Modeling LSH for Performance Tuning

Wei Dong, Zhe Wang, William Josephson, Moses Charikar, Kai Li
Department of Computer Science, Princeton University
35 Olden Street, Princeton, NJ 08540, USA
{wdong,zhewang,wkj,moses,li}@cs.princeton.edu

ABSTRACT

Although Locality-Sensitive Hashing (LSH) is a promising approach to similarity search in high-dimensional spaces, it has not been considered practical partly because its search quality is sensitive to several parameters that are quite data dependent. Previous research on LSH, though obtained interesting asymptotic results, provides little guidance on how these parameters should be chosen, and tuning parameters for a given dataset remains a tedious process.

To address this problem, we present a statistical performance model of Multi-probe LSH, a state-of-the-art variance of LSH. Our model can accurately predict the average search quality and latency given a small sample dataset. Apart from automatic parameter tuning with the performance model, we also use the model to devise an adaptive LSH search algorithm to determine the probing parameter dynamically for each query. The adaptive probing method addresses the problem that even though the average performance is tuned for optimal, the variance of the performance is extremely high. We experimented with three different datasets including audio, images and 3D shapes to evaluate our methods. The results show the accuracy of the proposed model: the recall errors predicted are within 5% from the real values for most cases; the adaptive search method reduces the standard deviation of recall by about 50% over the existing method.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms

Algorithms, Performance

Keywords

similarity search, locality sensitive hashing

1. INTRODUCTION

The K nearest neighbors (K-NN) problem is a common formulation of many similarity search tasks such as content-based retrieval of multimedia data[14]. Although the problem has been studied extensively for several decades, no satisfactory general solution is known. The exact K-NN problem suffers from the *curse of dimensionality*, *i.e.* either the search time or space requirement is exponential in D, the number of dimensions [6, 16]. Both theory and practice have shown that traditional tree based indexing methods, *e.g.* [9, 3, 11], degenerate into linear scan in sufficiently high dimensions[21]. As a result, various approximate algorithms have been proposed to trade precision for speed. One of the most promising approximate algorithms is Locality Sensitive Hashing (LSH)[10, 8, 4].

The key insight behind LSH is that it is possible to construct hash functions such that points close to each other in some metric space have the same hash value with higher probability than do points that are far from one another. Given a particular metric and corresponding hash family, LSH maintains a number of hash tables containing the points in the dataset. An approximate solution to the K-NN query may be found by hashing the query point and scanning the buckets which the query point is hashed to.

As originally proposed, LSH suffers from two drawbacks. First, it requires significant space, usually hundreds of hash tables to produce good approximation. The recently proposed Multi-probe LSH algorithm[15] addresses this problem in practice, showing a space reduction of more than 90% in experiments. The basic idea (building on [18]) is to probe several buckets in the same hash table — additional buckets probed are those with addresses close to the hash value of the query. The second significant drawback of LSH is that its performance is very sensitive to several parameters which must be chosen by the implementation. A previously proposed scheme, LSH Forest[2], partially addressed this problem by eliminating the need to fix one of the parameters; however, the implementation is still left with the issue of finding good values for others; and there is a similar problem, as in Multi-probe LSH, to determine how many nodes in the trees to visit. The process of parameter tuning is both tedious and a serious impediment for practical applications of LSH. Current research on LSH and its variants provides little guidance on how these parameter values should be chosen.

This paper presents a performance model of Multi-probe LSH. Given a particular data type, a small sample dataset and the set of LSH parameters, the model can accurately predict the query quality and latency, making the parameter tuning problem easy. The optimal setting of parameters depends on the dataset of interest, which, in practice, might not be available at implementation time. Our method does not require the full dataset. We identify relevant statistical properties of the data, infer them from a sample dataset and extrapolate them to larger datasets. We show that our model is an accurate predictor of empirical performance.

In addition, we use our performance model to devise an adaptive version of Multi-probe LSH with superior properties. Both our analysis and experiments show that the performance of LSH on a query point depends not only on the overall distribution of the dataset, but also on the local geometry in the vicinity of the particular query point. The fixed number of probes used in the original Multi-probe LSH may be insufficient for some queries and larger than necessary for others. Our adaptive probing method only probes enough buckets to achieve the required search result quality.

Our evaluation with three different datasets — images, audio, and 3D shapes — shows that our analytical model is accurate for predicting performance and thus reliable for parameter tuning. Furthermore, our adaptive probing method not only reduces the variance in search performance between different queries, but also potentially reduces the query latency.

2. BACKGROUND

2.1 Basic LSH

The K Nearest Neighbor (K-NN) problem is as follows: given a metric space $\langle \mathbb{M}, d\rangle$ and a set $S \subseteq \mathbb{M}$, maintain an index so that for any query point $v \in \mathbb{M}$, the set $I(v)$ of K points in S that are closest to v can be quickly identified. In this paper we assume the metric space is the D-dimensional Euclidean space $\mathbb{R}^D$, which is the most commonly used metric space.

Indyk and Motwani introduced LSH as a probabilistic technique suitable for solving the approximate K-NN problem [10, 8]. The original LSH hash function families were suitable only for Hamming space, but more recent families based on stable distributions and suitable for L_p, $p \in (0, 2]$ have been devised [4].

In our case of $\mathbb{R}^D$ with L_2 distance, the LSH family is defined as follows [4]:

$$H(v) = \langle h_1(v), h_2(v), \ldots, h_M(v)\rangle \tag{1}$$

$$h_i(v) = \lfloor \frac{a_i \cdot v + b_i}{W} \rfloor, \quad i = 1, 2, \ldots, M \tag{2}$$

where $a_i \in \mathbb{R}^D$ is a vector with entries chosen independently from the Gaussian distribution $N(0, 1)$ and b_i is drawn from the uniform distribution $U[0, W)$. For different i, a_i and b_i are sampled independently. The parameters M and W control the locality sensitivity of the hash function. The index data structure is L hash tables with independent hash functions, and the query algorithms is to scan the buckets which the query point is hashed to. As a data point might collide with the query point in more than one hash tables, a bitmap is maintained for each query to record the points scanned, so each data point is scanned at most once.

One drawback of the basic LSH scheme is that in practice it requires a large number of hash tables (L) to achieve good search quality. Panigrahy [18] proposed an entropy-based LSH scheme which reduced the required number of hash tables by using both the original query point and randomly perturbed, nearby points as additional queries. Lv, *et al.* [15] used a similar perturbation-based approach to develop Multi-probe LSH, which achieves the best results known so far. The study of this paper is based on Multi-probe LSH, which we review briefly below.

2.2 Multi-Probe LSH

Multi-probe LSH[15] is the current state of the art in LSH based schemes for nearest neighbor search. This scheme seeks to make better use of a smaller number of hash tables (L). To accomplish this goal, it not only considers the main bucket, where the query point falls, but also examines other buckets that are "close" to the main bucket.

For a single hash table, let v be the query point and $H(v)$ its hash. Recall that $H(v)$ consists of the concatenation of M integral values, each produced by an atomic hash function on v. Buckets corresponding to hash values that differ from $H(v)$ by ± 1 in one or several components are also likely to contain points near the original query point v. Buckets corresponding to hash values that differ from $H(v)$ by more than 1 in certain component are much less likely to contain points of interest[15], and are not considered.

Multi-probe LSH is to systematically probe those buckets that are closest to main bucket. For concreteness, consider the scenario shown in Figure 1 where $M = 3$. In this example, the query point is hashed to $\langle 5, 3, 9\rangle$. In addition to examining this main bucket, the algorithm also examines other buckets such as $\langle 6, 3, 9\rangle$, $\langle 5, 2, 9\rangle$ and $\langle 5, 3, 8\rangle$, which are close to the main bucket. Note that the closer the query point's hash value is to the boundary of the bin, the more likely it is that the bin bordering that boundary contains nearest neighbors of the query. In the above example, $\langle 5, 2, 9\rangle$ is the most promising of the additional buckets and the first of them to be examined.

In general, given a query point v and the main bucket $\pi_0 = \langle h_1, \ldots, h_M\rangle$, let the probing sequence be $\{\pi_0, \pi_1, \ldots, \pi_t, \ldots\}$, where $\pi_t = \langle h_1 + \delta_{t,1}, \ldots, h_M + \delta_{t,M}\rangle$ and $\langle \delta_{t,1}, \ldots, \delta_{t,M}\rangle$ is called the perturbation vector for step t. Because the chance of Kth nearest neighbor falling into buckets with $|\delta_{t,i}| \geq 2$ is very small, we restrict $\delta_{t,i}$ to the set $\{-1, 0, +1\}$ in order to simplify the algorithm. The buckets in the probing sequence are arranged in increasing order of the following query dependent score:

$$\text{score}(\pi_t) = \sum_{i=1}^{M} \Delta_i^2(\delta_{t,i}) \tag{3}$$

$$\text{where} \quad \Delta_i(\delta) = \delta \cdot [h_i + \frac{1}{2}(1 + \delta) - \frac{a_i \cdot v + b_i}{W}]. \tag{4}$$

$\Delta_i(\pm 1)$ is the distance from ith projection to the right/left window boundary for each i perturbed, and 0 for others, as illustrated in Figure 1.

Generating the probing sequence for each query is itself time consuming and so Multi-probe LSH uses a pre-calculated template probing sequence generated with the expected values of $\Delta_i(\pm 1)$ to approximate the query dependent probing sequence. For a specific query, its hash function components are ranked according to the $\Delta_i(\pm 1)$ values, and then adjusted according to the template probing sequence to produce the actual probing sequence used for the query. The precise details can be found in [15].

The length of the probing sequence T used by the algorithm is a further parameter to be tuned. Larger value of

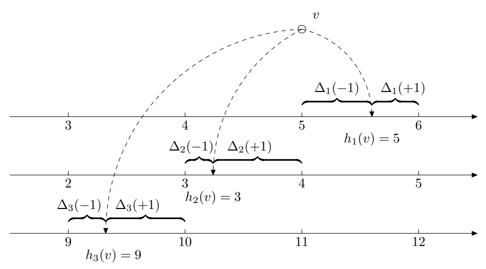


Figure 1: Illustration of LSH. The hash function consists of three components, each calculated by random projection and quantization. The point v is hashed to $H(v) = \langle 5, 3, 9\rangle$.

$S, N = \lvert S\rvert$	dataset and its size.
$I(v)$	true K-NNs of v.
$A(v)$	candidate set to be scanned.
X_k	distance to kth NN.
X	distance to an arbitrary point.
$H(\cdot) = \langle h_i(\cdot)\rangle_{i=1..M}$	the LSH function (1).
W	LSH window size (2).
M	# components in the LSH function.
L	# hash tables maintained.
T	# bins probed in each table.
$B_{t(,l)}$	tth bucket probed (in lth table).
$\Delta_i(\pm 1)$	distance to window boundary (4).
$\{\pi_t = \langle h_i + \delta_{t,i}\rangle_{i=1..M}\}$	the probing sequence.
$\eta_{W,\delta}(\cdot,\cdot)$	hash collision probability, see (12).
$\phi(\cdot)$	p.d.f. of Gaussian distribution $N(0, 1)$.

Table 1: Notation summary

T allow us to achieve the same quality of results with fewer hash tables. However, the likelihood of finding more relevant results falls rapidly as T increases. A very large value of T will only increase the candidate set size without significantly improving the quality of the results returned.

In summary, to achieve good performance from Multi-probe LSH, one needs to carefully tune four parameters: the window size W, the number of hash function components M, the number of hash tables L and the length of probing sequence T.

2.3 Other Related Works

A closely related work is LSH Forest [2], which represents each hash table by a prefix tree such that the number of hash functions per table can be adjusted. As new data arrive, the hash tables can grow on the fly. For a leaf node in the LSH forest, the depth of the node corresponds to parameter M in the basic LSH scheme. The method was designed for hamming distance. Although the idea may apply to L_2 distance with p-stable distribution-based hash functions, it must tune other parameters. Our data distribution modeling approach could be useful for this purpose.

The idea of intrinsic dimension has been used to analyze the performance of spatial index structures such as R-Tree [17]. The concepts in the paper have inspired our data model work, especially in the parameters of the gamma distribution and power law for modeling the distribution of distances.

3. ANALYTICAL MODEL

This section presents the performance model of Multi-probe LSH assuming fixed parameters, *i.e.* W, M, L and T. This model requires fitting a series of distributions that are dataset specific. Although there is not a universal family of such distributions, our experience indicates that the gamma distribution is commonly followed by multimedia data. For this special case, we then show how to extract statistical parameters from a small sample dataset to plug into the performance model.

Table 1 summarizes the notations we use in this paper.

3.1 Modeling Multi-Probe LSH

The first step is to formalize the performance measures. There are two aspects of performance — quality and cost. We use the percentage of true K-NNs found, or *recall*, to capture the quality of the query result. Formally, let v be a query point, $I(v)$ be the set of true K-NNs and $A(v)$ be the candidate set, which is the union of all the buckets probed. The approximate query result is the K elements of $A(v)$ closest to v (or the entire $A(v)$ if there are less than K points). Recall ρ is the percentage of $I(v)$ included in $A(v)$, or the following:

$$\rho(v) = \frac{|A(v) \cap I(v)|}{|I(v)|}. \tag{5}$$

We rank $A(v)$ and only return the best K points, thus precision is exactly the same as recall.

For cost, we are only interested in the online query time, as the space overhead and offline construction time are both linear to the dataset size and the number of hash tables (L). The main part of query processing is scanning through the candidate set and maintaining a top-K heap. Because most candidate points, being too far away from the query, are thrown away immediately after distance evaluation, heap updates rarely happen. Therefore, the query time is mostly spent on computing distances, and is thus proportional to the size of the candidate set. As a result, the percentage of the whole dataset scanned by LSH is a good indicator of query time. This is further justified by experiments in Section 5. We thus define *selectivity* $\tau(v) = |A(v)|/N$, the size ratio between the candidate set and the whole dataset, as the measure of query time. We use the average recall and selectivity as overall performance measure. Our performance model is to estimate these two values by their mathematical expectation: $\rho = E[\rho(v)]$ and $\tau = E[\tau(v)]$. If not otherwise stated, expectations in this paper are taken over the randomness of the data as well as the random parameters of the LSH functions, *i.e.* a_i and b_i in (2).

The probability that an arbitrary point u in the dataset is found in the candidate set $A(v)$ is determined by its distance to the query point v. We represent this distance with random variable X, and define the probability as a function of X, which we also call recall and use the same symbol ρ:

$$\rho(X) = \Pr[u \in A(v) \mid \|u - v\| = X\,]. \tag{6}$$

With the above definition of $\rho(.)$ as a function of distance, the expected selectivity can be written as

$$\tau = E[\tau(v)] = E\left[\frac{1}{|S|} \sum_{\substack{x=||u-v|| \\ u \in S}} \rho(x)\right] = E[\rho(X)]. \tag{7}$$

The same reasoning applies to the nearest neighbors. Let $X_k = ||I_k(v) - v||$ be the distance between v and its kth nearest neighbor, then $\rho(X_k)$ is the recall of v on its kth nearest neighbor, and the overall expected recall can be obtained by taking the average

$$\rho = E[\rho(v)] = \frac{1}{K}\sum_{k=1}^{K} E[\rho(X_k)]. \tag{8}$$

So far the modeling problem is reduced to finding the distributions of X and X_k, which we will address in the next subsection, and finding the detailed expression of $\rho(\cdot)$, as explained below.

To obtain the expression of $\rho(\cdot)$, we need to decompose the candidate set $A(v)$ into the individual buckets probed. Let $B_{l,t}(v), 1 \le l \le L, 1 \le t \le T$ be buckets in the probing sequences of the L hash tables. Because the hash tables are maintained by the same algorithm in parallel, with hash functions sampled from the same family, the subscript l actually does not matter when only probabilities are considered, thus we use $B_t(v)$ to denote the tth bucket for arbitrary l. It is obvious that $A(v) = \bigcup_{l,t} B_{l,t}(v)$. Let u be an arbitrary point such that $X = \|u - v\|$, then

$$\rho(X) = 1 - \Pr[u \notin \bigcup_{l,t} B_{l,t}(v)|] = 1 - \left\{\prod_{t=1}^{T}\Pr[u \notin B_t(v)]\right\}^L. \tag{9}$$

Recall that the corresponding hash function of the B_t in the probing sequence is $\langle h_1(v) + \delta_{t,1}(v), \dots, h_M(v) + \delta_{t,M}(v)\rangle$. Thus

$$u \notin B_t(v) \Leftrightarrow \neg \bigwedge_{1\le i\le M} h_i(u) = h_i(v) + \delta_{t,i}(v)$$

and

$$\Pr[u \notin B_t(v)] = 1 - \prod_{1\le i\le M}\Pr[h_i(u) = h_i(v) + \delta_{t,i}(v)]. \tag{10}$$

The probability $\Pr[h_i(u) = h_i(v) + \delta_{t,i}(v)]$ depends on the perturbation value $\delta_{t,i}(v)$ as well as v's distance to the corresponding window boundary on the ith projection. The detailed expression depends on the specific atomic hash function, which is (2) in our case for L_2 distance. We directly give the formula here:

$$\Pr[h_i(u) = h_i(v) + \delta] = \eta_{W,\delta}[X, \Delta_i(\delta)] \tag{11}$$

$$\text{where} \quad \eta_{W,\delta}(d,z) = \begin{cases} \frac{1}{d}\int_0^W \phi(\frac{z-x}{d})\mathrm{d}x & \text{if } \delta = 0 \\ \frac{1}{d}\int_0^W \phi(\frac{z+x}{d})\mathrm{d}x & \text{if } \delta = \pm 1 \end{cases} \tag{12}$$

where $\phi(\cdot)$ is the probability density function of the standard Gaussian distribution. Because $\eta_{W,0}(d,z)$ occurs frequently, we use the following approximation to simplify computation:

$$\eta_{W,0}(d,z) \sim E_{z\in[0,W)}[\eta_{W,0}(d,z)]. \tag{13}$$

The values $\Delta_i(\delta)$ are functions of the query v and the hash function parameters. We make some approximations to simplify calculations by taking advantage of the template probing sequence. First, note that the actual order of the M components of the hash function does not affect the evaluation of (10). Without loss of generality, we can assume that the components are ordered by their minimal distance to window boundary, as in the template probing sequence. Second, the value of $h_i(u)$ and $h_i(v)$ do not appear in (11) and only $\Delta_i(\delta)$ is interesting. Finally, we use the expected values of $\Delta_i(\delta)$ instead of their actual values:

$$\begin{aligned} \Delta_i(+1) &= E[U_{(i)}] = \frac{i}{2(M+1)} \\ \Delta_i(-1) &= 1 - E[U_{(i)}]. \end{aligned} \tag{14}$$

These assumptions would allow us to calculate the probabilities directly from the template probing sequence.

The performance model of both recall and selectivity is given by (7–14).

3.2 Modeling Data Distribution

To apply the performance model to real datasets, we still need to determine a series of distributions: the distribution of the distance X between two arbitrary points, and the distributions of the distance X_k between an arbitrary query point and its kth nearest neighbor. These distributions are dataset specific and there is not a universal family that fits every possible dataset. However, we show that many existing multimedia datasets do fit a common family — the gamma distribution. In this subsection, we show how to extract statistical parameters from a small sample dataset and do parameter estimation specific to the gamma distribution.

A previous study [20] has shown with multiple datasets that the distribution of L_1 distance between two arbitrary points follows the log-normal distribution without giving any intuitive explanation. Actually a log-normal variable is conceptually the multiplicative product of many small independent factors, which can not be easily related to distance distributions. In this paper, we propose to fit the squared L_2 distances, both X^2 and X_k^2, by the gamma distribution, whose probability density function is

$$f_{\kappa,\theta}(x) = \alpha(\frac{x}{\theta})^\kappa e^{-x/\theta} \tag{15}$$

where κ is the shape parameter, θ is the scale parameter, as it always appears as the denominator under x, and α is the normalizing coefficient such that the integral of the function over R^+ is 1. The gamma distribution and log-normal distribution have similar skewed shapes and are usually considered as alternatives. However, for our purpose, the gamma distribution has the following advantages.

First, it fits many multimedia datasets. Figure 2 and 3 show that both X^2 and X_k^2 can be accurately fitted (see Section 5.1 for detailed description of the datasets). Various other examples exist, but are not shown due to the page limit.

Second, the gamma distribution has an intuitive explanation which is related to the dataset's intrinsic dimensionality. Assume that the feature vector space can be embedded into R^D, D being the intrinsic dimensionality. For two points u and v, assume the difference of each dimension, $u_i - v_i$, follows an identical but mutually independent Gaussian distribution, then $||u - v||^2 = \sum_{i=1}^{D}(u_i - v_i)^2$ is the sum of D squared Gaussian variables, which can be proved to follow χ^2 distribution, a special case of gamma distribution. The parameter κ in the distribution is determined by the dimensionality of the space. When D is integer, we have the relationship $D = 2(\kappa + 1)$. Because gamma distribution does not require κ to be integer, we extend this to the non-integer case, and call $2(\kappa + 1)$ the *degree of freedom* of the distribution, which we expect to capture the intrinsic dimensionality of the datasets. A dataset with higher degree

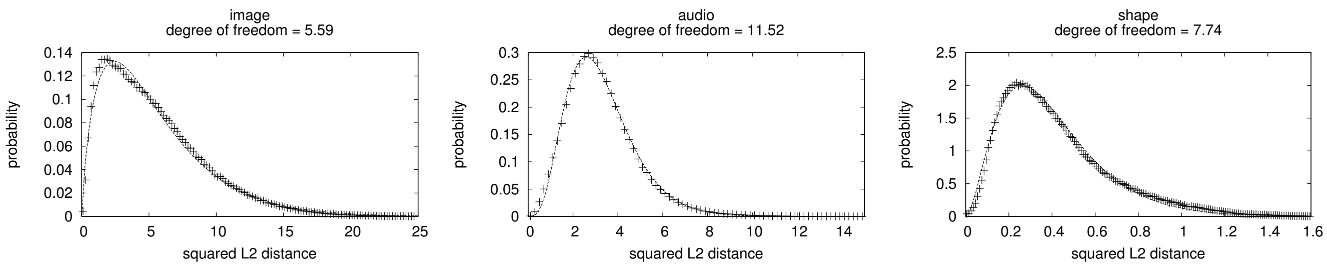


Figure 2: The squared distance between two arbitrary points (X^2) follows the gamma distribution.

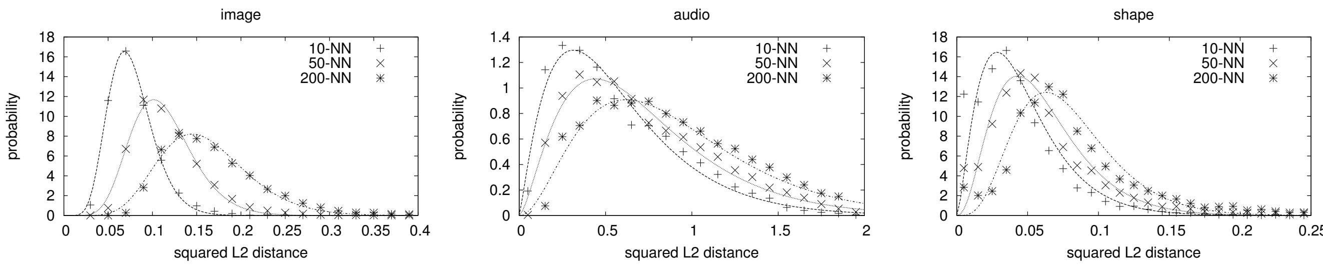


Figure 3: The squared distance between the query point and kth NN (X_k^2) follows the gamma distribution

of freedom will be harder to index. The relative order of the three dataset with regard to degree of freedom matches the experimental results in Section 5.

Finally, there exists an effective method to estimate the distribution parameters. The parameters of gamma distribution can be estimated by Maximum Likelihood Estimation (MLE), and depends only on the arithmetic mean E and geometric mean G of the sample, which can be easily extracted from the dataset. Given E and G, κ and t can be solved from the following set of equations.

$$\begin{cases} \kappa\theta = E \\ \ln(\kappa) - \psi(\kappa) = \ln(E) - \ln(G) \end{cases} \tag{16}$$

where $\psi(x) = \Gamma'(x)/\Gamma(x)$ is the digamma function.

To obtain the estimation of the distributions of X^2 and X_k^2, we need to calculate from the sample set the corresponding arithmetic and geometric means of these random variables. For X^2, the means E and G can be obtained simply by sampling random pairs of points. For X_k^2, we need E_k and G_k for each k under consideration. As the total number K of nearest neighbors can be big, maintaining K pairs of means is not practical. Further more, the distribution of X_k^2 depends on the size N of the dataset. At the performance modeling stage, there is usually only a small subset of the whole dataset available. In such case, we need to extrapolate the parameters according to the available data. Pagel *et al.* [17] studied the expected value of X_k as a function of k and N, and proved the following relationship:

$$E[X_k] = \frac{[\Gamma(1+\frac{D_1}{2})]^{\frac{1}{D_1}}}{\sqrt{\pi}} \left(\frac{k}{N-1}\right)^{\frac{1}{D_2}}. \tag{17}$$

where D_1 and D_2 are the embedding dimensionality and intrinsic dimensionality, respectively, and N is the size of dataset. Based on this result, we propose to empirically model both E_k and G_k of X_k^2 as power functions of both k and N:

$$E_k = \alpha k^{\beta} N^{\gamma} \qquad G_k = \alpha' k^{\beta'} N^{\gamma'} \tag{18}$$

To extract the parameters, a subset of the dataset is sampled as anchor points, and subsets of various sizes are sampled from the rest of points to be queried against. The K-NNs of all the anchor points are found by sequential scan, resulting in a sample set of X_k for different values of k and N. We then obtain the six parameters of (18) via least squares fitting. As shown by Figures 4 and 5 (for now, disregard the curve "arith. mean + std") this power law modeling is very precise.

The fact that for fixed k, X_k is a power function of N is very important for practical system design. It indicates that X_k changes very slowly as dataset grows, the optimal LSH parameters should also shift very slowly as data accumulate This allows us to keep high performance of LSH by only reconstructing the hash tables when dataset doubles, and this kind of reconstruction could be achieved with low amortized cost.

With the above method, 8 parameters are collected in total. Given the full dataset size to be indexed, we can obtain via MLE the distribution function $f(.)$ of X^2, and distribution functions $f_k(.)$ of X_k^2. The performance of LSH is then calculated by

$$\rho = \frac{1}{K}\sum_{k=1}^{K}\int_0^{\infty} \rho(\sqrt{x}) f_k(x)\mathrm{d}x \tag{19}$$

$$\tau = \int_0^{\infty} \rho(\sqrt{x}) f(x)\mathrm{d}x \tag{20}$$

4. APPLICATIONS OF THE MODEL

This section presents two applications of the performance model. First is parameter tuning for optimal average performance and second is adaptive probing, which dynamically determines for each query how many buckets to probe at runtime.

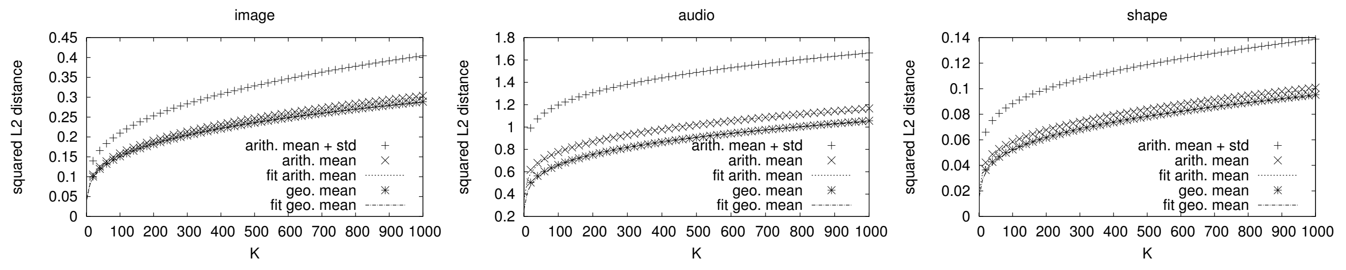


Figure 4: Arithmetic and geometric means of squared k-NN distance (X_k^2) follow the power law with regard to k.

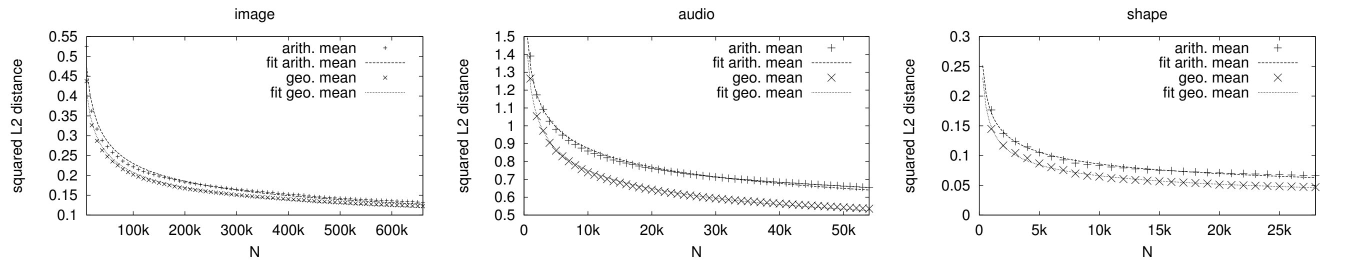


Figure 5: Arithmetic and geometric means of squared k-NN distance (X_k^2) follow the power law with regard to N. Here we use $k = 50$.

4.1 Offline Parameter Tuning

There are four parameters related to LSH. Among them, W, M and L need to be fixed when creating the hash tables, and T is related to the query processing algorithm and can either be fixed offline, or change from query to query. According to our model, larger L results in higher recall with the same selectivity, thus L should be tuned to the maximal affordable value, as limited by the storage available. Note that in practice if L is really high ($L >> 10$, which is not likely to happen for large datasets), query time will again start increasing as the cost to generate the probing sequences becomes dominant.

We would like to tune W and M for optimal when creating the hash tables, while having T adaptively determined at query time. However, our model requires a fixed T value to predict the performance, and thus to tune W and M. As a workaround, we choose a fixed T value of medium size (adding an equation $T = M$ is a good choice, as the first few buckets are the most fruitful[15]) to tune for (near) optimal W and M, and again determine the T value for each query online (to be explained in next subsection). The optimization problem is as follows:

$$\begin{aligned} \text{min.} \quad & \tau(W, M) \\ \text{s.t.} \quad & \rho(W, M) \geq \text{required value.} \end{aligned} \tag{21}$$

In our implementation, we use the following simple method to solve this problem. Assume M is fixed, the relationships between W and both recall and selectivity are monotonic (see Figure 7A-C), and the optimal W can be found using binary search. We then enumerate the value of M from 1 to some reasonably large value max (30 for our datasets) to obtain the optimal. On a machine with Pentium 4 3.0GHz CPU, our code runs for no more than one minute for each of the datasets we have.

4.2 Adaptive Query Processing

In the previous subsection, we tuned M and W for optimal average performance. However, for the following reason, we do not want to determine the T value offline as first proposed in [15]: even if we tune for optimal average performance, the actual performance can be different from query to query. That is because recall and selectivity are both determined by the local geometry of the query point. The "arith. mean + std" curves in Figure 4 show a large standard deviation, which means that the local geometry of different query points can be very different. For each specific query, probing the default T buckets can be either too few or too many to achieve the required recall. In this subsection, we address this problem by the adaptive probing method.

The basic idea of adaptive probing is simple: maintain an online prediction of the expected recall of current query, and keep probing until the required value is reached. If we knew the true K-NN distances precisely, we can predict the expected recall with our performance model. The problem is then turned to predicting the true K-NN distances by the partial result when processing the query, and refine the prediction iteratively as new buckets are probed. A natural approach is to use the partial results themselves as approximations of the true K-NNs. This approximation actually has an advantage that the estimated values are always larger then the real values, and the estimated expected recall is thus always lower than the real value. Also, as the probing sequence goes on, the partial K-NNs will quickly converge to the real ones. In practice, if one is to tune for 90% recall, the estimated value should be more or less the same as the true one.

Adaptive probing requires calculating the current expected recall after each step of probing and this computation is time consuming. To prevent this from slowing down the query process, a small lookup table is precomputed to map K-NN

Dataset	# Feature Vectors	Dimension
image	662,317	14
audio	54,387	192
3D shape	28,775	544

Table 2: Dataset summary

distances to expected recalls, for each different T value. Our experiments show that the run time overhead of using this lookup table is minimal.

5. EVALUATION

In this section, we are interested in answering the following two questions by experimental studies:

1. How accurate is our model in predicting LSH performance, when the model parameters are obtained from a small portion of the whole dataset?
2. How does our adaptive probing method improve over the fixed method?

The evaluation of our methods with three real-life datasets gives satisfactory answer to these questions.

5.1 Datasets

We employ three datasets to evaluate our methods: images, audio clips and 3D shapes. Table 2 provides a summary of them. These datasets are chosen to reflect a variety of real-life use cases, and they are of different number of dimensions, from tens to hundreds. Experiments with a couple of other datasets have shown equally good results, but are not shown here due to the space limit.

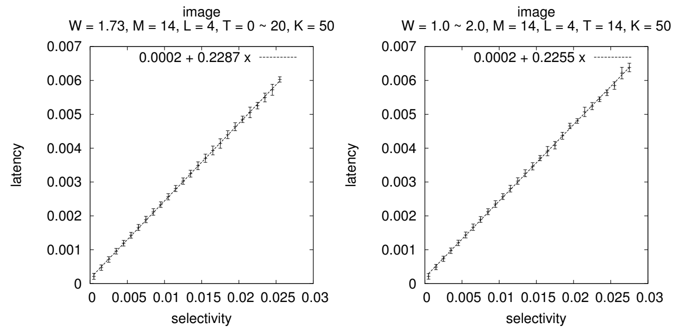
Image Data: The image dataset is drawn from the Corel Stock Photo Library, a dataset for evaluating content-based image retrieval algorithms. For feature extraction, we use JSEG [5] to segment the images into regions and use the method in [13] to extract a feature vector from each region. The feature vector is of 14 dimensions, among which, 9 is for color moments and 5 is for shape and size information. There are 66,000 images in the dataset, and each image is segmented into roughly 10 regions, resulting in 662,317 feature vectors in total.

Audio Data: The audio dataset is drawn from the DARPA TIMIT collection [7]. The TIMIT collection is an audio speech database that contains 6,300 English sentences spoken by 630 different speakers with a variety of regional accents. We break each sentence into smaller segments and extract features from each segment with the Marsyas library [19]. There are 54,387 192-dimensional feature vectors in total.

Shape Data: The third dataset we use in our study contains about 29,000 3D shape models, which is a mixture of 3D polygonal models gathered from commercial viewpoint models, De Espona Models, Cacheforce models and from the Web. Each model is represented by a single Spherical Harmonic Descriptor(SHD) [12], yielding 28,775 544-dimensional feature vectors in total.

5.2 LSH Model Evaluation

First of all, we need to justify modeling query latency with selectivity τ under the assumption that the most of the query time is spent scanning the candidate data points.



CPU is Intel P4 Xeon 3.2GHz and all data fit in main memory.

Figure 6: Latency vs. selectivity. The different selectivities in the two figures are obtained by varying T and W respectively. The matching of the two figures confirms the reliability of selectivity as a proxy of latency.

Since τ is not a tunable input parameter, we conduct two different experiments by varying different input parameters for the image dataset. With all other parameters fixed, one experiment changes τ by using different window sizes (W), and the other by using different probing sequence lengths (T). For each configuration of parameters, 1,000 queries for 50-NN are executed, with latency and selectivity τ recorded. These numbers are then binned by τ, and the average and the standard deviation of the latencies in each bin are calculated and plotted. Our results are shown in Figure 6, the height of the error bar representing two standard deviations. The linear relationship between query time and τ is obvious.

Strictly speaking, apart from the time spent on scanning the candidate set, there is a tiny cost to initialize the query data structure, and a tiny cost to locate the bucket to scan at each probing step. If these costs are not small enough, they should be seen in the plots. The former should create a non-zero y-intercept, and the latter will make the slopes of the two curves different. Figure 6 shows a minimal initial cost and a very small divergence between the slopes of the two curves which can be safely ignored in practice. As a result, it is safe to use selectivity as the machine-independent time cost instead of latency which is machine-dependent.

We then go on to evaluate the accuracy of the model itself. As the model involves four input parameters, *i.e.* W, M, L and T, and two output parameters, *i.e.* recall ρ and selectivity τ, it is hard to evaluate an overall accuracy. Also, not every point in the input parameter space is equally interesting because only the parameter configurations that result in high recall and low selectivity are of practical importance. We thus design our experiments in the following way. First, a set of baseline parameters are chosen for each dataset, which achieves about 90% recall with a reasonably low selectivity. Then each time we fix all but one of the four parameters and change it around the baseline value. For each configuration of the parameters, we build LSH data structure with the whole dataset, and run 50-NN queries for 1,000 randomly sampled query points. The average of the recall and selectivity are recorded and plotted. We also predict the average recall and selectivity with our performance model. The data parameters are extracted from one 10th the whole dataset according to the method discussed in Section 3.2.

The effects of the four parameters on the three datasets are shown in Figure 7A-L, with error bars representing two standard deviations. According to the figures, the predicted values are close to the real experimental values, and most of the errors are within one standard deviation. Our prediction of recall values is especially accurate. The error is within 5% the actual recall value for most cases. When $M > 15$, the error ratio is a little high, but these cases have very low recall, and are not interesting in practice. Further more, our predictions correctly follow the trends of the actual curves. This implies that our model is reliable for real system design and the automatically tuned parameters are close to optimal.

By experimenting with the parameters one by one while leaving the others fixed, we also see the performance impact of each parameter.

5.3 Adaptive Query Evaluation

In this subsection, we first conduct an experiment to show the impact of local geometry around query points on the performance, and demonstrate the advantage of adaptive method over the original method with fixed T, which we call the *fixed method*. We then show by another experiment that the adaptive method is capable of working with different K values at query time. Note that so far our discussion has been assuming a fixed K value.

The local geometry is best described by local K-NN distances (X_k as in the model), and we want to see how LSH performs for query points with different K-NN distances. For each dataset, we sample 1,000 queries at random, and group them into about 30 bins according to the distances to the 50th nearest neigbhor (X_{50}). We then run both fixed and adaptive versions of the query algorithm, and plot the average recall and selectivity for each bin. The parameters M and W are tuned for 90% recall of 50-NN queries, assuming $T = M$. For adaptive method, T is again dynamically determined at query time. The results are shown in Figure 8, and there is an obvious difference between the behaviors of the two methods. These results are better understood when compared with Figure 3, which shows the probability distribution of K-NN distance. The average and standard deviations of the recall and selectivity values can also be found in the $K = 50$ rows of Table 3.

Both recall and selectivity of the fixed method drop dramatically as K-NN distance grows beyond certain point. This is because LSH is tuned to perform well for average K-NN distance, which is relatively small. For queries with larger K-NN distance, both the nearest neighbors and the background points have smaller chances of falling into the probed buckets, and thus the fixed method gets excessively low recall. To compensate this effect, the fixed method has to achieve exceptionally high recall, close to 100%, with the easy cases. Because selectivity grows faster at higher recall values (see Figure 7), such compensation will result in higher overall cost.

The adaptive method, however, is able to probe more buckets for those difficult points and achieve a high recall to meet the requirement. As a result, the adaptive method significantly reduces the variance of recall and better meets the quality requirements for individual queries. For the difficult queries on the right end of the curves, it costs more than average for the adaptive method to achieve the required recall.

			recall/stdev (%)			selectivity (%)	
Data	K	T	fixed	adaptive	reduction	fixed	adap.
Image	10	8	94/12	95/08	39%	0.3	0.3
	50	14	91/12	95/06	51%	0.5	0.5
	100	22	90/12	95/05	57%	0.6	0.6
Audio	10	10	95/13	98/05	58%	22	17
	50	14	94/12	98/04	70%	25	21
	100	20	95/11	98/03	68%	28	23
Shape	10	7	97/09	97/07	23%	11	08
	50	14	97/09	96/05	45%	16	11
	100	22	97/09	96/04	49%	19	12

Parameters W,M and L are the same as in Figure 8. The T values shown are only for fixed method.

Table 3: Adaptive vs. fixed method on dealing with different K requirement. To achieve the same recall, the fixed method needs to have T tuned for different Ks, while the adaptive method does not. We can also see a significant reduction of recall standard deviation by the adaptive method.

Our second experiment is to show that the adaptive method works with different K values better compared to the fixed method, which is tuned for a single K value. To compare the two methods, we use the same hash tables constructed with the parameters tuned for 50-NN queries, the same as those used in Figure 8. And we use these tables to answer queries for 10, 50, 100-NNs to see how the two methods behave. For the fixed method, we use our model to pre-calculate the T needed for each of the three cases so as to achieve 90% recall on average (as shown in the third column of Table 3) , and for the adaptive method, T of each query is dynamically determined. For each configuration of parameters, we run 1,000 queries and take the average recall and selectivity as well as the standard deviation. The results are shown in Table 3.

As we can see from the results, the adaptive method significantly reduces the performance variation between different queries. The standard deviation reductions for recall are 50%, 65% and 40% for three datasets respectively, around 50% on average. Also in most cases, the adaptive method produces a higher recall than the fixed method, with a lower selectivity.

The above two experiments allow us to demonstrate the effectiveness of our adaptive query process method on reducing the variation of performance among different queries as well as the average cost to achieve the same recall.

6. CONCLUSION

Our study shows that it is possible to model Multi-probe LSH and data distribution accurately with small sample datasets and use the models for automatic parameter tuning in real implementations.

We have proposed a performance model for multi-probe LSH and a data model to predict the distributions of K-NN distances in a dataset. Our experiments with three datasets show that the models fitted with small sample datasets are accurate; the recalls are within 5% of the real average values for most cases.

We have derived an adaptive search method based on the performance model to reduce performance variance between different query points. Our experimental results show that the adaptive method can reduce the standard deviation of recalls by about 50%, while achieving the same recall with lower latency.

We have implemented the automatic tuning in the toolkit which will be made available to the public domain[1].

Acknowledgments

This work is supported in part by NSF grants EIA-0101247, CCR-0205594, CCR-0237113, CNS-0509447, DMS-0528414 and by research grants from Google, Intel, Microsoft, and Yahoo!. Wei Dong is supported by Gordon Wu Fellowship.

7. REFERENCES

[1] `http://www.cs.princeton.edu/cass`.

[2] M. Bawa, T. Condie, and P. Ganesan. Lsh forest: self-tuning indexes for similarity search. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 651–660, New York, NY, USA, 2005. ACM.

[3] J. L. Bentley. K-d trees for semidynamic point sets. In *SCG '90: Proceedings of the sixth annual symposium on Computational geometry*, pages 187–197, New York, NY, USA, 1990. ACM.

[4] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *SCG '04: Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262, New York, NY, USA, 2004. ACM Press.

[5] Y. Deng and B. Manjunath. Unsupervised segmentation of color-texture regions in images and video. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 23(8):800–810, August 2001.

[6] D. Dobkin and R. J. Lipton. Multidimensional searching problems. *SIAM Journal on Computing*, 5(2):181–186, 1976.

[7] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, and N. L. Dahlgren. DARPA TIMIT acoustic-phonetic continuous speech corpus, 1993.

[8] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB '99: Proceedings of the 25th International Conference on Very Large Data Bases*, pages 518–529, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[9] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *SIGMOD '84: Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pages 47–57, New York, NY, USA, 1984. ACM.

[10] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, New York, NY, USA, 1998. ACM.

[11] N. Katayama and S. Satoh. The sr-tree: an index structure for high-dimensional nearest neighbor queries. In *SIGMOD '97: Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pages 369–380, New York, NY, USA, 1997. ACM.

[12] M. Kazhdan, T. Funkhouser, and S. Rusinkiewicz. Rotation invariant spherical harmonic representation of 3d shape descriptors. In *SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 156–164, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[13] Q. Lv, M. Charikar, and K. Li. Image similarity search with compact data structures. In *CIKM '04: Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 208–217, New York, NY, USA, 2004. ACM.

[14] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Ferret: a toolkit for content-based similarity search of feature-rich data. In *EuroSys '06: Proceedings of the ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, pages 317–330, New York, NY, USA, 2006. ACM.

[15] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-probe lsh: Efficient indexing for high-dimensional similarity search. In *VLDB '07: Proceedings of the 24rd International Conference on Very Large Data Bases*, Vienna, Austria, 2007.

[16] S. Meiser. Point location in arrangements of hyperplanes. *Inf. Comput.*, 106(2):286–303, 1993.

[17] B.-U. Pagel, F. Korn, and C. Faloutsos. Deflating the dimensionality curse using multiple fractal dimensions. In *ICDE '00: Proceedings of the 16th International Conference on Data Engineering*, page 589, Washington, DC, USA, 2000. IEEE Computer Society.

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

[19] G. Tzanetakis and P. Cook. Marsyas: a framework for audio analysis. *Organized Sound*, 4(3):169–175, 1999.

[20] Z. Wang, W. Dong, W. Josephson, Q. Lv, M. Charikar, and K. Li. Sizing sketches: a rank-based analysis for similarity search. In *SIGMETRICS '07: Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 157–168, New York, NY, USA, 2007. ACM.

[21] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB '98: Proceedings of the 24rd International Conference on Very Large Data Bases*, pages 194–205, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

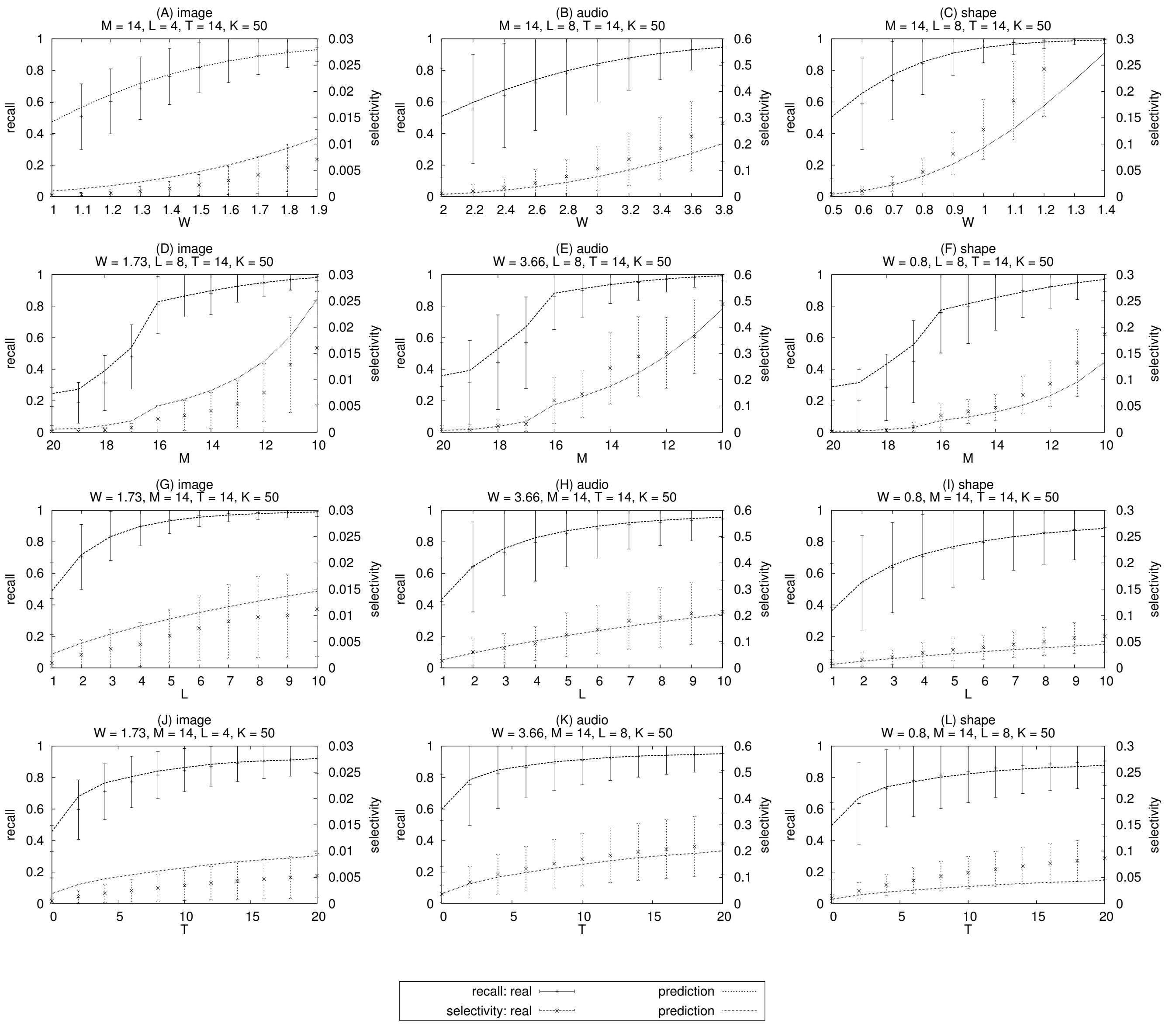


Figure 7: Recall and selectivity vs. W, M, L & T respectively. The predicted values are close to the average experimental results.

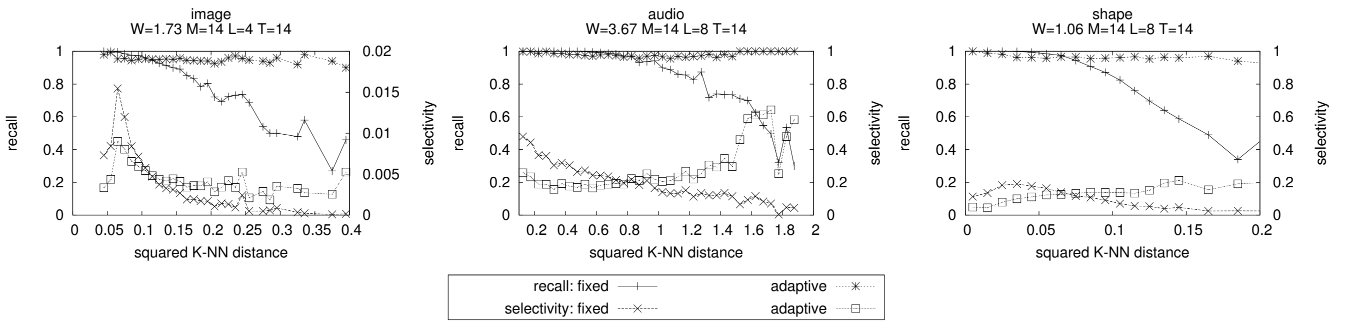


Figure 8: Adaptive vs. fixed method in terms of recall and selectivity. The recall of the fixed method degrades as distance increases, while the adaptive method performs much more consistently. For the queries with large distance, the adaptive method does more probes to achieve high recall.