similar search quality, LSH_point and LSH_hash methods can reduce the space usage by a factor of 5 compare with the LSH_basic method.
- LSH_hash is faster than LSH_point when number of perturbations grows.

5.3.1 Performance Comparison of LSH Methods

Table 2 and table 3 show performance comparison of different LSH methods on image and audio data. The tables show

the extra space requirement (in terms of both the number of hash tables used and extra memory consumed by these hash tables) and search time for different LSH methods to achieve certain recall scores. We can see that LSH_hash method is most efficient in terms of space and time usage while achieving similar recall score comparing with other methods.

We can make several observations on the results. First, LSH methods can speed up the search time by an order of magnitude by only losing 0.1 recall score. This is the main incentive of using LSH method compare with the brute force approach.

Second, LSH_hash method has both faster query time and smaller space requirement of all LSH based methods for image data. For audio data, its search time is similar to other LSH methods but has much smaller space requirement comparing with LSH_basic method.

5.3.2 Space Usage of Different LSH Methods

One of the key issues of using the basic LSH method is the amount of hash table needed in order to achieve good search quality. In our experiments, we have observed that in order to get recall score about 0.9, we would need around 50 hash tables. This poses a system implementation problem when one can have millions of objects and the memory requirement for these hash tables can be very big. In our experiments, they can take about 2GB or more of memory.

The perturbation-based methods (LSH_point and LSH_hash) can reduce the space requirement by a large factor, compared with the basic LSH method. Figure 6 shows the recall scores in terms of the number of hash tables used. In order to achieve similar recall score, perturbation based method only need to use 1/5 amount of hash tables for both image and audio data.

5.3.3 Search Time Comparison

Figure 7 shows search time used by LSH_hash and LSH_point methods in terms of number of perturbations used. Usually the more perturbation used, the better recall score we can achieve. From the figure 7, we can see that the LSH_hash method uses less time than LSH_point when the number of perturbation used grows. It is an important attribute for the LSH_hash because this will give us better recall value when we are allowed to use more perturbations for each query.

6. RELATED WORK

The similarity search problem is closely related to the nearest neighbor search problem, which has been studied extensively in the theory community. A number of data structures have been devised for nearest neighbor searching; examples include R-trees [14], K-D trees [1], and SR-trees [17]. These data structures are capable of supporting similarity queries, but do not scale satisfactorily to large, high-dimensional data sets. The exact nearest neighbor problem suffers from the "curse of dimensionality" – *i.e.* either the search time or the search space is exponential in dimension d [7, 19]. As a result, recent research has focused on the approximate nearest neighbor problem. In this version of the problem, the objective is to find points whose distance from the query point is at most $1 + \epsilon$ times the exact nearest neighbor's distance. Chávez *et al.* have written an extensive survey of these and other techniques for searching and indexing in metric spaces [5]. Recent work has attempted to improve the complexity of nearest neighbor algorithms by taking advantage of the "intrinsic dimensionality" of datasets [16, 18, 3]. For an overview of dimension reduction techniques, see Imola Fodor's survey [12].

The notion of locality sensitive hashing (LSH) was first introduced by Indyk and Motwani in [15] for approximate nearest neighbor search. The key idea is to use hash functions such that the collision probability is much higher for objects that are close to other than objects that are far part. The LSH functions of Indyk and Motwani [15, 13] are for the binary Hamming space. More recently, Datar, Immorlica, Indyk, and Mirrokni have proposed an LSH scheme for approximate nearest neighbor under the l_p norms that is based on p-stable distributions [6]. The newest development in locality sensitive hashing is the perturbation-based scheme of Panigrahy [21], which perturbs the query object to generate several random objects in the neighborhood of the query objects and hashes these perturbed objects to locate buckets that may contain the query object's nearest neighbors.

Domain specific methods for content-based similarity search have been studied extensively. Usually they focus on extracting good feature vectors which can effectively measure the similarity between objects. For image similarity search, low-level features such as color, texture, and shape are usually used[8, 22, 25, 30, 23]. Previous work on audio similarity has tended to focus on similarity search either for music [29, 2] or the spoken word. Some recent systems [20] have used "phone"-based approaches for speech audio search.

7. CONCLUSIONS

Our experimental evaluation of the three LSH approaches with two datasets shows several results:

- The proposed hash-perturbation LSH approach is both time and space efficient. It is better than either the basic LSH approach or the point-perturbation LSH approach.
- To achieve similar recall scores, the hash-perturbation LSH approach typically achieves similar time efficiency to the basic LSH approach while reducing the number of hash tables by a factor of five.
- With the image dataset, the proposed approach improves the time efficiency over the entropy-based LSH approach by a factor of two, while having a similar space efficiency.

The dataset sizes are chosen to allow the basic LSH approach to fit all hash tables into the main memory, in order to compare all three LSH approaches. Although our experimental results also show that the LSH approaches are an order-of-magnitude faster than the linear scan approach, the performance gaps should be larger as the dataset sizes increase.

Throughout the experiments, we learned a main limitation of the LSH approaches with respect to real systems designs is

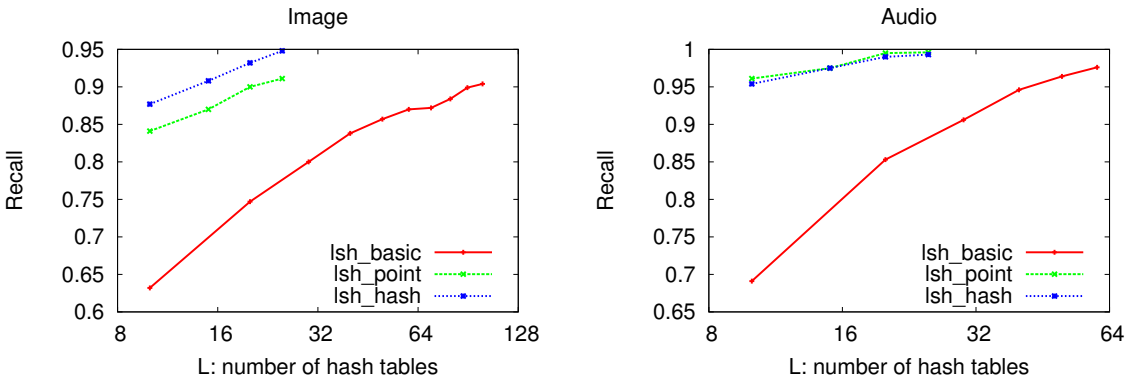

Figure 6: Recall score as function of number of hash tables.

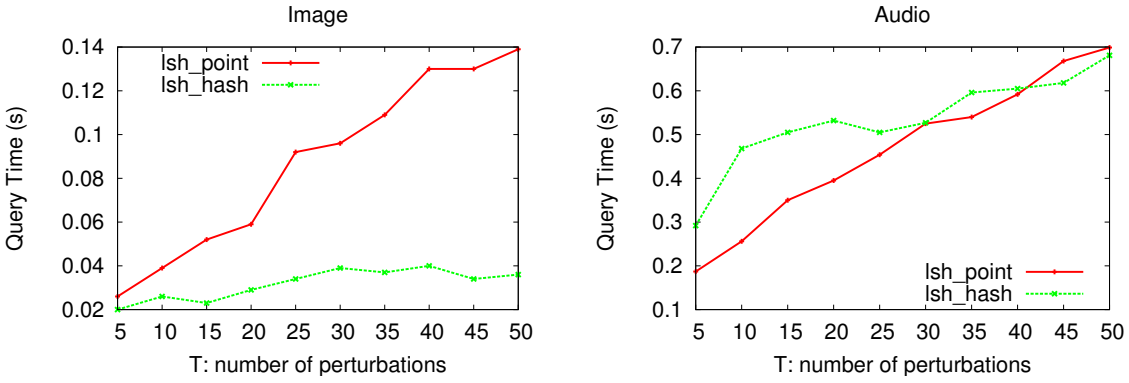

Figure 7: Query time as function of number of perturbations.

to find the right set of parameters for both LSH approaches. Without an accurate analytical model, it is difficult to experimentally explore among many highly sensitive parameters such as the number of hash tables, the number of hash functions, the parameters of the hash functions, the parameters of perturbations and the number of perturbations.

8. REFERENCES

[1] J. L. Bentley. K-D trees for semi-dynamic point sets. In *Proceedings of the 6th Annual ACM Symposium on Computational Geometry*, pages 187–197, 1990.

[2] A. Berenzweig and D. Ellis. Locating singing voice segments within music signals. In *Proc. of IEEE Workshop on Applications of Signal Processing t o Acoustics and Audio*, October 2001.

[3] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. http://hunch.net/jl/projects/-cover_tree/icalp_3/icalp_1.ps.

[4] C. Bohm, S. Berchtold, and D. A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Comput. Surv.*, 33(3):322–373, 2001.

[5] E. Chávez, G. Navarro, R. A. Baeza-Yates, and J. L. Marroqun. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.

[6] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the 20th annual symposium on Computational geometry(SCG)*, pages 253–262, 2004.

[7] D. Dobkin and R. Lipton. Multidimensional search problems. *SIAM J. Computing*, 5:181–186, 1976.

[8] J. P. Eakins and M. e. Graham. Content-based image retrieval: A report to the JISC technology applications programme. Technical report, University of Northumbria at newcastle, Institute for Image Data Research, 1999.

[9] C. Faloutsos, R. Barber, M. Flickner, W. Niblack, D. Petkovic, and W. Equitz. Efficient and effective querying by image content. *Journal of Intelligent Information Systems*, 3:231–262, 1994.

[10] *FIRE: Flexible Image Retrieval Engine.*

[11] 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. *IEEE Computer*, 28:23–32, 1995.

[12] I. K. Fodor. A survey of dimension reduction techniques. Technical Report UCRL-ID-148494, Lawrence Livermore National Laboratory, 2002.

[13] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of 25th International Conference on Very Large Data Bases(VLDB)*, pages 518–529, 1999.

[14] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 47–57, 1984.

[15] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimens ionality. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 604–613, 1998.

[16] D. Karger and M. Ruhl. Finding nearest neighbors in growth restricted metrics. pages 63–66, 2002.

[17] N. Katayama and S. Satoh. The SR-tree: An index structure for high-dimensional nearest neighbor queries. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 369–380, 1997.

[18] R. Krauthgamer and J. R. Lee. Navigating nets: Simple algorithms for proximity search. In *Proceedings of the 15th ACM Symposium on Discrete Algorithms*, pages 798–807, 2004.

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

[20] N. Moreau, H. G. Kim, and T. Sikora. Phone-based spoken document retrieval in conformance with the mpeg-7 standard. *Proc. of the Audio Engineering Society 25th Intl. Conf.*, 2004.

[21] R. Panigrahy. Entropy based nearest neighbor search in high dimensions. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms(SODA)*, Jan 2006.

[22] Y. Rui, T. S. Huang, and S.-F. Chang. Image retrieval: Current techniques, promising directions and open issues. *J. of Visual Communication and Image Representation*, 10(4):39–62, 1999.

[23] R. Schettini, G. Ciocca, and S. Zuffi. A survey of methods for color image indexing and retrieval in image databases. *Color Imaging Science: Exploiting Digital Media*, 2001.

[24] T. Sellis, N. Roussopoulos, and C. Faloutsos. Multidimensional access methods: Trees have grown everywhere. In *Proc. of the 23rd International Conference on Very Large Data Bases*, pages 13–15, 1997.

[25] A. W. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-base image retrieval at the end of the early years. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 22(12), 2000.

[26] *Stanford WebBase Project.*

[27] *SWITCHBOARD-1 Release 2.*

[28] G. Tzanetakis and P. Cook. *MARSYAS: A Framework for Audio Analysis.* Cambridge University Press, 2000.

[29] G. Tzanetakis and P. Cook. Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing*, 10(5), July 2002.

[30] R. C. Veltkamp and M. Tanase. Content-base image retrieval systems: A survey. Technical Report UU-CS-2000-34, Utrecht University, Information and Computer Sciences, 2000.

[31] R. Weber, H. Schek, and S. Blott. A quantitative analysis and performance study for similarity search methods in high dimensional spaces. In *Proc. of the 24th International Conference on Very Large Data Bases*, pages 194–205, 1998.

[32] V. Zolotarev. One-dimensional stable distributions. *Translations of Mathematical Monographs*, 65, 1986.