High-Dimensional Similarity Search for Large Datasets

Wei Dong

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
by the Department of
Computer Science
Adviser: Kai Li

November 2011

Abstract

The volume of images and other non-text data is growing exponentially in today's digital universe. A popular way of extracting useful information from such data is to conduct content-based similarity search, e.g., to search for images with content similar to a query image. How to build data structures to support efficient similarity search on a large scale is an issue of increasing importance. The challenge is that feature-rich data are usually represented as high-dimensional feature vectors, and the *curse of dimensionality* dictates that as dimensionality grows, any search strategy examines an increasingly large portion of the dataset and eventually degenerates to brute-force scan. In this dissertation, we study several key issues to improve the accuracy and efficiency of high-dimensional similarity search. Specifically, we make the following contributions:

First, we enhance Multi-Probe LSH, the state-of-the-art high-dimensional similarity search technique, by developing an accurate performance model that predicts search accuracy with an error rate of less than 5% using only a 1/10 sample of the whole dataset. We use this model to realize automatic performance tuning, which would otherwise be tedious, and to develop an adaptive query processing strategy to ensure the high search quality of each individual query.

Second, we develop an efficient offline method for simultaneously finding the K nearest neighbors of every point in the dataset. Our method is 2 to 16 times faster than the state-of-the-art technique when evaluated with different datasets. Furthermore, we show how to use the offline computed nearest-neighbor information to double the speed of online similarity search.

Third, we develop compact representations of high-dimensional feature vectors optimized for similarity search tasks. We present an asymmetric distance estimation framework to exploit the information in the uncompressed query data. We use that to further compress the indexed dataset, by 10% to 40%, without compromising search

quality.

Fourth, we develop a scheme to compactly represent sets of feature vectors, an increasingly popular data representation that is more accurate than single vectors, but also more expensive. Our method dramatically reduces the matching cost in both space and time. In an image classification task, our method achieves 25 times speedup over one of the best existing techniques.

Finally, we demonstrate some of the proposed techniques by building a large-scale near-duplicate image search engine. Our system serves more than 50 million web images on a single commodity server and responds to queries within a few seconds.

Acknowledgements

I would like to thank the following people:

My advisor Kai Li for giving me the opportunity to work on the exciting topic of similarity search, and for his guidance and encouragement throughout the course of this study.

My official readers Andrea LaPaugh and Moses Charikar for carefully reading several drafts of this dissertation and providing helpful suggestions, and Moses for collaborating in the development of each component of this dissertation.

The other three professors who served in my thesis committee: Brian Kernighan, Olga Troyanskaya and Tom Funkhouser.

My colleague Zhe Wang for inspiring discussion, data collection and infrastructure maintenance, and for being a great friend.

Jia Deng for help collecting the ground truth images used in the experiments of Chapter 7.

My wife Yuanfang Guan and my parents for their love and support.

This work was supported in part by Gigascale Systems Research Center, Google Research Grant, Yahool Research Grant, Intel Research Council, National Science Foundation grants CSR-0509447 and CSR-0509402, and Gordon Wu Fellowship.

The materials of Chapter 2—5 have been used in the following publications and have been presented in the corresponding conferences:

[Chapter 2] Wei Dong, Zhe Wang, William Josephson, Moses Charikar, and Kai Li. Modeling LSH for performance tuning. In *CIKM '08: Proceedings of the 17th ACM Conference on Information and Knowledge Management*. ACM, New York, NY, USA, 669-678.

[Chapter 3] Wei Dong, Moses Charikar, and Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *WWW '11: Proceedings of the 20th International Conference on World Wide Web*. ACM, New York, NY, USA, 577-586.

[Chapter 4] Wei Dong, Moses Charikar, and Kai Li. Asymmetric distance estimation with sketches for similarity search in high-dimensional spaces. In *SIGIR '08: Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, New York, NY, USA, 123-130.

[Chapter 5] Wei Dong, Zhe Wang, Moses Charikar, and Kai Li. Efficiently matching sets of features with random histograms. In *MM '08: Proceedings of the 16th ACM International Conference on Multimedia*. ACM, New York, NY, USA, 179-188.

Contents

Chapter 1

Introduction

In this chapter, we introduce the problem of high-dimensional similarity search in the context of its main application: building a system to efficiently search data objects according to content similarity. We formally define the problem and demonstrate the dimensionality challenge. After that, we briefly review the history and state of the art and summarize the contributions we make in this dissertation.

1.1 Application: Content-Based Similarity Search

The world is producing data at an exponentially increasing speed (Figure 1.1). It was estimated in 2008 [27], and later confirmed in 2011 [28], that in the past five years (2006—2011), yearly worldwide data production grew from 180 exabytes [1] to 1,800 exabytes. How to effectively manage and make use of these data is one of the fundamental problems people need to solve in the information age. Searching is a common way people extract useful information from data. Various techniques for searching particular data types have been developed over the past decades. Most successful among these techniques are the relational database management system

[1] 1 exabyte = 1,000,000,000,000,000,000 bytes.

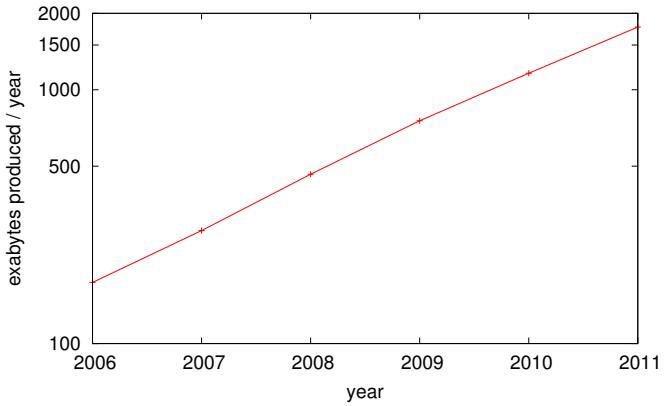


Figure 1.1: Yearly amount of data produced worldwide from 2006 to 2011. Data source: IDC [27].

for structured data (tables) [2] and the web search engine for unstructured web text data. However, tables and text cover only a small fraction of all the data the world is producing, while predominant in today's digital universe are non-text, feature-rich data like images, audio and video [27]. Unfortunately, a generally applicable system to search such data on a large scale is yet to be developed.

It is generally agreed that the search paradigm for such feature-rich data is to search by example, i.e., the user submits a query object, and the system returns objects that are similar to the query. This paradigm is entirely different from either Structured Query Language (SQL) for relational databases or keyword-based search for web search engines, both with known scalable solutions. Even though over the years researchers and companies have built many systems to search feature-rich data on small to medium scales, due to reasons we will shortly explain, constructing such systems for web-scale datasets remains a formidable task.

We start by explaining how a general content-based search system works, so as to define the application context of the core similarity search problem we try to address in this dissertation. Figure 1.2 shows the data flow in a general content-based search system. Depending on the design choices, one or more components may be absent

[2] A database management system has many functions, search being an important one of them.

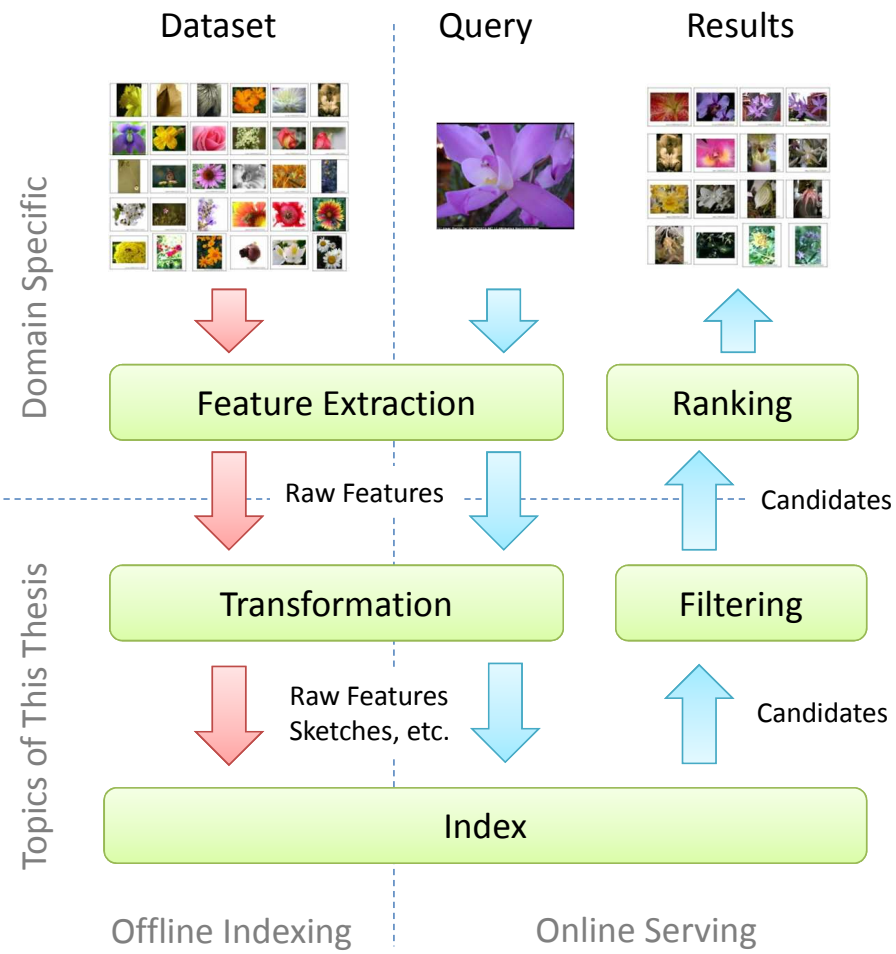


Figure 1.2: Data flow in a general content-based search system.

from a real search system. The full system is made up of two subsystems: an offline indexing subsystem for preprocessing and indexing the dataset, and an online query processing subsystem for answering user queries with the offline-built index. Next we explain in more detail what happens in these subsystems.

The dataset to be searched against needs to be preprocessed and indexed by the offline indexing subsystem. This includes the following stages:

1. **Feature extraction.** Domain expertise is used to extract feature data from data objects. Feature data are usually in the form of single high-dimensional vectors or sets of vectors. They can be used to evaluate the similarity between data objects. For example, a popular feature for finding visually similar images is the color histogram.

2. **Feature transformation.** The raw feature data, despite containing the most information, could be too large for efficient processing. In some systems, raw feature data are converted to spatially efficient approximate internal representations, e.g., sketches discussed in Chapter 4 and random histograms discussed in Chapter 5.

3. **Indexing.** The transformed and/or raw feature data are organized into the index, a data structure to speedup online search.

After the index is built, it can be loaded onto the query processing subsystem to answer user queries. Typically a query is answered in the following steps:

1. **Feature extraction and transformation.** These two steps are the same as in offline processing.

2. **Index lookup.** The search scope is quickly narrowed down to a *candidate set* which is a small fraction of the whole dataset that is likely to contain the search targets.

3. **Filtering.** The candidates are scanned through with the space-efficient internal representation, and items that can be assessed as irrelevant are discarded.

4. **Ranking.** The remaining high-quality candidates are ranked with raw feature data, and the highest-ranked ones are presented to the user.

There are two key problems involved in building a content-based search system. The first problem is how to design feature extraction methods and similarity measures that faithfully capture the similarity between objects. In most cases, features are in the form of high-dimensional vectors and the similarity measure is a standard distance metric like l_1 and l_2. There also exist other feature types and non-standard distance metrics, and even similarity measures that are not proper distance metrics. For example, in Chapter 5, we present a technique to approximate a complex distance metric called the Earth Mover's Distance, which measures similarity between images represented as weighted sets of features.

The second problem is *similarity search*: given a query, how to quickly find the objects most similar to the query. A technique to improve similarity search for a standard or abstract distance metric or even generic similarity measures may benefit search systems of many data types. Therefore, the problem of similarity search is of general interest.

1.2 Similarity Search and the Curse of Dimensionality

We devote this dissertation to techniques that make similarity search faster in various circumstances. There are two variants of the similarity search problem: to search for the *K-nearest neighbor* (K-NN) or the *ϵ-nearest neighbor* (ϵ-NN), which are formally defined as follows.

Given a set V of N points with a similarity measure (distance metric) and an arbitrary query point v, the K-NN of v, denoted by $B_K(v)$, is the set of K points in V that are closest to v and the ϵ-NN of v, denoted by $B_\epsilon(v)$, is the set of points in V whose similarity (distance) to v is not less (greater) than ϵ.

The most common case is that V is a subset of the D-dimensional Euclidean space and the similarity measure is defined by the standard Euclidean distance l_2. Among the two variants, it is usually easier for a user to pick K rather than a non-intuitive threshold ϵ, so K-NN search is more popular. In this dissertation, we put our emphasis on the K-NN search problem.

Without specifying the data representation and similarity measure, we can not talk about complexity in primary arithmetic operations. For example, evaluating the l_2 distance between two D-dimensional vectors costs $O(D)$ arithmetic operations, while evaluating the Earth Mover's Distance between two sets of D-dimensional vectors with n items each could cost $O(n^3 + Dn^2)$ arithmetic operations. For K-NN search, we also need to maintain a top-K heap to efficiently track the K-NN, and after scanning each data point in the candidate set, we need to apply an $O(\log K)$ heap updating operation. However, heap updating is usually cheaper than distance evaluation, which is typically of $\Omega(D)$ complexity for D-dimensional vectors, and D is typically tens or even hundreds, and almost always larger than $\log K$. Moreover, heap updates rarely happen as most data points have a relatively large distance to the query and are immediately discarded after one comparison. Therefore in the rest of this dissertation, unless otherwise stated, we always ignore the cost of heap updates and solely measure time complexity in *number of similarity or distance evaluation*.

Searching for K-NN by brute force costs $O(N)$ comparisons, and is only applicable to relatively small datasets. The standard approach to speeding up search for large datasets is to build an index. A large number of indexing techniques are based on the idea of dataset partitioning. The following toy example illustrates the idea of

Figure 1.3: A toy index that bi-partitions the dataset with a hyper-plane, such that the two partitions contain roughly the same number of points. The figures show 2-dimensional projections of points in imaginary low- and high-dimensional spaces, respectively (therefore the distances shown in the figures are not necessarily equal to the distances in the original spaces). For each case, the radius of the green circle is the distance from the query (red) to the unknown nearest neighbor (blue). A partition could potentially contain the nearest neighbor if it overlaps with the green circle, and all such partitions must be examined. In the low-dimensional case (a), the hyper-plane successfully limits the search scope into half the dataset. In the high-dimensional case (b), when the distance to the nearest neighbor grows larger, both partitions have to be examined to guarantee the finding of the nearest neighbor.

indexing (Figure 1.3).

Assume that we are to support 1-NN queries on a set of points sampled from the D-dimensional Euclidean space. We can use a hyper-plane to partition the dataset into two subsets, such that the numbers of points belonging to the two subsets are roughly the same. Given a query, if it is sufficiently far away from the partition boundary (Figure 1.3a), then we know its K-NN will only be found in one subset and can narrow down the search scope by half. There are also cases where the distance from the query to the partition boundary is not as large as the expected distance to the nearest neighbor, and we cannot rule out the possibility that its nearest neighbors may be found in both partitions. If this happens, we either have to give up speedup by examining both subsets, or we have to tolerate the possibility of accuracy loss.

In practice, more sophisticated data partitioning schemes have been proposed. For example, various trees are constructed by recursively partitioning the dataset;

Locality-Sensitive Hashing involves partitioning the dataset into many hash buckets via random projection; and partitioning by K-means clustering has also been shown to be efficient. We will discuss these techniques in more detail in the following sections and provide citations there.

The challenge is that the K-NN problem suffers from the *curse of dimensionality*, which dictates that *as the dimension of data grows, all indexing techniques based on data partitioning scan an increasingly large fraction of the dataset, and will eventually degrade to brute-force search.*

Formal treatment of the dimensionality curse can be found in Weber et al. [78]. Here, we use the above toy example to illustrate the problem, assuming that now the dimensionality D is an increasing number and that the data points have independent similar distributions along each dimension. We already see that the index fails when the query point lies sufficiently close to the partition boundary. It is no longer safe to look into only one partition if the distance δ between the query point and the partition boundary becomes smaller than the distance $\epsilon(D)$ between the query point and its nearest neighbor. We parameterize ϵ with the dimensionality D to emphasize its dependence on D. The fact is that δ is upper-bounded by the range of points along the normal direction to the hyper-plane, and the upper bound is roughly fixed no matter how the hyper-plane is posed. Meanwhile, δ generally grows with D, because according to the definition of the l_2 distance

$$l_2(x, y) = \sqrt{\sum_{i=1}^{D} |x_i - y_i|^2}, \qquad \forall x, y \in \mathbb{R}^D,$$

the added components $|x_i - y_i|^2$ are always larger than or equal to zero (zero is unlikely at the presence of randomization or noise). When D grows, $\epsilon(D)$ will eventually become larger than any δ for most points, and the index becomes useless.

In practical datasets, there are cases when interdependence exists between differ-

ent dimensions. In such cases, the number of dimensions used to represent the data is different from the *intrinsic dimension*, i.e. the minimal number of components needed to describe the data. The hardness of the K-NN indexing problem is determined by the intrinsic dimensionality of the dataset. For example, if one replicates a scalar 100 times to make a 100-dimensional vector, the intrinsic dimension is still 1, and the efficiency of a well-designed indexing algorithm should remain the same as indexing individual numbers.

1.3 The History and State of the Art

The similarity search problem has a long history: the 2-dimensional version of nearest-neighbor search appeared in the 1973 book by Knuth [46] as the *post office problem*; an $O(2^D \log N)$ solution was proposed by Dobkin and Lipton [23] in 1976. Early solutions to the problem are mostly various kinds of tree data structures, e.g., R-tree [35], KD-tree [6], SR-tree [40]. Such methods are all designed to solve the nearest-neighbor search problem exactly. Böhm et al. [8] presented a comprehensive survey that treats these methods under a unified framework.

The dimensionality challenge began to receive more attention in the late 90s when applications required indexing datasets of higher and higher dimensions while tree-based indices failed to deliver the fast search speed as promised by their theoretical $O(\log N)$ complexity (with a hidden coefficient that is exponential in D, which is typically treated as a constant). Weber et al. [78] proved that when dimensionality is sufficiently high, all data partitioning techniques, including all tree-based methods ever designed and those to be designed in the future, will degrade to brute-force search. They also demonstrated that sequential scan outperforms such methods whenever the dimensionality is above 10, which is much smaller than the dimensionality that would cause the indices to fail in theory. The reason is that in out-of-core implementations,

trees usually require random disk access, and the throughput is typically only 1/10 that of sequential access. For a tree-based method to outperform sequential scan, it should be able to find the nearest neighbors by scanning less than 1/10 of the whole dataset, which is hard when dimensionality is high. Clarkson [18] presented a survey of approaches to the nearest-neighbor search problem with an emphasis on the relationship between the problem and data dimensionality.

Recently the trend has shifted to conducting approximate nearest-neighbor search in the hope that speedup can be achieved by sacrificing a limited amount of accuracy. After all, errors are already introduced, in most cases, during data acquisition and feature extraction, and an additional loss of accuracy during K-NN search is usually tolerable. Most recent works fall within the following three categories.

Locality Sensitive Hashing LSH [36, 31] is the state-of-the-art technique of approximate K-NN search. The idea is that it is possible to construct randomized hash functions such that nearby points are more likely to be hashed into the same bucket than points far away from each other. The search scope is limited to the buckets to which the query point is hashed. The accuracy-cost trade-off is realized by varying the average bucket size, the number of hash tables, and the number of buckets probed during the search process. Many studies have been conducted to design hash functions for different similarity measures [20, 14, 47], to reduce storage overhead [55, 39] and to improve search strategy [38]. A recent study [66] showed that the pre-LSH K-means clustering method [51, 72] can also be viewed as a kind of hashing, and can achieve competitive performance when combined with the recently developed LSH-based techniques like multi-probing [55] and query-adaptive querying [38].

Compact Data Representation This line of research follows the seminal work of Weber et al. [78]. The idea is that when dimensionality is high, indexing techniques cannot avoid scanning a significant portion of the whole dataset, and linear scan could be more efficient as sequential read entails a 10 times speedup over random access.

Further speedup can be achieved by using compact data representation. Sketch construction is such a technique for embedding high-dimensional vectors into compact bit-vectors and to approximate the original distance metric with hamming distance. The idea of sketch is closely related to LSH. After all, as we will discuss in Chapter 4, a bit in the sketch is simply a one-bit locality-sensitive hash value. Bi-partitioning the space with a random hyper-plane is an effective way of generating sketches without any training data [14, 53, 77]. Methods with an offline training stage for better online performance have also been recently proposed [79, 85].

Approximate Search with Trees Although trees are well-known to be slow for exact high-dimensional similarity search, significant speedup can actually be achieved when the accuracy requirement is relaxed. ANN [59] is a library for approximate K-NN search with various tree data structures. LSH Forest [4] is a tree-based generalization of LSH, where heavily loaded hash buckets are recursively partitioned with embedded hash tables. The goal is to eliminate the data-dependent parameters that must be hand-tuned and to improve the performance for skewed data distribution. The randomized KD-tree [60] is a similar independently developed technique that aims at improving the performance of the KD-tree for approximate similarity search by simultaneously maintaining multiple independent KD-trees that are constructed with randomization.

Despite the fact that the K-NN problem has been intensively studied for more than two decades, there has not been a satisfactory general solution when the dimensionality of data is high. Rather than a cure to the curse of dimensionality, the approximate techniques are more of a compromise when exact search is simply unable to finish in time. Experiments show that even the state-of-the-art LSH index often has to scan above 10% of the whole dataset in order to achieve acceptable accuracy (e.g., $> 90\%$ recall). Further speeding up of similarity search, even with approximation, remains a challenge.

1.4 Contribution Summary

In this dissertation, we study several key issues to improve the accuracy and efficiency of high-dimensional similarity search. Specifically, we make the following contributions.

Modeling and Improving Multi-Probe LSH

Although Locality-Sensitive Hashing (LSH) has achieved promising results, one practical problem remains that its search quality is sensitive to several parameters that are data-dependent. Previous works on LSH have obtained interesting asymptotic results, but they provide 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 [55], a state-of-the-art variant of LSH. Our model can accurately predict the average search quality and latency by gathering statistical data from a small sample of the dataset. Apart from automatic parameter tuning with the performance model, we also use the model to devise an adaptive search algorithm that determines the probing parameter dynamically for each individual 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. Evaluation results show that the relative error of the recall predictions are less than 5% for most cases; the adaptive search method reduces the standard deviation of recall by about 50% over the existing method with lower average cost.

Efficient K-NN Graph Construction

K-NN graph construction is a variant of the K-NN search problem, in which we want to find the K-NN of every point in the dataset. Unlike the online search task for

which the single query speed is optimized, K-NN graph construction is an offline task, and the goal is to achieve maximal throughput. Existing methods for K-NN graph construction either do not scale, or are specific to certain similarity measures. We present *NN-Descent*, a simple yet efficient algorithm for approximate K-NN graph construction with arbitrary similarity measures. Our method is based on local search, has minimal space overhead and does not rely on any shared global index. Hence, it is especially suitable for large-scale applications where data structures need to be distributed over the network. We show with a variety of datasets and similarity measures that the proposed method typically converges to above 90% recall with each point comparing only to a few percent of the whole dataset on average. Our method is from 2 to 16 times faster than the state-of-the-art technique [15]. The offline computed nearest-neighbor information could be used to double the speed of online similarity search.

Compact Feature Representation with Sketches

Recent studies [53, 54, 77] have shown that sketches can effectively approximate the l_1 distance in high-dimensional spaces and that filtering with sketches can speed up similarity search by an order of magnitude. However, there is not yet an equivalent technique for the popular l_2 distance. It also remains a challenge to further reduce the size of sketches, which are already compact, without compromising accuracy of distance estimation. We present an efficient sketch algorithm for similarity search with l_2 distances and a novel asymmetric distance estimation technique. Our new asymmetric estimator takes advantage of the original feature vector of the query to boost the distance estimation accuracy. We also apply this asymmetric method to existing sketches for cosine similarity and l_1 distance. Evaluations with datasets extracted from images and telephone records show that our l_2 sketch outperforms existing dimension reduction methods like PCA and random projection, and the

asymmetric estimators consistently improve the accuracy of different sketch methods. To achieve the same search quality, asymmetric estimators can reduce the sketch size by 10% to 40%.

Compact Feature Set Representation with Random Histograms

The commonly used representation of a feature-rich data object has evolved from a single feature vector to a set of feature vectors. How to compactly represent sets of feature vectors becomes a significant problem. To address the problem, we present a randomized algorithm to embed a set of features into a single high-dimensional vector. The main idea is to project feature vectors into an auxiliary space using LSH and to represent a set of features as a histogram in the auxiliary space, which is simply a high-dimensional vector. The experimental results show that the proposed approach is indeed effective and flexible. It can achieve accuracy comparable to the feature set-matching methods, while requiring significantly less space and time. For object recognition with the Caltech 101 dataset [24], our method runs 25 times faster to achieve the same precision as Pyramid Matching Kernel [33], one state-of-the-art feature set-matching method.

A Large-Scale Image Search Engine

Finally, we demonstrate some of the proposed techniques by building a large-scale near-duplicate image search system that is capable of serving more than 50 million web images on a single commodity server (a cluster is needed for offline indexing). We also developed domain-specific techniques that make such a system possible. First, we show that entropy-based filtering eliminates ambiguous SIFT features [52] that cause most of the false positives and enables claiming near-duplicity with a single match of the retained high-quality features. Second, we show that graph cut [3] can be used for query expansion with a duplicity graph computed offline to substantially improve

search quality. Evaluation with web images shows that when combined with sketch embedding, our methods achieve a false positive rate orders of magnitude lower than the standard visual word approach [81].

Chapter 2

LSH Performance Modeling

LSH is the state-of-the-art technique of high-dimensional similarity search. A major practical problem is that the search quality of LSH is sensitive to several data-dependent parameters, which are hard to tune by hand. To address this problem, we present a statistical performance model of LSH that 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. Experiments with three different datasets show that the recall errors predicted are within 5% from the real values for most cases and that the adaptive search method reduces the standard deviation of recall by about 50% over the existing method.

2.1 Introduction

Locality-Sensitive Hashing (LSH) [36, 31, 20, 55] is the state of the art for approximate K-NN search. One drawback of LSH, and its space-efficient variant Multi-Probe LSH [55] in particular, is that its performance is very sensitive to several parameters which must be chosen by the implementation. The tedious process of parameter

tuning is 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 chapter 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 accuracy 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 data size.

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 accuracy.

Evaluation with three different datasets, i.e., 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.

Although this chapter focuses on LSH performance modeling, it is worth emphasizing that, at least for the datasets, our data model is capable of capturing the distribution of l_2 distances between a query point and its kth nearest neighbor for any k, as well as the distribution of l_2 distance between two arbitrary points in the dataset with as few as six parameters. This data model is of independent interest and

we expect it to be valuable in modeling the performance of other search and indexing algorithms as well.

2.2 Background

2.2.1 Locality-Sensitive Hashing

LSH was introduced as a randomized technique suitable for solving the approximate K-NN problem [36, 31]. The key insight behind LSH is that it is possible to construct hash functions such that points close to each other under some similarity measure have the same hash value with higher probability than do points that are far from one another. Specifically, given a metric space (V, d), a family $\mathcal{H}$ of hash functions is called (r, cr, p_1, p_2)-sensitive if for any $p, q \in V$ both the following two conditions hold:

- If $d(p, q) \leq r$, then $\Pr_{H \in \mathcal{H}}[H(p) = H(q)] \geq p_1$,
- If $d(p, q) \geq cr$, then $\Pr_{H \in \mathcal{H}}[H(p) = H(q)] \leq p_2$.

Here $\Pr_{H \in \mathcal{H}}[e(H)]$ means the probability of event $e(H)$ over all $H \in \mathcal{H}$. Note that the hash value is not necessarily a single integral value, but could be any data type that supports equality comparison. In practice, vector-valued hash functions with integral components are most commonly used.

More generally, similarity between points can be measured by a similarity measure s, with larger s meaning higher similarity. For example, the similarity measure of the metric space (V, D) is simply $s = -d$. A family $\mathcal{H}$ of hash functions induces the following similarity measure:

$$s_{\mathcal{H}}(p, q) \triangleq \Pr_{H \in \mathcal{H}}[H(p) = H(q)].$$

$\mathcal{H}$ is considered locality sensitive if $s_{\mathcal{H}}$ can be expressed as a increasing function of the similarity measure s of interest, or a decreasing function of the distance metric d if the similarity is measured by the distance:

$$s_{\mathcal{H}}(p,q) = f[s(p,q)], \quad f \text{ is monotonically increasing.}$$

or

$$s_{\mathcal{H}}(p,q) = g[d(p,q)], \quad g \text{ is monotonically decreasing.}$$

Given a particular similarity measure and the corresponding hash family $\mathcal{H}$, LSH maintains L hash tables containing the points in the dataset, each constructed with an independent hash function randomly sampled from $\mathcal{H}$. An approximate solution to the K-NN query may be found by hashing the query point and scanning the buckets to which the query point is hashed. 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.

LSH families have been devised for various similarity measures. Some of the most popular examples are bit sampling for hamming distance [36], min-wise independent permutation for set similarity (Jaccard index) [11], random hyper-plane for cosine similarity [14] and random projection for l_p distance with $p \in (0, 2]$ [20].

A particularly interesting case is the high-dimensional Euclidean space $\mathbb{R}^D$ with l_2 distance, for which the LSH family is defined as follows [20]:

$$H(v) = \langle h_1(v), h_2(v), \ldots, h_M(v) \rangle \tag{2.1}$$

$$h_i(v) = \lfloor \frac{a_i \cdot v + b_i}{W} \rfloor, \quad i = 1, 2, \ldots, M \tag{2.2}$$

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 hash function H is a vector-valued function made by concatenating M scalar hash functions. We call each h_i an atomic LSH function. The parameters M and W control the locality sensitivity of the hash function.

2.2.2 Multi-Probe LSH

One drawback of the basic LSH scheme is that in practice it requires a large number of hash tables (L) to achieve good search accuracy. Panigrahy [64] 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. [55] used a similar perturbation-based approach to develop Multi-probe LSH, which represents the state of the art of K-NN search. To make better use of a smaller number of hash tables, Multi-Probe LSH not only considers the main bucket, where the query point falls, but also examines other buckets that are "close" to the main bucket. Although the idea of multi-probing applies to all vector-valued hash functions with integral components generated by quantizing some real-valued atomic hash functions, so far the random projection-based LSH for l_2 (2.1–2.2) is the only one reported to have been implemented and evaluated in the literature.

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, and are not considered.

Multi-probe LSH is to systematically probe those buckets that are closest to the main bucket. For concreteness, consider the scenario shown in Figure 2.1 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.

Specifically, 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 [55], 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:

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

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

$\Delta_i(\pm 1)$ is the distance from ith projection to the right/left window boundary for each i perturbed, and 0 for others, as illustrated in Figure 2.1.

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 length of the probing sequence T used by the algorithm is a further parameter to be tuned. A larger value of T allows us to achieve the same accuracy of results with fewer hash tables. However, the likelihood of finding more relevant results falls

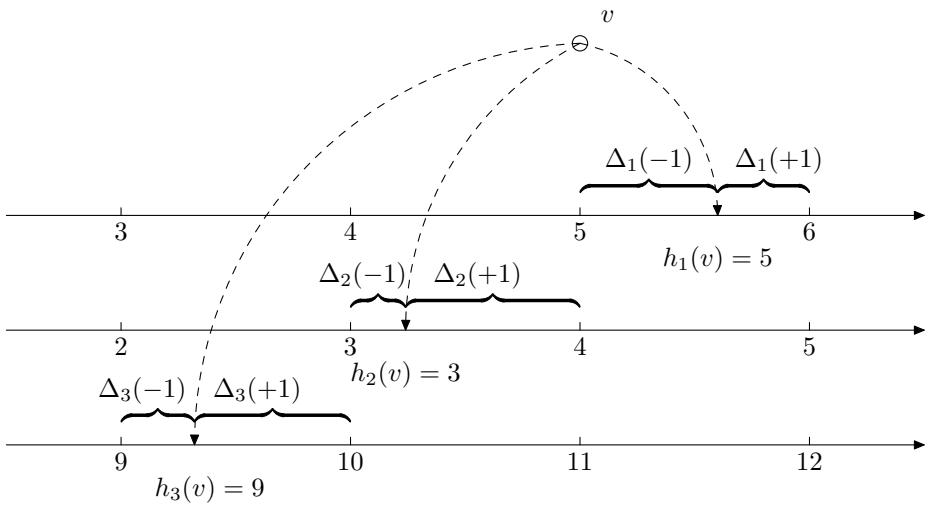


Figure 2.1: Illustration of LSH. The hash function consists of three components, each calculated by random projection and quantization. The point v is hashed to $H(v) = \langle 5, 3, 9 \rangle$.

rapidly as T increases. A very large value of T will only increase the candidate set size without significantly improving accuracy.

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 Performance Measures

First of all we need to formalize the performance measures. There are two aspects of performance — accuracy and cost. For the results to be generally applicable to all applications, we avoid using an application specific gold standard, e.g., image similarity rated by human, except in Chapter 5 and Chapter 6, which are more application specific. Instead, we use the true K-NN obtained with brute-force scan as gold standard, and use the percentage of true K-NN successfully found, or *recall*, to capture the accuracy of the query result. Formally, let v be a query point, $B_K(v)$ be the set of true K-NN 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 $|A(v)| \leq K$). The recall ρ is the percentage of $B_K(v)$ included in $A(v)$:

$$\rho(v) = \frac{|A(v) \cap B_K(v)|}{|B_K(v)|}.$$

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 N times the number of hash tables L. The main part of query processing is scanning through the candidate set and maintaining a top-K heap, which can be updated in $O(\log K)$ time. Because most candidate points are far away from the query and are discarded 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 2.8. 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 chapter 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.

2.4 Modeling Multi-Probe LSH

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:

$$\rho(X) = \Pr[u \in A(v) \,|\, \|u - v\| = X\,].$$

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

$$\tau = E[\tau(v)] = E\left[\frac{1}{|S|}\sum_{\substack{x=\|u-v\| \\ u\in S}} \rho(x)\right] = E[\rho(X)]. \tag{2.3}$$

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 $I_k(v)$, 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

$$\rho = E[\rho(v)] = \frac{1}{K}\sum_{k=1}^{K} E[\rho(X_k)]. \tag{2.4}$$

So far the modeling problem is reduced to finding the distributions of X and X_k, which we will address in the next section, 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 independently, the subscript l is not significant when only probabilities are considered, therefore 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 $d = \|u - v\|$, then

$$\rho(d) = 1 - \Pr[u \notin \bigcup_{l,t} B_{l,t}(v)|] = 1 - \left\{\prod_{t=1}^{T} \Pr[u \notin B_t(v)]\right\}^{L}. \tag{2.5}$$

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

$$\Pr[u \notin B_t(v)] = 1 - \prod_{1 \le i \le M} \Pr[h_i(u) = h_i(v) + \delta_{t,i}(v)]. \tag{2.6}$$

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. Following we deduct the formulas for our l_2 atomic hash function (2.2).

We use $h'_i(v)$ to denote (2.2) without the floor operation:

$$h'_i(v) = \frac{a_i \cdot v + b_i}{W}, \quad i = 1, 2, \ldots, M$$

According to the 2-stability of Gaussian distribution [20], we have that

$$h'_i(v) - h'_i(u) = \frac{a_i \cdot (u - v)}{W} \sim N\left(0, \frac{\|u - v\|^2}{W^2}\right) = N(0, \frac{d^2}{W^2}). \tag{2.7}$$

Let $\phi(\cdot)$ be the probability density function of the standard Gaussian distribution, and let $\phi_{\sigma^2}(\cdot)$ be that of the Guassian distribution $N(0, \sigma^2)$. We have

$$\phi_{\frac{d^2}{W^2}}(x) = \frac{W}{d}\phi(\frac{W}{d}x) \quad \text{and} \quad \phi(-x) = \phi(x), \qquad \forall x \in \mathbb{R}.$$

Following we separately consider the three possible cases of the event $h_i(u) = h_i(v) + \delta_{t,i}(v)$, i.e., when $\delta_{t,i}(v) = 0, +1$ and -1.

Case 1, $\delta_{t,i}(v) = 0$ and $h_i(u) = h_i(v)$. This is equivalent to the event $h'_i(u) \in [h'_i(v) - \Delta_i(-1), h'_i(v) + \Delta_i(+1))$, or $h'_i(u) - h'_i(v) \in [-\Delta_i(-1), \Delta_i(+1))$. According

to (2.7), we have

$$\begin{aligned}\Pr[h_i(u) = h_i(v)] &= \Pr\{h'_i(u) - h_i(v) \in [-\Delta_i(-1), \Delta_i(+1))\} \\ &= \int_{-\Delta_i(-1)}^{\Delta_i(+1)} \phi_{\frac{d^2}{W^2}}(x)\, dx = \int_0^1 \phi_{\frac{d^2}{W^2}}[x - \Delta_i(-1)]\, dx \\ &= \int_0^1 \frac{W}{d}\phi\{\frac{W}{d}[x - \Delta_i(-1)]\}\, dx \\ &= \frac{1}{d}\int_0^W \phi[\frac{x - W\Delta_i(-1)}{d}]\, dx.\end{aligned}$$

We also have

$$\begin{aligned}&\frac{1}{d}\int_0^W \phi[\frac{x - W\Delta_i(-1)}{d}]\, dx \\ =&\frac{1}{d}\int_0^W \phi[\frac{W - x - W\Delta_i(-1)}{d}]\, dx \\ =&\frac{1}{d}\int_0^W \phi[\frac{W\Delta_i(+1) - x}{d}]\, dx \\ =&\frac{1}{d}\int_0^W \phi[\frac{x - W\Delta_i(+1)}{d}]\, dx\end{aligned}$$

That is, either $\Delta_i(-1)$ or $\Delta_i(+1)$ can be used in the formula.

Case 2, $\delta_{t,i}(v) = +1$ and $h_i(u) = h_i(v) + 1$. This is equivalent to the event $h'_i(u) \in [h'_i(v) + \Delta_i(+1), h'_i(v) + \Delta_i(+1) + 1)$, or $h'_i(u) - h'_i(v) \in [\Delta_i(+1), \Delta_i(+1) + 1)$. According to (2.7), we have

$$\begin{aligned}\Pr[h_i(u) = h_i(v) + 1] &= \Pr\{h'_i(u) - h_i(v) \in [\Delta_i(+1), \Delta_i(+1) + 1)\} \\ &= \int_{\Delta_i(+1)}^{\Delta_i(+1)+1} \phi_{\frac{d^2}{W^2}}(x)\, dx = \int_0^1 \phi_{\frac{d^2}{W^2}}[x + \Delta_i(+1)]\, dx \\ &= \int_0^1 \frac{W}{d}\phi\{\frac{W}{d}[x + \Delta_i(+1)]\}\, dx \\ &= \frac{1}{d}\int_0^W \phi[\frac{x + W\Delta_i(+1)}{d}]\, dx.\end{aligned}$$

Case 3, $\delta_{t,i}(v) = -1$ and $h_i(u) = h_i(v) - 1$. Similar to the above, we have

$$\begin{aligned}\Pr[h_i(u) = h_i(v) - 1] &= \Pr\{h'_i(u) - h_i(v) \in [-\Delta_i(-1) - 1, -\Delta_i(-1))\} \\ &= \int_{-\Delta_i(-1)-1}^{-\Delta_i(-1)} \phi_{\frac{d^2}{W^2}}(x)\, dx = \int_0^1 \phi_{\frac{d^2}{W^2}}[x - \Delta_i(-1) - 1]\, dx \\ &= \int_0^1 \phi_{\frac{d^2}{W^2}}[-x - \Delta_i(-1)]\, dx \\ &= \int_0^1 \frac{W}{d}\phi\{\frac{W}{d}[-x - \Delta_i(-1)]\}\, dx \\ &= \frac{1}{d}\int_0^W \phi[\frac{-x - W\Delta_i(-1)}{d}]\, dx \\ &= \frac{1}{d}\int_0^W \phi[\frac{x + W\Delta_i(-1)}{d}]\, dx.\end{aligned}$$

Combining three cases, we have

$$\Pr[h_i(u) = h_i(v) + \delta] = \begin{cases} \frac{1}{d}\int_0^W \phi[\frac{x - W\Delta(\pm 1)}{d}]\, \mathrm{d}x & \text{if } \delta = 0 \\ \frac{1}{d}\int_0^W \phi[\frac{x + W\Delta(\delta)}{d}]\, \mathrm{d}x & \text{if } \delta = \pm 1 \end{cases}. \tag{2.8}$$

Because Case 1 occurs frequently, we use the following approximation to simplify

computation:

$$\Pr[h_i(u) = h_i(v)] = \frac{1}{d}\int_0^W \phi[\frac{x - W\Delta(\pm 1)}{d}]\,\mathrm{d}x \sim E_{\Delta\in[0,1]}\{\frac{1}{d}\int_0^W \phi[\frac{x - W\Delta}{d}]\mathrm{d}x\}. \tag{2.9}$$

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 (2.6). 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 (2.8) and only $\Delta_i(\delta)$ is interesting. Finally, we use the expected values of $\Delta_i(\delta)$ instead of their actual values:

$$\begin{aligned} \Delta_i(+1) &= E[U_{(i)}] = \frac{i}{2(M+1)} \\ \Delta_i(-1) &= 1 - E[U_{(i)}]. \end{aligned} \tag{2.10}$$

where $U_{(i)}$ denotes the ith order statistic of M independent samples from the uniform distribution $U[0, 1)$. 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 (2.3–2.10).

2.5 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 distribution that fits every possible dataset. However, we show that many existing

multimedia datasets do fit a common distribution — the gamma distribution. In this section, we show how to extract statistical parameters from a small sample dataset and do parameter estimation specific to the gamma distribution.

A previous study [77] 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 cannot be easily related to distance distributions. In this dissertation, 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

$$f_{\kappa,\theta}(x) = \alpha(\frac{x}{\theta})^{\kappa} e^{-x/\theta} \tag{2.11}$$

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 probability density function (2.11) over $\mathbb{R}^+$ 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 2.2 and 2.3 show that both X^2 and X_k^2 can be accurately fitted (see Section 2.8.1 for detailed description of the datasets).

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 $\mathbb{R}^D$, 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 a higher degree of freedom will be harder to index. The relative order of the three datasets with regard to degree of freedom matches the experimental results in Section 2.8.

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.

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

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. Furthermore, 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.

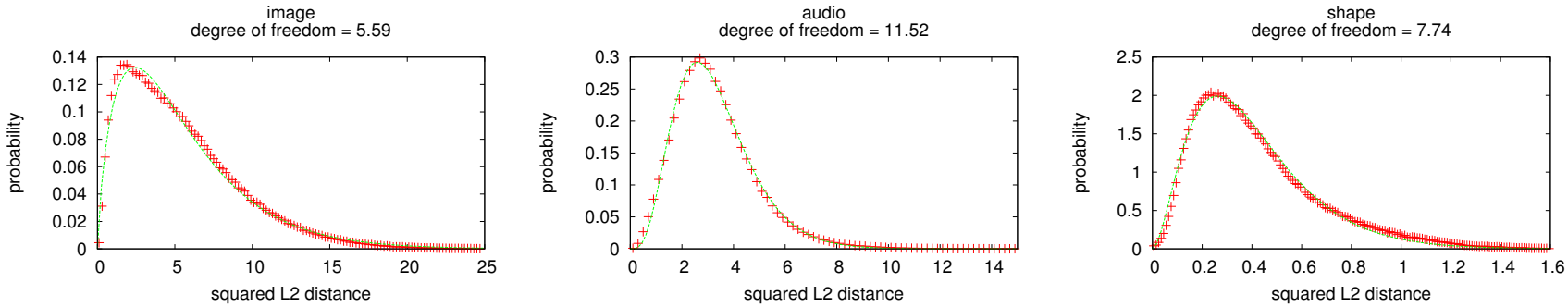


Figure 2.2: The squared distance between two arbitrary points (X^2) follows the gamma distribution.

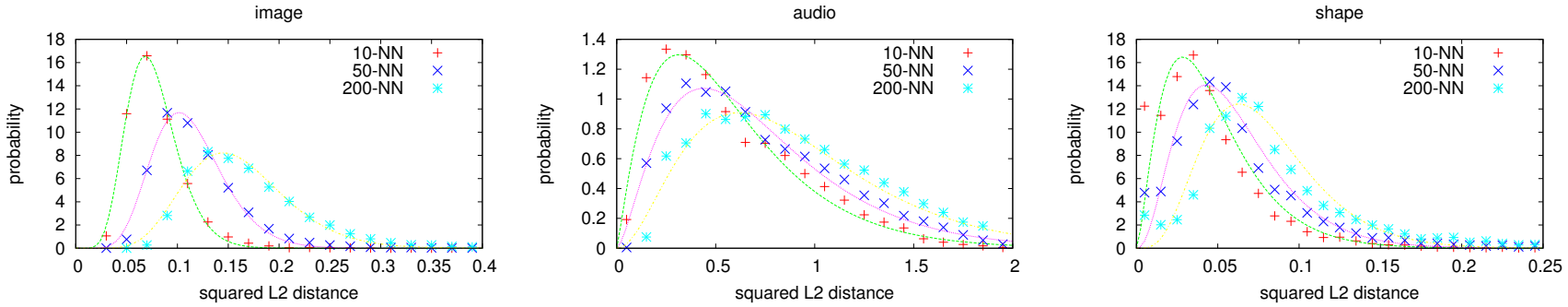


Figure 2.3: The squared distance between the query point and kth NN (X_k^2) follows the gamma distribution

Pagel et al. [63] studied the expected value of X_k as a function of k and N, and proved the following relationship:

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

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:

$$E_k = \alpha k^{\beta} N^{\gamma} \qquad G_k = \alpha' k^{\beta'} N^{\gamma'} \tag{2.12}$$

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-NN of all 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 (2.12) via least squares fitting.

As shown by Figure 2.4 and 2.5, (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.

Finally, we list in Table 2.5 the clues of intrinsic dimensionality of the data sets as suggested by the three different formulas, i.e., $D_k = 1/\beta$, $D_N = -1/\gamma$ by fitting E_k, and degree of freedom (DoF) of gamma distribution. We see that the three measurements all suggest that the audio dataset has highest intrinsic dimensionality,

Dataset	DoF	D_k	D_N
image	5.58	15.78	6.76
audio	11.52	25.41	10.85
shape	7.74	14.43	6.64

Table 2.1: Different clues of intrinsic dimensionality

and that the rest two datasets have similar intrinsic dimensionality.

With the above method, 8 parameters are collected in total, i.e., $\alpha, \beta, \gamma, \alpha', \beta', \gamma'$ in (2.12) and the arithmetic and geometric means of background distance distribution X^2. 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

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

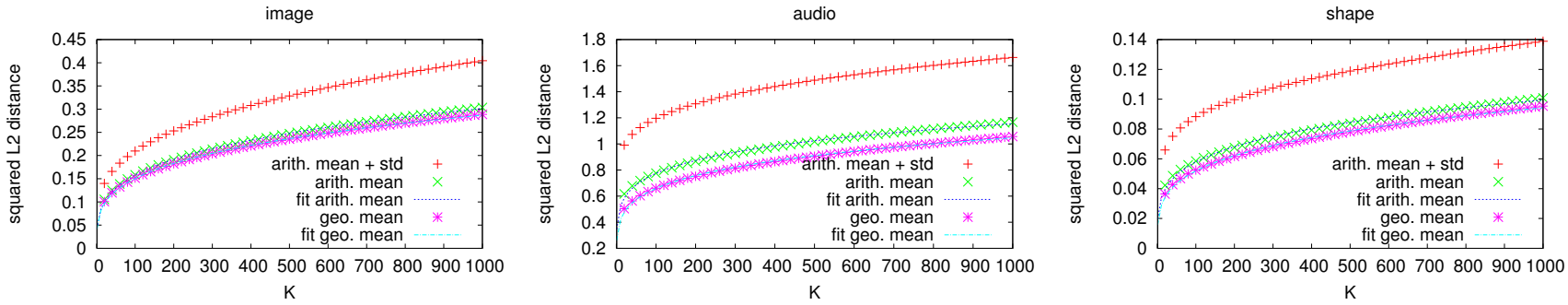


Figure 2.4: Arithmetic and geometric means of squared k-NN distance (X_k^2) follow the power law with regard to k.

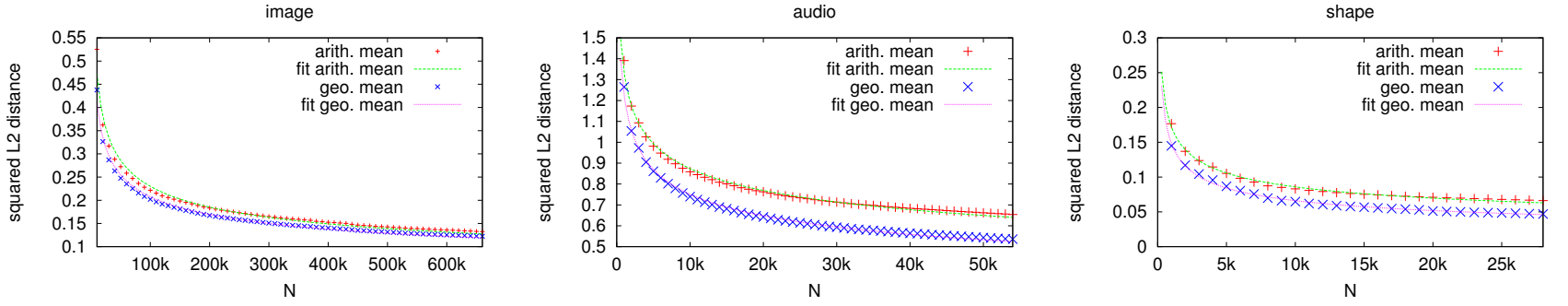


Figure 2.5: Arithmetic and geometric means of squared k-NN distance (X_k^2) follow the power law with regard to N. Here we use $k = 50$.

2.6 Automatic Parameter Tuning

There are four parameters related to LSH. Among them, W, M and L need to be chosen when creating the hash tables, and T is related to the query processing algorithm and can either be chosen 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 ($L \gg 10$, 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 [55]) to tune for near-optimal W and M, and again determine the T value for each query online (to be explained in the next section). The optimization problem is as follows:

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

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 2.8A-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.

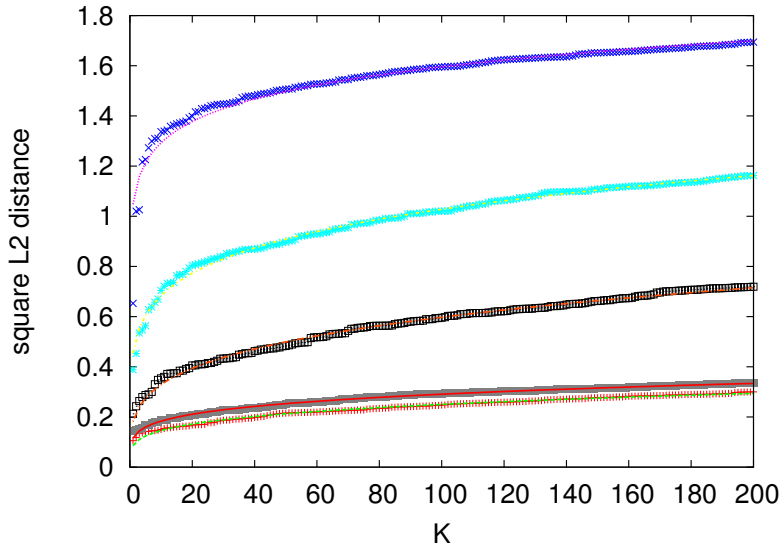


Figure 2.6: Square K-NN distance vs K curves of several sample queries on audio dataset show large variation.

2.7 Adaptive Query Processing

In the previous section, we tuned M and W for optimal average performance. However, for the following reason, we do not want to determine the T value offline, because 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 2.4 show a large standard deviation, which means that the local geometry of different query points can be very different. To directly illustrate the different local geometry of different query points, we sample five query points at random from the audio dataset, and plot their K-NN distance curves in Figure 2.7. For each specific query, probing the default T buckets can be either too few or too many to achieve the required recall. In this section, 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 the 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 refining the prediction iteratively as new buckets are probed. A natural approach is to use the partial results themselves as approximations of the true K-NN. This approximation actually has an advantage that the estimated values are always larger than the real values, and the estimated expected recall is thus always lower than the real one. Also, as the probing sequence goes on, the partial K-NN will quickly converge to the real ones. In practice, if one is to tune for 90% recall, the estimated value should be roughly the same as the real value.

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

2.8 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.

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

Table 2.2: Dataset summary

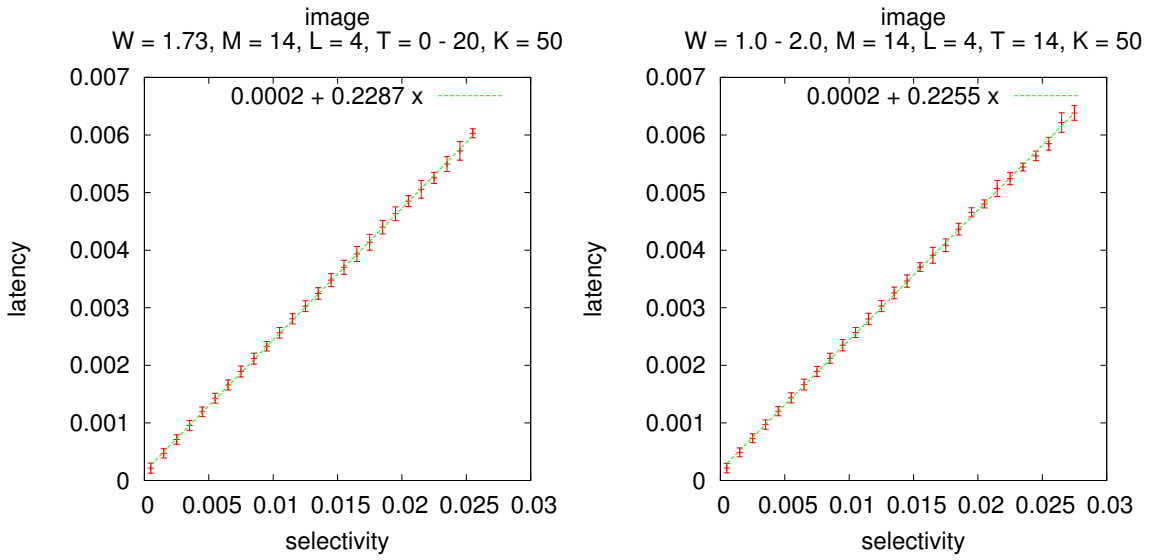
2.8.1 Datasets

We employ three datasets to evaluate our methods: images, audio clips and 3D shapes. Table 2.2 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.

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 [22] to segment the images into regions and use the method by Lv et al. [53] to extract a feature vector from each region. The feature vector is of 14 dimensions, among which, 9 are for color moments and 5 are 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 [29]. 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 [74]. 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) [41], yielding 28,775 544-dimensional feature vectors in total.



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

Figure 2.7: 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.

2.8.2 Multi-Probe LSH Model

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. 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, we run 1,000 queries for 50-NN and record the latency and selectivity. 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 2.7, 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 2.7 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 (shown in Figure 2.8), which achieves about 90% recall with a reasonably low selectivity. The baseline represents a configuration that would be used in real systems. 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 averages 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 2.5.

The effects of the four parameters on the three datasets are shown in Figure 2.8A-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

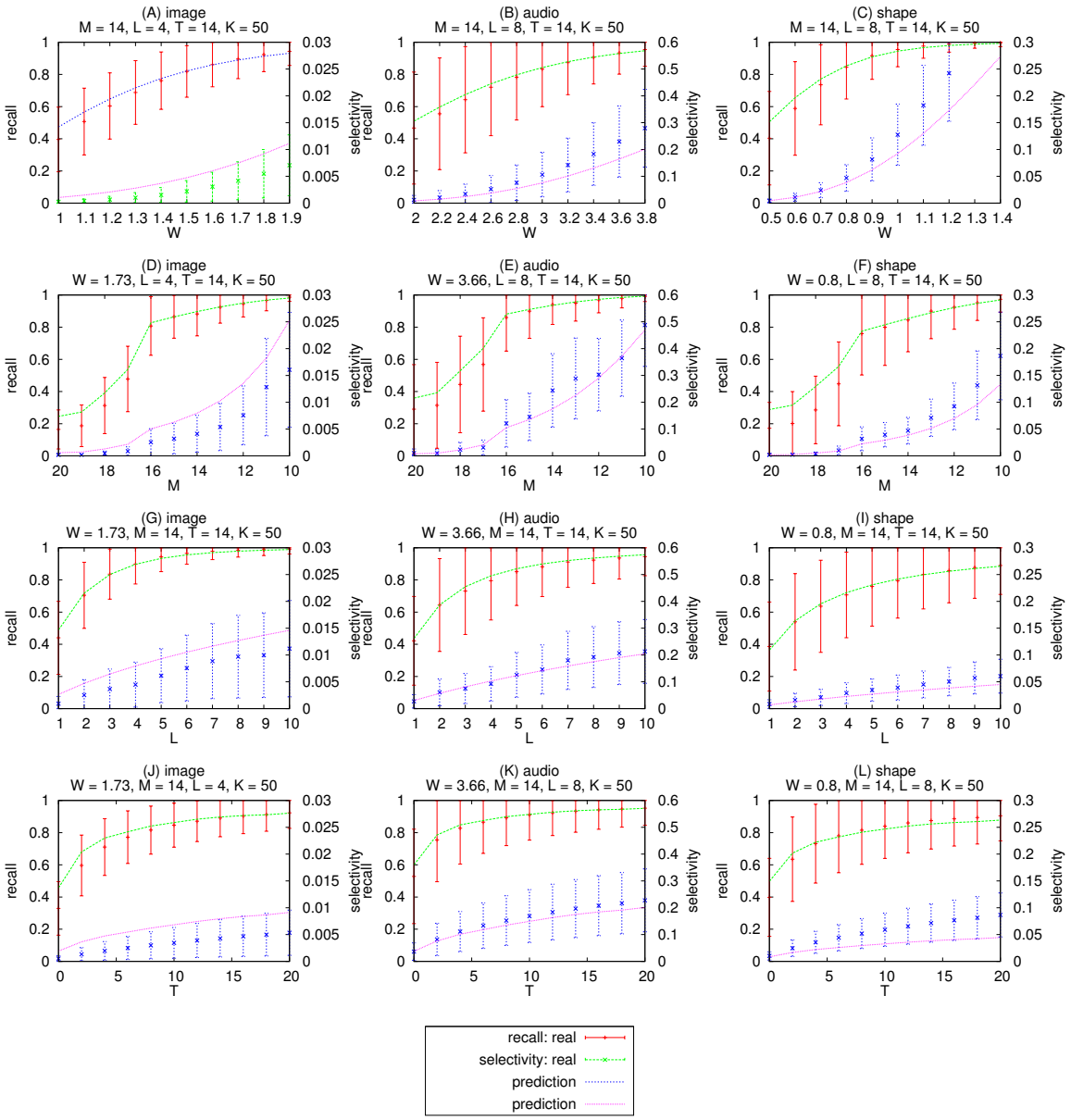


Figure 2.8: Recall and selectivity vs. W, M, L and T respectively. The predicted values are close to the average experimental results.

ratio is a little high, but these cases have very low recall, and are not interesting in practice. Furthermore, 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. We see that enlarging any of the parameters either increase or decrease both performance measures as expected. Most such changes are smooth, except in Figure 2.8(D-F), the the grow rate of recall have a sudden drop at $M = 16$. This suggests that practically $M \leq 16$ should be considered.

2.8.3 Adaptive Query Processing

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 the 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 2.9, and there is an obvious difference between the behaviors of the two methods. These results are better understood when

compared with Figure 2.3, 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 2.3.

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, which could drop below 0.5. Queries with low recall could have much larger impact on average recall than the easier queries whose recall are close to 1. For example, to keep the average recall at 0.9, a query with recall 0.5 has to be compensated by eight queries with recall 0.95. The result is that even though the average recall is achieved, performance variation is high.

The adaptive method, on the contrary, probes more buckets for those difficult points and achieves a high recall to meet the requirement. As a result, the adaptive method significantly reduces the variance of recall and better meets the accuracy 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.

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 2.9. 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 2.3) , and for the adaptive method, T of each query is dynamically

Data	K	T	recall/stdev (%) fixed	adaptive	reduction	selectivity (%) fixed	adaptive
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 2.9. The T values shown are only for fixed method.

Table 2.3: 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.

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 2.3.

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 demonstrate the effectiveness of the adaptive query process method on reducing performance variation as well as the average cost to achieve the same recall.

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.

2.9 Related Works

Existing works related to LSH have been discussed in Section 1.3. A work closely related to the goal of this chapter is the LSH Forest [4], 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. Casey et al. [12] analyzed the minimal distance distribution of high-dimensional musical features for tuning the parameter ϵ for ϵ-NN search so as to achieve optimal precision-recall performance w.r.t. a domain specific gold standard (cover song retrieval). If the ϵ-NN search task is to be implemented with LSH, then relevant LSH parameters still have to be determined, for which our study could be helpful.

Automatic parameter tuning with black-box methods has long been studied in operating research community, see [1] for an example. Although these methods are general, they need to run the algorithm many times. Thus, it is not suitable to tune the offline table construction phase of LSH, especially for large datasets.

The idea of intrinsic dimension has been used to analyze the performance of spatial index structures such as R-tree [63]. The concepts in the paper have inspired our data modeling work, especially in the parameters of the gamma distribution and power law for modeling the distribution of distances. Clarkson [18] has surveyed methods for nearest neighbor search with an emphasis on the relationship with metric space dimensions.

2.10 Summary

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 developed 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 also proposed an adaptive query processing method based on the performance model to reduce performance variance between different query points. This method not only makes on average a 50% reduction on the standard deviation of recall, but is also able to achieve the same average recall with lower latency.

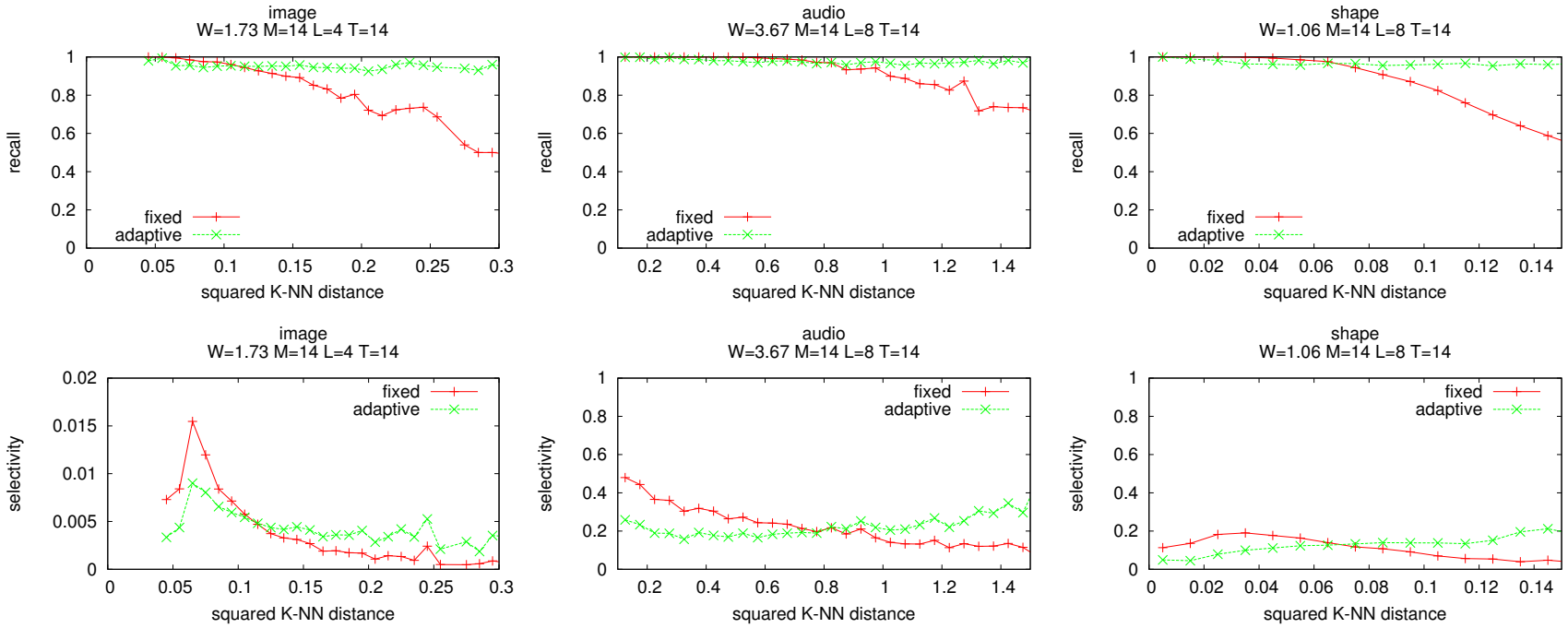


Figure 2.9: Adaptive vs. fixed method in terms of recall and selectivity. The recall of the fixed method degrades as distance increases, while the adaptive method performs much more consistently. For the queries with large distance, the adaptive method does more probes to achieve high recall.

Chapter 3

Offline K-Nearest Neighbor Graph Construction

An important variant of K-NN search is to find the K-NN of every object in the dataset or to construct a K-NN graph. Existing methods for K-NN graph construction either do not scale or are specific to certain similarity measures. We present NN-Descentc*, a simple yet efficient algorithm for approximate K-NN graph construction with arbitrary similarity measures. Our method has minimal space overhead and does not rely on any shared global index. We show with a variety of datasets and similarity measures that the proposed method typically converges to above 90% recall with each point comparing only to a few percent of the whole dataset on average, and is from 2 to 16 times faster, depending on data size, than the state-of-the-art technique. We also show that a K-NN graph can be used to double the speed of LSH-based online K-NN search.*

3.1 Introduction

The K-NN graph for a set of objects V is a directed graph with vertex set V and an edge from each $v \in V$ to its K most similar objects in V under a given similarity

measure, e.g., cosine similarity for text, l_2 distance of color histograms for images, etc.

For search applications whose queries are limited to known objects, offline compute the K-NN for every point would make online search as simple as table lookup. It also makes it unnecessary to keep the feature data if the only purpose of such data is to conduct K-NN search. This could mean substantial space saving for datasets of high dimensionality, e.g. for a 1000-D dataset, saving a 100-NN graph costs only 1/10 the space. Actually, as we will show, even for queries that have not been seen before, such information computed offline could be used to improve online search.

The K-NN graph is also a key data structure for many established methods in data mining [10] and machine learning [70], especially manifold learning [84]. Further more, an efficient K-NN graph construction method will enable the application of a large pool of existing graph and network analysis methods to datasets without an explicit graph structure.

K-NN graph construction by brute-force has cost $O(N^2)$ and is only practical for small datasets. A straightforward approach would be to construct the graph by simply conduct a K-NN search for every point, and any existing K-NN index can be used to speedup the process. However, an algorithm designed to jointly find the K-NN for every point might be able to share or avoid certain computation which would be necessary when solving the K-NN problem individually. In this chapter we will presents such an algorithm. We will also show that a K-NN graph can be used to double the speed of LSH-based online K-NN search with unknown queries.

Paredes et al. [65] proposed two methods for K-NN graph construction in general metric spaces with low empirical complexity, but both require a global data structure and are hard to parallelize across machines. Efficient methods for l_2 distance have been developed based on recursive data partitioning [15] and space filling curves [19], but they do not generalize to other distance metrics or general similarity measures.

In the text retrieval community, methods based on prefix-filtering have been developed for ϵ-NN construction, a.k.a. all pairs similarity search or similarity join [5, 82, 75]. An ϵ-NN graph is different from a K-NN graph in that undirected edges are established between all pairs of points with a similarity above ϵ. These methods are efficient with a tight similarity threshold, when the ϵ-NN graphs constructed are usually very sparse and disconnected.

In summary, efficient K-NN graph construction is still an open problem, and none of the known solutions for this problem is general, efficient and scalable. In this chapter, we present *NN-Descent*, a simple yet effective K-NN graph construction algorithm meeting these requirements with the following properties:

- *General.* Our method works with an arbitrary similarity oracle — a function that produces a similarity score for two objects.

- *Space efficient.* In principle, the only data structure we need is an approximate K-NN graph which is also the final output: our method can iteratively improve the graph in place. For optimization, or in a distributed implementation, minimal extra data are maintained.

- *Fast and accurate.* We demonstrate with real-life datasets that our method typically converges to above 90% recall with each point comparing only to a few percent of the whole dataset on average.

- *Easy to implement.* Our single-node implementation with all optimizations described in this chapter takes less than 200 lines of C++ code (excluding I/O and evaluation code).

We compare NN-Descent against two existing methods, i.e., Recursive Lanczos Partitioning (RLB) [15] and LSH [20] for the special case of the l_2 metric. We show that our method is from 2 to 16 times faster than RLB, depending on the size of the

dataset, and that more speedup can be achieved for larger datasets[1]. We also show how to use the K-NN graph to double the speed of LSH-based online K-NN search.

3.2 Preliminaries

Let V be a dataset of size $N = |V|$, and let $\sigma : V \times V \to \mathbb{R}$ be a similarity measure. For each $v \in V$, let $B_K(v)$ be v's K-NN, i.e., the K objects in V (other than v) most similar to v. Let $R_K(v) = \{u \in V \mid v \in B_K(u)\}$ be v's *reverse K-NN*. In the algorithm, we use $B[v]$ and $R[v]$ to store the approximation of $B_K(v)$ and $R_K(v)$, together with the similarity values, and let $\overline{B}[v] = B[v] \cup R[v]$, referred to as the general *neighbors* of v. $B[v]$ is organized as a heap, so updates cost $O(\log K)$.

We are particularly interested in the case when V is a metric space with a distance metric $d : V \times V \to [0, +\infty)$ for which more specific analysis can be done. Since smaller distance means higher similarity, we simply let $\sigma = -d$. For any $r \in [0, +\infty)$, the *r-ball* around $v \in V$ is defined as $B_r(v) = \{u \in V \mid d(u, v) \leq r\}$.

A metric space V is said to be *growth restricted* if there exists a constant c, s.t.

$$|B_{2r}(v)| \leq c|B_r(v)|, \quad \forall v \in V.$$

The smallest such c is called the *growing constant* of V, which is a generalization of the concept of dimensionality and captures the complexity of the dataset.

3.3 The Basic Algorithm

Our method is based on the following simple principle: *a neighbor of a neighbor is also likely to be a neighbor.* In other words, if we have an approximation of the K-NN

[1]LSH typically works better when accuracy requirement is not high, and its speed would become close to linear scan in order to achieve the accuracy of our method.

for each point, then we can improve that approximation by exploring each point's neighbors' neighbors as defined by the current approximation.

This observation can be quantified by the following *heuristic* argument when V is a growth restricted metric space. Let c be the growing constant of V and let $K = c^3$. Assume we already have an approximate K-NN graph B, and for every $v \in V$, let $B'[v] = \bigcup_{v' \in B[v]} B[v']$ be the set of points we explore trying to improve B. If the accuracy of B is reasonably good, such that for certain fixed radius r, for all $v \in V$, $B[v]$ contains K neighbors that are uniformly distributed in $B_r(v)$, then *assuming independence of certain events and that* $K \ll |B_{r/2}(v)|$ (needed for approximation), we can conclude that $B'[v]$ is likely to contain K neighbors in $B_{r/2}(v)$. In other words, we expect to halve the maximal distance to the set of approximate K nearest neighbors by exploring $B'[v]$ for every $v \in V$. This can be seen from the following.

For any $u \in B_{r/2}(v)$ to be found in $B'[v]$, we need to have at least one v' such that $v' \in B[v] \wedge u \in B[v']$. Any $v' \in B_{r/2}(v)$ is likely to satisfy this requirement, as we have:

1. v' is also in $B_r(v)$, so $\Pr\{v' \in B[v]\} \geq K/|B_r(v)|$.
2. $d(u, v') \leq d(u, v) + d(v, v') \leq r$, so $\Pr\{u \in B[v']\} \geq K/|B_r(v')|$.
3. $|B_r(v)| \leq c|B_{r/2}(v)|$, and $|B_r(v')| \leq c|B_{r/2}(v')| \leq c|B_r(v)| \leq c^2|B_{r/2}(v)|$.

Combining 1—3 and assuming independence, we get

$$\Pr\{v' \in B[v] \wedge u \in B[v']\} \geq K/|B_{r/2}(v)|^2$$

In total, we have $|B_{r/2}(v)|$ candidates for such v', so that

$$\Pr\{u \in B'[v]\} \geq 1 - \left(1 - K/|B_{r/2}(v)|^2\right)^{|B_{r/2}(v)|} \approx K/|B_{r/2}(v)|.$$

The first order approximation holds only when the assumption $K \ll |B_{r/2}(v)|$ remains

true.

Let the diameter of the whole dataset be Δ. The above heuristic argument suggests that so long as we pick a large enough K (depending on the growing constant), even if we start from a random K-NN graph approximation, we are likely to find for each object K items within a radius $\Delta/2$ by exploring its neighbors' neighbors. The process can be repeated to further shrink the radius until the nearest neighbors are found.

Algorithm 1: NNDescent

Data: dataset V, similarity oracle σ, K
Result: K-NN list B
begin
  $B[v] \longleftarrow \textsc{Sample}(V, K) \times \{\infty\}, \quad \forall v \in V$
  loop
    $R \longleftarrow \textsc{Reverse}(B)$
    $\overline{B}[v] \longleftarrow B[v] \cup R[v], \quad \forall v \in V;$
    $c \longleftarrow 0$ //update counter
    for $v \in V$ **do**
      for $u_1 \in \overline{B}[v], u_2 \in \overline{B}[u_1]$ **do**
        $l \longleftarrow \sigma(v, u_2)$
        $c \longleftarrow c + \textsc{UpdateNN}(B[v], \langle u_2, l \rangle)$
    return B **if** $c = 0$

function $\textsc{Sample}(S, n)$
return Sample n items from set S

function $\textsc{Reverse}(B)$
begin
  $R[v] \longleftarrow \{u \mid \langle v, \cdots \rangle \in B[u]\} \quad \forall v \in V$
  return R

function $\textsc{UpdateNN}(H, \langle u, l, \ldots \rangle)$
Update K-NN heap H; return 1 if changed, or 0 if not.

Our basic NN-Descent algorithm, as shown in Algorithm 1, is just a repeated application of this observation. We start by picking a random approximation of K-NN for each object, iteratively improve that approximation by comparing each object against its current neighbors' neighbors, including both K-NN and reverse K-NN, and

stop when no improvement can be made.

The approximate K-NN graph can be viewed as $K \times N$ functions, each being the distance between one of the N objects and its k-th NN. The algorithm is simply to simultaneously minimize these $K \times N$ functions with the gradient descent method, hence the name "NN-descent". But unlike regular gradient descent, which is applied to a function on $\mathbb{R}^D$ and always explores a small neighborhood of fixed radius, our functions are defined on the discrete set V, and the radius we explore around an object is determined by the previous iteration's approximation of the K-NN graph. In fact, the radius starts from a large value as the initial approximation is randomly formed, and shrinks when the approximation is improved through iterations (the number of objects we examine within the radius remains roughly the same). The idea of gradually shrinking search radius through iterations is similar to decentralized search of small-world networks [45] (global optimization). The effect is that most points can reach their true K-NN in a few iterations.

3.4 Optimizations

The basic algorithm already performs remarkably well on many datasets. In practice, it can be improved in multiple ways as discussed in this section.

3.4.1 Local Join

Given point v and its neighbors $\overline{B}[v]$, a *local join* on $\overline{B}[v]$ is to compute the similarity between each pair of different $p, q \in \overline{B}[v]$, and to update $B[p]$ and $B[q]$ with the similarity. The operation of having each point explore its neighbors' neighbors can be equally realized by a local join on each point's neighborhood, i.e., each point introducing its neighbors to know each other. To see that, consider the following relationship: $a \to b \to c$, meaning that $b \in B_K(a)$ and $c \in B_K(b)$ (the directions does

not matter as we also consider reverse K-NN). In the basic algorithm, we compare a and c twice, once when exploring around either a or c (the redundancy can be avoided by comparing only when $a > c$). Equally, the comparison between a and c is guaranteed by the local join on $\overline{B}[b]$.

Even though the amount of computation remains the same, local join dramatically improves data locality of the algorithm and makes its execution much more efficient. Assume that the average size of $\overline{B}[\cdot]$ is $\overline{K}$, in the basic algorithm, exploring each point's neighborhood touches $\overline{K}^2$ points; the local join on each point, on the contrary, touches only $\overline{K}$ points.

For a single machine implementation, the local join optimization may speed up the algorithm by several percent to several times by improving cache hit rate. For a MapReduce implementation, local join largely reduce data replication among machines.

3.4.2 Incremental Search

As the algorithm runs, fewer and fewer new items make their way into the K-NN graph in each iteration. Hence it is wasteful to conduct a full local join in every iteration as many pairs are already compared in previous iterations. We use the following incremental search strategy to avoid redundant computation:

- We attach a Boolean flag to each object in the K-NN lists. The flag is initially marked true when an object is inserted into the list.
- In local join, two objects are compared only if at least one of them is new. After an object participates in a local join, its flag is marked as false.

3.4.3 Sampling

So far there are still two problems with our method. One is that the cost of local join could be high when K is large. Even if only objects in K-NN are used for a local join, cost of each iteration is K^2N similarity comparisons. The situation is worse when reverse K-NN is considered, as there is no limit on the size of reverse K-NN. Another problem is that it is possible that two points are both connected to more than one point, and are compared multiple times when local join is conducted on their common neighbors. We use the sampling strategy to alleviate both problems:

- Before local join, we sample γK out of the K-NN items marked true for each object to use in local join, $\gamma \in (0, 1]$ being the *sample rate*. Only those sampled objects are marked as false after each iteration.
- Reverse K-NN lists are constructed separately with the sampled objects and the objects marked as false. Those lists are sampled again, so each has at most γK items.
- Local join is conducted on the sampled objects, and between sampled objects and old items.

Note that the objects marked true but not sampled in the current iteration still have a chance to be sampled in future iterations, if they are not replaced by better approximations.

We found that the algorithm usually converges to acceptable recall even when only a few items are sampled. Both accuracy and cost decline with sample rate γ, though cost declines much faster (evaluated in Section 3.6.6). The parameter γ is used to control the trade-off between accuracy and speed.

3.4.4 Early Termination

The natural termination criterion is when the K-NN graph can no longer be improved. In practice, the number of K-NN graph updates in each iteration shrinks rapidly. Little real work is done in the final few iterations, when the bookkeeping overhead dominates the computational cost. We use the following early termination criteria to stop the algorithm when further iteration can no longer bring meaningful improvement to accuracy: we count the number of K-NN list updates in each iteration, and stop when it becomes less than δKN, where δ is a precision parameter, which is roughly the fraction of true K-NN that are allowed to be missed due to early termination. We use a default δ of 0.001.

3.5 The Full Algorithm

The full NN-Descent algorithm incorporating the four optimizations discussed above is listed in Algorithm 2. In this chapter we are mainly interested in a method that is independent of similarity measures. Optimizations specialized to particular similarity measures are possible. For example, if the similarity measure is a distance metric, triangle inequality could be potentially used to avoid unnecessary computation.

Our optimizations are not sufficient to ensure that the similarity between two objects is only evaluated once. Full elimination of redundant computation would require a table of $O(N^2)$ space, which is too expensive for large datasets. Space efficient approximations, like Bloom filter, are possible, but come with extra computational cost, and would only be helpful if similarity measure is very expensive to compute.

Our method involves three parameters: K, sample rate γ, and termination threshold δ (note that K would have to be enlarged if the value required by the application is not big enough to produce high recall). Unless otherwise mentioned, we use a default $K = 20$ for all datasets, except that for DBLP we use $K = 50$. The DBLP

Algorithm 2: NNDescentFull

Data: dataset V, similarity oracle σ, K, γ, δ
Result: K-NN list B
begin
  $B[v] \longleftarrow \textsc{Sample}(V, K) \times \{\langle \infty, true \rangle\} \quad \forall v \in V$
  loop
    parallel for $v \in V$ **do**
      $old[v] \longleftarrow$ all items in $B[v]$ with a false flag
      $new[v] \longleftarrow \gamma K$ items in $B[v]$ with a true flag
      Mark sampled items in $B[v]$ as false;
1   $old' \leftarrow \textsc{Reverse}(old)$, $new' \leftarrow \textsc{Reverse}(new)$
    $c \longleftarrow 0$ //update counter
    parallel for $v \in V$ **do**
      $old[v] \longleftarrow old[v] \cup \textsc{Sample}(old'[v], \gamma K)$
      $new[v] \longleftarrow new[v] \cup \textsc{Sample}(new'[v], \gamma K)$
      for $u_1, u_2 \in new[v], u_1 < u_2$
      or $u_1 \in new[v], u_2 \in old[v]$ **do**
        $l \longleftarrow \sigma(u_1, u_2)$
        // c and $B[.]$ are synchronized.
2       $c \longleftarrow c + \textsc{UpdateNN}(B[u_1], \langle u_2, l, true \rangle)$
3       $c \longleftarrow c + \textsc{UpdateNN}(B[u_2], \langle u_1, l, true \rangle)$
    return B **if** $c < \delta NK$

dataset is intrinsically more difficult than the others and need a larger K to reach about 90% recall. We use a default sample rate of $\gamma = 1.0$, i.e., we do not conduct sampling except for trimming reverse K-NN to no more than K elements. For some experiments, we also report the performance numbers for $\gamma = 0.5$, as a faster but less accurate setting. We use a default termination threshold of 0.001.

3.6 Evaluation

We are interested in answering the following questions by experimental studies:

- How does our method perform under typical settings?
- How does our method compare against existing approaches?
- How do accuracy and cost change as dataset scales?
- How to pick a suitable set of parameters?
- How K-NN graph improves online search?
- How does intrinsic dimensionality affect performance?

The last question is answered by an empirical study with synthetic data.

3.6.1 Datasets and Similarity Measures

We use 5 datasets and 5 similarity measures, divided into three categories as described below. These datasets are chosen to reflect a variety of real-life use cases, and to cover similarity measures of different natures. Table 3.1 summarizes the salient information of these datasets.

Four of the datasets are each experimented with two similarity measures (l_1 and l_2, or Jaccard and cosine). Our results show that the two similarity measures for the

same dataset usually produce similar performance numbers and overlapping curves, so in some plots and tables we only report results with one similarity measure per dataset. However, this is not to suggest that different similarity measures for the same dataset are interchangeable. For example, if we test our method with l_1 distance on a benchmark generated with l_2 distance (or vise-versa), only around 70% recall can be reached as opposed to above 95% when the right measure is used.

Dataset	# Objects	Dimension	Similarity Measures
Corel	662,317	14	l_1, l_2
Audio	54,387	192	l_1, l_2
Shape	28,775	544	l_1, l_2
DBLP	857,820	N/A	Cosine, Jaccard
Flickr	100,000	N/A	EMD

Table 3.1: Dataset summary

Dense Vectors

We use the same three datasets, i.e., image, audio and shape, as used in Chapter 2. These datasets consist of dense feature vectors, and are equipped with l_1 and l_2 metrics. We will refer the image dataset as Corel, so as to distinguish it from the other image dataset with EMD distance.

Text Data

We tokenize and stem text data and view them as, depending on the context, multisets of words, or sparse vectors. We apply two popular similarity measures on text data:

- Cosine similarity (vector view) : $C(x, y) = \frac{x \cdot y}{\|x\| \cdot \|y\|}$;
- Jaccard similarity (multiset view): $J(x, y) = \frac{|x \cap y|}{|x \cup y|}$.

DBLP: This dataset contains 0.9 million bibliography records from the DBLP web site. Each record includes the authors' full names and the title of a publication. The same dataset was used in a previous study [82] on similarity join of text data.

Earth Mover's Distance

The Earth Mover's Distance (EMD) [2] [69] is a measure of distance between two weighted sets of feature vectors and has been shown effective for content-based image retrieval [53]. Let $P = \{\langle w_i, v_i \rangle\}$ and $Q = \{\langle w_j, v_j \rangle\}$ be two sets of features with normalized weights $\sum_i w_i = \sum_j w_j = 1$, and let d be the feature distance, we use the following definition of EMD:

$$\text{EMD}(P, Q) = \min \sum_i \sum_j f_{ij} d(v_i, v_j)$$
$$s.t. \quad \sum_j f_{ij} = w_i, \quad \sum_i f_{ij} = w_j.$$

Evaluating EMD is costly as it involves solving a linear programming. We use EMD to show the generality of our method.

Flickr: We apply the same feature extraction method [53] as used for the Corel dataset to 100,000 randomly crawled Flickr images. We set the weights of the feature vectors to be proportional to the number of pixels in their corresponding image regions. Following the original paper [53], we use l_1 as feature distance.

3.6.2 Performance Measures

We use *recall* as an accuracy measure. The ground truth is true K-NN obtained by scanning the datasets in brute force. The recall of one object is the number of its true K-NN members found divided by K. The recall of an approximate K-NN graph is the average recall of all objects.

We use the number of similarity evaluations conducted as an architecture independent measure of computational cost. The absolute number depends on the size of the dataset and is not directly comparable across datasets. So we use a normalized

[2] Code obtained from `http://www.cs.duke.edu/~tomasi/software/emd.htm`, minor changes made to support parallelization.

Dataset	Measure	Default		Fast ($\gamma = 0.5$)	
		Recall	**Cost**	**Recall**	**Cost**
Corel	l_1	0.989	0.00817	0.983	0.00467
Corel	l_2	0.997	0.00782	0.995	0.00436
Audio	l_1	0.972	0.0764	0.945	0.0450
Audio	l_2	0.985	0.0758	0.969	0.0445
Shape	l_1	0.995	0.137	0.989	0.0761
Shape	l_2	0.997	0.136	0.994	0.0754
DBLP	Cosine	0.894	0.0271	0.839	0.0146
DBLP	Jaccard	0.886	0.0309	0.848	0.0173
Flickr	EMD	0.925	0.047	0.877	0.0278

Table 3.2: Overall performance under default setting ($\gamma = 1.0$) and a "fast" setting ($\gamma = 0.5$), with cost measured in scan rate. The default setting ensures high recall (near or above 90%). When minor loss in recall is acceptable, the fast setting reduces scan rate by nearly half. Shape has a particularly high scan rate due to its small size.

version of the measure:

$$\text{scan rate} = \frac{\#\text{ similarity evaluations}}{N(N-1)/2}.$$

The denominator is the total number of possible similarity evaluations of a dataset, and should not be exceeded by any reasonable algorithm.

When scan rate is not an appropriate cost measure for an existing method, we simply compare costs by measuring CPU time. We used commodity servers with dual quad core Intel E5430 2.66GHz CPU and 16GB main memory.

3.6.3 Overall Performance

Table 3.2 summarizes the performance of our method on all the datasets and similarity measures under two typical settings: the default setting ($\gamma = 1.0$) achieving highest possible accuracy and a "fast" setting ($\gamma = 0.5$) with slightly lower accuracy. We see that even with the fast setting, our method is able to achieve $\geq 95\%$ recall, except for DBLP and Flickr. for which recall is below 90%. By putting in more computation

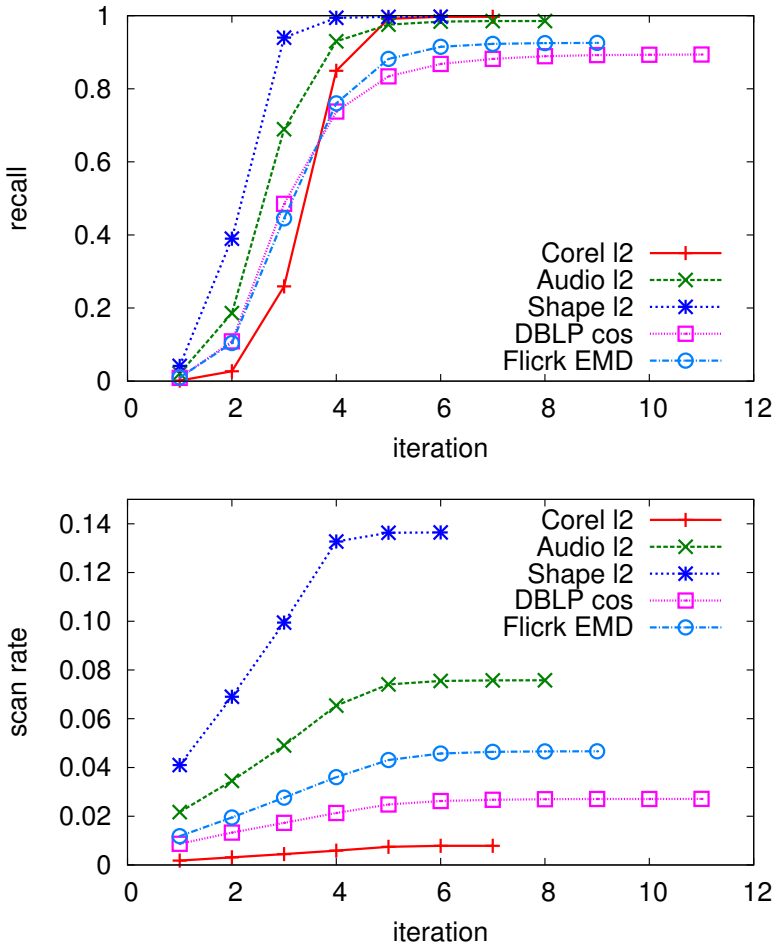


Figure 3.1: Recall and scan rate vs. iterations. Recall increases fast in the beginning and then quickly converges.

with the default setting, we are able to boost recall for the more difficult datasets to close or above 90%.

We see that scan rate has a larger variation across datasets, ranging from below 0.01 for Corel to 0.15 for Shape. Multiple factors could affect scan rate, but we will show (Section 3.6.5) that the size of the dataset is the dominant factor, and with all other parameters fixed, scan rate shrinks as dataset grows. The scan rate of Shape is relatively high mainly because its size is small. In general, at million-object level, we expect to cover a few percent of the total $N(N-1)/2$ comparisons.

For a closer observation of the algorithm's behavior, Figure 3.1 shows the accumu-

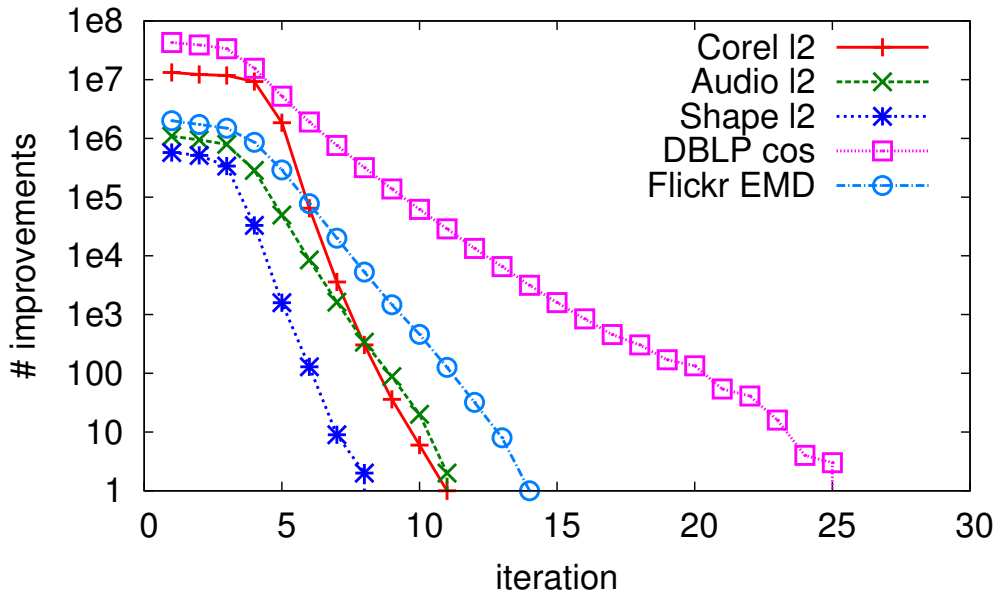


Figure 3.2: Number of total K-NN improvements made in each iteration without early termination. After three to four initial iteration, the numbers drop exponentially.

lative recall and scan rate through iterations. The curves of different data have highly similar trends. We see a fast convergence speed across all datasets — the curves are already close to their final recall after five iterations, and all curves converge within 12 iterations.

As another way to show the fast convergence of our method, Figure 3.2 plots the overall number of K-NN list updates of each iteration. We see that after three to four initial iterations, the number of K-NN list updates starts to reduce exponentially. This is important for us to safely control the loss of recall caused by early termination.

Figure 3.3 shows the approximate K-NN distance ($N \times K$ distance values) distributions of the Corel datasets during the first five iterations. The peaks of the first three iterations spread equally on a log-scaled horizontal axis, i.e., the search radius around each object shrink exponentially in the initial iterations. This remotely confirms our observation made in Section 3.3.

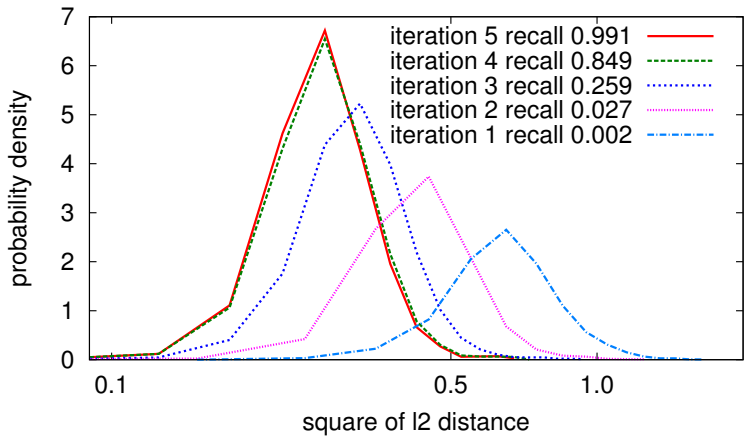


Figure 3.3: Approximate K-NN distance distributions of Corel dataset after each iteration. The peaks of the bottom three curves spread equally on the log-scaled horizontal axis, suggesting the exponential reduction of the radius covered by approximate K-NN during the execution of our method.

3.6.4 Comparison Against Existing Methods

We compare our method with two recent techniques, both specific to l_2 distance, so only the three dense-vector datasets are used. The brute-force approach, even though achieving 100% accuracy, is too expensive for large datasets and is not considered here.

Recursive Lanczos Bisection

Recursive Lanczos Bisection (RLB)[15] [3] is a divide-and-conquer method for approximate K-NN graph construction in Euclidean space. According to the two configurations supported by RLB, we conduct comparison under two settings, one for speed ($R = 0.15$ for RLB and $\gamma = 0.5$ for ours) and one for quality ($R = 0.3$ for RLB and $\gamma = 1.0$ for ours). Table 3.3 summarizes the recall and CPU time of both methods under those settings. Our method consistently achieves both higher recall and faster speed (2× to 16× speedup) in all cases. Actually, even the recall of our low-accuracy setting beats RLB's high-accuracy setting in all cases except for the Corel dataset, where there is only a difference of 0.001.

[3] Code obtained from `http://www.mcs.anl.gov/~jiechen/research/software.html`.

Dataset	Method	For Speed		For Accuracy	
		recall	time	recall	time
Corel	RLB	0.988	1844s	0.996	5415s
	Ours	0.995	252s	0.997	335s
Audio	RLB	0.906	84.6s	0.965	188.6s
	Ours	0.969	21.1s	0.986	31.5s
Shape	RLB	0.961	29.7s	0.989	56.0s
	Ours	0.994	14.0s	0.997	24.4s

Table 3.3: Comparison against Recursive Lanczos Bisection (RLB) under two settings. Our method runs 2 to 16 times faster with consistently higher recall.

Locality-Sensitive Hashing

LSH is a promising method for approximate K-NN search in high dimensional spaces. We use LSH for offline K-NN graph construction by building an LSH index (with multiple hash tables) and then running a K-NN query for each object. We use plain LSH [20] rather than the more recent Multi-Probing LSH [55] in this evaluation as the latter is mainly to reduce space cost, but could slightly raise scan rate to achieve the same recall. We make the following optimizations to the original LSH method to better suit the K-NN graph construction task:

- For each query, we use a bit vector to record the objects that have been compared, so if the same points are seen in another hash table, they are not evaluated again.
- We simultaneously keep the approximate K-NN lists for all objects, so whenever two objects are compared, the K-NN lists of both are updated;
- A query is only compared against objects with a smaller ID.

These optimizations eliminate all redundant computations without affecting recall. We translate the cost of LSH into our scan rate measure, so the two methods are directly comparable (we ignore the cost of LSH index construction though).

Dataset	LSH		Ours	
	recall	scan rate	recall	scan rate
Corel	0.906	0.004	0.995	0.004
Audio	0.615	0.047	0.969	0.045
Shape	0.925	0.076	0.994	0.075

Table 3.4: Comparison against LSH. We achieve much higher recall at similar scan rate. It is impractical to tune LSH to reach our recall as it would become slower than brute-force search.

It is hard for LSH to achieve even the recall of our low-accuracy settings, as the cost would be higher than brute-force search. Hence we tune the scan rate of both methods to an equal level and compare recall. For LSH, we use the same default parameters as used in Chapter 2, except that we tune the number of hash tables to adjust scan rate. For our method, the low-accuracy setting is used ($\gamma = 0.5$).

Table 3.4 summarizes recall and scan rate for both method. We see that our method strictly out-performs LSH: we achieve significantly higher recall at similar scan rate. Also note that the space cost of LSH is much higher than ours as tens of hash tables are needed, and the computational cost to construct those hash tables are not considered in the comparison.

3.6.5 Scalability

It is important that an algorithm has a consistent accuracy and a predictable cost as data scales, so that parameters tuned with a small sample are applicable to the full dataset. To study the scalability of our algorithm, we run experiments on subsamples of the full datasets with fixed parameter settings and observe the changes in recall and scan rate as sample size grows.

Table 3.5 shows recall vs. sample size. We see that as dataset grows, there is only a minor decline in recall. This confirms the feasibility of parameter tuning with a sampled dataset.

Size	Corel	Audio	Shape	DBLP	Flickr
	l_2	l_2	l_2	cos	EMD
1K	1.000	0.999	1.000	0.959	0.999
5K	1.000	0.996	0.992	0.970	0.991
10K	1.000	0.993	0.998	0.970	0.983
50K	0.999	0.988	-	0.951	0.953
100K	0.999	-	-	0.940	0.925
500K	0.997	-	-	0.907	-

Table 3.5: Recall vs. dataset size. The impact of data size growth is small.

Figure 3.4 plots scan rate vs. dataset size, in log-log scale, and we see a very interesting phenomenon: all curves form parallel straight lines, and the curves of all datasets except DBLP almost coincide. This suggests that our method has a polynomial time complexity regardless of the complexity of the dataset (which affects the accuracy rather than speed when parameters are fixed). Table 3.6 shows the empirical complexity, *measured by number of distance evaluations*, of our method on various datasets obtained by fitting the scan rate curves, which is roughly $O(N^{1.14})$ for all datasets. *It is possible for datasets to exist on which our algorithm would perform worse than this. How to obtain a provable upper bound remains an open problem.*

The main reason that the DBLP curve is higher is that we use $K = 50$ for DBLP and $K = 20$ for other datasets. As a local join costs $O(K^2)$, we expect the scan rate of DBLP to be about $(50/20)^2 = 6.25$ times the scan rate of other datasets if they all converge at the same speed. The real value we estimate from the curves (by dividing the intercept of the DBLP curve and the average intercept of the rest) is 5.2, which is close to the expected value.

3.6.6 Parameter Tuning

We need to fix three parameters for the algorithm: K, sample rate γ and termination threshold δ. The meaning of δ is clear: the loss in recall tolerable with early termination. Here we study the impact of K and γ on performance.

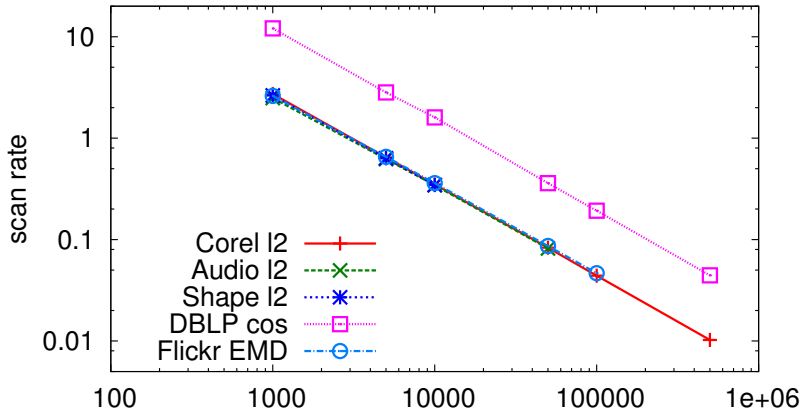


Figure 3.4: Scan rate vs. dataset size. The conincidence of various datasets (except DBLP) suggests that when parameters are fixed, the time complexity of our method depends polynomially on the dataset size, but is independent on the complexity of the dataset (DBLP is different due to a larger K value used).

Dataset & Measure	**Empirical Complexity**
Corel/l_2	$O(N^{1.11})$
Audio/l_2	$O(N^{1.14})$
Shape/l_2	$O(N^{1.11})$
DBLP/cos	$O(N^{1.11})$
Flickr/EMD	$O(N^{1.14})$

Table 3.6: Empirical complexity of different datasets and similarity measures under the default parameter setting. The values are obtained by fitting the scan rate curves in Figure 3.4.

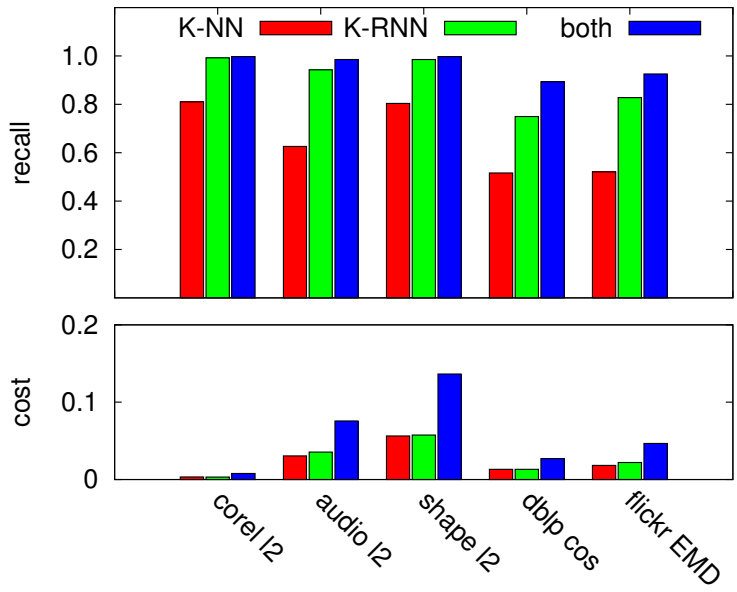


Figure 3.5: Performance with local join of K-NN and/or reverse K-NN (K-RNN).

K-NN and/or Reverse K-NN

Figure 3.5 compares the performance of using K-NN only, reverse K-NN only and both K-NN and reverse K-NN in our algorithm. We see that using K-NN only is not sufficient to achieve good recall (above 90%). Using both K-NN and reverse K-NN always achieves the best recall. Even though cost is also higher when using both, and in some cases good recall can also be achieved with reverse K-NN only, we choose to use both K-NN and reverse K-NN by default for robustness in accuracy.

Tuning K as a Parameter

The application determines the minimal K required. Meanwhile, a sufficiently large K is necessary to achieve high recall. The minimal value required by the application might need to be enlarged.

Figure 3.6 plots the relationship between K and performance. We see that $K \geq 10$ is needed for the dense vector datasets to achieve good recall, and for text, recall

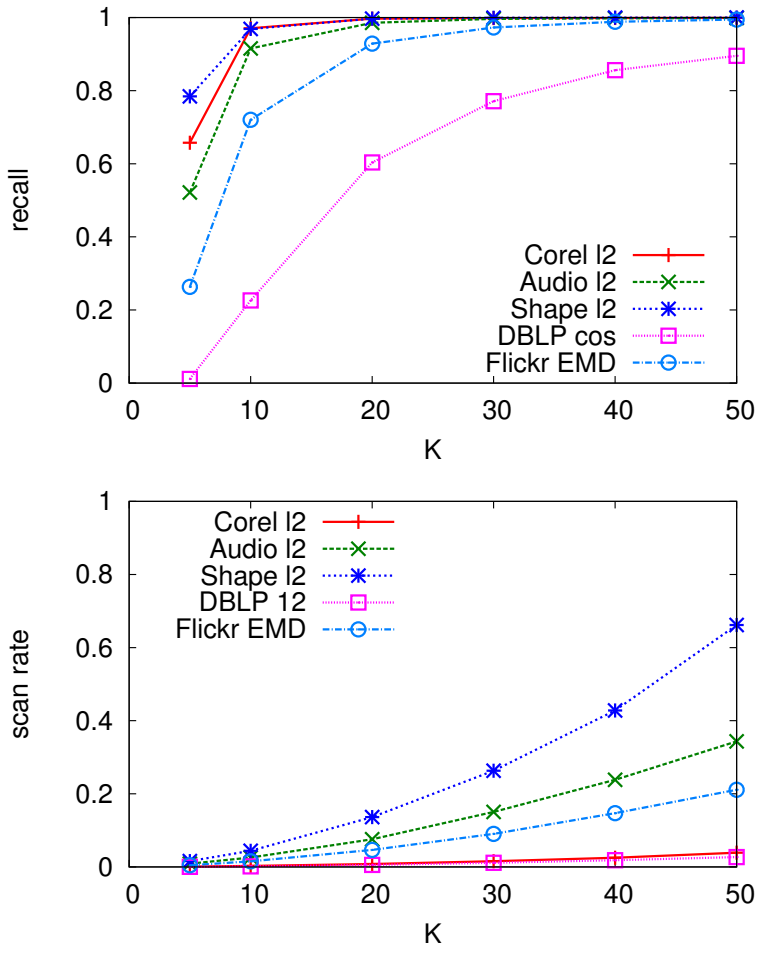


Figure 3.6: Recall and scan rate vs. K. For a particular dataset, a sufficiently large K is needed for recall above 90%. Beyond that, recall only improves marginally.

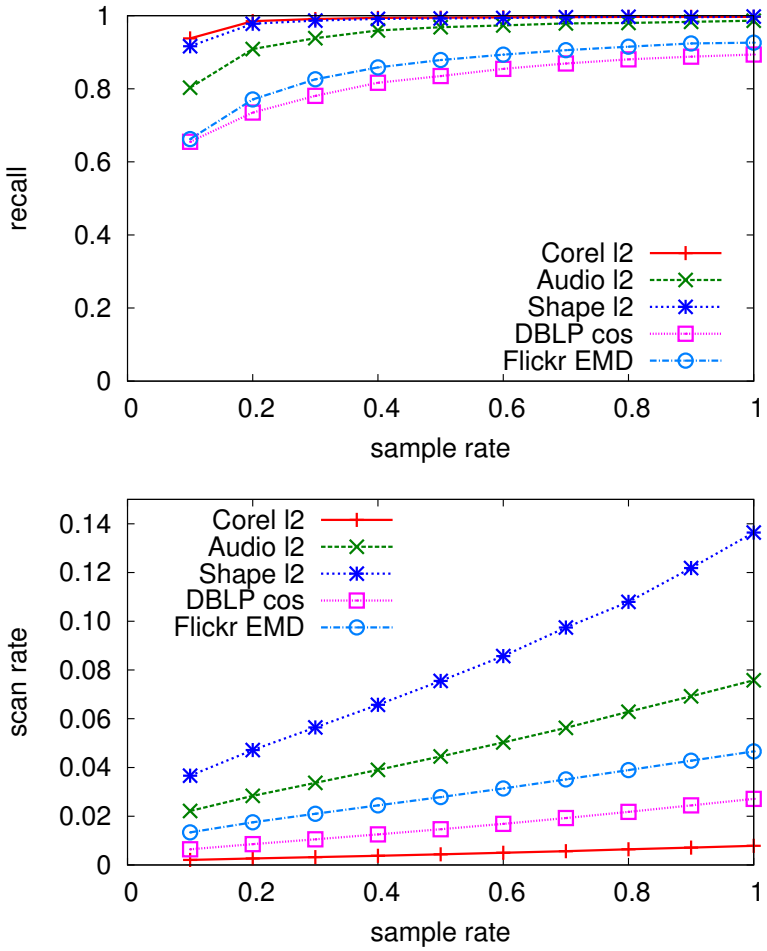


Figure 3.7: Recall and scan rate vs. sample rate γ. Recall grows marginally after $\gamma > 0.5$ while cost grows in a near-linear fashion across the whole range of γ.

reaches 90% only with $K \geq 50$. This suggests that text has a high intrinsic dimensionality than the other datasets. We also see that beyond a critical point, recall only improves marginally as K further grows.

Sample Rate for Accuracy-Cost Trade-Off

The sample rate γ can be used to control accuracy-cost trade-off. Figure 3.7 plots sample rate vs. performance. We see that even with a sample rate of 0.1, reasonably high recall can be achieved for most datasets, and recall grows very slowly beyond $\gamma = 0.5$. Scan rate, on the other hand, has a near-linear relationship with γ across

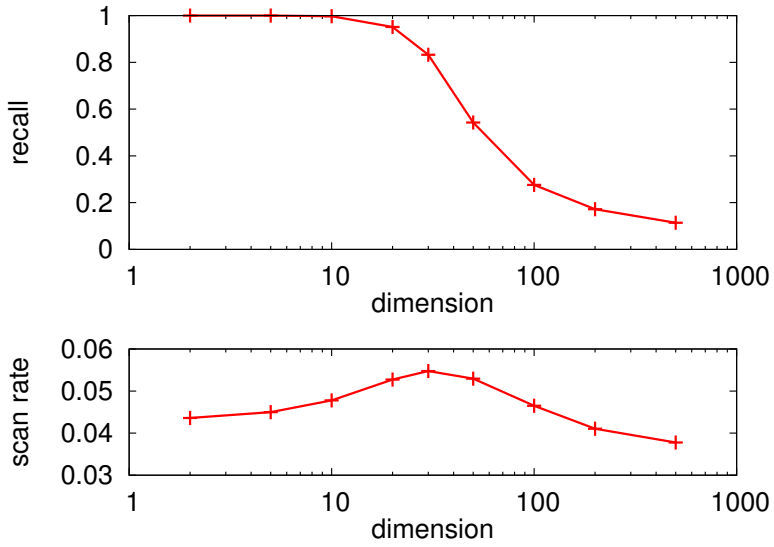


Figure 3.8: Recall and scan rate vs. dimensionality with $K = 20$. There is a limit of dimensionality up to which the algorithm is able to achieve high recall.

the whole range. We suggest $\gamma = 0.5$ for applications without a critical dependent on high recall.

3.6.7 Impact of Intrinsic Dimensionality

We use synthetic data to study the impact of dimensionality on the performance of our algorithm, as the intrinsic dimensionality of real-life datasets is not obvious and can not be controlled. We generate a D dimensional vector by concatenating D independent random values sampled from the uniform distribution $U[0, 1]$. Data generated in this way have the same intrinsic dimensionality and full dimensionality. We then test the performance of our method with different dimensionality, each with a dataset size of $100,000$.

Figure 3.8 plots the performance of our algorithm, with the default setting, under different dimensionality with a fixed $K = 20$. We see that recall decreases as dimensionality increases, and our method performs well (recall above 95%) with dimensionality ≤ 20 (which happens to be the value of K). Beyond that, recall rapidly

drops to about 50% as dimensionality grows to 50, and eventually approaches 0 as dimensionality further grows.

The impact of dimensionality on cost, although not as large, is also interesting. When dimension is low and most points are able to reach their true K-NN (global optima), cost is low. Convergence speed slows as dimensionality increases and scan rate also increases. Scan rate peaks at around 30 dimensions. After that, most points are not able to reach their true K-NN and get more and more easily trapped at local optima, so scan rate begins to shrink as dimensionality further grows. Overall, the fluctuation of cost is low, within a range of $2\times$.

We then study how recall can be improved by enlarging K. Table 3.7 summarizes the recall and scan rate of representative K values at various dimensionality. The results can be categorized into three zones of dimensionality:

- Small dimensionality ($D = 2, 5$): extremely high recall (close to 1) and very low scan rate (< 0.01) can be achieved.

- Medium dimensionality ($D = 10, 20$): recall reaches 95% with $K = D$; scan rate is relatively higher, around 5% (due to a larger K). Recall still increases as K grows, but a recall close to 1 is no longer practical as scan rate would grow too high.

- Large dimensionality ($D = 50, 100$): recall peaks at $K = 50$ and declines beyond that, and the peak recall shrinks as D grows (94% for $D = 50$ and 78% for $D = 100$). Scan rate is around 1/4, already too high for the algorithm to be practically useful.

This suggests that our method is best applied to dataset with intrinsic dimensionality around 20. It still works, but with relatively high cost, for dimensionality around 50, and start to fail as dimensionality further grows. Fortunately, all datasets we experimented with, and actually most real-life datasets where K-NN is meaningful [7],

are of relatively low dimensionality.

3.7 Online Search with K-NN Graph Expansion

An offline constructed K-NN graph can be used to improve online search with LSH and other approximate search algorithms, as the true K-NN members missing from the initial search results are likely to be neighbors of the items found, and can be recovered by exploring the K-NN graph neighborhood of the initial search results. Like in NN-Descent, such expansion can be carried out recursively. We will show with experiments that K-NN graph expansion can double the speed of Multi-Probe LSH without compromising search quality. K-NN graph expansion comes with the offline computational cost of K-NN graph construction and online storage cost to keep the K-NN graph. These extra costs must be considered by implementation.

In this section, we evaluate how online search with Multi-Probe LSH can be improved by K-NN graph expansion. We use the three dense-vector datasets, i.e., Corel, Audio and Shape, with l_2 distance. The evaluation protocol is the same as used in Chapter 2. We find the true 50-NN of 1000 randomly sampled queries by brute-force scan and use that as benchmark. We measure the search quality by average recall with regard to the benchmark, and measure cost by selectivity, which includes the points scanned during K-NN expansion. As exact K-NN graph construction is not practical for large dataset, we use NN-Descent with default parameters to construct K-NN graphs. By default, we explore the 10-NN in the K-NN graph during expansion. For Multi-Probe LSH, we use the same default parameters as used in Chapter 2, except that we also evaluate a setting with 1/4 the number of hash tables. Therefore, the experimental results can be directly compared against those obtained in Chapter 2.

D = 2			D = 5			D = 10			D = 20			D = 50			D = 100		
K	recall	cost	K	recall	cost	K	recall	cost	K	recall	cost	K	recall	cost	K	recall	cost
4	0.875	0.004	5	0.862	0.006	9	0.925	0.014	19	0.942	0.0487	49	0.934	0.238	49	0.774	0.240
5	0.990	0.005	6	0.957	0.007	10	0.950	0.016	20	0.952	0.0527	50	0.939	0.245	50	0.781	0.248
6	0.996	0.006	7	0.980	0.009	11	0.967	0.019	21	0.957	0.0569	51	0.925	0.253	51	0.778	0.257

Table 3.7: Tuning K for various dimensionality. For a dataset of fixed size and when dimensionality is high ($D \geq 50$), recall cannot always be improved by enlarging K, and the highest achievable recall shrinks as dimensionality grows. Even though only three K values are shown for each case, the peak is found by testing a larger range of values.

There are three different Ks as in K-NN involved in the evaluation and we distinguish them as K, K' and K''.

- $K = 50$: This is the K-NN used in the benchmark, and is picked to reflect the real usage.
- K': This is the size of neighborhood we explore in the K-NN graph during query expansion and the default value is 10. Larger K' means higher recall but also higher selectivity.
- $K'' = 20$: This is the value used for K-NN graph construction. It is the upper-bound of valid K'. We choose it to be larger than K' to ensure the quality of K-NN graph. In practice, if a $K'' > K'$ is used, the neighbors with ranks from $K' + 1$ to K'' can be discarded after the K-NN graph is constructed as they are not used in query expansion.

We compare the following five strategies:

Original Original Multi-Probe LSH. This is the baseline. The default parameter setting of Chapter 2 is used. The parameter L is the number of hash tables, and is set to 4 for Corel, and 8 for both Audio and Shape.

One Level One level K-NN graph expansion. This strategy achieves less improvement in recall, but is particularly interesting for out-of-core implementation, as we can store the K-NN together with each point on disk, so that no extra disk seeks are needed.

Recursive Recursive expansion until no improvement can be made. This strategy achieves the maximal recall improvement, and is suitable for main memory implementation.

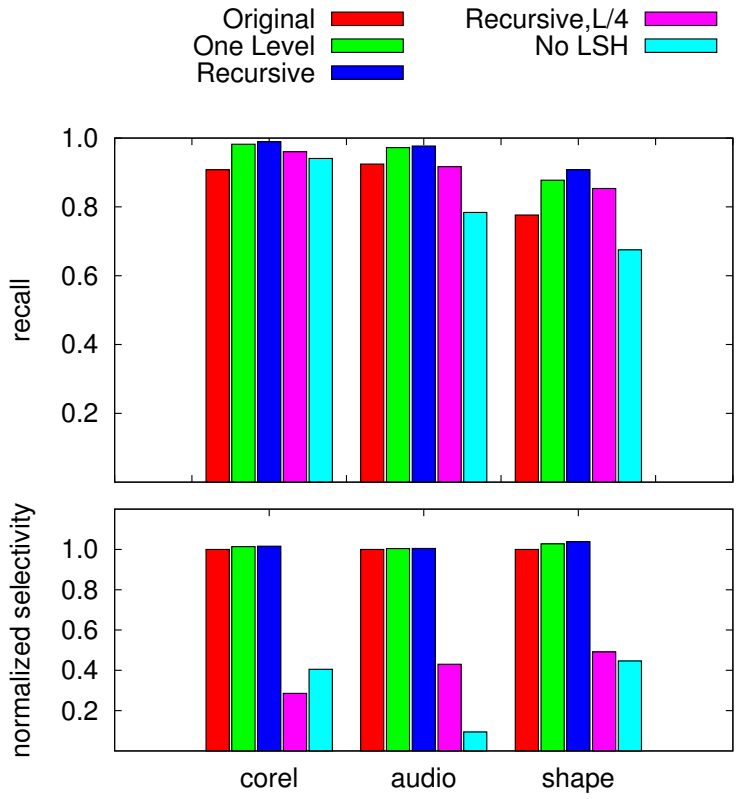


Figure 3.9: Online search performance with various K-NN graph expansion strategies. Selectivity is normalized against that of the original Multi-Probe LSH.

Recursive, $L/4$ Recursive expansion, but with 1/4 the original number of hash tables. This is to show how K-NN graph expansion can be used to reduce the search cost, and is the recommended setting for real implementation.

No LSH Recursive expansion starting from random initial results without using Multi-Probe LSH. We are interested to see if Multi-Probe LSH is necessary for online search.

Figure 3.9 shows the recall and selectivity of these five strategies. To make selectivity changes comparable across datasets, we normalize selectivity against that of the original Multi-Probe LSH. First, we see that by adding K-NN graph expansion to Multi-Probe LSH, we can improve recall by about 5% without significantly increasing selectivity. This demonstrates the power of K-NN expansion. Second, recursive

expansion achieves higher recall than one level expansion, while the cost overhead is minimal, so it should be preferred in main memory implementation. Third, by applying recursive expansion, we only need 1/4 the original number of hash tables to achieve similar recall, and at the same time cost can be reduced by more than half. Finally, K-NN graph expansion with random initialization achieves a competitive recall at less than half the cost of other methods for the Corel dataset. However, recall drops below 80% for the other two datasets. Even though its cost is consistently low, we believe its not robust enough for general application. The reason might be that most queries get trapped into local optimal. Overall, we think recursive expansion with 1/4 hash tables is a good choice for real implementation.

Next we study the impact of K-NN graph size used in expansion, or the parameter K', on search performance. Figure 3.10 plots recall and selectivity vs. K', with selectivity normalized against the case without K-NN graph expansion. We see that both recall and selectivity increase as K' grows, but the improvement of recall slows down around $K' = 10$, while selectivity continues to grow in a near-linear fashion. This confirms that $K' = 10$ is a proper choice.

3.8 Related Works

Paredes et al. [65] are the first to study K-NN graph construction for general metric space as a primary problem (rather than an application of K-NN search methods). Some general observations they made also apply to our work: the K-NN graph under construction can be used to improve the remaining construction work and cost can be reduced by solving the N K-NN queries jointly. They solved the K-NN graph construction problem in two stages: first an index is built, either a tree structure or a pivot table, and then the index is used to solve a K-NN query for each object. Despite the high level similarity to using a general K-NN search index for K-NN graph

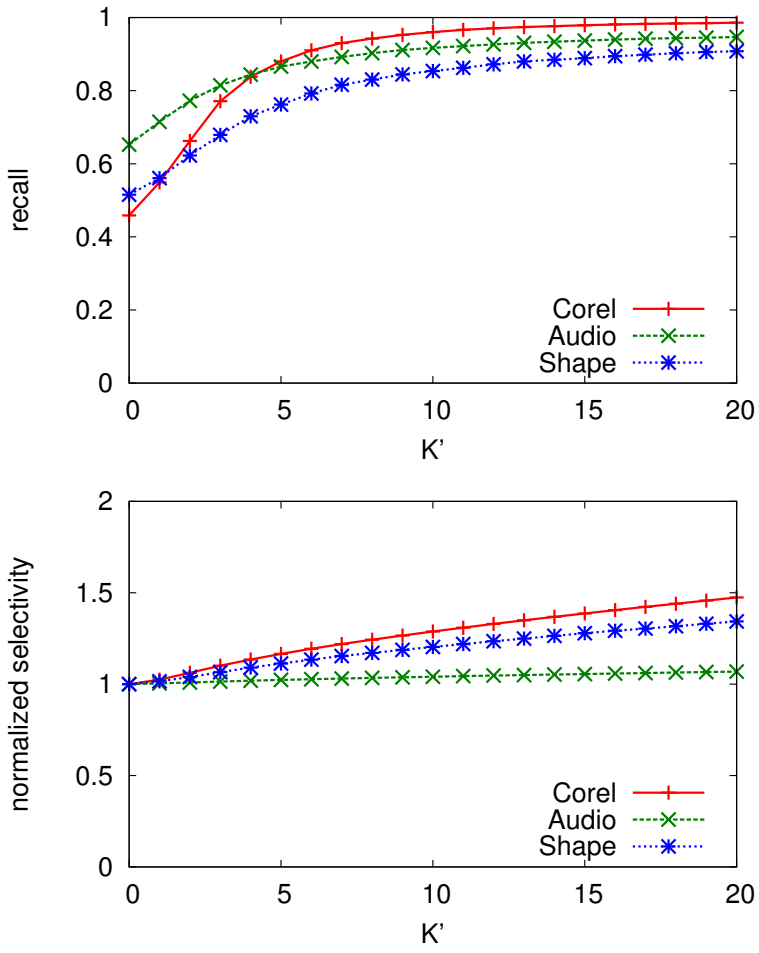


Figure 3.10: Online search performance at different K-NN graph sizes. Selectivity is normalized against the case without K-NN graph expansion.

construction, strategies to exploit the approximate K-NN graph already constructed are incorporated into the search process. They also studied the empirical complexity of their methods. For example, the pivot based method achieves a better empirical complexity, which is $O(N^{1.10})$ at 4 dimensions and $O(N^{1.96})$ at 24 dimensions.

Efficient K-NN graph construction methods have been developed specifically for l_2 metric. Recursive Lanczos Bisection [15] uses inexpensive Lanczos procedure to recursively divide the dataset, so objects in different partitions do not have to be compared. Connor et al. [19] used space filling curve to limit the search range around each object, and an extra verification and correction stage is used to ensure accuracy. These methods do not easily generalize to other distance metrics or general similarity measures.

In the text retrieval community, efficient methods based on prefix-filtering are developed for the ϵ-NN graph construction [5, 82, 75], a different kind of nearest neighbor graph which establishes an edge between all pairs of points whose similarity is above ϵ. The problem is that such methods are only efficient for a very tight similarity threshold, corresponding to a very sparse and disconnected graph.

3.9 Summary

We presented *NN-Descent*, a simple and efficient method for approximate K-NN graph construction with arbitrary similarity measures, and demonstrated its excellent accuracy and speed with extensive experimental study. Our method has a low empirical complexity of $O(N^{1.14})$ (on various tested datasets) and can be easily parallelized, potentially enabling the application of existing graph and network analysis methods to large-scaled dataset without an explicit graph structure. Experimental study showed that our method is from 2 times to 16 times faster than the state-of-the-art K-NN graph construction technique Recursive Lanczos Bisection on the datasets evaluated.

We also showed that our method can easily achieve above 95% recall, which is hard for approximate online K-NN techniques like LSH to achieve. Finally, we showed that the K-NN graph can be used improve the speed of LSH-based online search by a factor of two. Rigorous theoretical analysis of our method is an interesting problem to be solved.

Chapter 4

Compact Feature Representation with Sketches

The compactness of data representation, besides the number of items in the candidate set, is another key factor that determines search speed. This chapter presents a compact bit-vector, or sketch, embedding of high-dimensional vectors that is able to approximate the l_2 distance, and a novel asymmetric distance estimation technique. Our asymmetric estimator takes advantage of the original feature vector of the query to boost the distance estimation accuracy. We also apply this asymmetric method to existing sketches for cosine similarity and l_1 distance. Evaluations with datasets extracted from images and telephone records show that our l_2 sketch outperforms existing methods like PCA and random projection, and the asymmetric estimators consistently improve the accuracy of different sketch methods. To achieve the same search quality, asymmetric estimators can reduce the sketch size by 10% to 40%.

4.1 Introduction

Search speed is mainly determined by two factors: how many items are in the candidate set, and how efficient each item can be matched. In the previous two chapters, we presented methods to model and reduce the number of items in the candidate set. From this point on, we will focus on efficient representation and matching of large objects: this chapter on individual high dimensional feature vectors and the next chapter on sets of feature vectors.

Sketch construction is an effective way to approximate feature vectors for similarity search. Sketches are compact bit vectors that can be used instead of the original feature vectors to estimate distances. In a search system, sketches are scanned upon a query to quickly generate a small set of candidates which can be ranked later with original feature vectors to obtain the search result. Such a process is often called filtering. As reported by previous work [53, 54, 77], sketches are typically an order of magnitude smaller than their feature vectors, and can significantly improve the space requirement and search speed. For systems dealing with large datasets, reducing space cost usually means improving speed, as load time can be reduced (both from disk to main memory and from main memory to CPU), and it might be possible to use faster but smaller storage (from disk to main memory). A key challenge for sketch construction is to achieve a high ratio of distance estimation accuracy to sketch size.

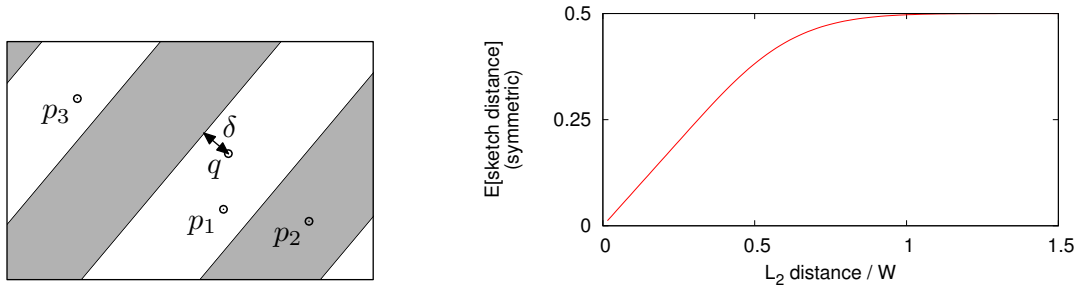
The traditional method of using sketches to estimate distances, like l_1 distance [53] and cosine similarity [14], is to construct a bit vector for each feature vector where each bit is determined by the position of the feature vector with respect to a random hyperplane. The Hamming distance of such sketches will then be used to approximate the original distance measure. Since computing Hamming distance (counting the number of bits that are different) is simple, the filtering process can be an order of magnitude faster than scanning the original feature vectors. However, the distance estimation with such sketches is a crude approximation; accuracy can be low when sketches are

very compact. A challenging question is whether the accuracy of distance estimation can be substantially improved using the same sketches.

Distance estimation using sketches is typically *symmetric* in that two sketches are compared to produce an estimate of the distance. In this chapter, we propose a novel *asymmetric* approach to estimate the distance between the feature vector of a query and the sketch of a data object. As the distance estimation is done in two different spaces, we call such algorithms *asymmetric estimators*. Asymmetric estimators do not impose additional space requirements because they use the same sketches for all data points and there is only one query point. These methods can achieve higher accuracy because they take advantage of the position information of the query point in the feature vector space. Another way to look at the effect of the proposed approach is that it allows the sketches to be more compact to achieve the same accuracy.

As a specific example, we first present a new l_2 sketch construction and its asymmetric estimator. Our new method has an interesting property particularly desirable for similarity search: it is only sensitive to a small distance range covering most k-nearest neighbors. We then provide general guidelines on designing asymmetric estimators and show how to design asymmetric estimators for two existing sketch constructions for l_1 distance [53] and cosine similarity [32, 14].

We evaluate all three sketch constructions with symmetric and asymmetric estimators using real-life datasets extracted from images and telephone records, and demonstrate the significant space saving (10% to 40%) of the asymmetric estimators, as well as the superiority of our new l_2 sketch over the existing methods.



(a) Partitioning scheme

(b) Sketch vs. l_2 distance

Figure 4.1: Illustration of l_2 Sketch. (a) The space is partitioned randomly into gray and white stripes, inducing a 0/1 sketch function. (b) The left half of the curve shows a near-linear relationship, meeting the requirement of similarity search.

4.2 Sketch and Asymmetric Estimator for l_2 Distance

In this section we present our new sketch algorithm for l_2 distance, which we call l_2 Sketch, and show how to exploit the raw feature vectors for asymmetric distance estimation.

4.2.1 l_2 Sketch Construction

The goal of sketches for similarity search is different from that for general distance estimation. First, we only need to know the distance relationships (smaller or larger), rather than exact values. Second, we only need high accuracy for small distances instead of uniform accuracy/inaccuracy across the whole range of distances. These two relaxations potentially allow data reduction more aggressive than traditional dimension reduction methods.

We are able to exploit both opportunities by a striping technique. Figure 4.1(a) illustrates the scheme in $\mathbb{R}^2$ space, and the idea is exactly the same for high-dimensional spaces. We randomly partition the space into interleaving white and gray stripes of

equal width, and use stripe colors to determine the sketch values of the points: gray for 0 and white for 1. Points closer to each other are more likely to fall into the same stripe, and subsequently, the XOR of their sketch values has a smaller expectation. We do the random striping multiple times to produce a sequence of bits for each point, and use the Hamming distance of the bit vectors as a proximity estimator. Specifically,

l_2 **Sketch.** *For a point $p \in \mathbb{R}^D$, its l_2 sketch is a bit vector $\sigma(p) \in \{0,1\}^B$, with each bit $\sigma_i(p)$ produced by*

$$\begin{aligned} \sigma_i(p) &= \lfloor h_i(p) \rfloor \mod 2, \\ h_i(p) &= \frac{a_i \cdot p + b_i}{W}, \qquad\qquad \forall i = 1, 2, ..., m \end{aligned}$$

where $a_i \in \mathbb{R}^D$ is a random vector with each dimension sampled independently from the standard Gaussian distribution $N(0,1)$, and $b_i \in \mathbb{R}$ is sampled from the uniform distribution $U[0, W)$. W is called the window size. *Note that h_i is exactly the same atomic hash function introduced in (2.2). For two points $p, q \in \mathbb{R}^D$, their sketch distance is defined as*

$$d_\sigma(p,q) = \frac{1}{B}\sum_{i=1}^{B} \sigma_i(p) \oplus \sigma_i(q) = \frac{1}{B} d_H[\sigma(p), \sigma(q)].$$

where d_H is Hamming distance.

The above algorithm specifies how to produce a random partitioning: a_i determines the direction of the stripes, $\|a_i\|$ and W together determine the width of the stripes, and the random shift b_i determines the 'phase'. Figure 4.1(b) shows the relationship between the expectation of symmetric sketch distance and l_2 distance (see Section 4.2.3 for formulas). The monotonicity of the curve satisfies our ranking purpose. For small l_2 distances, we can see a near-linear relationship, and when l_2

distance grows large, sketch distance quickly converges to the limit 0.5. The window size W determines the sensitive distance range.

4.2.2 Asymmetric Estimator for l_2 Sketch

Figure 4.1(a) explains how the original query point is useful for asymmetric distance estimation. Assume p_1, p_2 and p_3 are data points, with only sketches available, and q is the query point, with both sketch and feature vector available. We know the precise location of q, but only know that p_1 and p_3 are in white stripes and p_2 is in a gray stripe. Since p_2 and q are in stripes of different colors, the distance between p_2 and q is lower-bounded by the minimum distance from q to a stripe boundary. This lower bound (marked δ in the figure) replaces the Hamming distance 1 in asymmetric distance estimation. On the other hand, p_1 and p_3 are in stripes of the same color as q, and the value 0 is used. The asymmetric distance estimator is defined below.

l_2 Sketch. *Under the setting of l_2 Sketch 1, for data point p and query point q, both in $\mathbb{R}^D$, the asymmetric sketch distance is defined as*

$$d^*_\sigma(p,q) = \frac{1}{B}\sum_{i=0}^{B} \delta_i(q)[\sigma_i(p) \oplus \sigma_i(q)]$$

where $\delta_i(q) = \min\{\lceil h_i(q)\rceil - h_i(q),\ h_i(q) - \lfloor h_i(q)\rfloor\}$.

The asymmetric sketch distance is a weighted Hamming distance of the sketches. The weight $\delta_i(q)$ is the distance from q to the closest stripe boundary.

4.2.3 Statistical Analysis

The following lemma gives the relationship between symmetric and asymmetric sketch distances and original l_2 distance.

Lemma 1. *Under the setting of l_2 Sketch, for any $p, q \in \mathbb{R}^D$, let $d = \|p - q\|$ be the l_2 distance, $d_\sigma = d_\sigma(p, q)$ be the symmetric sketch distance and $d^*_\sigma = d^*_\sigma(p, q)$ be the asymmetric sketch distance, then for any $k \in \mathbb{R}$,*

$$E[d_\sigma] = f_0(d/W) \qquad E[d^*_\sigma] = f_1(d/W)$$

where

$$f_k(t) = \int_0^1 \int_0^1 \frac{\min\{x, 1-x\}^k}{t} \sum_{j \in \mathbb{Z}} \phi\left[\frac{2j + x + y}{t}\right] dxdy$$

and $\phi(\cdot)$ is the probability density function of the standard Gaussian distribution $N(0, 1)$.

Proof. The B bits in the sketch are identically distributed, so we only consider the case when $B = 1$ and omit subscript i as in a_i, b_i and h_i. For $B \neq 1$, the expectation remains the same, and variance is scaled by $1/B$.

We shift the real line by $\lfloor h(p) \rfloor$, so that $h(p) = y \in [0, W)$, following uniform distribution. Let $j \in \mathbb{Z}$ be an arbitrary integer. When $h(q) \in [2jW, (2j+1)W)$, $d_\sigma = d^*_\sigma = 0$. Thus only the case when $h(q) \in [(2j+1)W, (2j+2)W)$ is interesting. In this case, we have have $d_\sigma = 1$ and $d^*_\sigma = \min\{x, W - x\}$, where $x \in [0, W)$ is the distance from $h(q)$ to the left window boundary $(2j+1)W$. The probability for this to happen is

$$\begin{aligned}
&\sum_{j \in \mathbb{Z}} \Pr[h(p) = y \wedge h(q) = (2j+1)W + x] \\
=&\sum_{j \in \mathbb{Z}} \frac{1}{W} \Pr[h(q) - h(p) = (2j+1)W + x - y] \\
=&\sum_{j \in \mathbb{Z}} \frac{1}{W} \Pr[a \cdot (q - p) = (2j+1)W + x - y] \\
=&\sum_{j \in \mathbb{Z}} \frac{1}{Wd} \phi\left[\frac{(2j+1)W + x - y}{d}\right]
\end{aligned}$$

The last step is due to the 2-stability of Gaussian distribution [20], which states that

for any $p \in \mathbb{R}^D$, $a \cdot p \sim \|p\| \times N(0,1)$.

The expectations of d_σ and d^*_σ are obtained by averaging the corresponding values weighted by the above probability across all $x, y \in [0, W)$. □

Choosing a proper window size W is a practical issue. For K-NN search, twice the typical distance of the kth nearest neighbor is a good starting point for tuning.

4.3 Asymmetric Estimators for Other Sketch Algorithms

In this section we describe a general framework for designing asymmetric estimators and use this to devise asymmetric estimators for two existing sketch methods for *cosine* similarity and l_1 distance.

4.3.1 Asymmetric Estimator in General

The asymmetric estimator proposed in the previous section can be generalized into the following framework

First, a sketch algorithm σ for a metric space $\langle V, d\rangle$ specifies a family $\sigma = \{\sigma_i \mid i \in I = \{1, 2, 3, \ldots\}\}$ of bipartitions of the space, mapping an arbitrary point $p \in V$ into a bit sequence $\sigma(p) = \langle \sigma_i(p)\rangle_{i\in I}$. The sketch distance between two points $p, q \in V$ is the expectation of XOR of the corresponding bits in sketches: $d_\sigma(p, q) = E_{i\in I}[\sigma_i(p) \oplus \sigma_i(q)]$. A well-designed sketch algorithm should establish a predictable relationship between the sketch distance d_σ and original distance d. A closely related concept is LSH: a bipartition σ_i is a 0/1 valued LSH function.

Second, an asymmetric distance estimator specifies a distance function d^* (not necessarily the original d) between a point and a partition. In the case of l_2 Sketch, d^* is the l_2 distance measure under a random projection. If we denote by $[0]_{\sigma_i}$and

$[1]_{\sigma_i}$ the two parts of σ_i, then the asymmetric sketch distance is simply the average distance from the query point q to the part that the data point p is in: $d^*_\sigma(p,q) = E_{i\in I}\{d^*([\sigma_i(p)]_{\sigma_i}, q)\}$. Because d^* gives a refined lower bound of the crude XOR, the asymmetric estimator potentially provides higher accuracy.

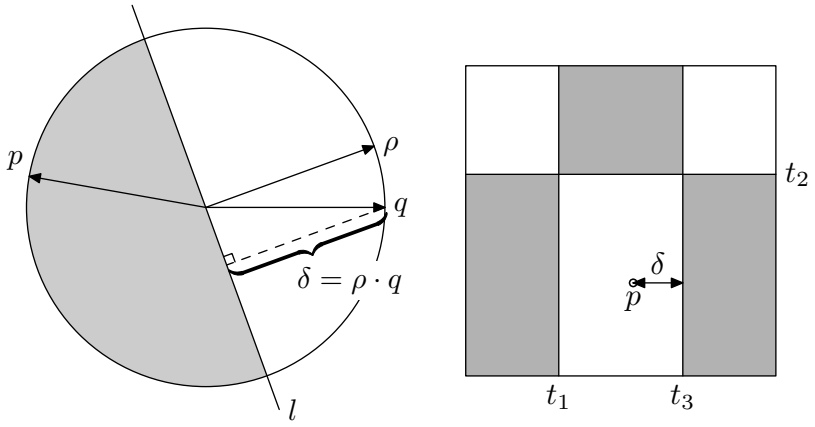
Given a sketch algorithm, the challenge then lies in finding a meaningful asymmetric distance function d^*. Though the sketch algorithm and the original distance measure usually give useful clues, such an estimator is not always obvious and does not always exist. This is one limitation of the asymmetric distance estimation method. It is possible that an asymmetric estimator results in a distance measure different from the original one, but is still helpful in improving the search quality. We later discuss such an estimator for the l_1 case. It is also possible that more than one asymmetric estimator might exist for a single sketch algorithm, each having its own advantage.

Another limitation of the asymmetric method is the accuracy improvement. The error in distance estimation comes from uncertainty in the positions of the data point and the query point. Even if the asymmetric method fully eliminates the error introduced by the query point, it eliminates only half the source of error. In practice (mainly due to the randomization in sketch algorithms), this 50% upper bound is usually far from tight, even though the asymmetric estimators do make a significant difference in many cases.

In the following two subsections, we restate two existing sketch algorithms in our framework, and devise asymmetric estimators for them. We call these algorithms Cosine Sketch and l_1 Sketch based on the distance measure they approximate.

4.3.2 Cosine Sketch

Cosine similarity is not a strict distance measure, but is still important to many applications like text document retrieval. For any $p, q \in \mathbb{R}^D$, the cosine similarity is



(a) Cosine Sketch (b) l_1 Sketch

Figure 4.2: The space partitioning schemes of Cosine Sketch and l_1 Sketch, both based on random hyper-plane method.

defined by $d_{cos}(p,q) = \frac{p \cdot q}{\|p\| \cdot \|q\|}$, and is related to l_2 distance by

$$[d_{l_2}(p,q)]^2 = \|p\|^2 + \|q\|^2 - 2\, d_{cos}(p,q)\, \|p\|\, \|q\|.$$

Thus, l_2 distance and cosine similarity have the same ranking effect for normalized datasets, and for unnormalized datasets, given the 2-norms of the points, a sketch algorithm for cosine similarity automatically induces one for l_2 distance.

The following sketch algorithm for cosine distance is implicitly given in [32, 14]. For simplicity of presentation, we assume all the points are normalized to unit vectors.

Cosine Sketch. *For a point $p \in S^{D-1} = \{r \in \mathbb{R}^D \mid \|r\| = 1\}$, its cosine sketch is a bit vector $\sigma(p) \in \{0,1\}^B$, with each bit $\sigma_i(p)$ produced by*

$$\sigma_i(p) = \begin{cases} 0 & \text{if } \rho_i \cdot p < 0 \\ 1 & \text{if } \rho_i \cdot p \geq 0 \end{cases} \qquad \forall i = 1, 2, \ldots, m$$

where ρ_i for each i is sampled uniformly at random from the unit hypersphere S^{D-1}. The symmetric sketch distance between $p, q \in S^{D-1}$ is defined as $d_\sigma(p,q) = \frac{1}{B} d_H[\sigma(p), \sigma(q)]$.

The idea of the random hyper-plane method is illustrated in Figure 4.2(a). The random vector $\rho \in S^{D-1}$ determines a hyper-plane l (the orthogonal complement of ρ) which bipartitions the sphere. A bit in the sketch records whether or not the point falls on the same side of the hyper-plane as ρ. To design an asymmetric distance estimator, we need to assess the distance from the query point q to its neighboring half space (the gray one). We find the distance from q to the hyper-plane l, indicated by δ in the figure, a natural choice. Simple enough, δ is exactly $|\rho \cdot q|$. Thus, we have the following asymmetric estimator.

Cosine Sketch. *Under the setting of Cosine Sketch 1, for data point p and query point q, both in S^{D-1}, the asymmetric sketch distance is defined as*

$$\begin{aligned} d^*_\sigma(p,q) &= \frac{1}{B}\sum_{i=0}^{B} |\rho_i \cdot q| \times [\sigma_i(p) \oplus \sigma_i(q)] \\ &= \frac{1}{B}\sum_{i=0}^{B} -\mathrm{sgn}(\rho_i \cdot p) \times (\rho_i \cdot q) \times [\sigma_i(p) \oplus \sigma_i(q)] \end{aligned}$$

The following lemma gives the relationship between symmetric and asymmetric sketch distances and the original cosine similarity.

Lemma 2. *Under the setting of Cosine Sketch, for any $p, q \in S^{D-1}$, let $\theta = \widehat{pq}$ be the angle between p and q, $d = \cos(\theta)$ be the cosine similarity, $d_\sigma = d_\sigma(p,q)$ be the symmetric sketch distance and $d^*_\sigma = d^*_\sigma(p,q)$ be the asymmetric sketch distance, then*

the following relations hold:

$$E[d_\sigma] = \frac{\theta}{\pi} \tag{a}$$

$$E[\cos(\pi d_\sigma)] = \sum_{k=0}^{B} \binom{B}{k} \frac{\theta^k (\pi - \theta)^{B-k}}{\pi^B} \cos\left(\frac{\pi}{B} k\right) \tag{b}$$

$$E[d_\sigma^*] = \frac{B(\frac{D}{2}, \frac{1}{2})}{2\pi}(1 - d) \tag{c}$$

$$var[d_\sigma^*] = \frac{1}{B}\{\frac{\theta - \frac{1}{2}\sin 2\theta}{D\pi} - E[d_\sigma^*]^2\} \tag{d}$$

where $B(\cdot, \cdot)$ is the Beta function.

Proof. (a) See [32, 14].

(b) Given the angle θ between p and q, the probability that the sketches have different values of the ith bit is θ/π. Thus, the probability that the sketches have Hamming distance k is given by the binomial distribution $binom(B, \theta/\pi)$. The expectation of $\cos \pi d_\sigma$ can be obtained by taking the average for k from 0 to B.

(c) By linearity of expectation, we only need to consider the case $B = 1$, and we drop the subscript i. We use hyperspherical coordinate system, where each point is represented with one radial coordinate r and $D-1$ angular coordinates $\phi_1, \ldots, \phi_{D-1}$. Because the radius is always 1 in S^{D-1}, we simply omit this coordinate. Moreover, for particular p and q, we rotate the coordinate system so that

$$\phi_i(p) = \phi_i(q) = \frac{\pi}{2} \quad \forall i = 1, \ldots, D-2$$

$$\phi_{D-1}(p) = 0, \quad \phi_{D-1}(q) = \theta.$$

The corresponding Cartesian coordinates are $p = \langle 0, \ldots, 1, 0\rangle$ and $q = \langle 0, \ldots, \cos(\theta), \sin(\theta)\rangle$. Let the hyper-spherical coordinates of the random vector $\rho \in S^{D-1}$ be $\langle \phi_1, \ldots, \phi_{D-1}\rangle$.

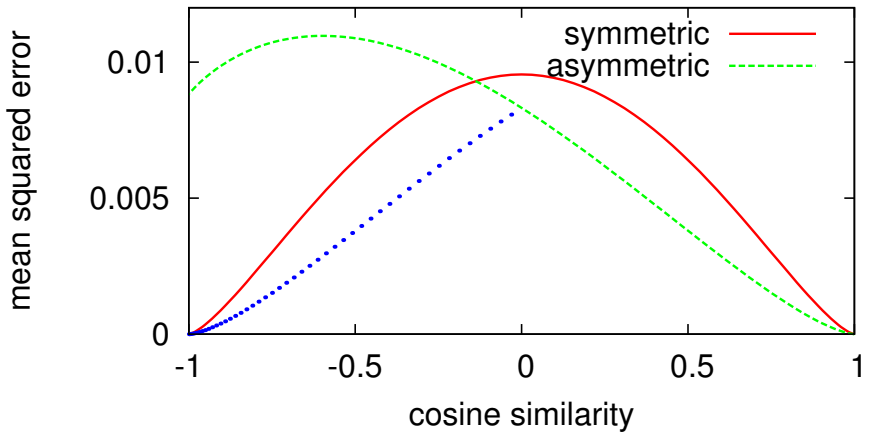


Figure 4.3: Mean squared error of Cosine Sketch estimators vs. cosine similarity ($D = 128, B = 256$). The asymmetric curve has its left half higher than the symmetric one. We use a trick (described in Section 4.3.2) to lower it down to the dotted curve, which mirrors the right half of the asymmetric curve.

We thus have

$$\rho \cdot q \;=\; \sin(\phi_1) \cdots \sin(\phi_{D-2}) \cos(\phi_{D-1} - \theta)$$

Only the hyper-planes that separate p and q make a non-zero contribution to the asymmetric distance. These hyper-planes correspond to $\rho \in \Omega_1 \cup \Omega_2$, where

$$\Omega_1 = \{\rho \,|\, \rho \cdot p < 0, \rho \cdot q > 0\} = \{\rho \,|\, \frac{1}{2}\pi < \phi_{D-1}(\rho) < \frac{1}{2}\pi + \theta\},$$
$$\Omega_2 = \{\rho \,|\, \rho \cdot p > 0, \rho \cdot q < 0\} = \{\rho \,|\, \frac{3}{2}\pi < \phi_{D-1}(\rho) < \frac{3}{2}\pi + \theta)\}.$$

Thus

$$E[d^*_\sigma(p,q)] = \frac{\int_{\Omega_1} \rho \cdot q \, d\rho + \int_{\Omega_2} -\rho \cdot q \, d\rho}{\int_{S^{D-1}} d\rho} = \frac{B(\frac{D}{2}, \frac{1}{2})}{2\pi}(1 - \cos\theta)$$

(d) Similar to (c). □

The symmetric estimator estimates the angle θ instead of the cosine similarity. In practice we use $\cos(\pi d_\sigma)$ as a biased estimator of cosine similarity.

Figure 4.3 shows the mean squared error of both symmetric and asymmetric estimators when $D = 128$ and $B = 256$ based on Lemma 2. We can see that the

asymmetric estimator works well when cosine similarity is close to 1, but degrades badly when smaller than 0. For similarity search under cosine similarity, this works well, for only similarity close to 1 is interesting. For estimating l_2 distance, however, we actually want low error across the whole range. We use the following trick to circumvent this flaw.

For arbitrary $p, q \in S^{D-1}$, we have $d_{cos}(-p, q) = -d_{cos}(p, q)$; and at any time, either $d_{cos}(p, q)$ or $d_{cos}(-p, q)$ is non-negative, and is estimated more accurately. Furthermore, the sketch $\sigma(-p)$ of $-p$ is exactly the binary complement of that of p, and is thus available at query time. As a result, we always compute an estimation for both p and $-p$, and use the better one to produce the final estimation.

4.3.3 l_1 Sketch

Lv et al. [53] introduced a sketch algorithm for weighted l_1 distance. To simplify the presentation, we only consider the unweighted case here. Our asymmetric estimator can be easily adapted to work with the weighted case. Following is the unweighted version of the l_1 sketch algorithm proposed in Lv et al. [53]. Note that the algorithm works only for limited space, and requires the range of each dimension as input.

l_1 Sketch. *Given a closed space $V = [l_1, u_1] \times \cdots \times [l_D, u_D] \subset \mathbb{R}^D$, for a point $p \in V$, its l_1 sketch is a bit vector $\sigma(p) \in \{0, 1\}^B$, each bit $\sigma_i(p)$ produced by*

$$\sigma_i(p) = \sigma_{i,1}(p) \oplus \sigma_{i,2}(p) \oplus \cdots \oplus \sigma_{i,b}(p)$$

$$\sigma_{i,j}(p) = \begin{cases} 0 & \text{if} \quad p_{s_{i,j}} < t_{i,j} \\ 1 & \text{else} \end{cases}$$

$$\forall i = 1, 2, ..., B, \quad j = 1, 2, ..., b.$$

Each bit in the sketch is the XOR of b independently generated bits, and b is a pa-

rameter specified by the user. For each $\langle i, j \rangle$pair, the index $s_{i,j}$ and threshold $t_{i,j}$ are generated in the following two steps:

1. *Pick $s_{i,j}$ such that $\Pr[s_{i,j} = k] \propto (u_k - l_k), k = 1, 2, ...D$;*

2. *Pick $t_{i,j}$ uniformly at random from $[l_{s_{i,j}}, u_{s_{i,j}}]$.*

The symmetric sketch distance between $p, q \in S^{D-1}$ is defined as $d_\sigma(p, q) = \frac{1}{B} d_H[\sigma(p), \sigma(q)]$.

It can be proved that if $b = 1$, the expectation of sketch distance is proportional to l_1 distance. XORing b bits has the effect of increasing the sensitivity to small distance range, similar to what is shown in Figure 4.1(b). Though the theoretical properties of this XOR idea are interesting, no intuitive explanation is given in [53, 54, 77]. Here we give a geometrical view of the XOR method, which is rather straightforward.

Figure 4.2(b) illustrates a bipartition of a 2D space according to the sketch algorithm, with $b = 3$. The three random dimension-threshold pairs partition the space into six rectangular regions, and the XOR scheme induces a gray-white coloring of the regions such that touching regions are always colored differently. Note that regions of the same color, as in l_2 Sketch, are considered as one partition.

Given the geometrical view of the bipartition scheme, one asymmetric estimator is straightforward. We take the distance from the query point q to the closest boundary of the region, which is t_3 in the figure, as the asymmetric distance function d^*. This distance is indicated in the figure by δ. Following is the algorithm of the asymmetric estimator.

l_1 **Sketch.** *Under the setting of l_1 Sketch 1, for data point p and query point q, both in V, the asymmetric sketch distance is defined as*

$$d^*_\sigma(p, q) = \frac{1}{B} \sum_{i=0}^{B} \delta_i(q)[\sigma_i(p) \oplus \sigma_i(q)]$$

where $\delta_i(q) = \min\{|q_{s_{i,j}} - t_{i,j}| \,|\, j = 1, 2, \ldots, b\}$.

Dataset	# Feature Vectors	Dimension
image	4,459,549	128
audio	2,663,040	192

Table 4.1: Summary of datasets.

The drawback of the above asymmetric estimator is that it is not a proper estimator of l_1 distance. It can be shown that when $b = 1$, the asymmetric distance is in fact proportional to the l_2 distance. Nevertheless, as the experimental results in Section 4.4 show, it does improve search quality in practice.

4.4 Evaluation

In this section, we conduct experiments with two real-life datasets to study the performance of the methods described in the previous two sections. We first measure the accuracy of different methods to demonstrate the advantage of sketch methods over traditional dimension reduction methods and the improvement of asymmetric estimators over the basic symmetric ones. We then plug our methods into a similarity search algorithm and evaluate the space reduction achievable by the new asymmetric estimators.

We have shown in Section 4.3.2 that l_2 distance and cosine similarity are essentially equivalent, so we evaluate both l_2 Sketch and cosine sketch under l_2 measure by using the Cosine Sketch to estimate l_2 distance.

4.4.1 Datasets

We use two high-dimensional datasets, i.e., images and audio, in our evaluation. These datasets are reasonably large and reflect real-life usage. Table 1 is a summary of these datasets.

Image: The image dataset is extracted from the Caltech 101 [24] object recognition

benchmark. This dataset contains 101 object categories and a background clutter category, with 9144 images in total. We convert the images into grayscale PGM format and use the SIFT [52] [1] algorithm with default parameters to extract feature vectors, obtaining nearly 4.5 million 128-dimensional feature vectors. We treat feature vectors extracted from the same image as independent objects, as we only consider similarity search on the feature vector level.

Audio: The audio dataset is the LDC-SWITCHBOARD-1 [2] collection, which is a collection of about 2,400 two-side telephone conversations among 543 speakers from all areas of the United States. The conversations are segmented into individual words based on human transcription. We use the Marsyas library [74] to extract feature vectors from the word segments. For each word segment, we take a 512-sample sliding window with variable stride to obtain 32 windows, and from each window extract the first six MFCC parameters, resulting in a 192-dimensional feature vector. About 2.5 million feature vectors are extracted.

4.4.2 More on the Methods

For l_2 distance, we compare the following six methods.

- l_2 Sketch with symmetric and asymmetric estimators. We use window size $W = 900$ for the image dataset, and $W = 23$ for the audio dataset. These parameters are tuned to be optimal for 100-NN search. We build lookup tables based on Lemma 1 to map the sketch distances to l_2 distance for explicit l_2 distance estimation. For similarity search, the sketch distances are directly used.

- Cosine Sketch with symmetric and asymmetric estimators. We use Lemma 2 to directly convert the sketch distances to cosine similarity, and further convert

[1] Code obtained from `http://www.cs.ubc.ca/~lowe/keypoints/`.

[2] `http://www.ldc.upenn.edu/Catalog/docs/switchboard/`

cosine similarity to l_2 distance with saved 2-norms.

- PCA (Principal Component Analysis) and RP (Random Projection), also considered as sketches to serve as performance baselines.

For l_1 distance, we simply compare the symmetric and asymmetric estimators of l_1 Sketch.

We always allocate sketches in full bytes to simplify implementation. Actually, varying sketch size by less than one byte does not make a practically significant difference in performance. For Cosine Sketch, we need the 2-norms of the vectors to estimate l_2 distance, and count 4 extra bytes into sketch size. For PCA and RP, we use 32-bit floating-point representation, and thus count each dimension as four bytes.

We implement asymmetric sketch distances in the following way. Assume an B-bit sketch length, we pre-calculate for each query point an $B \times 2$ matrix M, $M[i][j]$ being the value to be added if bit-i of the data sketch is j. Therefore, evaluating the asymmetric sketch distance for each data point involves B floating-point additions.

4.4.3 Evaluation of Distance Estimation

Because our asymmetric estimator of l_1 Sketch does not produce a proper estimation of l_1 distance, we do not consider l_1 distance here. We only evaluate the l_2 distance related methods. Also, we only use the image dataset, which is sufficient to demonstrate the behavior of different methods.

First, we measure the mean squared error of various methods at different distance values. We randomly sample 100,000 pairs of points from the image dataset, create sketches for them, estimate the distance for each pair with sketches, and then compare the estimations with real distances. We then bin the squared error values according to the corresponding real distances and take the average of each bin. For asymmetric estimators, we use the raw feature vector of one arbitrary point from each pair.

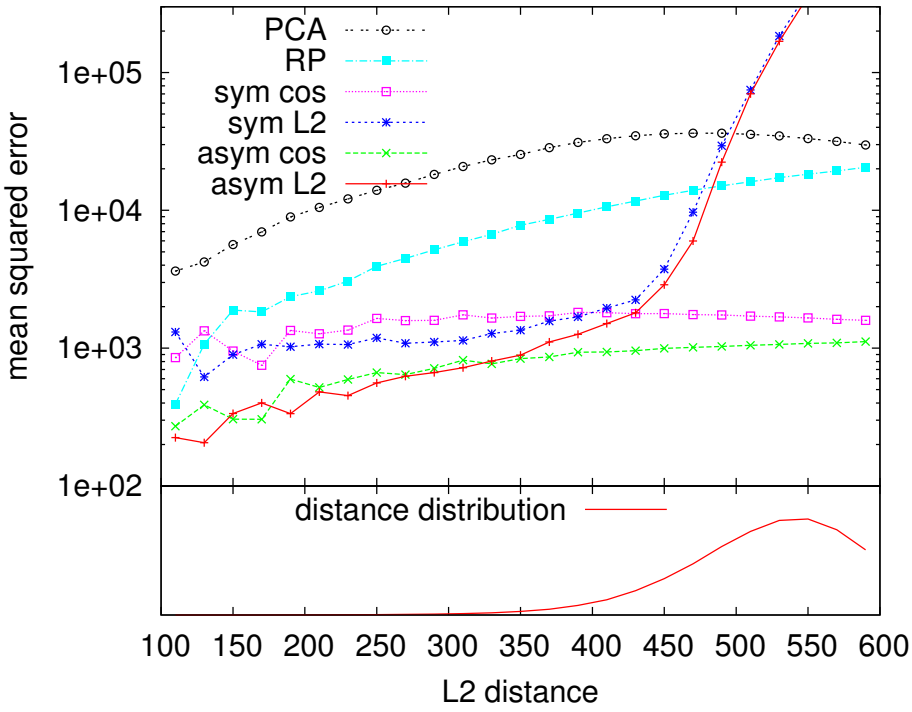


Figure 4.4: Accuracy of various l_2 distance estimation methods on different distance values of the image dataset. Note that only the small distances are interesting to similarity search.

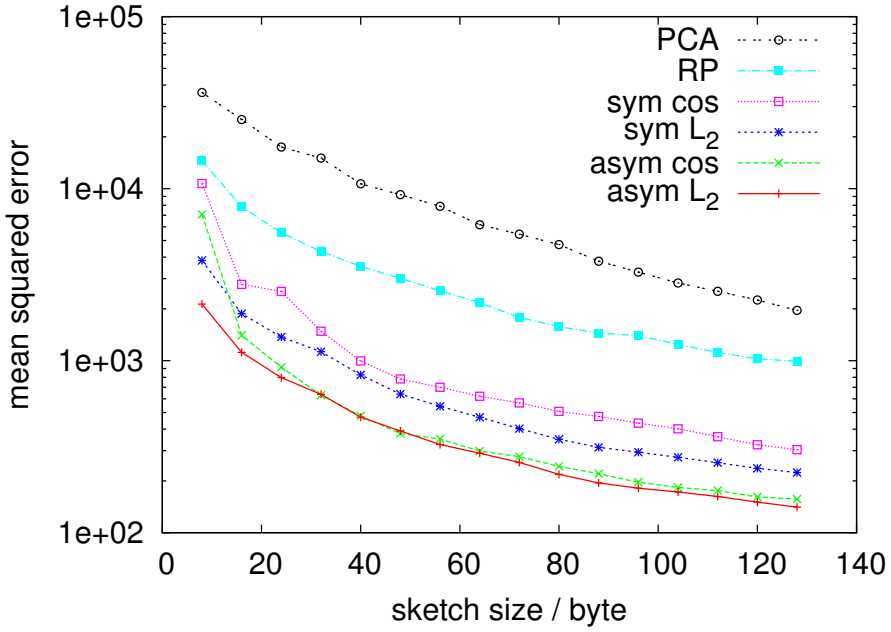


Figure 4.5: Mean squared error of different methods vs. sketch size, image dataset.

The results are plotted in Figure 4.4. We also attach the distribution of real l_2 distance in the bottom of the figure to show the relative importance of different distance values. The mean square error values in Figure 4.4 appear to be large. For example, a means square error of 1000 at l_2 distance 100 translates to a relative error of $\sqrt{1000}/100 \approx 32\%$. However, with sketch construction, we are not interested in estimating the actual distance values, but rather want to filter away those distances that are obviously large, and this error level is acceptable in practice.

Figure 4.4 gives a clear view of how different methods behave for different distance values. The l_2 Sketch curves are most interesting, for the errors are low when distance is small, but beyond 400, the errors increase dramatically as distance grows. Note that the turning point is tunable in practice via the window size W. The curves of the other methods are relatively flat. Especially, the Cosine Sketch estimators are pretty consistent across the whole distance range, and are the first choice if one is interested in estimating distances for arbitrary points rather than for the nearest neighbors only.

We then consider the overall accuracy vs. sketch size. For the purpose of K-NN search, it does not make sense to cover the whole distance range from zero to infinite. Instead, we only consider the distance range from 0 to 300, which is around one standard deviation (~ 60) beyond the average distance of the 100th nearest neighbor (~ 250). We sample from the image dataset 1000 pairs of points that are within this distance range, and take the mean squared error of different methods at different sketch sizes. The results are shown in Figure 4.5.

The curves in Figure 4.5 can be grouped into three categories, from low to high in accuracy: dimension reduction methods, sketches with symmetric estimators and sketches with asymmetric estimators. The relative performances of these methods are mostly consistent across the whole figure. The more than 50% error reduction of the sketch methods over PCA and RP clearly shows their superiority. The figure also shows that the improvement of an asymmetric estimator (from "sym cos" to "asym

cos") is larger than using a better sketch scheme (from "sym cos" to "sym l_2 ").

In the above two figures, RP consistently out-performs PCA in terms of mean squared error. This is mainly because PCA has a systematic bias that makes it always lower-estimate actual distance. This bias does not affect similarity search and we will see that PCA is actually better at similarity search.

4.4.4 Evaluation of Similarity Search

With the accuracy improvements of our new methods confirmed, in this subsection we evaluate the methods in the scenario of similarity search, and see how they improve search quality, and equivalently, reduce the space requirement to achieve a fixed quality.

Here is how we create the evaluation benchmark: for each dataset, we pick 100 points at random as query points, and use the rest of the points as the data points indexed. For each query point, we sequentially scan all the data points for K-NN under each distance measure concerned, and use these results as the ground truth. We use $k = 100$ in our experiments.

We measure search quality by *recall*, which is the percentage of the true K-NN found by the query algorithm. We do not consider *precision* here, but use the equivalent *filter ratios* t and t' as input parameters to the query algorithm explained below.

We use the same query algorithm as modeled in [77]. The query algorithm with symmetric estimators proceed in two steps: first, scan the sketches for a candidate set of $t \times k$ points with smallest sketch distances to the query point; second, scan the raw feature vectors of the candidate set, and find the top k within them as the final result. We use $t = 20$ in all our experiments.

Asymmetric estimators take more computation than the symmetric ones, and filtering all the sketches with asymmetric estimators is not always affordable. As a work-around, we extend the query algorithm with an extra filtering step. We first

scan the sketches with a symmetric estimator for $t' \times t \times k$ candidate points, and then use an asymmetric estimator to rank them and choose the top $t \times k$ candidates for final re-ranking with raw feature vectors. Our experience shows that $t' = 10$, which we use throughout the following experiments, provides nearly identical recall as using asymmetric estimators to scan all the sketches.

The relationships between t, k and recall are thoroughly studied in [77] and are not our focus in this thesis. Instead, we focus on the recall/size performance of different sketch algorithms. Figure 4.6 shows the experimental results of l_2 distance, and Figure 4.7 shows those for l_1 distance.

Figure 4.6 shows that for each sketch method, the asymmetric estimator always out-performs the symmetric one. The improvement is especially significant when the sketch is small. But even at a baseline recall of 0.90, the asymmetric methods still achieve a 0.05 improvement in most cases. The results for l_1 distance, as shown in Figure 4.7, are not as good; however, we do see a consistent improvement.

The performance of PCA on the audio dataset is pretty interesting, as the recall grows very quickly when the size is smaller than 32, or 8 dimensions, reaching one of the sketch methods at 8 dimensions, and then slows down significantly. This suggests that the intrinsic dimension of the audio dataset might be around 10.

Table 4.2 shows the minimal sketch sizes of symmetric and asymmetric methods to reach various recall values. For l_2 distance, the asymmetric estimators achieve sketch size reduction from 20% to 43%. For l_1 distance, the reduction is smaller, from 10% to 27%.

4.5 Related Works

Sketches were originally used to answer aggregate queries over data streams [2], and were recently used as a filter to accelerate similarity search [53, 54, 77]. In existing

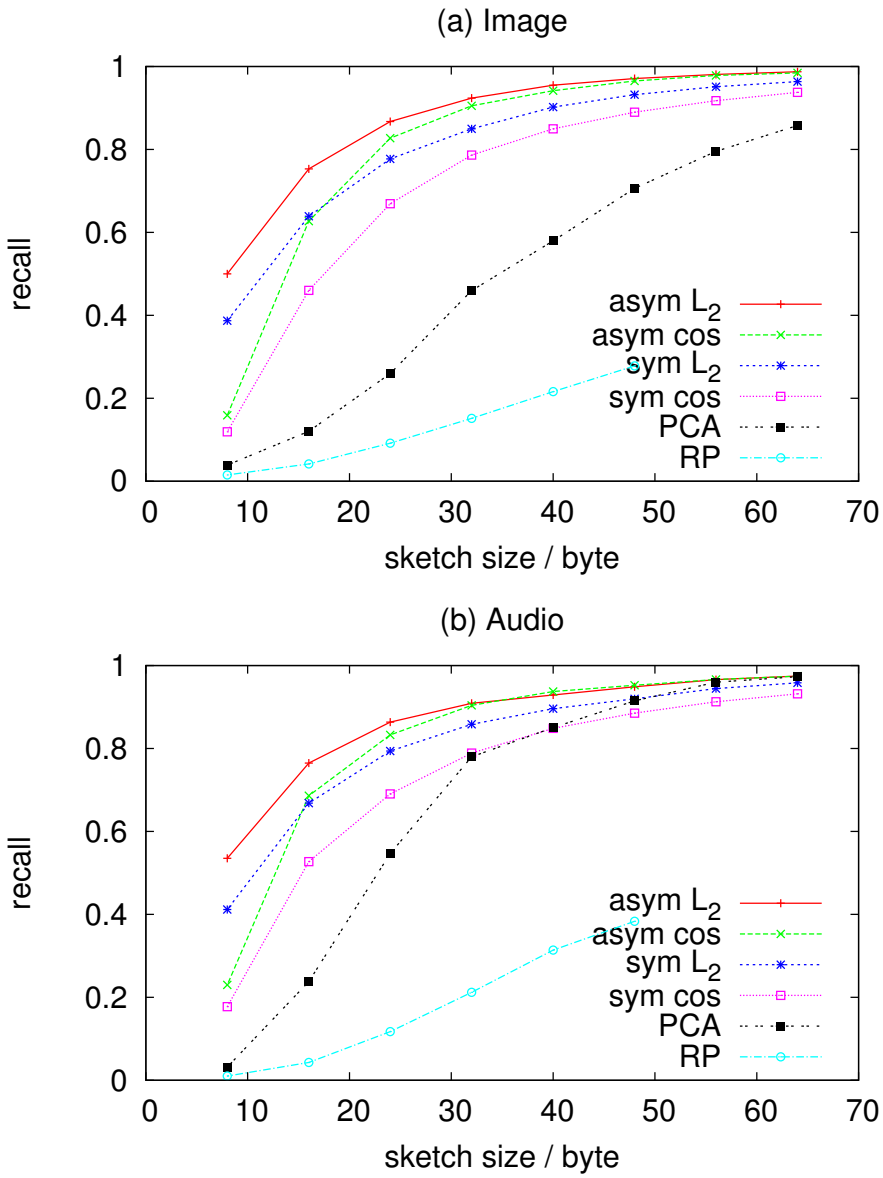


Figure 4.6: Recall vs. sketch size, l_2 distance.

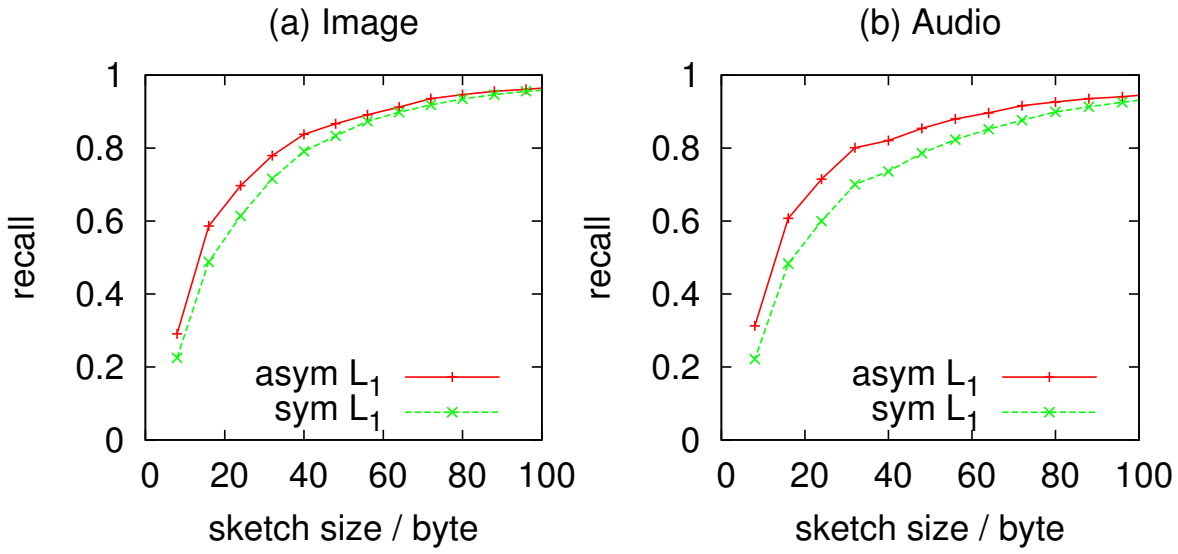


Figure 4.7: Recall vs. sketch size, l_1 distance.

		Image			Audio		
Recall		0.85	0.9	0.95	0.85	0.90	0.95
l_2	sym	32	40	56	32	42	60
	asym	22	29	39	23	32	48
Reduction		31%	28%	30%	28%	24%	20%
Cos	sym	40	54	73	41	54	73
	asym	26	32	42	25	32	46
Reduction		35%	41%	43%	30%	41%	37%
l_1	sym	51	66	92	64	80	120
	asym	43	59	83	47	64	106
Reduction		16%	11%	10%	27%	20%	12%

Table 4.2: Space requirement (in bytes) of various methods to achieve specific recalls. Here l_2 sketch and Cosine sketch are evaluated with l_2 distance, and l_1 sketch is evaluated with l_1 distance.

methods for all these applications, however, the estimating operation is carried out completely in the sketch space. Similarity search is a typical scenario where additional information other than sketches can be exploited to improve estimation accuracy. We believe our idea of asymmetric distance estimation will find other applications as well.

A sketch algorithm for l_1 distance was proposed by Lv et al. [53], and later applied to other datasets [54]. The random hyper-plane technique, first proposed by Goemans and Williamson [32] for solving the max-cut problem and later used for LSH [14], implicitly gives a sketch algorithm for cosine similarity, which is essentially equivalent to l_2 distance. However, its performance in similarity search has not been experimentally studied. In this chapter, we propose asymmetric estimators for both methods, and experimentally evaluate all these methods as well as our newly proposed sketch for l_2 distance.

An analytical performance model of similarity search using sketches was recently given in [77]. We use the same query-processing algorithm, so the performance model can be easily adapted to work with our method.

Dimension reduction [25] is an active research area with many applications, and various methods exist. Though related in concept, sketches are not merely new dimension reduction methods. They are task-specific (similarity search in our case), and the emphasis is more on reducing the size rather than keeping high precision. PCA and random projection are evaluated in this chapter as performance baselines.

4.6 Summary

In this chapter, we proposed the idea of asymmetric distance estimators to exploit the raw query points, which are not used by traditional methods when estimating distances with sketches. We apply the idea to three sketch algorithms, one of them newly proposed in this chapter. Our experimental results confirm the precision im-

provement of the asymmetric estimators, and show that to achieve the same search quality, the asymmetric estimators can reduce the space requirement by 10% to 40%.

Chapter 5

Compact Feature Set Representation with Random Histograms

In this chapter, we present a randomized algorithm to embed a set of features, an increasingly common representation of a feature-rich data object, into a single high-dimensional vector for efficient matching and searching. The main idea is to project feature vectors into an auxiliary space using LSH and to represent a set of features as a histogram in the auxiliary space. We evaluate the proposed approach under two different task settings, i.e. content-based image search and image object recognition. The experimental results show that the proposed approach is indeed effective and flexible. For object recognition with the Caltech 101 dataset, our method runs 25 times faster to achieve the same precision as Pyramid Matching Kernel, the state-of-the-art feature set-matching method.

5.1 Introduction

Using sets of local features to represent multimedia data objects for content-based similarity search or object recognition has been gaining popularity recently because of the superior discriminating power of local features over global features. However, matching sets of unordered high-dimensional feature vectors imposes much higher complexity in both space and time. The challenge is to develop a method to substantially reduce the complexity of the set-matching operation while maintaining the search or recognition quality of using sets of features.

For example, the state-of-the-art approach to object recognition is to extract local features such as SIFT [52] from "interesting" points in an image and form a set of feature vectors for each image. The number of such feature vectors for each image varies depending on its content, but it is common for an image to have a set of hundreds of SIFT features, each with 128 dimensions. Recent search systems for audio [54] and video [80] also use sets of unordered high-dimensional feature vectors as their metadata representations. It is therefore important to develop a flexible and efficient representation for sets of features.

There are two challenging aspects in solving the problem of matching sets of features. The first is that its space requirement is high. Local feature vectors are high dimensional. For example, A SIFT feature vector has 128 dimensions. A typical MFCC feature vector, a commonly-used feature type for audio data, typically has about 200 dimensions. The number of SIFT feature vectors or MFCC feature vectors for each image or audio sentence is typically on the order of hundreds and sometimes thousands. The space requirement for such a set of feature vectors can easily exceed 100KB which is often larger than the size of the original data object.

The second is that the complexity to compute a commonly-used set similarity measure is high. Since the set representation disregards the spatial orders or semantic relationships among the feature vectors, the similarity between two sets of

feature vectors is typically defined as an overall measure of the similarity between the matched points under certain condition from the two sets. A commonly-used similarity measure is the sum of point distances of the one-to-one best matched points from the two sets [69]. The complexity of computing this measure is $O(n^3)$, where n is the average number of feature vectors in a set. If a search engine had to compare the set of features of a query image with that of each of billions of images, the computation cost would be formidably expensive, especially when n is on the order of hundreds and sometimes thousands.

The state-of-the-art light-weight set matching algorithm is Pyramid Matching Kernel (PMK) [34]. Its main idea is to uniformly quantize the feature space in multiple resolutions, and to represent a set of features as a list of the space quanta that cover the set members. Such lists are ordered, and can be matched in linear time, which is a substantial improvement over the quadratic or even cubic set-matching algorithms. However, PMK has two drawbacks. First, it works with feature space of limited dimensionality. For high-dimensional feature vectors, it requires applying a dimension reduction method first which may cause significant loss of information if the intrinsic feature dimensionality is high. Second, the sparse and hierarchical representation imposes a significant space overhead.

This chapter presents a randomized algorithm to embed a set of features into a single high-dimensional vector to simplify the feature-set matching problem. The main idea, as illustrated in Figure 5.1, is to project all feature vectors into an auxiliary space using LSH [36, 31] and to represent a set of features as histograms, which are simply high dimensional vectors. Histogram similarity measures will be employed to approximate feature-set distance measures. By doing so, the challenging feature-set matching problem is reduced to a simpler similarity match problem in a high-dimensional space.

To evaluate the effectiveness of the proposed approach, we conducted experiments

under two different task settings: content-based image similarity search and object recognition. The experimental results show that the proposed approach is indeed effective and flexible. It can achieve search and recognition accuracy comparable to the feature-set matching methods, while requiring significantly less space and time. For object recognition with Caltech 101 dataset, our method achieves the same precision as the feature-set matching method, but runs 25 times faster than Pyramid Matching Kernel.

In addition, the proposed approach is flexible and can easily be used to construct search systems for multiple feature-rich data types with different similarity measures. By using different LSH families, one can approximate different point distance functions. Users can define different histogram distance metrics to satisfy desired set matching conditions. Since histograms are straightforward vectors, it is easy to pass them to existing software modules. Since the dimensionality of the histogram vectors does not explicitly depend on the feature vector's dimensionality nor the set size, users can configure it according to their needs.

5.2 Preliminaries

5.2.1 Similarity, Distance and Kernel

An LSH family $\mathcal{H}$ induces the following similarity measure:

$$s_{\mathcal{H}}(x, y) \triangleq \Pr_{H \in \mathcal{H}}[H(x) = H(y)].$$

Given an arbitrary function $h \in \mathcal{H}$, we also define the corresponding similarity measure as

$$s_h(x, y) = \delta_{h(x),h(y)} = \begin{cases} 1 & \text{if } h(x) = h(y) \\ 0 & \text{if } h(x) \neq h(y), \end{cases}$$

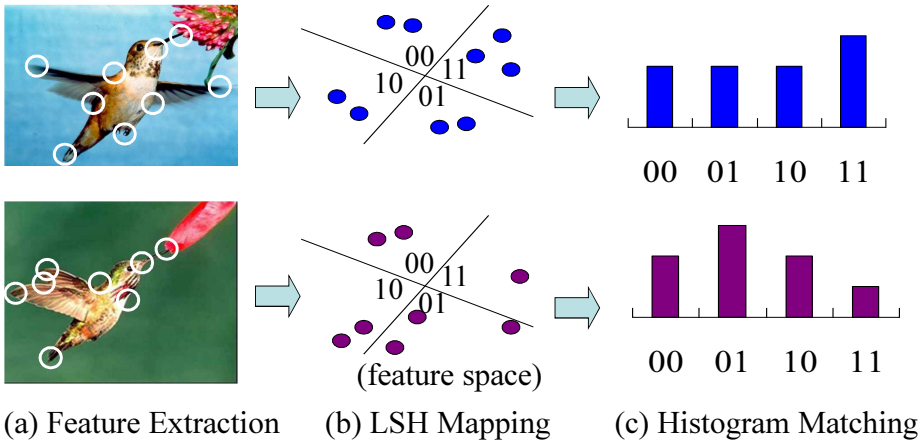


Figure 5.1: Illustration of the proposed approach. (a) Features are extracted from interesting points. (b) Objects are represented as sets of points in the feature space. These points are mapped to discrete values via locality preserving random mappings (LSH). A mapping is equivalent to a partitioning of the feature space. (c) Sets of features are embedded to histograms over the mapping images. The histograms are matched with standard vector similarity measures.

such that $E_{h\in\mathcal{H}}[s_h(x,y)] = s_{\mathcal{H}}(x,y)$.

Set similarities are defined in terms of point similarities. Let X and Y be two finite sets in an arbitrary feature space, and let n be the average set size. We use s to denote an arbitrary point similarity, i.e., $s(x,y)$ is the similarity between points x and y, larger values indicate higher similarity. We use S for the corresponding set similarity, i.e., $S(X,Y)$ is the similarity between sets X and Y of features. We study the following two set similarities which are commonly used in practice.

The first set similarity S_1 is defined in a similar way as the Earth Mover's Dis-

tance [69]:

$$
\begin{aligned}
S_1(X,Y) &= \max \sum_{x\in X}\sum_{y\in Y} a_{x,y}s(x,y) \\
s.t. \qquad & \sum_{x\in X} a_{x,y} \le 1, \quad \forall y \in Y \\
& \sum_{y\in Y} a_{x,y} \le 1, \quad \forall x \in X \\
& a_{x,y} \in \{0,1\}, \quad \forall x \in X, y \in Y.
\end{aligned}
$$

The above assignment problem formation (a special case of integer linear programming) searches for an optimal one-to-one matching between the set members. The coefficient $a_{x,y} = 1$ means that the point x is matched to the point y, and a point can be used at most once. The best-known method to find the optimal matching is the Hungarian algorithm, which takes $O(n^3)$ time. A similar measure can be defined for weighted sets which can be expressed as the optimum of a linear program (in fact min cost flow), which can also be solved in polynomial time.

The second set similarity S_2 is for one-to-many matching, which is defined as

$$
S_2(X,Y) = \sum_{x\in X}\sum_{y\in Y} s(x,y).
$$

The above similarity obviously takes $O(n^2)$ time.

Distance and kernel are two related concepts in measuring similarity (for either points or sets). Distances are more often used in data retrieval tasks, which are commonly formulated as the K-NN problem. Though a properly defined distance metric should be positive, symmetric, and satisfy the triangle inequality, certain measures violating one or more of these conditions, like the Kullback–Leibler divergence, are also considered distance measures in practice. Kernels are most commonly used with support vector machine (SVM) related methods. A well-defined kernel should follow

the Mercer condition (positive definiteness), which ensures the convergence of the SVM learning process. Taking a Mercer kernel between two objects is equivalent to taking the inner product between vectors obtained by mappings of these objects into certain high dimensional l_2 space. This equivalence allows SVM to apply to arbitrary feature space so long as there is a well-defined kernel. Note that higher similarity means a larger kernel value, and a smaller distance.

For set similarity, our proposed approach explicitly maps (embeds) sets of features to high-dimensional vectors, and is thus a complement to the kernel method, where the mapping is implicit. The Mercer condition is automatically guaranteed by the explicit mapping to a vector space. One can essentially try out any distances and kernels with the embedded vectors. In this dissertation, we limit our discussion to l_1 and l_2 distance for set similarity. Given two arbitrary points $x, y \in \mathbb{R}^D$, the l_1 and l_2 distances are defined as follows:

$$l_1(x,y) = \sum_{i=1}^{D} |x_i - y_i| \qquad l_2(x,y) = \{\sum_{i=1}^{D} |x_i - y_i|^2\}^{1/2}$$

We also define the corresponding K_1 and K_2 kernels as follows:

$$K_1(x,y) = \sum_{i=1}^{D} \min\{x_i, y_i\} \qquad K_2(x,y) = \sum_{i=1}^{D} x_i \cdot y_i$$

The following two relationships can be easily verified:

$$l_1(x,y) = K_1(x,x) + K_1(y,y) - 2 \times K_1(x,y)$$
$$[l_2(x,y)]^2 = K_2(x,x) + K_2(y,y) - 2 \times K_2(x,y)$$

The above two equalities essentially have the same form. For normalized vectors, we have $K_i(x,x) = K_i(y,y) = 1, i = 1, 2$, and the relationships are exactly linear. For

unnormalized vectors, there is still a conceptual equivalence. Thus later when set similarity is concerned, we will only discuss in the kernel setting, knowing that the results apply to the corresponding distance metric.

5.3 The Proposed Approach

In this section, we formalize the random histogram construction algorithm and establish a correspondence between the histogram kernels K_1, K_2 and the set similarity measures S_1, S_2.

5.3.1 Random Histograms Construction

Our approach is to represent a set of features as a histogram. However, creating a histogram in the original feature space is not always possible if the feature space is not simply a vector space, or may lead to several issues: the histogram can be extremely large and sparse if the dimensionality is high, and multiple resolutions may be needed to attain reasonable accuracy. To avoid such issues, we propose to first project the feature space into an auxiliary space, and then create histograms on the auxiliary space. The mapping used for projection should preserve the locality information, so point similarity is preserved in the auxiliary space. LSH is a perfect tool for this purpose.

We first formalize the concept of *histogram* as used in this dissertation. Given a finite set $\mathbb{A}$, let $\{\mathbf{e}[i] \,|\, i \in \mathbb{A}\}$ be the standard basis of the $|\mathbb{A}|$-dimensional vector space, so that for any $i, j \in \mathbb{A}$,

$$\mathbf{e}[i] \cdot \mathbf{e}[j] = \delta_{i,j}.$$

A histogram G over the set $\mathbb{A}$ is represented as a point in the $|\mathbb{A}|$-dimensional vector space, i.e., $G = \sum_{i \in \mathbb{A}} \alpha_i \mathbf{e}_i$, with α_i being the "count" of bin i.

We then formalize how to embed a set of features into a histogram. Let $\mathbb{M}$ be an arbitrary feature space, and $\mathcal{H}$ be an LSH family of mappings from $\mathbb{M}$ to $\mathbb{A}$. Note that $\mathbb{M}$ can be any space that admits an LSH family, not necessarily a vector space. We define the family of mappings

$$\mathcal{G}_{\mathcal{H}} = \{g_h : 2^{\mathbb{M}} \to \mathbb{R}^{|\mathbb{A}|} \mid h \in \mathcal{H}\},^{1}$$

such that for any $X \subset \mathbb{M}$ and $h \in \mathcal{H}$, X is mapped to a histogram on $\mathbb{A}$:

$$g_h(X) = \sum_{x \in X} \mathbf{e}[h(x)].$$

Because the histogram $g_h(X)$ is determined by the hash function h chosen at random from $\mathcal{H}$, we call it a *random histogram*.

The above scheme can be easily extended to work with sets of weighted features. If $X' \subset \mathbb{R} \times \mathbb{M}$ is a set of weighted features, then we define

$$g_h(X') = \sum_{\langle w,x \rangle \in X'} w \times \mathbf{e}[h(x)].$$

We assume an unweighted set for the rest of this chapter because this extension is straightforward.

In practice, we maintain N independent random histograms to improve accuracy. Specifically, we choose N hash functions $H = \langle h_1, \ldots, h_N \rangle$ from $\mathcal{H}$ independently, and concatenate them to form a "super" histogram:

$$g_H(X) = \langle g_{h_1}(X), \ldots, g_{h_N}(X) \rangle.$$

We extend any similarity measure function $f(\cdot, \cdot)$ on single histograms to this "super"

[1] For an arbitrary set $\mathbb{M}$, the power set $2^{\mathbb{M}}$ is the set of all subsets of $\mathbb{M}$.

histogram:

$$f[g_H(X), g_H(Y)] = \frac{1}{N}\sum_{i=1}^{N} f[g_{h_i}(X), g_{h_i}(Y)].$$

In doing so, we keep the expectation the same, while lowering variance by a factor of $1/N$.

For most applications, the scaling factor $1/N$ will not affect the search or learning result, and the "super" histogram can be treated as a single vector.

5.3.2 Matching Random Histograms

Recall that we set out to describe our technique to efficiently estimate similarity between sets of feature vectors. So far we have explained the representation of sets of features as histograms. Now we explain how we compute the similarity for these histograms. In principle, one can use any histogram matching distances or kernels with the random histogram. In this dissertation, we discuss two special cases, i.e., the K_1 and K_2 kernels, corresponding to two set similarity measures S_1 and S_2 (introduced earlier) that have been proved practically effective. Given two sets $X, Y \subset \mathbb{M}$, and a mapping $h \in \mathcal{H}$, the kernels $K_{1,h}$ and $K_{2,h}$ induced by the mapping h are just the K_1 and K_2 kernels in the histogram space:

$$K_{1,h}(X,Y) = \sum_{i\in\mathbb{A}} \min\{g_h(X)\cdot\mathbf{e}[i],\ g_h(Y)\cdot\mathbf{e}[i]\}$$

$$K_{2,h}(X,Y) = \sum_{i\in\mathbb{A}} \{g_h(X)\cdot\mathbf{e}[i]\} \times \{g_h(Y)\cdot\mathbf{e}[i]\}.$$

We will explain how these two histogram kernels relate to the two set similarity measures S_1 and S_2.

One-to-One Matching

The kernel $K_{1,h}$ corresponds to one-to-one matching. The intuition is that points hashed into the same bin by h can be matched to each other, and for each bin, the set with fewer points in this bin is always guaranteed to have its members matched. Formally, the point similarity measure induced by h is s_h and the corresponding set similarity measure for one-to-one matching is $S_{1,h}$. It can be easily verified that

$$K_{1,h}(X,Y) = S_{1,h}(X,Y) \quad \forall X, Y \in \mathbb{M}.$$

However, this equality does not strictly carry over to the similarity $s_{\mathcal{H}}$ and $S_{1,\mathcal{H}}$ which are actually interesting. Though $s_{\mathcal{H}}(X,Y) = E[s_h(X,Y)]$, we only have $E[K_{1,h}(X,Y)] \geq S_{1,\mathcal{H}}(X,Y)$. In the worst case, $E[K_{1,h}(X,Y)]$ can be significantly larger than $S_{1,\mathcal{H}}(X,Y)$.

Nevertheless, s_h does contain information about $s_{\mathcal{H}}$. In practice, using $K_{1,h}$ as a biased approximation of $S_{1,\mathcal{H}}(X,Y)$ yields good results when one-to-one matching is desirable.

One-to-Many Matching

The kernel $K_{2,h}$ corresponds to one-to-many matching, and we show a direct connection between the histogram kernel $K_{2,h}$ and the set similarity measure $S_{2,\mathcal{H}}$. We have

$$E_{h\in\mathcal{H}}[K_{2,h}(X,Y)] = S_{2,\mathcal{H}}(X,Y), \quad \forall X, Y \subset \mathbb{M}.$$

This can be seen by the following:

$$\begin{aligned} & K_{2,h}[X,Y] \\ =& g_h(X) \cdot g_h(Y) = \left\{ \sum_{x \in X} \mathbf{e}[h(x)] \right\} \cdot \left\{ \sum_{y \in Y} \mathbf{e}[h(y)] \right\} \\ =& \sum_{x \in X} \sum_{y \in Y} \mathbf{e}[h(x)] \cdot \mathbf{e}[h(y)] = \sum_{x \in X} \sum_{y \in Y} \delta_{h(x),h(y)} \end{aligned}$$

Because $E_{h \in \mathcal{H}}[\delta_{h(x),h(y)}] = \Pr_{h \in \mathcal{H}}[h(x) = h(y)] = s_{\mathcal{H}}(x,y)$, the relationship can be obtained by linearity of expectation.

5.4 Point Similarity and LSH

In order to facilitate tuning of histogram sizes to provide a size/performance trade-off, we choose LSH families of a special form: the B-bit sketch presented in Chapter 4. We pick an atomic 0/1-valued LSH family $\mathcal{F}$ according to the similarity measure of interest, and concatenate B independent atomic hash functions to make a B-bit hash function: $\mathcal{H} = \{\langle f_1, \ldots, f_B \rangle \mid f_i \in \mathcal{F}\}$. We use $\mathcal{H}$ to build random histograms. Because $\mathcal{H}$ is a B-bit function with 2^B possible values, the histograms are of size 2^B.

Concatenating B atomic mapping has the effect of enhance locality sensitiveness. The similarities induced by $\mathcal{H}$ and $\mathcal{F}$ have the relationship $s_{\mathcal{H}}(x,y) = [s_{\mathcal{F}}(x,y)]^B$ for any $x, y \in \mathbb{M}$. Figure 5.2 shows such a relationship. The effect of a larger B is to have a smaller range (at the high end) of point similarity that actually contributes to the set similarity (i.e. only very similar pairs of points have an effect on the set similarity value). In practice, this often means higher discriminative power, and thus higher retrieval or recognition accuracy, but at the same time, a larger histogram size is required. Such threshold effect has been proved effective in previous studies [53, 57].

An easier approach to tune histogram size is to use mappings that produce an integer value, and use the modulo operation to limit the range of this value. One

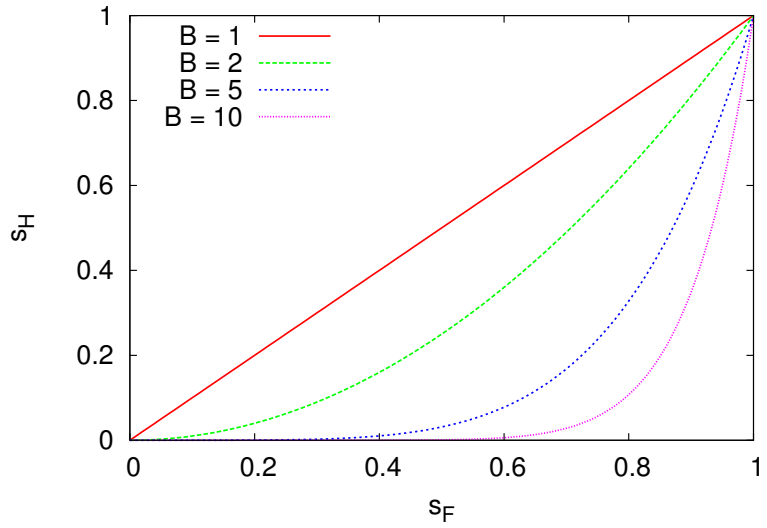


Figure 5.2: Concatenating B atomic mappings enhances the locality sensitiveness.

potential drawback of this approach is that the mapping might concentrate on a few values, resulting in a biased histogram with some bins more often used than others.

5.5 Practical Issues

5.5.1 Compact Histogram Representation

Usually the histogram size needs to be large for high discriminative power. The result is that 1) the histogram can be very sparse and 2) the count of each bin is small. Next, we describe two practical techniques for compact histogram representation. These methods are specially designed to work with random histograms for unweighted features. Other general dimension reduction methods like PCA, and sketches can be applied to the histograms for lossy compression.

Folding

The folding technique applies to the K_2 histogram kernel (inner product). The main objective of folding is to produce a more accurate estimator of similarity without increasing the space and time requirement. Basically the histograms corresponding to several different hash functions are added together; hence the name. The K_2 histogram kernel applied to this folded histogram has roughly the same expectation as the kernel applied to each individual histogram, but has lower variance. This technique is based on the following observations: the same LSH function causes similar points to collide, but if two points are mapped with different LSH functions, the values are independent. Specifically, for two points x, y, and two different B-bit LSH functions h_1, h_2, if a random atomic hash function is equally likely to map a point to 0 or 1 and if the atomic hash functions are picked independently of each other, we have

$$\Pr[h_1(x) = h_2(y)] = 2^{-B}$$

The probability is close to 0 when B is large. As a result, for two sets X and Y, $g_{h_1}(X) \cdot g_{h_2}(Y)$ should also be small:

$$E[g_{h_1}(X) \cdot g_{h_2}(Y)] = \sum_{x \in X} \sum_{y \in Y} \Pr[h_1(x) = h_2(y)] = 2^{-B} |X|\,|Y|.$$

Our folding technique is simply to add multiple independent random histograms together to make a single vector. Given M independent LSH mappings $H = \{h_1, h_2, \ldots, h_M\} \subset \mathcal{H}$, we use the following conceptual random histogram

$$g_H(X) = \frac{1}{\sqrt{M}} \sum_{i=1}^{M} g_{h_i}(X).$$

We have

$$
\begin{aligned}
& E[g_H(X) \cdot g_H(Y)] \\
=& E[\left\{ \frac{1}{\sqrt{M}} \sum_i g_{h_i}(X) \right\} \cdot \left\{ \frac{1}{\sqrt{M}} \sum_j g_{h_j}(Y) \right\}] \\
=& E[\frac{1}{M} \sum_i g_{h_i}(X) \cdot g_{h_i}(Y) + \frac{1}{M} \sum_{i \neq j} g_{h_i}(X) \cdot g_{h_j}(Y)] \\
=& E[g_{h_1}(X) \cdot g_{h_1}(Y) + (M-1) E[g_{h_1}(X) \cdot g_{h_2}(Y)] \\
=& E[g_{h_1}(X) \cdot g_{h_1}(Y)] + (M-1) 2^{-B} |X|\,|Y|.
\end{aligned}
$$

Thus the value of the folded estimator is essentially an unbiased main term plus an error term. The error term only depends on the sizes of X and Y, and is not locality sensitive. The effect of folding is that the error is reduced by lowering the variance of the main term (roughly by a factor of M), with a small amount of extra error introduced. When the similarity value is higher than a certain threshold, the main term is dominant, and the overall effect of folding is positive. In practice, high similarity values are more important, and thus folding can potentially improve the end performance. The limitation is that folding is only helpful when the histograms are very sparse, and the parameter M cannot be too large. We use both folding and concatenation in our final scheme.

Bit-Coding

If bin sizes of the histogram are still small after folding, it will be wasteful to use full integers or floating point numbers to represent such small sizes. We can fit multiple bins into a single byte via bit-encoding. Assume we encode each bin with $C = 1, 2, 4, 8$ bits, then each byte can hold $8/C$ bins. If the count of a bin is larger than 2^C, the exceeding part is trimmed. When computing the kernel functions online, it would be

slow if we first extract the bin counts from the bit-encoding. We thus use a lookup table of 256×256 entries with pre-computed kernel values. By doing so, the $8/C$ multiplications or minimum operations are substituted with a single memory lookup. However, bit-encoding is a specialized representation which might not be supported by existing software, and we do not use it in the experimental evaluation.

5.5.2 Overall Scheme and Parameter Tuning

Figure 5.3 illustrates the whole scheme of our proposed method, showing all the tunable parameters. To summarize, there are three operations involved in constructing the final vector representation:

1. A B-bit LSH function generates a simple histogram.
2. The vector sum of M independent simple histograms forms a folded histogram.
3. N independent folded histograms are concatenated to form the final vector representation.

There are in total $M \times N$ B-bit LSH functions involved. These LSH functions are randomly sampled at system initialization and remain fixed throughout the lifespan of the system. It is meaningful to match two random histograms only if they are generated with the same set of LSH functions.

The complexity of our method has two aspects — off-line and online, and in practice, the latter is usually more important. Assume the feature space has a dimension of d, and the average set size is n. The off-line part involves generating the hash values and adding to the histogram bins. The histogram has $2^B N$ entries, but it is usually very sparse. After an empty histogram is created, the embedding process takes time $O(dnBMN)$. The online part depends entirely on the histogram size, and thus takes time $O(2^B N)$. The online part does not explicitly depend on the set size or the dimensionality, and can be potentially very fast.

We briefly comment on how to tune each of the parameters for a given dataset.

- The parameter B determines the size of a single histogram. Larger B usually improves the accuracy, but exponentially enlarges the histogram size. So B is more limited by available space. A good starting point for tuning B is to make the histogram size (2^B) similar to the average set size.

- The parameter N lowers the variance of the result. This is necessary as our method is a randomized algorithm. Our suggested range of values for N is 10 to 50.

- The parameter M determines the number of histograms folded together. It depends on the histogram size, and can be slightly enlarged as the histogram expands. Our experiments show that M is mostly useful when less than 5.

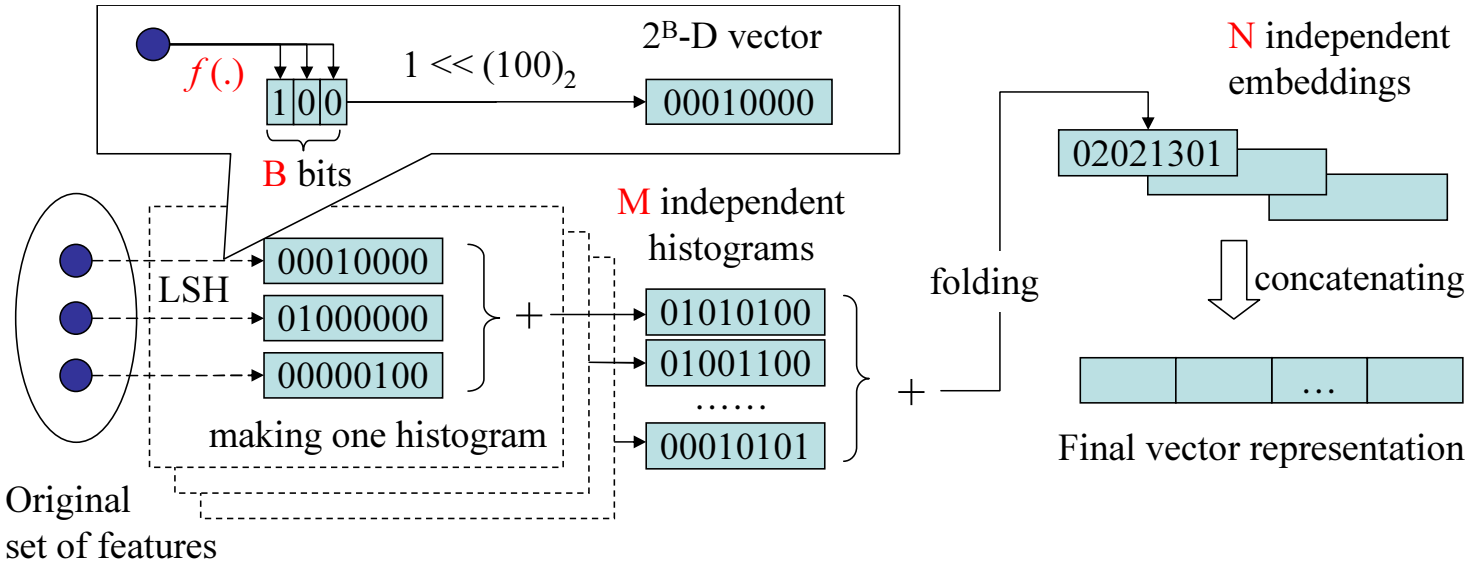


Figure 5.3: Overview of the random histogram embedding scheme

- The LSH family used actually has the largest impact on the result. The right LSH family depends on the dataset, and different choices of LSH need to be tested if the feature extraction algorithm does not suggest an obvious choice.

5.6 Evaluation

In this section, we evaluate the random histogram technique with real-life datasets under two different task settings, i.e., object recognition and content-based image retrieval. The tasks are experimented with different configurations of parameters to illustrate the flexibility of our method—they either use different LSH families or different set similarity measure. We show the comparable accuracy of our method with the state-of-the-art set matching techniques, mainly EMD [69] and PMK [34], as well as our superiority in speed. The effect of different parameters to accuracy is evaluated in detail with the Caltech-101 dataset to serve as a guide for parameter tuning.

5.6.1 Object Recognition

This task is to recognize the semantic category of images. We evaluate our method with two datasets, i.e., Caltech-101 and Graz-01. The former focuses on distinguishing between different object categories, and the latter emphasizes on predicting whether an object of certain single category appears in the image or not,

Following the previous studies, we use SVM for machine learning. Because the random histograms are just high-dimensional vectors, off-the-shelf software can be directly applied.

Caltech-101

The Caltech-101 dataset [24] contains 101 object categories, with 40 to 800 images per category. The names of the categories were chosen from the Webster Collegiate Dictionary and the images were manually selected from Google image search results. Most of the images are of medium resolution, i.e., about 300×300 pixels, and contain little or no clutter. The objects tend to lie in the center of the images and to be presented in similar poses. Images from some categories are artificially rotated and have partially black background.

We evaluate our method with the same features as used in the original PMK paper [34]. The features were extracted on a uniform grid from the images, and each set has on average 1140 features. Each feature is the concatenation of a 10D PCA-SIFT [42] descriptor and a 2D positional feature. The standard evaluation protocol with the dataset is used, i.e., 15 images from each category are selected as training example, and a model is trained with the training images of all categories. The model is then used to predict the category of the remaining images. The recognition rate of each category is averaged as the overall performance.

Table 5.1 compares the performance of our method and some of the previous methods based on the set-of-feature model. Zhang [86] use an EMD based kernel, but their feature sets are slightly different from ours. First, they use four combinations of interesting point detectors (Harris + Laplacian) and descriptors (SIFT and SPIN), each channel is matched individually and the sum of the channel similarity is used as the kernel value for learning. Second, when matching in a single channel, the features of an image are first clustered, and EMD is applied upon the clusters as weighted features. This method uses more information than [34] and our method, and is the best result so far achieved by a pure set-of-feature model. The Spatial Pyramid Matching Kernel proposed by Lazebnik [49] represents the best result achieved on the dataset among all existing approaches. They divide the features into different

Grauman [34]	Zhang [86]	Lazebnik [49]	Ours A	Ours B
0.50	0.539	0.646	0.497	0.541

Table 5.1: Comparison of different methods with Caltech-101 benchmark. We test two different configurations of our methods. Both use the LSH for l_2 distance and match histograms with K_2 kernel. Other parameters are: (A, for speed) $B = 6, M = 5, N = 20$; (B, for accuracy) $B = 10, M = 20, N = 20$.

channels by clustering the features of all the images, and then do pyramid match in each channel with the positional features. We see in Table 5.1 that our fast configuration achieves roughly the same performance of PMK, and the slow configuration approaches that of Zhang [86]. We run our experiment on a Pentium 4 2.2GHz CPU. The fast configuration takes only 4×10^{-6} second to match a pair of images, while the number reported for PMK on a slightly faster machine (2.4GHz) was 1×10^{-4} [34]. Thus our method is 25 times the speed of PMK for matching images from Caltech-101 benchmark to achieve the same recognition performance.

Lazebnik [49] out-performs all other methods in this task. This is partly due to that the images in the dataset tend to present in similar poses, and Lazebnik [49] is specially optimized for such tasks. We can see that with the Graz-01 dataset, which has more variations, the performance difference is not this significant.

Figure 5.4 shows the effects of different parameters on the performance. These curves are just as expected in our discussion in the previous section. Generally, enlarging the single histogram size (B), folding more histograms (M) and concatenating more histograms (N) all lead to higher performance, and all have a threshold beyond which the performance improvement becomes very slow. For B the threshold is around $8 \sim 10$, and for N it is 20. For M, the threshold depends on the histogram size, and folding tends to add more to the performance when histograms are large.

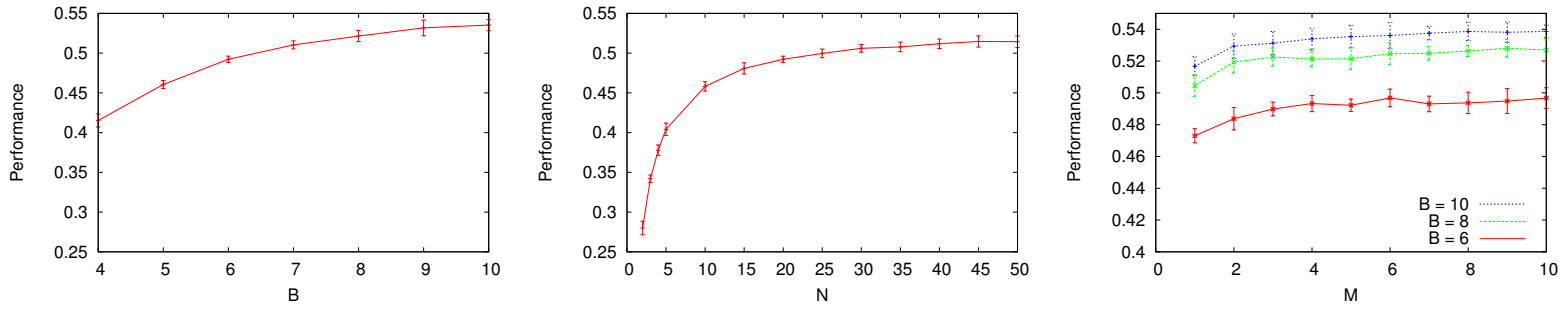


Figure 5.4: Recognition performance vs. different embedding parameters. The embedding has $N \times 2^B$ dimensions, and the figures indicate that accuracy generally increases with dimensionality. When histograms are sparse (with large B), folding (increasing M) can also improve accuracy without extra space requirement.

We also evaluated our method on a similar but easier dataset ETH-80 [50] with the same protocol as used in a previous study [34]. We achieve an accuracy of 84.3%, using the configuration $B = 8, M = 20, N = 20$, which is similar to the performance of PMK, which is 84%.

Graz-01

The Graz-01 dataset [62] contains two object classes, bike (373 images), person(460 images), and a background class(270 images). This dataset is of high intra-class scale and pose variation. We use the Difference-of-Gaussian (DoG) detector and SIFT descriptor for feature extraction [52].

We perform two-class (object vs. background) categorization using the same experimental setup as used in the original study [62]. For each object class, we train a model on 100 positive and 100 negative images (of which 50 are drawn from the other object class and 50 from the background class). According to the standard evaluation protocol of this dataset [62], we tune the parameters to make the following two rates equal to each other and report that rate:

$$\frac{\#\text{ true positive examples}}{\#\text{ positive examples}} = \frac{\#\text{ true negative examples}}{\#\text{ negative exampes}},$$

where true positive (negative) examples are the positive (negative) examples correctly classified. This measure is called the *ROC equal error rate* and captures the accuracy of the algorithm when it is equally sensitive to positive and negative examples.

We show the recognition rates in Table 5.2. We optimize the parameters of our method for accuracy, and our result is on a similar level as other methods. Here the gap between our method and Zhang [86] is more obvious, showing accuracy loss of our randomized algorithm and its potential limitation in application to tasks requiring very high accuracy.

	Opelt [62]	Zhang [86]	Lazebnik [49]	Ours
Bike	0.865	0.920	0.863	0.883
Person	0.808	0.880	0.823	0.805

Table 5.2: Comparison of different methods with Graz-01 benchmark. We use LSH for l_2 distance and match histograms with K_2 kernel. Other parameters are: $B = 12, M = 20, N = 20$

5.6.2 Content-Based Image Retrieval

We use the same image collection and feature extraction algorithm as in [53], where content-based image retrieval with a tweaked version of EMD distance is studied. The image collection consists about 10,000 general-purpose images. We use 32 sets of similar images as the ground truth in our evaluation. These sets are manually selected, and each of these sets contains images that are visually similar. We use JSEG [22] to segment the images into regions. The average number of regions per image is 7.16. We extract from each region a feature vector of 14-dimensions, including 9 dimensions of color moment information and 5 dimensions of bounding box information. We use the region weight that is proportional to the square-root of the region's size, which was shown to be better than the raw region's size [53].

We use average precision to measure the effectiveness of the retrieving method. Given a query and k relevant items excluding the query, let rank_i be the rank of the ith retrieved relevant item, then average precision is defined as following:

$$\text{average precision} = \frac{1}{k}\sum_{i=1}^{k}\frac{i}{\text{rank}_i}.$$

Table 5.3 compares four methods: SIMPLIcity [76], EMD, EMD* and random histogram. The performance of random histogram is the same as EMD. The highest performance is achieved by EMD*, which involves a thresholding tweak that is highly specialized to the particular feature extraction method.

Though we cannot directly compare the image retrieval speed with Lv et al. [53]

Method	SIMPLIcity	EMD	EMD*	Ours
Avg. Prec.	0.331	0.548	0.615	0.548
Time/match	N/A	5.0e-5	5.0e-5	3.1e-7

Table 5.3: Comparison of SIMPLIcity, EMD, EMD* and out method with a 10K image benchmark. We use LSH for l_2 distance, with $W = 1$ and match histograms with K_2 kernel. Other parameters are: $B = 4, M = 1, N = 50$.

without implementing the system, we note that the retrieval process of Lv et al. [53] involves first identifying with efficient sketch filtering a candidate set that covers 5% of the whole dataset, and then re-ranking the candidate set with EMD. Even if we ignore the time spent on sketch filtering, our method is at least $(5.0 \times 10^{-5}) \times 5\%/(3.7 \times 10^{-7}) \approx 8$ times faster than Lv et al.'s system, which is reported to be 3.5 times faster than SIMPLicity [53]. In practice, our method can be further accelerated by various high-dimensional indexing data structures.

5.7 Related Works

Existing methods for measuring set similarity roughly fall into two categories. Methods in the first category depend on a given point similarity measure, and directly work on the sets by matching the member points. The S_1 similarity for one-to-one matching [69] and S_2 similarity for one-to-many matching [57] are two commonly used approaches. Such methods usually involve high on-line computational cost. Furthermore, storing raw sets of high-dimensional features can easily outgrow the limited main memory, leading to an out-of-core implementation and further slowing down the computation.

Methods in the second category, on the other hand, calculate set similarity via certain intermediate representation. They assume that the points are from certain metric space, often the Euclidean space, making the point similarity measure implicit. The intermediate representation can be a histogram created by quantizing the space. For

example, Pyramid Matching [34, 37] corresponds to scalar quantization, and the bag-of-word model commonly used in computer vision often use vector quantization [86]. These methods either require large space to store the high dimensional sparse histograms, or long running off-line clustering. The intermediate representation can also be parameters of certain distributional model, e.g., Gaussian-mixture [56], though such methods also tend to be CPU intensive, and are not as general.

Among the various methods mentioned above, the pyramid method [34, 37] and the bag-of-word model are the two that are most related to our work. They scalar-quantize the feature space in multiple resolutions and maintain a histogram for each resolution. Though in practice a sparse data structure is used, the cost in space is still significant when the feature space is of high dimensionality, and dimension reduction methods like PCA need to be applied first.

The bag-of-word model works by clustering the feature vectors into a fixed number of key words, turning a set of points into a bag of words. The clustering process is essentially vector-quantization of the feature space. The mapping from feature vectors to key words induced by the clustering result can be viewed as a deterministic LSH, while from the other perspective, our method is just a randomized version of the bag-of-word model.

LSH can also be used to construct bloom filters like ordinary hash function [44]. Such bloom filters have a special property that allows one to estimate the distance between a point and the sets of point embedded in the bloom filter. The random histogram proposed in this chapter can be viewed as an extended version of such bloom filter. We not only record whether each bucket is empty, but also keep a count for each bucket.

The compact data structure proposed by Lv et al. [53] to approximate EMD distance is actually a specialized version of our method. Their final data structure is more compact than ours via a further level of sampling and sketching, but they

need to maintain the raw data for re-ranking due to the precision loss of compression. With our method, the raw data is no longer needed.

5.8 Summary

In this chapter we propose an efficient method for representing and estimating similarities between sets of features. Representing multimedia objects as sets of local features has been quite successful recently, but suffers from high storage requirements and high time complexity. Our main idea is to project the feature points into a compact auxiliary space with LSH mappings, and then represent the sets as histograms on the auxiliary space. The histograms are just standard high-dimensional vectors. Our method has the flexibility that allow the user to approximate different measures of point similarity by choosing corresponding LSH families, and implement different measures of set similarity with appropriate choice of histogram distances/kernels. Experiments with various datasets under different task settings show that our method has accuracy comparable to the best-known set matching algorithms, and has much higher speed. For example, to achieve the same recognition performance on the Caltech-101 dataset, our method is 25 times the speed of Pyramid Matching Kernel for set matching, the latter being the fastest known set matching algorithm. Because our histogram embedding is essentially a global feature extracted from the intermediate local feature representation, our results show that global features, when properly extracted, can also achieve high accuracy while maintaining a significant speed advantage.

Chapter 6

A Large-Scale Image Search Engine

In this chapter, we present a large-scale near-duplicate image search engine as a demonstration of the sketch technique we proposed. We also propose novel domain-specific techniques, namely entropy-based filtering and graph-based query expansion, to solve the challenges associated with the large data scale. Our search engine is capable of serving more than 50 million web images on a single commodity server and respond to queries within seconds. Our system also achieves a reduction, by three orders of magnitude, in the false positive rate (at 80% recall) over the state-of-the-art method.

6.1 Introduction

Near-duplicate image detection is the task of finding different versions of the same image, i.e., images that are not exact duplicates in binary form, but can be visually identified as the same image having undergone various editing steps such as color mapping, scaling, format changing, etc. In particular, we are also interested in detecting partial-duplicates, i.e., images with padding and cropping, but still containing sub-images that are near-duplicates. Near-duplicates are common in web images. Being able to efficiently detect them is important to applications such as copyright

violation detection and finding alternate versions of existing images. Eliminating near-duplicates is also an important preprocessing step for many applications.

Previous researches have achieved high quality search results in small to medium scale (one million images or less) experiments [43, 61, 17, 81], but the challenge remains in pushing the retrieval scale to the World Wide Web level. Today's web contains hundreds of billions of images and continues to grow rapidly. Both commercial search engines and academic experiments run on a much smaller scale. For example, Tineye (`http://www.tineye.com/`), the first commercial near-duplicate image search engine available to the public, indexes 1.8 billion images as of 2010. In the literature, a recent system [81] achieved a 1.9 second search time on one million images with a single machine.

Our goal is to design a building block for a large-scale near-duplicate image search engine — a single node system that substantially improves the search capacity over existing approaches without increasing search time, and most importantly, conducts searches with high confidence, i.e., low false positive rate. The false positive problem is not prominent in traditional content-based image search due to the lack of an objective and unambiguous definition of visual similarity. In our case, there is a clear yes-or-no answer to whether two images are near-duplicates, and one can easily spot irrelevant images in the search result, so it is important for the search result not to be dominated by false positives. The challenge is that there are usually only a few (a few to thousands) true positives, and when multiplied by a virtually unlimited number of background images (e.g., 1 billion), even a very small false positive rate (e.g., 0.1%) could potentially generate an overwhelming number of false positives.

Even though various approaches have been published, they all fail to meet our efficiency and/or quality requirements. As a local feature is essential for detecting partial-duplicity, and SIFT [52] has been firmly established in the literature as the feature of choice for a variety of image retrieval tasks, including near-duplicate image

detection, we focus on system designs using SIFT features.

The state-of-the-art approach to near-duplicate image detection is the Bag-of-Word (BoW) model, i.e., to convert SIFT features into visual words via vector quantization, and then apply well-developed text retrieval techniques like inverted index and TF-IDF ranking [61, 67, 81]. This approach, however, is not sufficient to lower the false positive rate to our desired level because: first, as we will show later, a small fraction of SIFT features are intrinsically ambiguous and can cause a large amount of false positives; second, vector quantization loses the discriminating power of raw SIFT features [9] and further increases the false positive rate. We will show by experiments that TF-IDF ranking and geometric verification have their limits on compensating for the information lost during vector quantization.

Another approach is to conduct retrieval directly on raw SIFT features with LSH [43, 17]. The problem is that the large raw feature size (typically a few hundred 128-dimension feature vectors per image) imposes a high storage and retrieval cost and could become a capacity bottleneck. It is known that plain LSH takes too many hash tables [55]. Although the recently developed Multi-Probe LSH [55] reduces storage cost, it requires too many disk seeks when implemented out of core. Expensive geometric verification is also needed for this approach to ensure a low false positive rate.

In this chapter, we achieve both efficiency and quality by making the following three contributions:

First, we eliminate the previously indispensable stage of geometric verification by making the following important observation: false positives are mainly caused by a small fraction of SIFT features that are intrinsically ambiguous due to the lack of internal structure within the SIFT region. We propose an entropy-based filtering method that can effectively filter out these non-discriminative features, and show that with the retained high quality features, *a single match is sufficient for claiming a near-*

duplicate relationship between images with high confidence. This finding dramatically simplifies our retrieval model and system design and also reduces the number of SIFT features to be indexed.

Second, we propose a query expansion method based on graph cut. We construct offline a duplicity graph by connecting all pairs of images predicted to be near-duplicates. Duplicate images are relatively few among all web images (Section 6.4.7), so constructing such a duplicity graph is practically viable. At query time, all vertices reachable from the initial search results are potentially positive results. We use a graph cut method to exclude pathological cases when large clusters of negative images are connected to the initial results. Our query expansion method substantially improves recall, or equivalently, reduces the false positive rate by orders of magnitude at the same recall level.

Third, we implement a search system based on the proposed techniques. For space and time efficiency, we represent SIFT features with sketches (Chapter 4), which is a bit-vector representation more compact than raw feature vectors and more accurate than visual words, and index the sketches with an efficient out-of-core data structure [58]. Our system is capable of indexing more than 50 million web images on a commodity server and returning search results in less than two seconds.

We evaluated our system, as well as some representative existing techniques, with a benchmark consisting of ten thousand manually verified near-duplicate web images and one million random background images. Our system achieves a false positive rate of 2.4×10^{-6} (at recall 0.8), three orders of magnitude smaller than the state of the art [81].

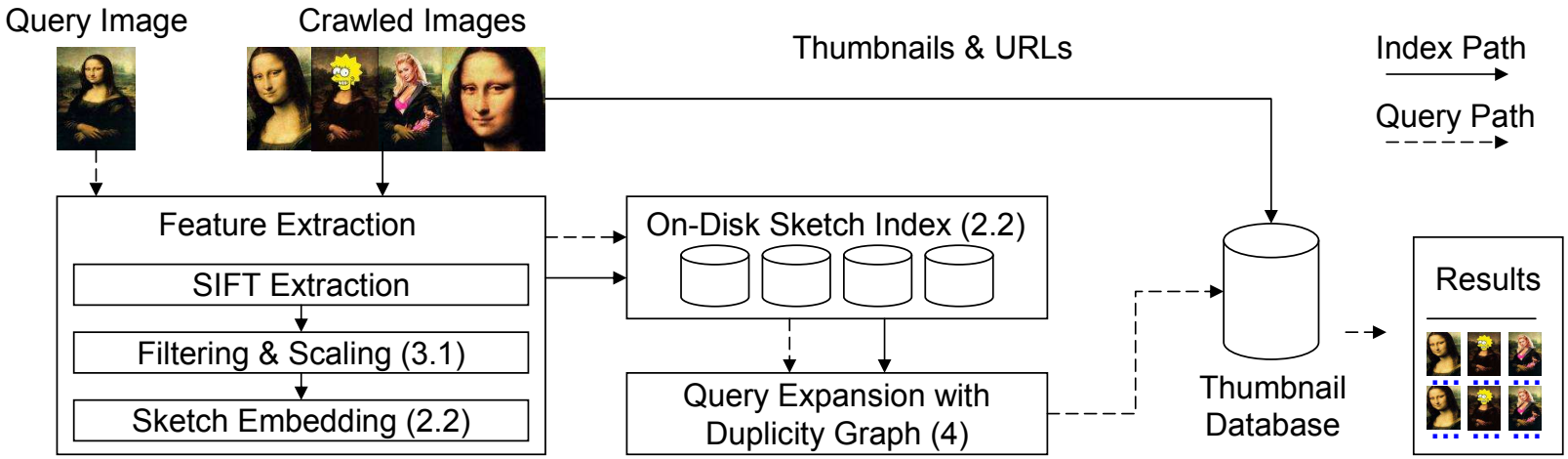


Figure 6.1: System architecture. Section numbers are indicated for the key components.

6.2 System Design

6.2.1 Overview

Our system is based on a very simple retrieval model. Each image is represented as an ID with an attached set of features: $I_i = \langle i, \{f_{ij}\}\rangle$. A flat database DB $= \{\langle f_{ij}, i\rangle\}$ of ID-feature pairs is maintained for the indexed image dataset. For each query image $Q = \langle \cdot, \{f_k\}\rangle$, a query is issued for each feature f_k to retrieve database records whose features are within a fixed distance threshold d:

$$N_d(f_k) = \{\langle f_{ij}, i\rangle \in \text{DB} | \, \|f_k - f_{ij}\|_2 \leq d\}.$$

The retrieved lists are merged: $N_d(Q) = \cup_k N_d(f_k)$, and all IDs in $N_d(Q)$ are returned as near-duplicates. An optional query expansion stage (Section 6.3.1) can be applied to improve recall.

Local features can get matched to different degrees and there can be multiple matched local features, with consistent or inconsistent spatial layout. It is non-trivial to design a principal measure to cover all these aspects. We circumvent the complexity by assuming that we can produce local features of sufficiently high quality, so that a single match of local feature provides a strong indication of image duplicity.

The system architecture is shown in Figure 6.1, with Section numbers indicated for the major components. The sketch embedding and indexing methods are based on existing techniques and are briefly described in the remainder of this section. The novel feature filtering and query expansion techniques are detailed in the following two sections.

6.2.2 Sketch Construction and Indexing

We use sketch embedding for l_2 distance described in Chapter 4. It does not require offline training and can be tuned to be sensitive to a particular distance range, which fits our distance thresholding retrieval model. We find a sketch size of $B = 128$ bit an appropriate choice which both retains good accuracy and provides an $8\times$ compression over raw SIFT features. We claim a match if the hamming distance is ≤ 3 for online search, or ≤ 2 for offline duplicity graph construction.

Our indexing data structure for online search is a simple special case of the technique proposed by Manku *et al* [58]. We divide the 128-bit into 4 blocks, and build a hash table with each block as a key. For a pair of matching sketches, there could be at most 3 different bits, so at least one hash key is identical. By searching all the 4 hash tables, we are guaranteed not to miss any matching sketches. Offline duplicity graph construction is carried out with the same technique. The sketch index and the duplicity graph are constructed with Hadoop on a cluster, but after being created, the data structures are compact enough to be served with a single server.

6.3 Feature Processing

The wide adoption of the visual word approach creates the impression that SIFT is a point feature (hence the phrase key point), and that a strong match between two images probably needs the agreement of multiple matches of key points. But in fact, as we can see from Figure 6.2, a SIFT feature describes a region that usually occupies a significant area in the image and has rich internal content. We believe that in most cases, one matched SIFT region, like B and C in Figure 6.2, is already a strong indication of near-duplicity. Our retrieval model is based on a simple dissertation: SIFT feature match $\Rightarrow$ image match. We will show with experiments that this seemingly loose assumption works sufficiently well with high quality SIFT features produced

Figure 6.2: Illustration of SIFT features. A random sample of SIFT features extracted from the Lena image is plotted. Each SIFT feature consists of 4×4 histograms of edge orientation. In general, a matching of SIFT features means that the spatial configuration of 16 small regions is consistent between two images. It is a strong signal of near-duplicity, except for features like A and D, which contain little internal structure. We use entropy-based filtering to remove such bad features.

with the two techniques described in this section.

We use VLFeat (`http://www.vlfeat.org/`) for SIFT feature extraction. To limit the number of features generated from each image, we resize large images to 300×300. After applying entropy-based filtering described below, roughly 80 features are extracted from each image.

Entropy-Based Filtering

The SIFT features best suitable for near-duplicate detection are those with rich internal structure. SIFT features associated with near-empty regions are the main source of false positives: they tend to occur frequently and get easily matched against one another. We use entropy to measure the richness of internal structure of a SIFT region. Specifically, we treat the SIFT feature $F = [f_1, f_2, \ldots, f_{128}]$ as 128 samples of

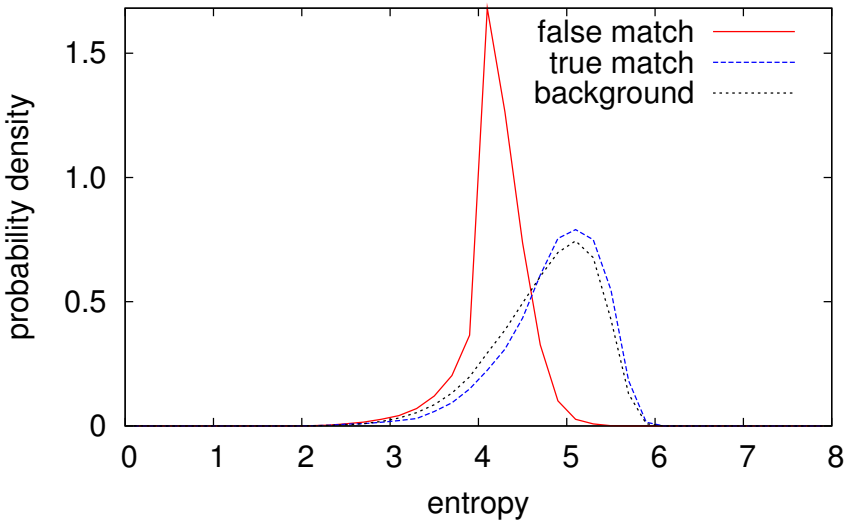


Figure 6.3: Entropy distributions of SIFT features that are sample from (1) false matches, (2) true matches, (3) random background. Compared to the others, the entropy distribution of wrongly matched SIFT feature is narrower, and the peak is at a smaller value.

a discrete random variable in $\{0, 1, \ldots, 255\}$:

$$H(F) = -\sum_{i=0}^{255} p_i(F) \log_2 p_i(F), \qquad p_i(F) = \frac{|\{k \mid f_k = i\}|}{128}.$$

Note that each SIFT dimension is an 8-bit integer, so the entropy has a range of $[0, 8]$.

We measure the entropy distributions of (1) SIFT features generating false matches, (2) SIFT features generating true matches and (3) randomly sampled SIFT features as background. The results are plotted in Figure 6.3. We see that the features causing false positives have a more concentrated distribution, with the peak at a relatively low entropy value of 4.1, while the good and background SIFT features have very similar distributions, both have a peak at 5.1. In our system, we discard all SIFT features with an entropy smaller than 4.4.

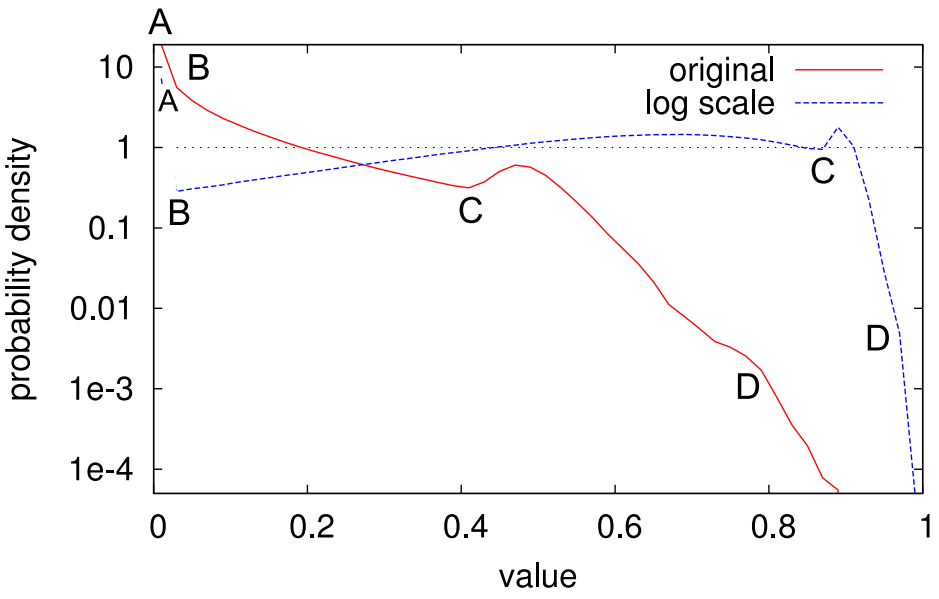


Figure 6.4: Distribution of SIFT feature values before and after log scaling. Log scaling makes the curve much more flat. Some corresponding points on the curves are marked with the letters. We assume that the raw SIFT values have been linearly scaled to fit the range $[0, 1]$.

Log Scaling

The values for individual dimensions of SIFT feature follow a near-exponential distribution, with small values dominating the whole distribution. A SIFT value has a range of $[0, 255]$, but find more than 97% of the values are smaller than 128 and 67% of the values are smaller than 25. This means that the range of the value is not efficiently used. We propose to scale the dimensions of SIFT features logarithmically. To simplify discussion, we first linearly scale the SIFT values into the range $[0, 1]$. We then use the following formula to re-scale each value:

$$v \leftarrow \frac{\log(\max\{v, v_0\}/v_0)}{\log(1/v_0)}, \qquad v_0 = 0.001$$

We take $\max\{v, v_0\}$ to ensure that the result value is always non-negative. The value v_0 is tuned with real data for optimal retrieval performance.

Figure 6.4 shows the SIFT value distribution before and after log scaling. Overall,

we see that the distribution is more uniform after log scaling. Two segments of the log-scaled curve, i.e. A–B and B–C, are seemingly abnormal and need some discussion.

First, the point B corresponds to the unscaled value v_0, and after log-scaling, all points within $[0, v_0]$ are converted to 0. That is why there is a gap between A and B in the log-scaled curve.

Second, the segment B–C is decreasing in the original curve and becomes increasing after log-scaling. We examine a simplified version of this paradoxical phenomenon and show that it is actually possible. We use $B[u, l]$ to denote the bucket containing all points falling within $[u, l]$ in the original scale, and use $B_{\log}[u, l]$ to denote the bucket containing all points falling within $[u, l]$ in the log scale. The simplified version of the paradoxical phenomenon is that in the original scale, $B[0.1, 0.2]$ is larger than $B[0.2, 0.3]$, but in log scale, $B_{\log}[0.1, 0.2]$ is smaller than $B_{\log}[0.2, 0.3]$. Figure 6.5 shows that 0.1, 0.2 and 0.3 in log-scale roughly correspond to 0.0020, 0.0040, 0.0079 in the original scale. So $B_{\log}[0.1, 0.2]$ is equivalent to $B[0.0020, 0.0040]$ and $B_{\log}[0.2, 0.3]$ is equivalent to $B[0.0040, 0.0079]$. $B_{\log}[0.2, 0.3]$ covers almost twice the range of $B_{\log}[0.1, 0.2]$ in the original scale, and it is not strange that $B_{\log}[0.2, 0.3]$ contains more points than $B_{\log}[0.1, 0.2]$.

Beyond the point C, the log scaling becomes near-linear, so it causes less distortion to the shape of the curve except for the aspect ratio. Indeed, the right portion of a log curve is always near-linear, no matter at which scale it is viewed. This can also be seen from Figure 6.5.

6.3.1 Query Expansion with Graph Cut

We set a tight threshold on similarity measure to ensure a low false positive rate, but that also cause us to fail to detect a significant portion of true positives. Query expansion can be used to recover some of the missed positive images. In this section, we propose an effective method of query expansion that particularly fits our retrieval

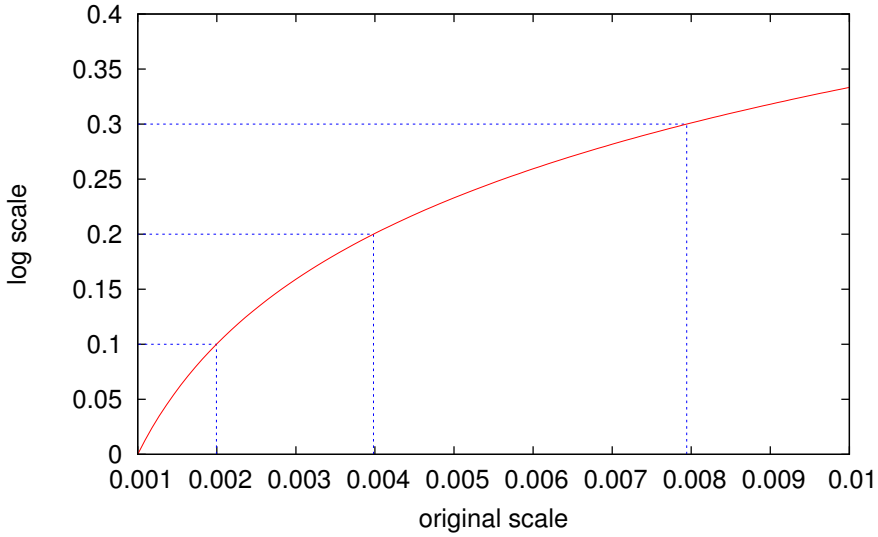


Figure 6.5: Illustration of scale conversion: $x \rightarrow log(x/v_0)/log(1/v_0)$ with $v_0 = 0.001$.

model.

The high level idea of query expansion is to use the search results to query the database again. The procedure can be recursively applied until no new results are found (transitive closure expansion). As the queries used in expansion are all images known to the system, we can accelerate the process by computing the near-duplicates of each indexed image offline.

To ease further discussion, we restate the above in the language of graph theory. We use a *duplicity graph* G to represent the offline computed duplicity relationship. Each indexed image is represented as a vertex in G, and an edge is established between each pair of images with at least one matched local feature. We use $\overline{G}$ to represent G augmented with the query image q and edges between the query and the initial search results. We use $B_k(q)$ to denote the set of vertices that can be reached from q within k steps, so $B_1(q)$ is exactly the initial search results without query expansion. We use $B_\infty(q)$ to denote the connected component of $\overline{G}$ including q, or equally, the results one will get with transitive closure expansion. $B_k(q)$ and $B_\infty(q)$ consists of our baseline query expansion strategies.

To inspire our improved strategy, we categorize potential sources of false positives involved in query expansion into the following three types:

Sporadic False Positives: This is also how false positives get into the initial search results, and is governed by the false positive rate.

Clusters: Dense clusters exist in G. They could either be meaningful clusters of popular images like *Mona Lisa*, or clusters of unrelated noisy images. One common cause of the latter is that images with dense patterns and rich in local feature, e.g., windows of a skyscraper, tend to get matched to a lot of false positives. Such "hub" images can even interconnect to create huge noise clusters. No matter how a cluster is generated, it could dramatically increase false positive rate when hit via a sporadic false positive.

Bridging Images: Even if we achieve 0 false positive rate in building G, false positives could still occur in query expansion via bridging images. For example, an image composed of both *Mona Lisa* and *The Last Supper* should be connected to both clusters by definition. When the query contains only *Mona Lisa*, transitive closure expansion will reach the cluster of *The Last Supper* via the bridging image.

Because our graph has extremely low false positive rate, we are relatively confident when query expansion leads to individual images only. What we want to avoid is hitting an irrelevant cluster through either sporadic false positives or bridging images. On the other hand, if a cluster has many different connections to the query, then we have reasons to believe that they are actually positive results. In all, we want the query expansion result $B(q)$ to be connected, have many internal connections but few connections to the rest of $\overline{G}$, and be large. We find that these requirements can be jointly captured by the *conductance* of a subgraph, defined as

$$\Phi(S) = \frac{|E[S, \overline{G} \setminus S]| \quad \text{//connections to the rest of graph}}{\mu(S) \quad \text{//all connections}},$$

where S is the subgraph of interest, $E[S, \overline{G} \setminus S]$ is the set of edges between S and $\overline{G} \setminus S$, and $\mu(S)$, called the volume of S, is the sum of degrees of vertices in S (the number of edges between S and the rest of the graph plus twice the number of edges within S)[1]. The sparse cut problem is to optimize for minimal conductance:

$$B(q) = \arg \min_{\substack{S \subset \overline{G}, q \in S \\ S \text{ connected}}} \Phi(S).$$

The problem is hard, but efficient approximate solutions exist [71, 3]. In our system, we use an adapted version of PageRank-Nibble [3], which we list in Figure 3. It simplifies the original algorithm by (1) removing many parameters that are necessary when the algorithm is used as a subroutine of a provable graph partitioning algorithm, and (2) choosing the sweep set with the smallest conductance rather than returning one only when it satisfies a number of requirements. The subroutine to compute approximate PageRank is exactly the same as in the original paper. The algorithm requires two parameters $\alpha, \epsilon \in (0, 1)$. We find the final results relatively independent of those parameters, and use $\alpha = 0.5$ and $\epsilon = 0.00001$ in our experiment.

The computational cost of the method, assuming G fits in main memory, is negligible when compared to sketch index lookup, and the improvement in recall is huge: from 45% to nearly 80%.

6.4 Evaluation

We are interested in evaluating how the proposed techniques improve search quality. We first describe the configuration of our experiments, including the dataset, performance measures and evaluated methods. We then report the evaluation results of the individual techniques proposed, and compare our full system to existing retrieval

[1]This definition is equivalent to the standard definition [71] under the assumption that S is a always smaller than $\overline{G} \setminus S$. This assumption is reasonable in our setting as we are never to return more than half of the indexed images.

Algorithm 3: PageRank-Nibble

Data: graph $\overline{G}$ (adjacent list), query q, α, ϵ
begin
  $P \longleftarrow$ ApproximatePageRank(q, α, ϵ)
  // P consists of vertices with non-zero PageRank,
  // sorted in descending order of PageRank
  $e \longleftarrow 0$ // Count $|E[S, \overline{G} \setminus S]|$
  $v \longleftarrow 0$ // Count volume of S
  min $\longleftarrow +\infty$, $B \longleftarrow \emptyset$ // Lowest conductance
  for $i \longleftarrow 1$ **to** $|P|$ **do**
    $v \longleftarrow v + |\overline{G}[P_i]|$
    $e \longleftarrow e + |\overline{G}[P_i] \setminus P_{1..i-1}| - |\overline{G}[P_i] \cap P_{1..i-1}|$
    $c \longleftarrow e/v$
    if $c <$ min **then**
      min $\longleftarrow c$
      $B \longleftarrow P_{1..i}$
  return B

methods. Finally, we provide some space/time performance numbers of our system.

6.4.1 Dataset

We gathered ground truth images by issuing text queries to the popular image search engines and then manually identifying near-duplicate images from the search results:

1. 81 titles of famous paintings, movies and CDs were picked;

2. Four popular image search engines, i.e., Google, Bing, Yahoo and Flickr, were queried with these titles, and original images were downloaded using the URLs returned by the image search engines;

3. Exact duplicates were removed from the crawled images using MD5 checksums. The remaining images were manually inspected, and between 40 and 120 near-duplicate images were identified from each group. The total dataset consists of 10839 images.

We did not use text information in our system so images gathered with text queries should allow us to obtain a good estimation of performance on real web images.

We use one million random Flickr images as background.

6.4.2 Performance Measures

We are mainly interested in evaluating the receiver operating characteristic (ROC). ROC is a standard method to evaluate the performance of a detecting algorithm, or in general, a binary classifier. It which includes the following two measures:

True Positive Rate (Recall) : number of retrieved near-duplicates divided by the total number of near-duplicates in the ground truth dataset;

False Positive Rate : number of falsely retrieved background images divided by the total number of background images. Because these numbers are so small, the figures show the actual numbers times 10^6, the absolute number of background images returned on average.

If we use true positive rate as horizontal axis and the false positive rate as vertical axis, then each parameter configuration determines a point in the two-dimensional space called the ROC space. An ROC curve can be produced by varying the parameters.

When evaluating a retrieval system, a common practice is to use a single measure of quality, e.g., precision@k, or average precision. Such measures are meaningful only when the ratio between positive and negative is fixed, and are not appropriate in our setting because we gather positive and negative examples separately, and the combination does not necessarily approximate the real-life ratio, which is subject to change depending on the application.

When evaluating false positive rate, we assumed that there are no near-duplicates of the ground truth images in the one million background images. This might not be true as we did not label the background data, which would require a huge amount

of human labor. However, we believe that this approximation is not a problem for our particular evaluation setup. First, the expected number of true positives in the background set is extremely small according to our experiments. We achieve a false positive rate of 4.9×10^{-7} at recall 0.43 (without query expansion). That means that the expected number of near-duplicates mixed in the one million background dataset is $4.9 \times 10^{-7} / 0.43 \times 10^{6} = 0.1$, less than one. Second, even if there are near-duplicates in the background images, that only means that the real false positive rate of our method is lower (better) than what we report. Note that we do not use ground-truth near-duplicate images, but only the background Flickr images to measure false positive rate because we want to avoid having to fix an artificial mixing ratio between the two datasets, and also because we believe the background images better represent "random" images on the web.

6.4.3 Simulation

We built a simulator to evaluate various retrieval methods. Our simulator is different from a full search engine in that it does not build an out-of-core index data structures but rather sequentially scans all the data images. All performance numbers reported are exactly the same as if full search engines were built, except that we cannot use the simulator to evaluate search speed.

The following retrieval methods are implemented in the simulator:

Raw The raw SIFT feature retrieval method as described in Section 6.2.1. This method provides the best search quality, but is too time-consuming for online search. It represents the highest search quality achievable with our retrieval model (without query expansion) when an ideal feature index is used.

TF-IDF The bag-of-word method with TF-IDF ranking. We use a dictionary of one million visual words and soft assign each feature to four visual words.

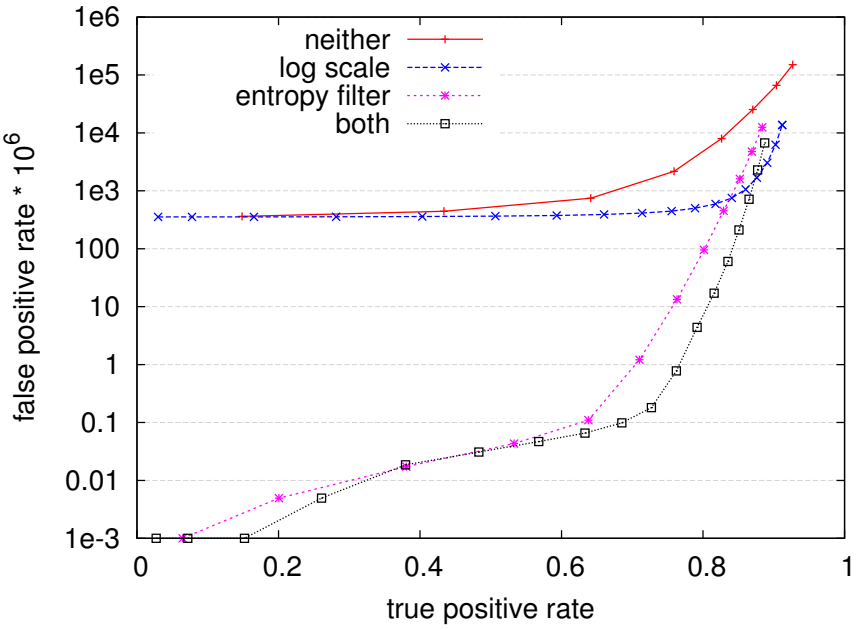


Figure 6.6: The effects of entropy-based filtering and log scaling. Entropy-based filtering eliminates most of the false positives caused by ambiguous SIFT features. Log scaling helps further reducing false positives in most cases.

Bundle The TF-IDF method augmented with bundle information [81].

Sketch The proposed sketch based retrieval method.

Sketch+Exp The sketch based retrieval method with result set expansion.

6.4.4 Entropy-Based Filtering and Log Scaling

Because entropy-based filtering and log scaling are independent of the retrieval models, we first show their improvement by applying them on the raw SIFT feature retrieval method. We then apply them to all the retrieval methods for fair comparison.

Figure 6.6 shows the huge impact these two feature processing steps can make. Entropy-based filtering helps to reduce false positive rate by orders of magnitude, which is especially prominent when true positive rate is low. Without entropy-based

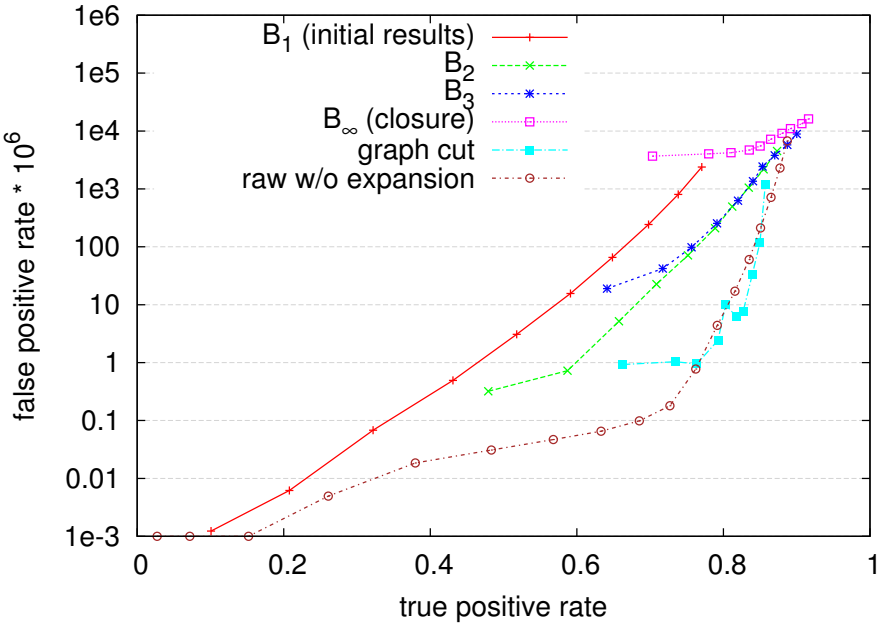


Figure 6.7: Comparison of various query expansion methods.

filtering, there are always 354 background hits no matter how we tighten the retrieval threshold. These potential false positives are caused by SIFT features that are intrinsically ambiguous, and would otherwise require geometric verification to get eliminated. With entropy-based filtering, false positive rate can be tuned very close to zero, which is the normal behavior desired by a retrieval method.

The effect of log scaling is not as gigantic. Unlike entropy-based filtering, it mainly works at the range of high true positive rate. Even when stacked with entropy-based filtering, it still helps to further lower the false positive rate.

6.4.5 Query Expansion

Figure 6.7 compares various strategies of query expansion. We observe that for the $B_k(q), k \geq 2$ family of strategies to achieve the same recall, false positive rate increases as k becomes larger, with $B_2(q)$ being the most practical strategy. This is because larger k means more aggressive query expansion, and is more susceptible to false

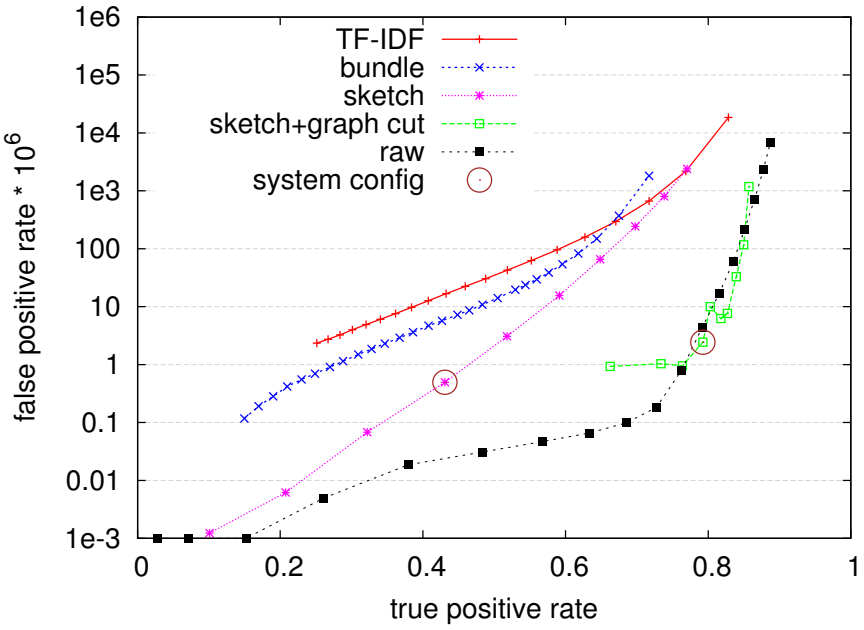


Figure 6.8: Comparison of various retrieval methods.

positives. Our graph-cut-based expansion strategy is way better in avoiding false positives, achieving false positive rate two orders of magnitude lower than $B_2(q)$ at recall around 0.8.

6.4.6 Comparison of Retrieval Methods

Figure 6.8 shows the performance or various retrieval methods. We see that the raw SIFT feature retrieval method has the best performance, except that it is too time-consuming to be practically applicable. Plain TF-IDF ranking with visual words has the worst performance. Furthermore, the items of around 20,000 background images (indicated by the rightmost point on the curve) have to be first retrieved from disk before ranking can be carried out, i.e., the retrieval cost is the same for all points on the curve. By attaching geometric information on the visual words, the bundling method helps to reduce background hit rate by a factor of 3 for most points on the curve. The curve of our sketch method lies in the middle between the BoW methods

and the raw SIFT method, and by applying query expansion, true positive rate can be significantly improved to make the performance very close to the raw SIFT curve.

In the real system, we use a hamming distance threshold of 3, which corresponds to true positive rate = 0.43 and false positive rate = 4.9×10^{-7} without query expansion, and true positive rate = 0.79 and false positive rate = 2.5×10^{-6} with query expansion. For the sketch curve, false positive rate increases exponentially with recall, and the point we choose represents a good trade-off between precision and recall.

6.4.7 Whole System Performance Numbers

We use our search engine to index 50 million web images. Our sketch index runs one sketch query for 0.069 second sequentially, and for our benchmark, each image requires 77.7 sketch queries on average (recall that, as defined in Section 6.2.1, the result is generated by searching with each interesting region of an image as an independent query and then merging the results). To reduce the search time, we scatter the index hash tables to four disks for parallel reading, and batch process the sketch queries so disk accesses can be properly scheduled [43]. The result is that feature extraction and sketch searching can be finished in 1.29 seconds on average. That is faster than the 1.9 second reported by the bundled feature paper [81] to search one million images.

Our sketch index for 50 million images takes 1.3TB disk storage and 2.8GB main memory. The duplicity graph for query expansion takes about 5GB main memory, which means that only a small fraction of the indexed images are offline detected as near-duplicate images. The thumbnails ($128x128$), densely stored in a single file, takes about 400GB disk space. Overall, less than a quarter of the server's memory and disk storage is used (it is common for a commodity server to be equipped with 32GB main memory and $4 \times 2TB$ disk storage), so we expect our single-server capacity to be above 200 million images. We use a cluster of 66 nodes for offline indexing, and processing 50 million images takes about one day.

Our system can be easily scaled up to index more images, e.g. 10 billion images, by taking the following measures:

1. Randomly partition the images into subsets of roughly the same size, e.g. 200 million images each.

2. Use a larger cluster (or longer processing time) to build the index data structure for each subset.

3. Use one machine to server each subset. A query is searched against all subsets and the results are merged.

6.5 Related Works

Near-duplicate image detection, especially detecting duplicates subject to cropping and padding, has only been practically viable after the invention of the groundbreaking SIFT local feature [52]. Ke et al. [43] were the first to study a SIFT based method for near-duplicate image detection. They used LSH to index PCA-SIFT (a variant of SIFT) and RANSAC for verification, and demonstrated superior retrieving performance at the scale of 20K images. They showed that running queries in a batch and linearizing disk accesses can bring a 20× speed-up. We also implemented similar disk scheduling algorithm in our system.

Most recent methods are based on the Bag-of-Word model [61, 67, 81]. Although the BoW approach has been successful on million image level, it has intrinsic limitations preventing it from scaling up to the level of a billion images. On one hand, a large visual dictionary is needed for better discriminating power and to limit the amount of data needed to be read from disk. On the other hand, the larger dictionary usually means lower recall when visually identical local features are over-quantized into different visual words. Soft assignment [68] can be used to recover some of the

lost recall due to over-quantization, but it also enlarges the retrieval cost by several times. One million is a widely accepted size for a visual dictionary [67, 81]. Such a large dictionary can be efficiently created by hierarchical clustering [61]. Assume we extract 100 features from each image on average, and these features are uniformly distributed across all the visual words. The chance for any two images to share at least one visual word is about 0.01. That means nearly 1% of all indexed images will be ranked for each query. When we need to index tens of millions of images on a single server, the retrieval and ranking cost would be too high to make a online search system possible.

Several approaches have been explored to improve the discriminating power of visual words. The visual phrase [87] method links near-by visual words to form phrases and conduct retrieval at phrase level. Visual phrases do have much higher discriminating power than visual words. However the number of visual phrases extracted from each image can easily grow out of control. Bundled Feature [81] is another method to improve the discriminating power of visual words. It attaches information on its relative position within its surrounding MSER region to each visual word. Thus explicit geometric verification can be avoided. However, the disk I/O problem is not solved because retrieval is still carried out at individual visual word level.

Foo and Sinha [26] first showed that SIFT features can be dramatically pruned for near-duplicate image detection. Two methods were studied: to keep the top N most significant key points, and to raise the inclusion threshold value to discard key points. Turcot and Lowe [73] proposed a SIFT feature pruning method based on the observation that a SIFT feature is only useful when it contributes to a matching within the database, assuming that each image in the database has at least one near-duplicate. This strategy is effective in reducing the number of features by more than an order of magnitude without compromising search quality, and its application is not limited to near-duplicate detection. The drawback is that, effectively only

images with duplicates in the database are indexed, which is not usually tolerable. Both SIFT feature pruning method aim at reducing the space overhead, rather than filtering out low-quality features. Our entropy-based filtering method is orthogonal to those methods and can be used as a complement.

Chum et al. [16] studied various query expansion methods, including transitive closure expansion, for local-feature based object retrieval with vector quantization and geometric verification. The other methods proposed, like average query expansion and multiple image resolution expansion, are potentially applicable to near-duplicate image detection, but involves reconstructing queries unknown to the system, and running those new queries could substantially increase search time.

Xu et al. [83] proposed to divide images into overlapping and non-overlapping blocks over multiple levels, and to compare and align the blocks with SIFT features and Earth Mover's Distance(EMD). Even though the method is reported to achieve high detection quality, it is only applicable to small datasets as it involves a very expensive similarity measure and requires scanning the full dataset in brute force.

6.6 Summary

We proposed two techniques for large-scale near-duplicate image detection:

- We found that entropy-based filtering can eliminate $> 99\%$ of false positives, and allows one to claim near-duplication relationship with a single match of the retained high quality features.
- We developed a query expansion method with graph cut to substantially improve search quality.

The combination of the two methods lowers false positive rate by three orders of magnitude (at 80% recall). With the proposed techniques, we built a search engine that

is capable of serving more than 50 million web images, and achieves a false positive rate three orders of magnitude smaller than the standard visual word approach.

Chapter 7

Conclusion

In this dissertation, we approached the high-dimensional similarity search problem by developing methods to reduce candidate set size — the limiting factor of search speed — along two directions: to reduce the number of items in the candidate set and to reduce the representation size of each individual object. In this chapter we summarize the contribution we have made and discuss potential directions for future research.

7.1 Summary of Contribution

In this dissertation, we studied several key issues to improve the accuracy and efficiency of high-dimensional similarity search. We made the following contributions:

- We presented a statistical performance model of Multi-Probe LSH, a state-of-the-art technique for high-dimensional similarity search. Our model can accurately predict the average search quality and latency by gathering statistical data from a small sample of the dataset. We used this model to realize automatic parameter tuning for optimal performance. We also used the model to devise an adaptive LSH search algorithm to determine the probing parameter dynamically for each query. The adaptive probing method addresses the prob-

lem that even though the average performance is tuned for optimal results, 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.

- We presented *NN-Descent*, a simple yet efficient algorithm for approximate K-NN graph construction, i.e., to search for K-NN for every point in the dataset. Our method supports arbitrary similarity measures, has minimal space overhead and does not rely on any shared global index. Hence, it is especially suitable for large-scale applications where data structures need to be distributed over the network. We have shown with a variety of datasets and similarity measures that the proposed method typically converges to above 90% recall with each point comparing only to several percent of the whole dataset on average. Our method is from 2 to 16 times faster than the state-of-the-art technique. We used the offline computed nearest-neighbor information to double the speed of online similarity search.

- We presented an efficient sketch algorithm for similarity search with l_2 distances and a novel asymmetric distance estimation technique. Our new asymmetric estimator takes advantage of the original feature vector of the query to boost the distance estimation accuracy. We also applied this asymmetric method to existing sketches for cosine similarity and l_1 distance. Evaluations with datasets extracted from images and telephone records show that our l_2 sketch outperforms existing dimension reduction methods such as PCA and random projection, and the asymmetric estimators consistently improve the accuracy of different sketch

methods. To achieve the same search quality, asymmetric estimators can reduce the sketch size by 10% to 40%.

- We presented a randomized algorithm to embed a set of features into a single high-dimensional vector to simplify the feature set-matching problem. The main idea is to project feature vectors into an auxiliary space using locality-sensitive hashing and to represent a set of features as a histogram in the auxiliary space, which is simply a high-dimensional vector. We have shown with experiments that the proposed approach is indeed effective and flexible. It can achieve accuracy comparable to the feature set-matching methods, while requiring significantly less space and time. For object recognition with the Caltech 101 dataset, our method runs 25 times faster to achieve the same precision as Pyramid Matching Kernel, one state-of-the-art feature set-matching method.

- Finally, we demonstrated some of the proposed techniques by building a large-scale near-duplicate image search system that is capable of serving more than 50 million web images on a single commodity server. We also developed domain-specific techniques that make such a system possible. First, we showed that entropy-based filtering eliminates ambiguous SIFT features that cause most of the false positives, and enables claiming near-duplicity with a single match of the retained high-quality features. Second, we showed that graph cut could be used for query expansion with a duplicity graph computed offline to substantially improve search quality. We have shown with experiments that when combined with sketch embedding, our methods achieve false positive rate orders of magnitude lower than the standard visual word approach.

7.2 Future Work

Even though we have made substantial improvements in this dissertation, the high-dimensional similarity search problem is far from being solved. In fact, it is unrealistic to hope to defeat the curse of dimensionality for datasets of an arbitrarily large number of dimensions. After all, dimensionality is the characterization of the intrinsic hardness of the dataset, which cannot be changed by an algorithm. However, the improvements we have made in various aspects of the problem, e.g., an order of magnitude speedup in offline K-NN graph construction, indicate that the complexity lower bound determined by the intrinsic hardness of the dataset is far from being reached. Below, we discuss some of the most promising directions for future research based on the findings of this dissertation.

7.2.1 Online Similarity Search with K-NN Graph

Our preliminary study on online search with K-NN graph showed promising results (Section 3.7) and opens an interesting direction for future exploration. The graph expansion, or essentially local search, approach is fundamentally different from most of the existing indexing techniques based on clustering or partitioning and has attracted little attention in previous research. This is natural as the very goal of indexing is to be able to make a big jump from the current search state to the search target. For example, if we can somehow measure the distance l between the current search state and the target, then a reduction in the distance by at least δl, for some fixed $\delta > 0$, should always be achieved in one search step so as for the index to be "efficient" in the traditional sense, i.e., to guarantee $O(\log N)$ search time. This is not possible with local search, in which the search range of each step is limited to a small amount that is prefixed and not related to l. However, for datasets of high dimensions, the curse of dimensionality calls for a different understanding of "efficiency": the $\log N$

part of the time complexity is dominated by the coefficient that is exponential to the number of dimension D, and is no longer as important as in low-dimensional cases. In fact, a previous study [7] has shown that as dimensionality grows, the smallest distance δ between any pair of points approaches the largest distance Δ between any pair of points. Note that Δ is also an upper bound of the distance between an arbitrary starting point and the search target. So long as improvement in the level of δ can be made in each step, convergence shall be reached in a few iterations. What is important to be optimized now is the size of the neighborhood that one needs to explore in each step. An $O(\log N)$ index, aiming at covering a larger search range in each step, usually also has to maintain a larger neighborhood to be examined, and has an intrinsic disadvantage over local search.

7.2.2 Combining Indexing and Filtering

Even though we have made improvements in both indexing and sketch filtering, how to integrate these two seemingly orthogonal techniques in a real system remains a practical issue. The problem is that both indexing and filtering are approximate. For example, assuming that each can achieve a recall of 90% independently, then the combined recall would be as low as 81%. That is why currently in the image search system we only use the filtering technique and rely on an exact but more expensive technique for indexing.

In order to answer the questions as to whether it is meaningful to integrate indexing and filtering and, if so, how to tune both components to minimize search cost, we need to further elaborate the unified framework shown in Figure 1.2 and build a performance model that covers both indexing and filtering. We have presented an accuracy performance model for the Multi-Probe LSH index, and Wang et al. [77] presented a similar model for sketch filtering. Future research could build on these existing works.

7.2.3 Similarity Search with Cloud Storage

In this dissertation, we have been evaluating our techniques within the main memory of a single machine. Meanwhile, many real-world datasets have out grown the capacity of a single machine, or even that of a medium-sized cluster of tens of machines. In order to solve real-world problems, cluster-based large-scale similarity search systems have to be built.

Recent advancements in cloud computing have shown that it is a scalable and cost-effective way to provide massive and high-throughput storage by federating the local file systems of a cluster of machines. Systems have been built that provide file system interface [30], key-value store interface [21, 48] and table interface [13]. It is, therefore, interesting to develop a similarity search system following the same model. We need to solve the following problems in order to build such a system.

First, we need to determine how to design a data layout algorithm that meets the requirements of replication, efficient search and load balancing. Data in a cloud storage have to be replicated for reliability. Copies of the same data item have to reside on different nodes, or even at different geologic sites. Meanwhile, data in a similarity search system, e.g., one based on Locality-Sensitive Hashing, also have to be replicated for better locality. Further more, it is desired that data be evenly distributed among the cluster nodes so as to maximize resource utilization.

Second, we need to determine how to handle incremental updates. We have already shown in Chapter 2 that the optimal parameter of Multi-Probe LSH depends on the size of the dataset. As data accumulate, the optimal parameter for the index will shift. To keep the system at a healthy state with growing data, we have to either design an index that is adaptive to data size, or to design a mechanism to periodically reconstruct the index in background without significantly degrading the quality of online service.

Third, we need to determine how to schedule queries to meet latency and/or

throughput requirements. For each query, a sufficiently large candidate set has to be scanned in order to reach the accuracy requirement, and it is probable that the candidates will reside on different nodes. When data are replicated, there could be multiple combinations of target nodes for a query to cover its candidate set. When multiple queries are present simultaneously, we have to devise mechanisms to optimize query plans so as to minimize the conflicts on node access, and for each node, we have to devise a scheduling strategy to maximize throughput while trying to satisfy any latency constraints.

7.3 Closing Remarks

High-dimensional similarity search is a significant and difficult problem. Despite more than two decades of intensive research, there is still no satisfactory general solution. Indeed, according to studies that revealed the difficult nature of the problem [78], an algorithm that is efficient (polynomial) with regard to the intrinsic dimensionality is unlikely to exist. However, this is not to say that the problem is insolvable in all practical cases. In this dissertation, we showed that for many real-world datasets, orders of magnitude speedup over brute-force scan can be achieved by exploiting the following two opportunities that are usually present in practice:

- Approximation in search results is usually acceptable,
- The intrinsic dimensionality is usually much lower than the appearing dimensionality.

Our results also indicate that there is still room for future improvement. Considering that a breakthrough in asymptotic complexity is less likely to happen, we envision that the most fruitful future direction is to engineer practical systems for searching large datasets.

Bibliography

[1] Belarmino Adenso-Diaz and Manuel Laguna. Fine-tuning of algorithms using fractional experimental designs and local search. *Oper. Res.*, 54:99–114, January 2006.

[2] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *STOC '96: Proceedings of the 28th Annual ACM Symposium On Theory of Computing*, pages 20–29, 1996.

[3] Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using pagerank vectors. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:475–486, 2006.

[4] Mayank Bawa, Tyson Condie, and Prasanna 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.

[5] Roberto J. Bayardo, Yiming Ma, and Ramakrishnan Srikant. Scaling up all pairs similarity search. In *WWW '07: Proceedings of the 16th International Conference on World Wide Web*, pages 131–140, New York, NY, USA, 2007. ACM.

[6] Jon Louis Bentley. K-d trees for semidynamic point sets. In *SCG '90: Proceedings of the 6th Annual Symposium on Computational Geometry*, pages 187–197, New York, NY, USA, 1990. ACM.

[7] Kevin S. Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? In *ICDT '99: Proceedings of the 7th International Conference on Database Theory*, pages 217–235, London, UK, 1999. Springer-Verlag.

[8] Christian Böhm, Stefan Berchtold, and Daniel A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Comput. Surv.*, 33:322–373, September 2001.

[9] Oren Boiman, Eli Shechtman, and Michal Irani. In defense of nearest-neighbor based image classification. In *CVPR '08: Proceedings of the 2008 IEEE Computer Society Conference on Computer Vision And Pattern Recognition*, Los Alamitos, CA, USA, June 2008. IEEE Computer Society.

[10] M. R. Brito, E. L. Chvez, A. J. Quiroz, and J. E. Yukich. Connectivity of the mutual k-nearest-neighbor graph in clustering and outlier detection. *Statistics & Probability Letters*, 35(1):33–42, August 1997.

[11] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-wise independent permutations (extended abstract). In *STOC '98: Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 327–336, New York, NY, USA, 1998. ACM.

[12] M. Casey, C. Rhodes, and M. Slaney. Analysis of minimum distances in high-dimensional musical spaces. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(5):1015 –1028, July 2008.

[13] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26:4:1–4:26, June 2008.

[14] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC '02: Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*, pages 380–388, New York, NY, USA, 2002. ACM.

[15] Jie Chen, Haw ren Fang, and Yousef Saad. Fast approximate knn graph construction for high dimensional data via recursive lanczos bisection. *Journal of Machine Learning Research*, 10:1989–2012, 2009.

[16] O. Chum, J. Philbin, J. Sivic, M. Isard, and A. Zisserman. Total recall: Automatic query expansion with a generative feature model for object retrieval. In *ICCV '07: IEEE 11th International Conference on Computer Vision*, October 2007.

[17] Ondřej Chum, James Philbin, Michael Isard, and Andrew Zisserman. Scalable near identical image and shot detection. In *CIVR '07: Proceedings of the 6th ACM International Conference on Image And Video Retrieval*, pages 549–556. ACM, 2007.

[18] Kenneth L. Clarkson. Nearest-neighbor searching and metric space dimensions. In Gregory Shakhnarovich, Trevor Darrell, and Piotr Indyk, editors, *Nearest-Neighbor Methods For Learning And Vision: Theory And Practice*, pages 15–59. MIT Press, 2006.

[19] Michael Connor and Piyush Kumar. Fast construction of k-nearest neighbor graphs for point clouds. *IEEE Transactions on Visualization and Computer Graphics*, 16:599–608, 2010.

[20] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab 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.

[21] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41:205–220, October 2007.

[22] Yining Deng and B. S. 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.

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

[24] Li Fei-Fei, Rob Fergus, and Pietro Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer Vision and Image Understanding*, pages 59–70, April 2004.

[25] Imola Fodor. A survey of dimension reduction techniques.

[26] Jun Jie Foo and Ranjan Sinha. Pruning SIFT for scalable near-duplicate image matching. In *ADC '07: Proceedings of the 18th Conference on Australasian Database - Volume 63*, pages 63–71. Australian Computer Society, Inc., 2007.

[27] John F. Gantz, Christopher Chute, Alex Manfrediz, Stephen Minton, David Reinsel, Wolfgang Schlichting, and Anna Toncheva. The diverse and exploding digital universe: an updated forecast of worldwide information growth through 2011. An IDC White Paper — sponsored by EMC, March 2008.

[28] John F. Gantz and David Reinsel. Extracting value from chaos. `http://www.emc.com/digital_universe`, June 2011.

[29] 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.

[30] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google file system. In *SOSP '03: Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 29–43, New York, NY, USA, 2003. ACM.

[31] Aristides Gionis, Piotr Indyk, and Rajeev 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.

[32] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, 1995.

[33] K. Grauman and T. Darrell. Pyramid match hashing: Sub-linear time indexing over partial correspondences. *CVPR '07: IEEE Conference on Computer Vision and Pattern Recognition, 2007.*, pages 1–8, 17-22 June 2007.

[34] Kristen Grauman and Trevor Darrell. The pyramid match kernel: Efficient learning with sets of features. *J. Mach. Learn. Res.*, 8:725–760, 2007.

[35] Antonin 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.

[36] Piotr Indyk and Rajeev 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.

[37] Piotr Indyk and Nitin Thaper. Fast image retrieval via embeddings. In *3rd International Workshop on Statistical And Computational Theories of Vision*, 2003.

[38] H. Jegou, L. Amsaleg, C. Schmid, and P. Gros. Query adaptative locality sensitive hashing. In *Acoustics, Speech And Signal Processing, 2008. ICASSP 2008. IEEE International Conference On*, pages 825 –828, 31 2008-april 4 2008.

[39] Alexis Joly and Olivier Buisson. A posteriori multi-probe locality sensitive hashing. In *MM '08: Proceeding of the 16th ACM International Conference on Multimedia*, 2008.

[40] Norio Katayama and Shin'ichi 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.

[41] Michael Kazhdan, Thomas Funkhouser, and Szymon 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.

[42] Yan Ke and Rahul Sukthankar. PCA-SIFT: A more distinctive representation for local image descriptors. In *CVPR '04: Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision And Pattern Recognition*, volume 02, pages 506–513, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[43] Yan Ke, Rahul Sukthankar, and Larry Huston. An efficient parts-based near-duplicate and sub-image retrieval system. In *MM '04: Proceedings of the 12th*

Annual ACM International Conference on Multimedia, pages 869–876. ACM, 2004.

[44] Adam Kirsch and Michael Mitzenmacher. Distance-sensitive bloom filters. In *ALENEX '06: Proceedings of the Ninth Workshop on Algorithm Engineering And Experiments*, pages 41–50, Philadelphia, PA, USA, 2006. Society for Industrial and Applied Mathematics.

[45] Jon Kleinberg. The small-world phenomenon: an algorithm perspective. In *STOC '00: Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pages 163–170, New York, NY, USA, 2000. ACM.

[46] Donald E. Knuth. *The art of computer programming, volume 3: sorting and searching*. Addison Wesley, MA, USA, 1973.

[47] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *ICCV '09: IEEE 12th International Conference on Computer Vision*, pages 2130 –2137, October 2009.

[48] Avinash Lakshman and Prashant Malik. Cassandra: structured storage system on a p2p network. In *PODC '09: Proceedings of the 28th ACM Symposium on Principles of Distributed Computing*, pages 5–5, New York, NY, USA, 2009. ACM.

[49] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision And Pattern Recognition*, pages 2169–2178, Washington, DC, USA, 2006. IEEE Computer Society.

[50] Bastian Leibe and Bernt Schiele. Analyzing appearance and contour based methods for object categorization. *CVPR '03: Proceedings of the 2003 IEEE Com-*

puter Society Conference on Computer Vision And Pattern Recognition, 02:409, 2003.

[51] Stuart P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28:129–137, 1982.

[52] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, 2004.

[53] Qin Lv, Moses Charikar, and Kai Li. Image similarity search with compact data structures. In *CIKM '04: Proceedings of the 13th ACM International Conference on Information And Knowledge Management*, pages 208–217, New York, NY, USA, 2004. ACM.

[54] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai 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.

[55] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. Multi-probe LSH: efficient indexing for high-dimensional similarity search. In *VLDB '07: Proceedings of the 33rd International Conference on Very Large Data Bases*, pages 950–961. VLDB Endowment, 2007.

[56] S. Lyu. A kernel between unordered sets of data: The gaussian mixture approach. In *ECML '05: European Conference on Machine Learning*, Porto, Portugal, 2005.

[57] Siwei Lyu. Mercer kernels for object recognition with local features. In *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision And Pattern Recognition*, volume 2, pages 223–229, Washington, DC, USA, 2005. IEEE Computer Society.

[58] Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. Detecting near-duplicates for web crawling. In *WWW '07: Proceedings of the 16th International Conference On World Wide Web*, pages 141–150. ACM, 2007.

[59] David M. Mount and Sunil Arya. ANN: A library for approximate nearest neighbor searching. `http://www.cs.umd.edu/~mount/ANN/`.

[60] Marius Muja and David G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory And Application VISSAPP'09)*, pages 331–340. INSTICC Press, 2009.

[61] David Nister and Henrik Stewenius. Scalable recognition with a vocabulary tree. In *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision And Pattern Recognition*, pages 2161–2168. IEEE Computer Society, 2006.

[62] A. Opelt, M. Fussenegger, A. Pinz, and P. Auer. *Weak Hypotheses and Boosting for Generic Object Detection and Recognition*, volume 3022/2004 of *Lecture Notes in Computer Science*, pages 71–84. Springer Berlin / Heidelberg, 2004.

[63] Bernd-Uwe Pagel, Flip Korn, and Christos 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.

[64] Rina 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.

[65] Rodrigo Paredes, Edgar Chávez, Karina Figueroa, and Gonzalo Navarro. Practical construction of -nearest neighbor graphs in metric spaces. In *WEA '06: Workshop on Experimental and Efficient Algorithms*, pages 85–97, 2006.

[66] Loïc Paulevé, Hervé Jégou, and Laurent Amsaleg. Locality sensitive hashing: A comparison of hash function types and querying mechanisms. *Pattern Recognition Letters*, 31(11):1348 – 1358, 2010.

[67] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *CVPR '07: Proceedings of the 2007 IEEE Computer Society Conference on Computer Vision And Pattern Recognition*, June 2007.

[68] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Lost in quantization: Improving particular object retrieval in large scale image databases. In *CVPR '08: Proceedings of the 2008 IEEE Computer Society Conference on Computer Vision And Pattern Recognition*, June 2008.

[69] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. The earth mover's distance as a metric for image retrieval. *Int. J. Comput. Vision*, 40(2):99–121, 2000.

[70] Gregory Shakhnarovich, Trevor Darrell, and Piotr Indyk. *Nearest-neighbor methods in learning and vision : theory and practice*. MIT Press, Cambridge, MA, 2005.

[71] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *STOC '04: Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, pages 81–90. ACM, 2004.

[72] Ertem Tuncel, Hakan Ferhatosmanoglu, and Kenneth Rose. VQ-index: an index structure for similarity searching in multimedia databases. In *MM '02: Proceed-*

ings of the 10th ACM International Conference on Multimedia, pages 543–552, New York, NY, USA, 2002. ACM.

[73] P. Turcot and D.G. Lowe. Better matching with fewer features: The selection of useful features in large database recognition problems. In *IEEE 12th International Conference on Computer Vision Workshops*, pages 2109 –2116, 2009.

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

[75] Rares Vernica, Michael J. Carey, and Chen Li. Efficient parallel set-similarity joins using mapreduce. In *SIGMOD '10: Proceedings of the 2010 International Conference on Management of Data*, pages 495–506, New York, NY, USA, 2010. ACM.

[76] James Z. Wang, Jia Li, and Gio Wiederhold. SIMPLIcity: Semantics-sensitive integrated matching for picture libraries. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(9):947–963, 2001.

[77] Zhe Wang, Wei Dong, William Josephson, Qin Lv, Moses Charikar, and Kai 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, 2007.

[78] Roger Weber, Hans-Jörg Schek, and Stephen 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.

[79] Yair Weiss, Antonio Torralba, and Rob Fergus. Spectral hashing. In *NIPS '08: Advances In Neural Information Processing Systems*, pages 1753–1760. 2008.

[80] Xiao Wu, Alexander G. Hauptmann, and Chong-Wah Ngo. Practical elimination of near-duplicates from web video search. In *MM '07: Proceedings of the 15th International Conference on Multimedia*, pages 218–227, New York, NY, USA, 2007. ACM.

[81] Zhong Wu, Qifa Ke, M. Isard, and Jian Sun. Bundling features for large scale partial-duplicate web image search. In *CVPR '09: Proceedings of the 2009 IEEE Computer Society Conference on Computer Vision And Pattern Recognition*, August 2009.

[82] Chuan Xiao, Wei Wang, Xuemin Lin, and Jeffrey Xu Yu. Efficient similarity joins for near duplicate detection. In *WWW '08: Proceeding of the 17th International Conference on World Wide Web*, pages 131–140, New York, NY, USA, 2008. ACM.

[83] Dong Xu, Tat-Jen Cham, Shuicheng Yan, and Shih-Fu Chang. Near duplicate image identification with patially aligned pyramid matching. In *CVPR '08: Proceedings of the 2008 IEEE Computer Society Conference on Computer Vision And Pattern Recognition*, pages 1 –7, June 2008.

[84] Shuicheng Yan, Dong Xu, Benyu Zhang, Hong-Jiang Zhang, Qiang Yang, and Stephen Lin. Graph embedding and extensions: A general framework for dimensionality reduction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29:40–51, 2007.

[85] Dell Zhang, Jun Wang, Deng Cai, and Jinsong Lu. Self-taught hashing for fast similarity search. In *SIGIR '10: Proceeding of the 33rd International ACM SIGIR Conference on Research And Development In Information Retrieval*, pages 18–25, New York, NY, USA, 2010. ACM.

[86] J. Zhang, M. Marszałek, S. Lazebnik, and C. Schmid. Local features and kernels for classification of texture and object categories: A comprehensive study. *Int. J. Comput. Vision*, 73(2):213–238, 2007.

[87] Shiliang Zhang, Qi Tian, Gang Hua, Qingming Huang, and Shipeng Li. Descriptive visual words and visual phrases for image applications. In *MM '09: Proceedings of the 17th ACM International Conference on Multimedia*, pages 75–84. ACM, 2009.