Fast Locality-Sensitive Hashing

Anirban Dasgupta Ravi Kumar Tamás Sarlós
Yahoo! Research
701 First Avenue
Sunnyvale, CA 94089
{anirban,ravikumar,stamas}@yahoo-inc.com

ABSTRACT

Locality-sensitive hashing (LSH) is a basic primitive in several large-scale data processing applications, including nearest-neighbor search, de-duplication, clustering, etc. In this paper we propose a new and simple method to speed up the widely-used Euclidean realization of LSH. At the heart of our method is a fast way to estimate the Euclidean distance between two d-dimensional vectors; this is achieved by the use of randomized Hadamard transforms in a non-linear setting. This decreases the running time of a (k, L)-parameterized LSH from $O(dkL)$ to $O(d \log d + kL)$. Our experiments show that using the new LSH in nearest-neighbor applications can improve their running times by significant amounts. To the best of our knowledge, this is the first running time improvement to LSH that is both provable *and* practical.

Categories and Subject Descriptors. H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; H.4.m [**Information Systems Applications**]: Miscellaneous; G.3 [**Mathematics of Computing**]: Probability and Statistics—*Probabilistic algorithms*

General Terms. Algorithms, Experimentation, Performance, Theory

Keywords. Nearest neighbor search, Locality-sensitive hashing, Hadamard transform, Dimension reduction

1. INTRODUCTION

Locality sensitive hashing (LSH) is a basic primitive in large-scale data processing algorithms that are designed to operate on objects (with features) in high dimensions. The idea behind LSH [23, 22] is the following: construct a family of functions that hash objects into buckets such that objects that are similar will be hashed to the same bucket with high probability. Here, the type of the objects and the notion of similarity between them determine the particular hash function family. Typical instances include the Jaccard coefficient as similarity when the underlying objects are sets and the ℓ_2 norm as distance (i.e., dissimilarity) or the cosine/angle as similarity when the underlying objects are vectors.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
KDD'11, August 21–24, 2011, San Diego, California, USA.
Copyright 2011 ACM 978-1-4503-0813-7/11/08 ...$10.00.

With such a powerful primitive, many large-scale data processing problems that previously suffered from the "curse of dimensionality" suddenly become tractable. For instance, in conjunction with standard indexing techniques, it becomes possible to search for nearest-neighbors efficiently [16]: given a query, hash the query into a bucket, use the objects in the bucket as candidates for near neighbors, and rank them according to their similarity to the query. Likewise, popular operations such as near-duplicate detection [7, 28, 19], all-pairs similarity [6], similarity join/record-linkage [25], temporal correlation [10], are made easier.

In this paper objects are vectors and the distance between vectors is the Euclidean (ℓ_2) norm. This choice is motivated by many applications in text processing, image/video indexing/retrieval, natural language processing, etc. When the similarity measure between vectors is their angle, Charikar [9] gave a very simple LSH called SIMHASH: the hash of an input vector is the sign of its inner product with a random unit vector. It can be shown that the probability that the hashes of two vectors agree is a function of the angle between the underlying vectors [17]. Datar et al. [12] present constructions based on stable distributions for the ℓ_p norm; their construct is almost identical to Charikar's in the ℓ_2 case. Since then, there have been efforts to make these LSH constructions more efficient and practical [27, 13, 5]. Each of these LSH constructions works by first using a randomized estimator of the similarity measure, pasting multiple such constructions to create one hash function with an appropriately small collision rate, and then using multiple such hash functions to get the right tradeoff between the collision rate and recall.

Main results. In this paper we obtain algorithmic improvements to the query time of the basic LSH in the ℓ_2 setting. At a high level, we improve the query time of the LSH for estimating ℓ_2 for d-dimensional vectors from $O(dkL)$ to $O(d \log d + kL)$, where roughly, each hash function maps into a vector of k bits and L different hash functions are used. Experiments on different, large, and high-dimensional datasets show that these theoretical gains translate to a typical improvement of 20% or more in the LSH query time. We also extend our results to the angle-based similarity, where we improve the query-time to ϵ-approximate the angle between two d-dimensional vectors from $O(d/\epsilon^2)$ to $O(d \log 1/\epsilon + 1/\epsilon^2)$; we postpone the details of this result to the full version of the paper. To the best of our knowledge, this is the first query-time improvement to LSH that is both provable and practical.

Our improvement consists of two new algorithms. The first algorithm, called `ACHash`, works by computing the following sequence applied to the input vector: randomly flip the signs of each coordinate of the vector, apply the Hadamard transform, and compute the inner product with a sparse Gaussian vector. This particular sequence of operations is directly inspired by the fast Johnson–

Lindenstrauss transform proposed by Ailon and Chazelle [1] for dimension reduction. The second algorithm, called `DHHash`, works by computing the following sequence applied to the input vector: randomly flip the signs of each coordinate of the vector, apply the Hadamard transform, multiply each coordinate by independent unit Gaussians, and apply another Hadamard transform. `DHHash`, even though it has only comparable theoretical guarantees to `ACHash`, performs much better in practice. The use of a Gaussian operator sandwiched by Hadamard transforms is a novel step in `DHHash`.

Clearly, both the algorithms exploit the computational advantages of the Hadamard transform and that is where we gain the improved query time. The space requirements of naive LSH lie between `ACHash` and `DHHash` whereas, the latter has a better query time than the former. Notice that our algorithms are extremely simple in practice; they have been implemented on top of a publicly available LSH package. A bulk of our contribution lies in proving that these algorithms do not compromise the basic LSH guarantees. The main technical difficulty we encounter is in the non-linear but inevitable operation of bucketizing the projection (or taking the sign). The proof of `ACHash` is relatively simple and relies on an application of the Bernstein's concentration bound. The proof of `DHHash` is intricate, since in addition we need to deal with dependent Gaussian random variables; it uses a novel analysis method based on matrix perturbation. The techniques we develop for the analysis may be of independent interest.

2. RELATED WORK

The related work falls into three main categories: the vast literature on data structures and indexing for similarity search, the extensive work on LSH and related methods, and the use of FFT-like methods in dimension reduction.

There have been several indexing data structures proposed for nearest-neighbor search and approximate nearest-neighbor search, such as the R-tree, the K-D tree, the SR-tree, etc. Unfortunately, these index structures do not scale well with the dimension of the data [32] and this is precisely where LSH-like techniques kick in. We do not delve into the myriad of searching and indexing techniques for similarity joins: the readers can refer to the survey [15] and the tutorial [25].

LSH was first articulated in a series of papers by Indyk et al. [23] and Indyk and Motwani [22]. Since then, it has become the state-of-the-art technique for similarity search in high dimensions. Charikar [9] developed an LSH for angles (called SIMHASH) and thus cosine similarities in Euclidean space. Datar et al. [12] presented LSH schemes based on stable distribution for ℓ_p norms.

Panigrahy [30] proposed the entropy-based LSH to (provably) reduce the space requirements of SIMHASH. In this scheme, in addition to considering the bucket corresponding to the query, buckets corresponding to perturbed versions of the query also considered. Unfortunately, while the space requirements are reduced, the query time is considerably increased. Lv et al. [27] designed a careful probing heuristic to look up multiple buckets that have a high probability of containing the nearest neighbors of a query. They obtain both space and query time improvements, but are not able to offer any provable guarantees of performance. Broder et al. [8] developed an LSH based on min-wise independent permutations for the Jaccard similarity for sets. There have been several followup work to LSH, improving its performance both theoretically (e.g., [30]) and practically (e.g., [27, 18]). Our results are mostly complementary and can be combined with these variants. We refer to the recent thesis of Andoni [3] and the survey by Andoni and Indyk [4] for further background.

Ailon and Chazelle [1] pioneered the use of FFT in dimensionality reduction; their main application was to obtain a fast version of the Johnson–Lindenstrauss transform. Our first algorithm is directly inspired by their work. Eshghi and Rajaram [13] proposed a class of LSH for angles based on the theory of concomitants from statistics; they use DFT as a heuristic to speed up their computations and do not offer any theoretical guarantees of correctness. Concurrently and independently of our work, Vybiral [31] obtained a version of the Johnson–Lindenstrauss theorem using circulant matrices and Gaussian random variables. His proof uses the fact that circulant matrices can be diagonalized using the discrete Fourier transform. Even though this appears syntactically close to our use of the double Hadamard transform, it is unclear if his analysis can be adapted either to the Hadamard transform setting or to the angle setting (as opposed to the distance setting in Johnson–Lindenstrauss theorem). Recently, Bachrach and Porat [5] and Feigenblat et al. [14] presented constructions that speed up the computation of the LSH for Jaccard similarity, namely, minwise independent fingerprints, by an exponential factor.

3. BACKGROUND

First, we set up some notation that will be used throughout the paper. Let X denote the set of input vectors in $\Re^d$ and let $|X| = n$. Let $x \in X$ denote an input vector and let $y \in \Re^d$ denote the query vector. Let $\|x\| = \|x\|_2$ denote the Euclidean norm of vector x unless otherwise noted and let $\|A\|_F$ be the Frobenius norm of matrix A. Let $[k]$ denote the set $\{1, \ldots, k\}$.

Let $N(0, \sigma)$ be the zero-mean Gaussian distribution with variance σ^2 and let $N(0, C)$ be the zero-mean multidimensional Gaussian distribution with covariance C. Let G be a $d \times d$ diagonal matrix where each diagonal entry is independently $N(0, 1)$. Let D be a $d \times d$ diagonal matrix where each diagonal entry is an independent Bernoulli random variable, i.e., D_{ii} is ± 1 equiprobably. Let M be a random permutation matrix of order d.

Hadamard matrices. We will use the following family of Hadamard matrices in our algorithms. Let $H_d \in \Re^{d \times d}$ denote the Hadamard matrix of order d, defined as follows: $H_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ and $H_{2^k} = H_2 \otimes H_{2^{k-1}}$, where $\otimes$ denotes the tensor product. For Hadamard matrices of order d, using an FFT-like algorithm, matrix-vector multiplication can be done in $O(d \log d)$ time; this fact will be important for our running time analysis. Since it will be clear from the context, we will drop the subscript d from H_d. Let D_{H_i} denote a $d \times d$ diagonal matrix such that $(D_{H_i})_{jj} = H_{ij}$. (For more background on Hadamard matrices and their properties, the readers are referred to standard CS undergraduate textbooks.)

An overview of LSH. We give a brief description of a basic LSH for ℓ_2 [12, 3]; we refer to this as the *naive* LSH. Let $X \subseteq \Re^d$ be the set of input points and let q be a query point. Given a distance parameter R and a recall parameter δ, our aim is to create a data structure to efficiently return at least $1 - \delta$ fraction of the neighbors of each query point that are at most a distance R (in ℓ_2 norm) from it. Let k, L, and w be parameters that we will choose later. Let A be a $k \times d$ random matrix where each A_{ij} is a random variable drawn independently from $N(0, 1)$ and let A_i denote the ith row of A. Let $b \in \Re^d$ be a random vector such that each b_i is chosen uniformly in $[w]$. For each $i \in [k]$, define $h_i(x) = \lfloor \frac{A_i \cdot x + b_i}{w} \rfloor$ to be the ith hash value for x. For $u = \|x - y\|$, let $p(u)$ be the (*collision*) probability that $h_i(x) = h_i(y)$ for any i. If $f(\cdot)$ denotes the density function of $N(0, 1)$, a simple argument in [12] shows that

$$p(u) = \int_0^w \frac{1}{u} f\left(\frac{t}{u}\right)\left(1 - \frac{t}{w}\right) dt,$$

which monotonically decreases with u.

Define $g(x) = (h_1(x), \ldots, h_k(x))$, i.e., a concatenation of the k hash values. The probability that $g(x) = g(y)$ is then $p^k(u)$. We then create L independent copies of $g(\cdot)$ as $g_1(x), \ldots, g_L(x)$. At preprocessing time, each point in the database is stored in each of the hash buckets $g_1(x), \ldots, g_L(x)$. During query time, for a given query point y, we compute $g_1(y), \ldots, g_L(y)$, and search all these L buckets to get a set of candidates. This candidate set may then be pruned based on whether we desire only the exact R-near neighbors. Following our previous calculation, the expected fraction of R-neighbors of the query point y that we get as candidates is at least $1 - (1 - p^k(R))^L$. We thus want this quantity to be at least $1 - \delta$, the targeted recall. This theoretically determines the values of the parameters k and L. In practice, as well as in our experiments, these parameters are determined by doing a grid search over the nearby k and L values using a sample of input points. The value of w is chosen to be 4, as suggested in [12]. The space used by the LSH data structure scales with L, the number of hash tables constructed. The time needed to find out the exact R-near neighbors of a query point consists of both the projection time and the time taken to prune the candidate set.

4. ALGORITHM ACHash

In this section we present the first LSH algorithm that we call ACHash. This algorithm is essentially a modification of the fast Johnson–Lindenstrauss transform (FJLT) of Ailon and Chazelle [1], with suitable rounding and thresholding steps for obtaining the hashing buckets. We first present ACHash and then argue that the collision probabilities of ACHash are very close to that of the naive LSH scheme.

We first recap FJLT whose goal is to obtain a faster version of the celebrated Johnson–Lindenstrauss transform. Given an input vector, FJLT first pre-conditions the vector so that it becomes dense while its length is unchanged. This is accomplished by using a random diagonal matrix and a Hadamard transform; this amounts to a random length-preserving rotation of the vector. As we mentioned earlier, the Hadamard transform can be computed in $O(d \log d)$ time. Once the input vector is densified, FJLT then projects the vector to a smaller dimension. The key observation is that given that the input vector has been densified, a very sparse Gaussian matrix is sufficient.

In ACHash, the steps are similar. We first pre-condition the input vector using a random diagonal matrix and a Hadamard transform, then apply a sparse Gaussian matrix followed by a rounding. We now proceed to a more formal description of ACHash (Algorithm 1). Let k and L be the parameters of LSH. Let $q = \min\left(\Theta\left(\frac{\log 1/\delta}{\epsilon^2}\frac{\log(d/\delta)}{d}\right), 1\right)$ and let P be a $k \times d$ matrix where $P_{ij} = 0$ with probability $1 - q$ and is $N(0, 1/q)$ with probability q. Note that the pre-conditioning needs to be computed only once. The time necessary to compute all the LSH buckets for one query point is thus $O(d \log d + kL \log^2 d)$.

We now proceed with the proof that the above method works. We first state the following key property of the pre-conditioning step shown in [1], which says that the vector $\hat{x}$ (Step 1) is sufficiently dense.

LEMMA 1 ([1]). *Let $x \in \Re^d$ be such that $\|x\| = 1$. Then, $\|HDx\|_\infty = O(\sqrt{\frac{\log(d/\delta)}{d}})$, with probability at least $1 - \delta$.*

Next we show a technical statement that guarantees that the subsampling using the matrix P preserves the Euclidean distances. For each $i \in [k]$, let S_i denote the set of coordinates such P_{ij} was chosen to be sampled from $N(0, 1/q)$. For a vector z, let z_{S_i} denote its restriction to the coordinate set S_i.

Algorithm 1 ACHash(x)

1: Compute $\hat{x} = HDx$.
2: **for** $i = 1, \ldots, L$ **do**
3: Generate $P \in \Re^{k \times d}$ where each P_{ij} is independently $N(0, 1/q)$ with probability q and 0 otherwise.
4: Generate $b \in \Re^k$ where each b_i is independently and uniformly in $[w]$.
5: Store
$$\texttt{ACHash}_i(x) = \left\lfloor \frac{P\hat{x} + b}{w} \right\rfloor$$
as the ith hash of x.

LEMMA 2. *Let $v \in \Re^d$ be such that $\|v\|_\infty = O(\sqrt{\frac{\log(d/\delta)}{d}})$ and $\|v\| = 1$. For all $\epsilon, \delta \in (0, 1)$, if $q = \Omega\left(\frac{\log(1/\delta)}{\epsilon^2}\frac{\log(d/\delta)}{d}\right)$, then*

$$\Pr\left[\left|\sum_{j \in S_i} v_j^2 - q\right| > \epsilon q\right] \le \delta.$$

PROOF. Let us define random variables $\delta_j = 1$ if $j \in S_i$, and apply Bernstein's inequality [29, Theorem 2.7] to the random variables $w_j = \delta_j v_j^2$. Note that

$$\sum_j E[w_j^2] = \sum_j q v_j^4 \le q\|v\|_\infty^2 \|v\|^2 = q\|v\|_\infty^2 = O\left(\frac{q \log(d/\delta)}{d}\right).$$

Therefore we have that

$$\begin{aligned} \Pr\left[\left|\sum_j w_j - q\right| > t\right] &\le \exp\left(-\frac{t^2/2}{\sum_j E[w_j^2] + \|w\|_\infty t/3}\right) \\ &\le \exp\left(-\frac{t^2/2}{O(q\log(d/\delta)/d + t\log(d/\delta)/(3d))}\right) \\ &\le \delta, \end{aligned}$$

for $t = \epsilon q$ and $q = \Omega\left(\frac{2}{\epsilon^2}\log(\frac{1}{\delta})\left(1 + \frac{\epsilon}{3}\right)\frac{\log(d/\delta)}{d}\right)$. □

The next theorem shows that the collision probabilities of ACHash are very similar to that of the naive LSH. Let $u = \|x - y\|$ and let

$$p_{\mathrm{AC}}(u) = \Pr[\texttt{ACHash}_i(x) = \texttt{ACHash}_i(y)]$$

denote the probability that all k hash values of ACHash agree for a generic $i \in [L]$.

THEOREM 3. *For all $\epsilon, \delta \in (0, 1)$, we have*

$$-(k+1)\delta + p^k((1+\epsilon)u) \le p_{\mathrm{AC}}(u) \le p^k((1-\epsilon)u) + (k+1)\delta.$$

PROOF. Let $\hat{x} = HDx$, $\hat{y} = HDy$, and $z = \hat{x} - \hat{y}$. Since HD is a unitary transformation, we have $\|z\| = u$. Using Lemma 1 we have that $\|\hat{x}\|_\infty, \|\hat{y}\|_\infty = O(\sqrt{\frac{\log(d/\delta)}{d}})$. By Lemma 2, we have that for all $i \in [k]$, with probability at least $1 - k\delta$,

$$(1 - \epsilon)u \le \sum_{j \in S_i} z_j^2/q \le (1 + \epsilon)u.$$

Thus each z_{S_i} preserves the distance u to within a factor of $1 \pm \epsilon$. Multiplying $\hat{x}_{S_i}$ by the Gaussian random variables (from P) and the bucketization performed in steps 4 and 5 corresponds to doing

a naive LSH on each of $\hat{x}_{S_i}$ and $\hat{y}_{S_i}$ vectors. Since the random variables are all independent, we get the corresponding guarantees from the naive LSH and the proof is complete by taking a union bound over all the bad events. □

5. ALGORITHM DHHash

In this section we present an improved hashing algorithm. While this algorithm is somewhat motivated by ACHash, it has two applications of the Hadamard transform and hence we call it DHHash. As before, we present the steps involved in computing the hash and then argue that the collision probabilities of DHHash are very close to that of the naive LSH scheme.

The first step is to pre-condition the input vector by applying a random diagonal matrix followed by a Hadamard transform. While this is the same as before, the rest of the steps are different. The next step is to apply a random permutation, followed by a random diagonal Gaussian matrix, and an another Hadamard transform. This will give us a vector from which k entries are sampled without replacement to give a particular hash bucket; the sampling step is then repeated L times independently. The formal description is given below (Algorithm 2). The time taken by DHHash to compute the buckets for a given query is thus $O(d \log d + kL)$, which is an improvement over ACHash.

The intuition behind the sequence of transformations used in DHHash is as follows.The first random diagonal matrix-Hadamard transform combination smoothens the input vectors so that the maximum coordinate is bounded w.h.p. For such smoothened vectors, the subsequent combination of the random *Gaussian* diagonal matrix and Hadamard transform simulates the multiplication of the input vectors by an i.i.d. Gaussian matrix – this is the key novelty in designing this transform. The final sampling then gives the indices of the hash buckets.

The crucial point to note in DHHash is that the vector itself is computed only once and kL indices are sampled with replacement from the resulting vector. This introduces dependence among the L hash buckets, but as our experiments will show, this dependence does not adversely affect the LSH performance. Note that it is possible to get a set of L independent buckets in $O(Ld \log k)$ time by repeatedly invoking a $O(d \log k)$ subroutine [2, 26] for computing k elements out of the randomized Hadamard transform. Recall the definitions of G, M and b from Section 3.

Algorithm 2 DHHash(x)

1: Compute
$$\zeta = \left\lfloor \frac{HGMHDx + b}{w} \right\rfloor .$$
2: **for** $i = 1, \ldots, L$ **do**
3: Store k entries sampled without replacement from ζ as DHHash$_i(x)$, the ith hash of x.

For ease of notation, let us recall the steps of the hashing operation as follows. For input vectors x and y, let $\hat{x}$ and $\hat{y}$ be their pre-conditioned and permuted versions, i.e., $\hat{x} = MHDx, \hat{y} = MHDy$. Let $g = \text{diag}(G)$ and let $g_i = (D_{H_i})\, g$. Observe that

$$\zeta_i(x) = \left\lfloor \frac{g_i^T \hat{x} + b_i}{w} \right\rfloor$$

is precisely the ith coordinate of vector ζ computed in Step 1 of Algorithm 2.

In what follows, we will show a bound on the collision probability of DHHash. Our guarantees will only be for a set of k indices sampled without replacement from ζ. Note that in the actual algorithm, we repeat this construction L times; while this seems to work well in practice, we do not have any theoretical guarantees on the joint distribution of the L choices.

Let S be a subset of k coordinates chosen at random without replacement, and without loss of generality we can assume $S = [k]$. Let $p_S(u)$ denote the probability that the hash bucket generated using these coordinates are the same for both x and y, where $u = \|\hat{x} - \hat{y}\| = \|x - y\|$. Since the b_i are i.i.d. uniform, observe that $p_S(u)$ can be written as

$$p_S(u) = \int_{\forall i, t_i = 0}^{t_i = w} f\left(t_1, \ldots, t_k\right) \prod_i \left(1 - \frac{t_i}{w}\right) dt_1 \ldots dt_k, \quad (1)$$

where $f(\cdot)$ denotes the joint pdf of the random variables τ_i where $\tau_i = g_i^T(\hat{x} - \hat{y})$.

Note that the distribution of $(\tau_1, \ldots, \tau_k)$ is a multidimensional Gaussian since each of $\tau_i = g^T D_{H_i}(\hat{x} - \hat{y})$ is a linear transformation of the Gaussian random variable g. If the Gaussian random vectors g_i were all independent, then we would have $p_S(u) = p^k(u)$. In our case the $g_i = D_{H_i} g$ are *not* independent. Nevertheless, we show strong lower and upper bounds on $p_S(u)$.

At a high level, our proof consists of two key steps. First we observe that the covariance of $(\tau_1, \ldots, \tau_k)$ depends on the "smoothness" of the vector $\hat{x} - \hat{y}$; ensuring this smoothness is precisely the role of the pre-conditioner. Second, we show that if the τ_i's are nearly uncorrelated, then $p_S(u)$ is close to $p^k(u)$. Let $c = O\left(\frac{\log d}{d}\right)$ and $\gamma = \sqrt{2c \log(d/\delta)}$ for the remainder of the proof.

LEMMA 4. *Let $\hat{x}$ and $\hat{y}$ be vectors in $\Re^d$ such that $\|\hat{x}\| = \|\hat{y}\| = 1$ and $\|\hat{x}\|_\infty \le \sqrt{c}$ and $\|\hat{y}\|_\infty \le \sqrt{c}$. Also, assume that $d > \log(d/\delta) \log(d)$ holds for a small fixed constant $\delta \in (0, 1)$. Then with probability $1 - \delta$, it holds that for all $i \ne j \in [d]$*

$$\left|\langle D_{H_i}\hat{x}, D_{H_j}\hat{y}\rangle\right| < \gamma.$$

PROOF. Consider a fixed i and j and let T be the set of coordinates t where $H_{it} = H_{jt}$. Since M is a random permutation, we can consider T as a random set of $d/2$ coordinates. Let $\delta_t = 1$ if $t \in T$ and 0 otherwise. Then,

$$\langle D_{H_i}\hat{x}, D_{H_j}\hat{y}\rangle = \sum_{t \in T} \hat{x}_t \hat{y}_t - \sum_{t \notin T} \hat{x}_t \hat{y}_t = 2 \sum_{t \in T} \hat{x}_t \hat{y}_t - \hat{x}^T \hat{y}.$$

Over the random choice of T, we have that

$$E\left[\sum_{t \in T} \hat{x}_t \hat{y}_t\right] = \hat{x}^T \hat{y}/2.$$

For $t \in [d]$ let $X_t = \delta_t \hat{x}_t \hat{y}_t - \hat{x}_t \hat{y}_t/2$. Note that the events $t \in T$ are not independent. However, since we can regard T as a set of $d/2$ coordinates sampled without replacement, we can employ the same form of Bernstein's inequality that is applicable to the independent case [20]. Note that

$$E[X_t^2] \le \hat{x}_t^2 \hat{y}_t^2/4 \le c\hat{x}_t^2/4,$$

and thus $\sum_t E[X_t^2] \le c/4$. Also observe that $|X_t| \le c$. Therefore, from Bernstein's inequality it follows that

$$\begin{aligned} \Pr\left[\left|\sum_{t \in T} \hat{x}_t \hat{y}_t - \frac{\hat{x}^T \hat{y}}{2}\right| > u\right] &\le 2 \exp\left(-\frac{u^2}{\sum_t EX_t^2 + cu/3}\right) \\ &\le 2 \exp\left(-\frac{u^2}{c/4 + cu/3}\right). \end{aligned}$$

By choosing $u = \gamma$, we have that $u < 1$ and $c/4 + cu/3 < 2c/3$. Thus it holds with probability $1 - 2(\delta/d)^3$ that

$$\left| \sum_{t \in T} \hat{x}_t \hat{y}_t - \frac{\hat{x}^T \hat{y}}{2} \right| \leq \gamma,$$

which means that for this i, j pair,

$$\left| \langle D_{H_i} \hat{x}, D_{H_j} \hat{y} \rangle - \frac{\hat{x}^T \hat{y}}{2} \right| = \left| 2 \sum_{t \in T} \hat{x}_t \hat{y}_t - \hat{x}^T \hat{y} \right| = \gamma.$$

Taking a union bound over all $i \neq j$ pairs, the statement of the lemma holds with probability at least $1 - O(\delta^3/d)$. □

We are now ready to state our main technical theorem. Observe that for a small enough γ, i.e., a large enough input dimension d, and for $k \ll d$ in Theorem 5, we have $u', u'' \approx u$ and $A_{k,\gamma}, B_{k,\gamma} \approx 1$ and therefore $p_S(u) \approx p^k(u)$.

THEOREM 5. *Assume that $\gamma = o(k^{-2})$. Let $u' = u\sqrt{\frac{1-k\gamma}{1-2k\gamma}}$, $u'' = u(1-k\gamma)^{1/2}$, and $A_{k,\gamma} = 1 + O(k^2\gamma)$ and $B_{k,\gamma} = 1 - \Theta(k^2\gamma)$ be constants depending on k and γ. Then,*

$$-2\delta + B_{k,\gamma} p^k(u'') \leq p_S(u) \leq A_{k,\gamma} p^k(u') + 2\delta.$$

PROOF. We give lower and upper bounds for the probability density function of the projections $f(\tau_1, \ldots, \tau_k)$ defined in (1).

Define the covariance matrix $C \in \Re^{k \times k}$ as $C_{ij} = E[\tau_i \tau_j]$. Thus, for all $i, j \in [1, k]$ we have that

$$\begin{aligned} C_{ij} &= (\hat{x} - \hat{y})^T D_{H_i} E[gg^T] D_{H_j} (\hat{x} - \hat{y}) \\ &= (\hat{x} - \hat{y})^T D_{H_i} D_{H_j} (\hat{x} - \hat{y}). \end{aligned}$$

It follows that $C_{ii} = u^2$. Using Lemma 1 with the vector $z = (1/u)(\hat{x} - \hat{y})$, we have that $\|z\|_\infty \leq \|x\|_\infty + \|y\|_\infty \leq 2\sqrt{c}$ with probability at least $1 - \delta$. Conditioning on this event, from Lemma 4 it follows that with probability at least $1 - \delta$, $|C_{ij}| \leq \gamma u^2$ for all $i \neq j$. Let $\Sigma = u^2 I_k$. Therefore C can be written as $C = \Sigma + u^2 E = u^2(I + E)$, where $|E_{ij}| \leq \gamma$ and $E_{ii} = 0$ and hence

$$\|E\| \leq \|E\|_F \leq k\gamma. \tag{2}$$

Now let

$$f_V(v) = \frac{1}{(2\pi)^{k/2}\sqrt{\det V}} \exp\left(-\frac{1}{2} v^T V^{-1} v\right)$$

denote the pdf of the k-dimensional Gaussian $N(0, V)$.

Using the perturbation result about the determinant [24, Corollary 2.14], we have that

$$\frac{|\det I - \det \frac{C}{u^2}|}{\det I} \leq \left(1 + \left\| I - \frac{C}{u^2} \right\|\right)^k - 1 \leq (1 + k\gamma)^k - 1,$$

and therefore

$$2 - (1 + k\gamma)^k \leq \det \frac{C}{u^2} \leq (1 + k\gamma)^k.$$

Also, for any vector v,

$$-v^T C^{-1} v + v^T \Sigma^{-1} v = v^T(-(I + E)^{-1} + I)v/u^2. \tag{3}$$

Using the standard matrix inverse perturbation bound [21, Section 5.8], we have that

$$\|(I + E)^{-1} - I\| \leq \frac{\|E\|}{1 - \|E\|}.$$

Therefore from (in)equalities (2) and (3),

$$\begin{aligned} v^T(-(I + E)^{-1} + I)v &\leq v^T v \|(I + E)^{-1} - I\| \\ &\leq v^T v \frac{\|E\|}{1 - \|E\|} \leq v^T v \frac{k\gamma}{1 - k\gamma}. \end{aligned}$$

Thus,

$$\begin{aligned} -v^T C^{-1} v &= -v^T \Sigma^{-1} v + v^T(\Sigma^{-1} - C^{-1})v \\ &\leq -u^{-2} v^T v + u^{-2} \frac{v^T v k\gamma}{1 - k\gamma} \\ &= -v^T v u'^{-2} = -v^T \Sigma'^{-1} v, \end{aligned}$$

where $u' = u/\sqrt{1 - \frac{k\gamma}{1-k\gamma}} = u\sqrt{\frac{1-k\gamma}{1-2k\gamma}}$ and $\Sigma' = u'^2 I$. Similarly,

$$\begin{aligned} -v^T C^{-1} v &\geq -u^{-2} v^T v - u^{-2} \frac{v^T v k\gamma}{1 - k\gamma} \\ &\geq -u^{-2} v^T v (1 - k\gamma)^{-1} = -v^T v u''^{-2} = -v^T \Sigma''^{-1} v, \end{aligned}$$

where $u'' = u(1 - k\gamma)^{1/2}$ and $\Sigma'' = u''^2 I$.

Now,

$$\begin{aligned} f(v) &= \frac{\exp(-v^T C^{-1} v/2)}{(2\pi)^{k/2}\sqrt{\det C}} \\ &\leq \frac{\exp(-v^T \Sigma'^{-1} v/2)}{(2\pi)^{k/2}\sqrt{\det \Sigma'}} \frac{\sqrt{\det \Sigma'}}{\sqrt{\det C}} \\ &\leq f_{\Sigma'}(v) \frac{(1 - k\gamma)^{k/2}}{(1 - 2k\gamma)^{k/2}\sqrt{2 - (1 + k\gamma)^k}} = f_{\Sigma'}(v) A_{k,\gamma}, \end{aligned}$$

where $A_{k,\gamma} := \frac{(1-k\gamma)^{k/2}}{(1-2k\gamma)^{k/2}\sqrt{2-(1+k\gamma)^k}}$. Similarly, using the lower bounds,

$$\begin{aligned} f(v) &\geq \frac{\exp(-v^T \Sigma''^{-1} v/2)}{(2\pi)^{k/2}\sqrt{\det \Sigma''}} \frac{\sqrt{\det \Sigma''}}{\sqrt{\det C}} \\ &\geq f_{\Sigma''}(v) \sqrt{\frac{(1 - k\gamma)^k}{(1 + k\gamma)^k}} = f_{\Sigma''}(v) B_{k,\gamma}, \end{aligned}$$

where we set $B_{k,\gamma} := \sqrt{\frac{(1-k\gamma)^k}{(1+k\gamma)^k}}$.

Using the above bounds for $f(\cdot)$ in (1), we have the main claim of the theorem. Observe that for $\gamma = o(k^{-2})$, we have $(1 + k\gamma)^{-k/2} = 1 - \Theta(k^2\gamma)$, $(1 - k\gamma)^{k/2} = 1 - \Theta(k^2\gamma)$, and $(1 - 2k\gamma)^{-k/2} = 1 + O(k^2\gamma)$. Substituting these into the definitions of $A_{k,\gamma}$ and $B_{k,\gamma}$ above, we obtain $A_{k,\gamma} = 1 + O(k^2\gamma)$ and $B_{k,\gamma} = 1 - \Theta(k^2\gamma)$. □

6. EXTENSIONS TO ANGLE SIMILARITY

In this section we sketch how to apply our methods to the setting when the similarity of two vectors is given by the angle between them. Along with the fast projection techniques of Sections 4 and 5, the idea is to use the signs of the projections [9]. Due to lack of space, we only provide a high-level description and omit the proofs.

To extend `ACHash` to the angle setting, we set the ith hash of the input vector x to be $\text{sgn}(PHDx)$. Provided that the angles are not very close to zero, we can show that the above hashing method has low bias in estimating the actual angle and that the estimates are correct with high probability. The mild technical condition of the angles not being to close to zero is easy to satisfy, say, by randomly perturbing all input vectors.

Name	# points	dimensions	# queries
FLICKR	1 million	1024	10k
QUERIES	1 million	376	10k
MNIST	60k	784	10k
P53	14.6k	5408	2k

Figure 1: Description of datasets.

To extend DHHash to the angle setting, we now compute $\zeta = \text{sgn}(\frac{HGMHDx+b}{w})$ instead; the rest of the steps are as before. Once again, we can show that this yields an LSH for the angle-based similarity of the input vectors.

7. EXPERIMENTS

In this section we describe the experimental results obtained by running our algorithms, compared against a standard LSH implementation as the baseline. We start by describing the datasets, at least two of which are publicly available for repeatability purposes.

We performed experiments in the R-near-neighbor setting. Given an input dataset and a query point, the goal is to return the set of points that lie within distance R of the query point.

7.1 Datasets

The experiments were performed on the following four datasets: FLICKR, QUERIES, MNIST, and P53. The basic statistics of the datasets are summarized in Figure 7.1.

The FLICKR dataset consists of images represented as dense vectors of dimension 1024 computed by a convolutional neural network. Out of these images, we sampled one million images in order to create the input dataset, and another 10,000 images to create the query set. When creating the query set, we ensured that the same image (according to the identifier present in the dataset) does not belong to both the input dataset and the query set. The QUERIES dataset was created by following the description given in [10], where the authors use an analogous dataset to demonstrate how temporal correlation of search queries is indicative of semantic similarity. In creating the QUERIES dataset, we calculated the frequencies of queries submitted to a major search-engine for 376 consecutive days. Each query q is then represented as a vector X_q of length 376, where the entry X_{qi} is the frequency of query q on the ith day. We kept only the 1.01 million most frequent queries, and again partitioned and sampled the set to obtain 1 million input vectors and 10,000 query vectors. Each of the input vectors and query vectors were also normalized using the mean and standard deviation as $\tilde{X}_{qi} = (X_{qi} - \mu(X_q))/\sigma(X_q)$, as described in [10].

For the sake of repeatability, we also perform the experiments on two smaller, publicly available datasets, namely, MNIST[1] and P53 [11][2]. The MNIST dataset consists of byte representations of images of handwritten digits that have been size normalized and pre-partitioned into training and test sets. We used this data set as-is, the 60k training images as the input points and 10k test images as query points. The P53 dataset consists of feature vectors that are each of size 5409, where each dimension indicates a feature value extracted from a biophysical model of the P53 protein. For this dataset, we removed all the data points that have missing entries, reducing the size of the dataset from 16.7k to 16.6k. These points were then partitioned randomly into 2k query points and 14.6k input points.

[1] yann.lecun.com/exdb/mnist/

[2] archive.ics.uci.edu/ml/datasets/p53+Mutants

7.2 Experimental method

We based our LSH implementation on the *Exact* Euclidean LSH (E^2LSH) implementation of Andoni et al.[3] E^2LSH implements R-near neighbor by first constructing a set of candidates using the LSH hash-buckets that the query point falls into, and then pruning the set of candidate by comparing each of them to the query point. The total query time is thus dependent on both the time taken to compute the LSH buckets and on the number of candidates returned by the LSH buckets.

E^2LSH also applies a pairing "trick" to speed up the LSH computation.[3] This reduces the time to compute all the probe locations in the LSH tables from $O(dkL)$ to $O(dk\sqrt{L})$. For any even k, L is set to $m(m-1)$. For a data point x, first it computes the $k/2$-wide LSH values $\{u^{(i)}(x) : i = 1, \ldots, m\}$ and then obtains L LSH values from these as $h^{(i,j)}(x) = (u^{(i)}(x), u^{(j)}(x))$, where $i \neq j$. Although the h values are not independent any more, the resulting scheme is still provably correct if L is set to slightly higher than in the fully independent case.

We compared four different algorithms, namely, DHHash, naive LSH (Naive), and two variants ACHash50 and ACHash25, where the sparsity parameter q of Algorithm 1 is set to 0.5 and 0.25 respectively. We ran E^2LSH with the pairing trick and modified the code to use each of these algorithms to compute the above described $u^{(i)}$ functions.

7.3 Metrics

For each dataset, we chose four different radii R, and these were chosen such that the average numbers of R-near neighbors are approximately 10, 25, 50, and 100. We present four different metrics for each dataset, for each radius:

- the average recall, i.e., fraction of R-near neighbors that are actually returned,
- the average query time per query measured in seconds elapsed,
- the time taken to compute the LSH indices,
- the space usage as measured by the number of hash tables constructed, L,
- the average number of nearest neighbor candidates that the E^2LSH algorithm had to sift through.

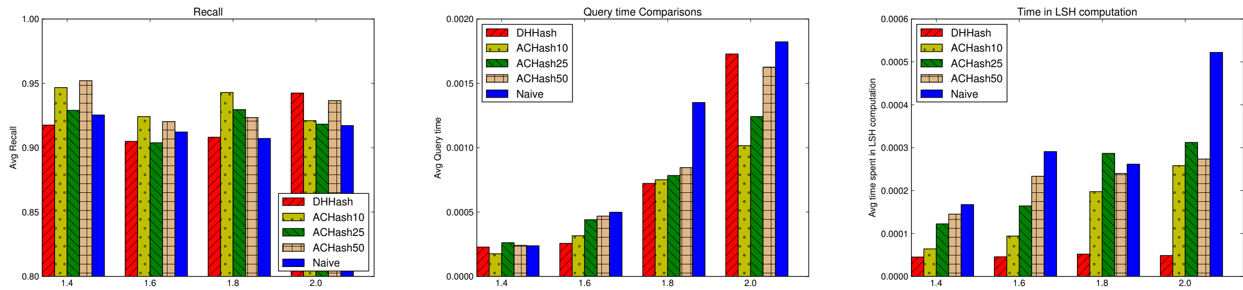
Note that E^2LSH filters set of candidates by computing the exact distances, it never reports false R-neighbors, i.e., its precision is always 1 irrespective of the LSH function used.

In order to be able to compare the query times of the different LSH for various algorithms, we fixed the targeted recall at 0.9. Since the performance of LSH schemes is sensitive to the parameters k and $L = m(m-1)$, we iterated over a range of parameters k and m and selected the parameter tuple that had the minimum average query time while having a macro-averaged recall over 0.9. Ideally, we would like to fix the recall of all the candidates exactly at 0.9 to be able to compare the query times. However, because of the small discrete jumps in the average recall in each of the LSH techniques, we were able to only approximately align the recall numbers of different algorithms. E^2LSH provides a functionality to estimate the optimal parameter set for a given recall and a given radius. We used this initial estimation to guide the construction of the grid of parameters that we searched over for each method.

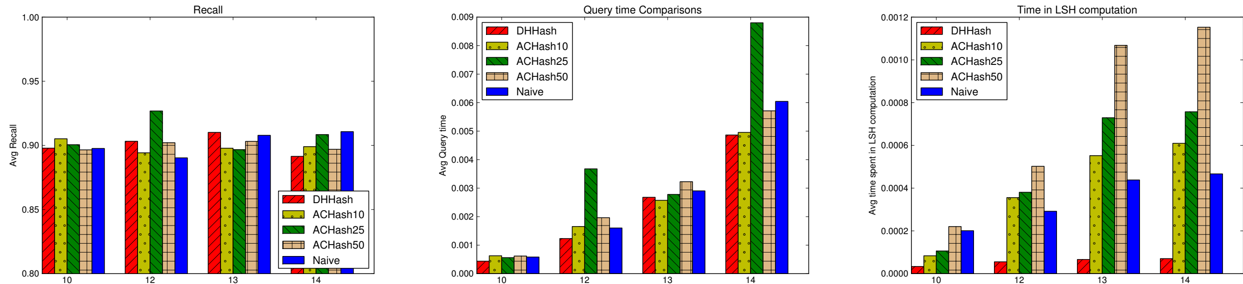
7.4 Results

We demonstrate the efficiency of our algorithms by carefully studying multiple metrics and datasets.

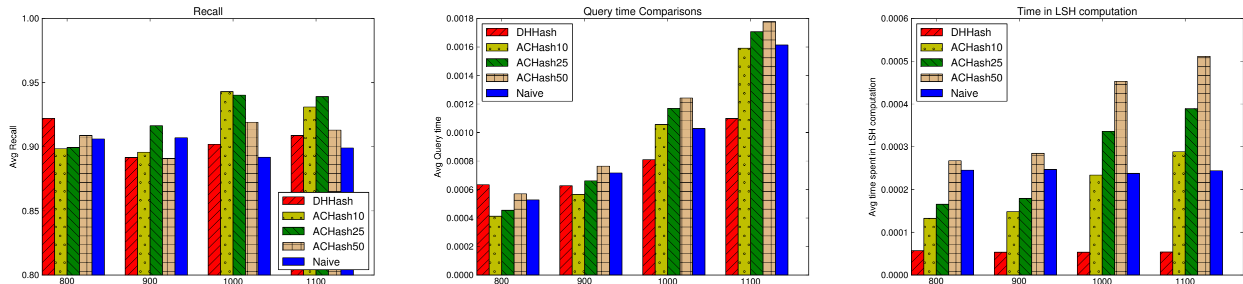
[3] www.mit.edu/~andoni/LSH/



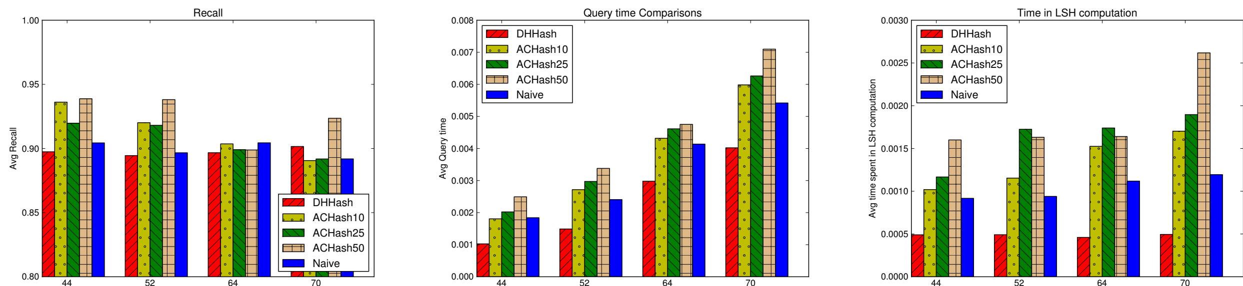
(a) Recall, LSH query time, and LSH computation time for FLICKR.



(b) Recall, LSH query time, and LSH computation time for QUERIES.



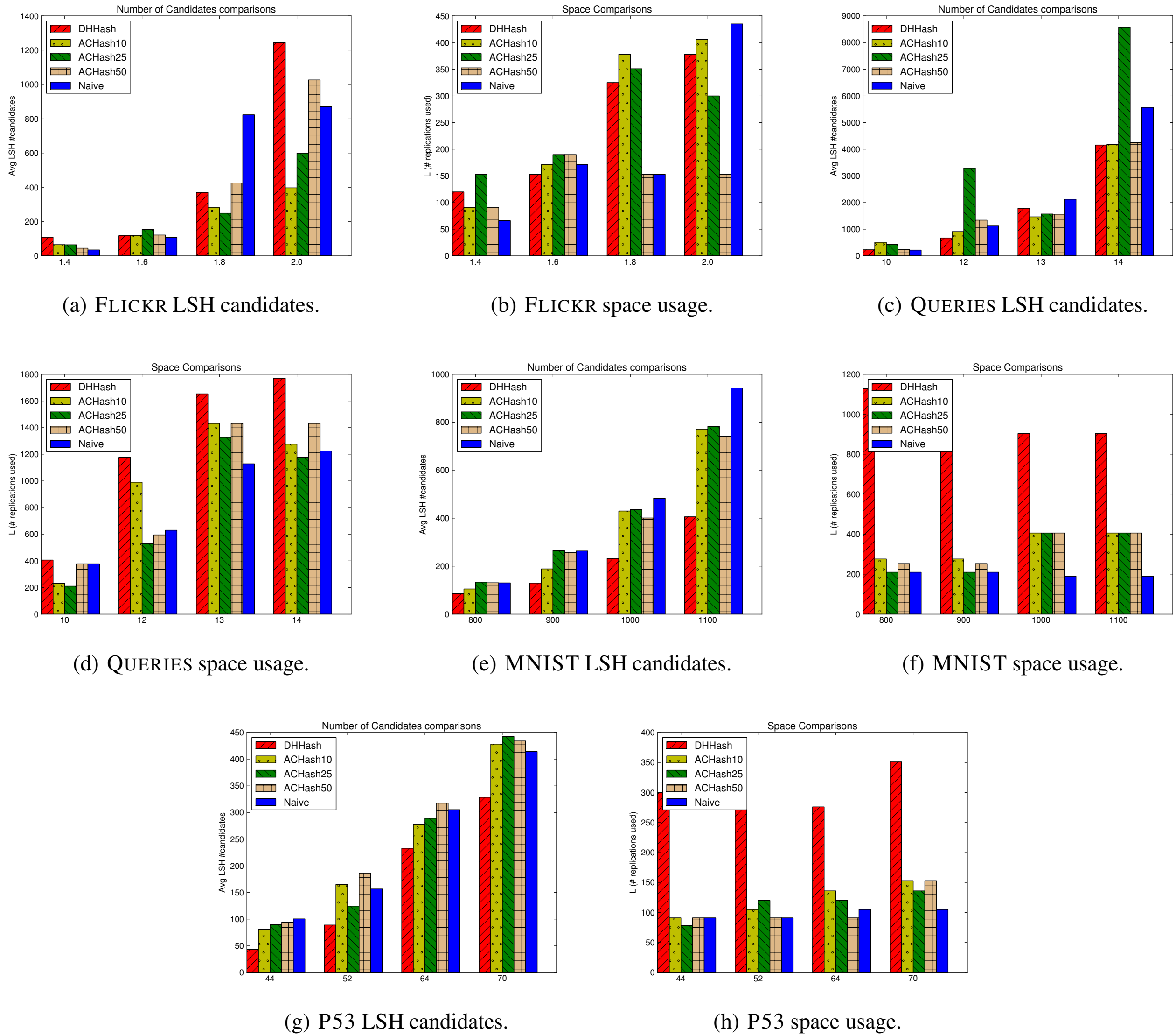
(c) Recall, LSH query time, and LSH computation time for MNIST.



(d) Recall, LSH query time, and LSH computation time for P53.

Figure 2: Recall, query times and LSH computation time for all four datasets

Recall. The first column in Figure 2 shows the recall levels achieved at the chosen parameter tuples for each of the radius values. As discussed above, the recall values are very close to each other, but are not aligned perfectly. The differences are at most a few percentage points. We observe that the recall for `DHHash` is almost always more than that of `Naive` at the chosen parameter tuple, thus making our claims of query time improvements justified.



(a) FLICKR LSH candidates. (b) FLICKR space usage. (c) QUERIES LSH candidates.

(d) QUERIES space usage. (e) MNIST LSH candidates. (f) MNIST space usage.

(g) P53 LSH candidates. (h) P53 space usage.

Figure 3: Number of LSH candidates and space usage for all four datasets.

Query time. The middle column in Figure 2 depicts the average query times obtained with the parameters that gave the aforementioned recalls. Overall, we see about a 15-20% improvement in most cases (and up to 50% maximum) by using `DHHash` over `Naive`. Using the two different variants of `ACHash`, however, does not always provide a uniform improvement in query time over `Naive`. Even at comparable (or lesser) recalls, we see `ACHash` variants performing slightly worse than `Naive`. We however, have not experimented with the sparsity settings of `ACHash` exhaustively. Overall this underlines the benefits of `DHHash` being parameter free.

LSH computation. The last column in Figure 2 shows the time taken in the computation of the LSH index for the average query point. As predicted by theory, `DHHash` is always an order of magnitude faster than `Naive`. `ACHash`, however, is not always faster than `Naive`, which is a result of two effects: at the sparsity setting we used, it still needs to compute an almost dense matrix-vector product, and the number of replications L needed for `ACHash` (and for `DHHash`) is typically more than that required for `Naive`.

Candidate set size. Figures 3(a), 3(c), 3(e), 3(g) illustrate the number of candidates that the LSH index returned as hits and the actual pairwise distance computed for by E^2LSH. The results are mostly correlated with the respective query times, except for the rare cases where in spite of having generated more candidates, `DHHash` runs faster as an effect of having computed the LSH functions much more efficiently.

Space usage. Finally, figures 3(b), 3(d), 3(f) and 3(h) show the space used by the chosen parameter settings, measured by the value of L, the number of replications performed. Except for FLICKR, `DHHash` has been able to improve runtime only at the expense of using more space than `Naive`.

Overall, we observe that due to the pairing "trick" described earlier, `Naive` spends the majority of time in filtering the candidates. Compared to `Naive`, `DHHash` achieves greater improvements in

query time than those possible by speeding up the LSH computation only for the same L and k. Its optimal k and L are larger, resulting in an increase in the number of cheap LSH operations performed and a decrease in the more expensive distance computations during filtering.

8. CONCLUSIONS

In this paper we proposed two new algorithms to speed up LSH for the Euclidean distance. Our algorithms exploit the property of being able to compute Hadamard transforms fast and consequently are able to reduce the hash index construction time to $O(d \log d + kL)$. While our algorithms are simple and easy to implement in practice, most of the difficulty is in showing provable guarantees on their performance. We develop novel analysis methods to this end. Our extensive experiments on four large datasets show that our algorithms achieve more than 20% improvement in query time over standard `DHHash` implementations.

Since LSH is so fundamental, our algorithms open up a wide possibility for their use in diverse settings. Interesting future directions include using our algorithms for applications such as de-duplication, clustering, similarity joins, all-pair similarity search, etc.

9. REFERENCES

[1] N. Ailon and B. Chazelle. The fast Johnson–Lindenstrauss transform and approximate nearest neighbors. *SIAM J. Comput.*, 39(1):302–322, 2009.

[2] N. Ailon and E. Liberty. Fast dimension reduction using Rademacher series on dual BCH codes. *Discrete and Computational Geometry*, 42(4):615–630, 2009.

[3] A. Andoni. *Nearest Neighbor Search: the Old, the New, and the Impossible*. PhD thesis, MIT, 2009.

[4] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, 2008.

[5] Y. Bachrach and E. Porat. Fast pseudo-random fingerprints. *Arxiv preprint arXiv:1009.5791*, 2010.

[6] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *Proc. 16th WWW*, pages 131–140, 2007.

[7] A. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. In *Proc. 6th WWW*, pages 391–404, 1997.

[8] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *J. Comput. Syst. Sci.*, 60(3):630–659, 2000.

[9] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proc. 34th STOC*, pages 380–388, 2002.

[10] S. Chien and N. Immorlica. Semantic similarity between search engine queries using temporal correlation. In *Proc. 14th WWW*, pages 2–11, 2005.

[11] S. Danziger, J. Zeng, Y. Wang, R. Brachmann, and R. Lathrop. Choosing where to look next in a mutation sequence space: Active Learning of informative p53 cancer rescue mutants. *Bioinformatics*, 23(13):i104, 2007.

[12] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proc. 20th SOCG*, pages 253–262, 2004.

[13] K. Eshghi and S. Rajaram. Locality sensitive hash functions based on concomitant rank order statistics. In *Proc. 14th KDD*, pages 221–229, 2008.

[14] G. Feigenblat, E. Porat, and A. Shiftan. Even better framework for min-wise based algorithms. *Arxiv preprint arXiv:1102.3537*, 2011.

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

[16] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proc. 25th VLDB*, pages 518–529, 1999.

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

[18] J. He, W. Liu, and S. Chang. Scalable similarity search with optimized kernel hashing. In *Proc. 16th KDD*, pages 1129–1138, 2010.

[19] M. R. Henzinger. Finding near-duplicate web pages: A large-scale evaluation of algorithms. In *Proc. 29th SIGIR*, pages 284–291, 2006.

[20] W. Hoeffding. Probability inequalities for sums of bounded random variables. *J. ASA*, 58(301):13–30, 1963.

[21] R. Horn and C. Johnson. *Matrix analysis*. Cambridge Univ Press, 1990.

[22] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. 30th STOC*, pages 604–613, 1998.

[23] P. Indyk, R. Motwani, P. Raghavan, and S. Vempala. Locality-preserving hashing in multidimensional spaces. In *Proc. 29th STOC*, pages 618–625, 1997.

[24] I. Ipsen and R. Rehman. Perturbation bounds for determinants and characteristic polynomials. *SIAM J. Matrix Analysis and Applications*, 30(2):762–776, 2008.

[25] N. Koudas and D. Srivatsava. Approximate joins: Concepts and techniques. In *Proc. 31st VLDB*, page 1363, 2005.

[26] E. Liberty, N. Ailon, and A. Singer. Dense fast random projections and lean Walsh transforms. In *Proc. 12th RANDOM*, pages 512–522, 2008.

[27] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-probe LSH: Efficient indexing for high-dimensional similarity search. In *Proc. VLDB*, pages 950–961, 2007.

[28] G. S. Manku, A. Jain, and A. D. Sarma. Detecting near-duplicates for web crawling. In *Proc. 16th WWW*, pages 141–150, 2007.

[29] C. McDiarmid. Concentration. In M. Habib, C. McDiarmid, J. Ramirez-Alfonsin, and B. Reed, editors, *Probabilistic Methods for Algorithmic Discrete Mathematics*, volume 16, pages 195–248. Springer, 1998.

[30] R. Panigrahy. Entropy-based nearest neighbor search in high dimensions. In *Proc. 17th SODA*, pages 1186–1195, 2006.

[31] J. Vybiral. A variant of the Johnson–Lindenstrauss lemma for circulant matrices. *J. Functional Analysis*, 260(4):1096–1105, 2011.

[32] R. Weber, H. Schek, and S. Blott. A quantitative analysis and performance study for similarity search methods in high dimensional spaces. In *Proc. 24th VLDB*, pages 194–205, 1998.