# Fast Image Retrieval via Embeddings

Piotr Indyk
LCS MIT
indyk@theory.lcs.mit.edu

Nitin Thaper
LCS MIT
nitin@theory.lcs.mit.edu

## Abstract

*In this paper we present a new algorithm for fast similarity search in large sets of images. The algorithm is obtained by employing the following two-step process. First, the dis-similarity metric is embedded into Euclidean space. Then, a fast nearest neighbor algorithm designed for the latter space is used.*

*The dis-similarity metric that we use is the Earth-Mover Distance (EMD). EMD has been experimentally verified to capture well the perceptual notion of a difference between images. However, the current algorithms for finding the nearest neighbor under EMD have running times linear in the number of images. This makes them inefficient when used for large data sets.*

*By embedding EMD into Euclidean space, and then using Locality-Sensitive Hashing (LSH) to find the nearest neighbor in the latter space, we obtain an approximate nearest neighbor search algorithm for the EMD metric that is order(s) of magnitude faster than the linear scan approach.*

## 1 Introduction

Image retrieval in large databases is a problem of interest in vision and database communities. The central questions in this area are:

- How to design a (dis)-similarity measure that quantifies the perceptual notion of two images being similar, and
- How to build a data structure that quickly identifies the images that are closest to a query image (the nearest neighbor search problem)

Early work on image retrieval typically solved the first issue by representing each image by a point in a multidimensional space, and using norms (e.g., Euclidean norm) to define the distance between two such points. For example, a natural method for extracting color information of an image is to compute its *color histogram*, i.e., split the color space into cells, and for each cell, count the number of pixels that fall into that cell. The advantage of this approach is that, in order to solve the nearest neighbor problem, one can use data structures specifically designed for points living in normed space (e.g., kd-trees or R-trees). Such data structures, especially if carefully implemented, provide mechanisms for query answering that are orders of magnitude faster than, say, naive linear scan for the nearest neighbor.

It was observed however, that the quality of retrieval achieved by such approaches is not always satisfactory (e.g., cf. [RTG00]) Thus, other metrics were proposed in the vision literature that *do not* satisfy norm axioms; examples include Hausdorff metric, Earth-Mover-Distance (EMD), variants of edit distance etc. Empirical evaluations suggest that such metrics often lead to better retrieval quality than when the normed-space approach is used. Unfortunately, for such metrics, the aforementioned efficient nearest neighbor data structures cannot be used. Instead, in such cases, the nearest neighbor search problem tends to be much more difficult. Typically, it is solved using simple linear scan that computes the distance between the query and all data points. Although it is possible to improve over a naive implementation of the above idea by exploiting triangle inequality and pruning (if we know that the distance to a given point is large, we do not need to actually to compute it), the speedup factors achieved by this method are typically low (less than 10). In the worst-case, the algorithms still suffer from linear query time.

In recent years, a novel approach for similarity search in un-normed metrics has been proposed [FCI99, CPSV00, MS00, CMS01, Cha02, ADG$^+$03]. The main idea is to *embed* the metric into a normed space, i.e., map each point in the metric space into a point in a normed space, so that the distance between the images of any two points is comparable to the distance between the points themselves. Given such a mapping (say $f$), we can implement similarity search data structure by constructing a nearest neighbor data structure for the images $f(p)$ of the data points $p$; then, in order to answer query $q$, we simply query our data structure with argument $f(q)$. The properties of the embedding imply that this approach guarantees that the answer is an *approximate* nearest neighbor, where the approximation factor depends on the distance distortion incurred by the mapping $f$.

Note that this approach requires the mapping $f$ to be well-defined and computable for *any* argument $q$, since in general $q$ is not known in advance. This means that $f$ needs to be designed for a *specific* metric. In particular, embedding techniques for general metrics (e.g., multidimensional scaling) cannot be used in this context, since they require advance knowledge of all metric points in order to construct the mapping.

In this paper, we apply the embedding methodology to the Earth-Mover Distance (EMD) metric [RTG00]. This metric has been experimentally verified to capture well the perceptual notion of a difference between images (e.g., with respect to their color or texture characteristic). The basic idea behind EMD is as follows. Assume that the features of an image are represented by a set of points in low-dimensional space $R^d$. For example, an image could be represented by a set of pixels, where each pixel is a point in 3-dimensional color space. The distance between two sets of points (representing two different images) is defined as the minimum amount of work needed to transform one set into another. Formally, this corresponds to the minimum weight matching (or flow) between the two sets of points.

We provide a low-distortion embedding of EMD into $l_1^d$ (i.e., $R^d$ equipped with Manhattan norm). Our mapping has a provable distortion upper bound of $O(\log \Delta)$, where $\Delta$ is the diameter of the underlying space (see later sections for formal definition). However, our experimental results show that the empirical distortion is much smaller (typically, around 10%). This allows us to reduce the nearest neighbor search problem in EMD to the equivalent problem in $l_1^d$, and use fast data structures for the latter problem.

The vectors $f(p)$ obtained from our embedding are sparse, but live in a very high-dimensional space (i.e., $d$ is very large). This means that we cannot use nearest neighbor data structures whose performance degrades quickly as the dimension increases (e.g., kd-trees). Instead, we use a different algorithm, namely Locality-Sensitive Hashing (LSH). The latter algorithm was introduced in [IM98, GIM99] and shown to solve the approximate nearest neighbor problem in time provably sublinear in the number of points $n$ even for large $d$ (see Preliminaries for more details on this method). In this paper we use a new variant of LSH, introduced in [DIIM03]. It is has the advantage of working directly in the Manhattan norm (the earlier method was applicable directly only to Hamming space). Since in our case the input vectors are high-dimensional but sparse, we adapted their algorithm to deal with this case.

By combining our embedding with the modified version of LSH, we obtain a very efficient algorithm for the nearest neighbor search under EMD. Our experiments (on 20,000 color signatures of Corel-Draw images) show that our algorithm is about 60 times faster than the naive linear scan, while most often reporting the exact (or almost exact) nearest neighbor. The speed-up is likely to increase if the number of data points grows.

As a by-product of our approach, we also obtain a very fast approximation algorithm for computing EMD between two images: to compute EMD between $p$ and $q$, we simply compute the Manhattan norm of $f(p) - f(q)$. The running time of our algorithm is linear in the image size; this should be contrasted with superpolynomial behavior of the simplex algorithm employed in [RTG00]. Thus, unlike in [RTG00], we do not need to precluster image pixels in order to make the computation of EMD feasible.[1] Although we did not pursue this direction, it could potentially lead to higher quality of retrieved images.

**Our techniques.** Our main technical contribution is the embedding of EMD into $R^d$ under Manhattan norm. The embedding can be viewed as resulting from a combination of the following two results[2] [3]

1. The result of [Cha02], who (implicitely) showed that the techniques of [KT99] imply the following: if a metric $M$ can be probabilistically embedded into trees with distortion $c$, then the EMD over $M$ can be embedded into $l_1$ with distortion $O(c)$.

2. The result of [CCG$^+$98] (cf. [Bar96]), who showed that the Euclidean metric over $\{1 \dots \Delta\}^d$ can be probabilistically embedded into trees with distortion $O(d \log \Delta)$. Again, that result is implicit in that paper.

In this paper, we present the resulting embedding explicitely, without going through the aforementioned sequence of steps. In addition, we give a direct proof of the properties of the embedding, not relying on the statements proved in [Bar96, CCG$^+$98, KT99, Cha02].

[1] We mention that we *do* use preclustered data for our experiments. This is because we need to compare the answers provided by our approximate algorithm with the exact ones. Computing the exact EMD between two raw images takes about 1 hour, which makes it infeasible to use it on a 20,000 image data set. Therefore, we had to resort to preclustered data in order to find the "ground truth" for our experiments.

[2] We skip the detailed definition of the terms used here, since we are not going to use them anywhere outside of this paragraph.

[3] We apologize for the following explanation being so convoluted. Unfortunately, this seems to be the most accurate way to depict the related work.

The main idea behind the embedding is as follows.[4] To embed $P$, we will compute and concatenate together several weighted histograms of $P$. Each histogram is defined by partitioning of the color space using a randomly shifted grid. The side lengths of the cells in different partitions form an exponential sequence (e.g., 1,2,4, etc). In addition, each entry in the histogram is multiplied by a weight that is proportional to the length of the grid cell.

It is easy to see that *one* color histogram fails to express similarity between images whose dominant colors fall to adjacent but different cells; one can call it a *quantization problem.* Intuitively, our embedding avoids this problem by combining several different quantization levels in a consistent fashion.

**Embeddings and high-dimensional nearest neighbor search.** Designing low-distortion embeddings of complex metrics into simpler ones, as well as designing fast nearest neighbor search algorithms for high-dimensional normed spaces, has been recently a subject of extensive research in the theoretical computer science community. For more background information about those areas, the reader is referred to surveys [Ind03, IM03].

# 2 The embedding

In this section we formally show how to construct an embedding of EMD into $l_1$, and prove a bound on its distortion.

Let $P, Q$ be two point sets of cardinality $s$, each in $\Re^k$ and $V = P \bigcup Q$. For any pair $p \in P, q \in Q$, the weight of $(p, q)$ is the Euclidean ($l_2$) distance between $p$ and $q$.

Recall that the EMD metric $D_M(P, Q)$ between these two point sets is defined as the cost of the minimum weight matching in the weighted bipartite graph consisting of all edges between points in $P$ and $Q$.[5]

Assume that the smallest inter-point distance is 1, and let $\Delta$ be the diameter of $V$. The embedding is defined as follows. We impose grids on the space $\Re^k$ of sides $1/2, 1, 2, 4, \ldots, 2^i \ldots \Delta$. Let $G_i$ be grid of side $2^i$. We impose the condition that the grid $G_i$ is a refinement of grid $G_{i+1}$. Moreover, the grid is translated by a vector chosen uniformly at random from $[0, \Delta]^k$.

For each grid $G_i$, we construct a vector $v_i(P)$ with one coordinate per cell, where each coordinate counts the number of points in the corresponding cell. In other words, each $v_i(P)$ forms a histogram of $P$. We define mapping $f$ by setting $f(P)$ to be the vector

$$v_{-1}(P)/2, v_0(P), 2v_1(P), 4v_2(P), \ldots, 2^i v_i(P), \ldots.$$

Note that $v(P)$ lives in an $O(\Delta^k)$-dimensional space, but only $O(\log(\Delta) \cdot |P|)$ entries in this vector are non-zero (i.e., the vector $v(P)$ is sparse).

This completes the description of the embedding. In the following, we provide two lemmas that bound the distortion induced by the embedding. For simplicity, we will show the claims for the case when $k = 2$ (i.e., when EMD operates on sets of points in the plane). The generalization to any *constant* dimension is straightforward, and only changes the constant factor in the distortion.

[4] For simplicity we describe here the embedding in the context of EMD over the color space. However, it can be used for EMD over other feature spaces (e.g., that describe the texture of images).

[5] In [RTG00] the authors provided a more general definition of EMD which does not assume $|P| = |Q|$. We do not consider this generalization here, since in such a case EMD does not form a metric.

**Lemma 1** *There is a constant $C$ such that for any $P, Q$, we have $D_M(P,Q) \leq C \cdot |v(P) - v(Q)|_1$.*

**Proof:** Let us consider the matching induced by pairing points within the same cells of grids $G_{-1}, G_1, \ldots$ etc. Firstly, observe that there are no pairings induced by $G_{-1}$, since all points fall to different cells; this implies $|v_{-1}(P) - v_{-1}(Q)|_1 = 2s$. Then, there are $s - |v_0(P) - v_0(Q)|/2$ pairs of points from $P$ and $Q$ that can be matched together within the same cells of grid $G_0$, which induces cost $\sqrt{2}[s - |v_0(P) - v_0(Q)|/2]$. Of the remainder, $|v_0(P) - v_0(Q)| - |v_1(P) - v_1(Q)|$ are matched in cells $G_1$, which induces cost $\sqrt{2} \cdot 2[|v_0(P) - v_0(Q)|/2 - |v_1(P) - v_1(Q)|/2]$, etc. In general, the grid $G_i$ induces the cost $\sqrt{2} \cdot 2^i[|v_{i-1}(P) - v_{i-1}(Q)|/2 - |v_i(P) - v_i(Q)|/2]$. By summing the costs up, we observe that

$$
\begin{aligned}
D_M(P,Q) &\leq \sqrt{2}[s + \frac{1}{2}(|v_0(P) - v_0(Q)| \\
&\quad +2|v_1(P) - v_1(Q)| + \ldots)] \\
&\leq \frac{1}{\sqrt{2}}(|v_{-1}(P) - v_{-1}(Q)| \\
&\quad +|v_0(P) - v_0(Q)| + \ldots) \\
&\leq C \cdot |v(P) - v(Q)|_1
\end{aligned}
$$

□

**Lemma 2** *There is a constant $C$ such that, for a fixed pair $P, Q$, if we shift grids randomly, then the expected value of $|v(P) - v(Q)|_1$ is at most $C \cdot D_M(P,Q) \log \Delta$.*

**Proof:** Observe that:

$$E[|v(P) - v(Q)|_1] = \sum_i 2^i E[X_i]$$

where $X_i = |v_i(P) - v_i(Q)|_1$.

Consider the best matching $M$ between the point sets. Let $n_i$ be the number of edges in $M$ with lengths between $2^{i-1}$ and $2^i$. By averaging argument we have

$$D_M(P,Q) \geq \sum_i n_i 2^{i-1}$$

Observe that any edge of $M$ left "uncut" by grid $G_i$ contributes nothing to $X_i$ and any edge "cut" by the grid contributes at most 2 to $X_i$. Thus $X_i$ can be bounded by:

$$E[X_i] \leq 2 \sum_j E[Y_{ij}]$$

where $Y_{ij}$ is the number of edges between lengths $2^{j-1}$ and $2^j$ cut by grid $G_i$.

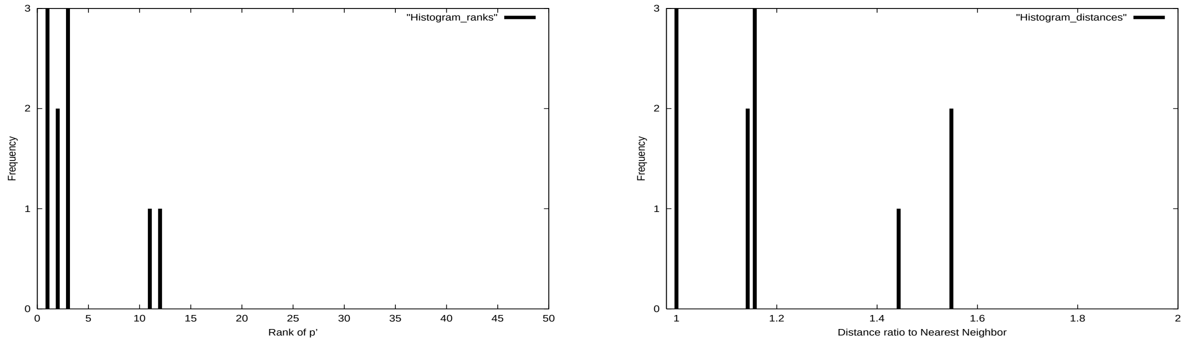
Since the probability of an edge of length $l$ being cut by grid $G_i$ is bounded by $l/2^i$, it follows that:

$$E[X_i] \leq 2 \sum_j n_j 2^j / 2^i \leq 2 D_M(P,Q)/2^i$$

from which it follows that

$$E[|v(P) - v(Q)|_1] \leq 2 \log \Delta D_M(P,Q)$$

□




(a) Rank

(b) Distance ratio

Figure 1: Quality of the nearest neighbors obtained via embeddings

### 2.1 Empirical distortion

The theoretical bounds on the distortion showed in the previous section are not strong enough to give meaningful practical guarantees. However, in practice, the distortion induced by the embedding is much lower. To verify that, we performed the following experiment. We took the 20,000 point data set (described in detail in the next section) and embedded all data elements into $\Re^d$. Then, for a few query points $q$, we computed their exact nearest neighbors $p$ (with respect to EMD metric) and $p'$ (with respect to the $l_1$ distance between $f(q)$ and $f(p')$). As Figure 1 illustrates, the nearest neighbor $p'$ in the embedded space is very likely to be among the top 10 near neighbors of the image in the original EMD space. Also, typically, the reported point was a $(1+\epsilon)$-approximate nearest neighbor, with $\epsilon < 20\%$.

If we repeat the randomized embedding process a few times, the chances of finding the nearest neighbor of the image increase. We did not perform additional experiments in this direction, since they would have been superseded by the experiments described in the next section.

## 3 Locality-Sensitive Hashing

In this section we describe the Locality-Sensitive Hashing algorithm [IM98, GIM99, DIIM03] for fast nearest neighbor search in $\Re^d$ under the $l_p$ norm.

We employ the following notation. For any point $\boldsymbol{v} \in \Re^d$, we denote by $||\vec{\boldsymbol{v}}||_p$ the $l_p$ norm of the vector $\vec{\boldsymbol{v}}$. Let $\mathcal{M} = (X, D)$ be any metric space, and $v \in X$. The *ball* of radius $r$ centered at $v$ is defined as $B(v, r) = \{q \in X \mid D(v, q) \leq r\}$.

### 3.1 Nearest Neighbor and its decision version

In this paper we focus on solving the *approximate decision* version of the nearest neighbor problem, and show how to use our solution for the optimization version.

Following [IM98], we define the $(R,\epsilon)$-Point Location in Equal Balls (PLEB) problem as follows.

**Definition 1 ($(R,\epsilon)$-PLEB)** *Given $n$ radius-$R$ balls centered at $P = \{p_1, \ldots, p_n\}$ in $\mathcal{M} = (X, D)$, devise a data structure which for any query point $q \in X$ does the following:*

- *if there exists $p \in P$ with $q \in B(p,R)$ then return* YES *and a point $p'$ such that $q \in B(p',(1+\epsilon)R)$,*
- *if $q \notin B(p,(1+\epsilon)R)$ for all $p \in P$ then return* NO*,*
- *if for the point $p$ closest to $q$ we have $R \leq D(q,p) \leq ((1+\epsilon)R)$ then return either* YES *or* NO*.*

Henceforth we will use $c$ to denote the approximation factor $(1+\epsilon)$ and refer to the $(R,\epsilon)$-PLEB problem as $(R,c)$-PLEB problem.

Observe that $(R,c)$-PLEB is simply a decision version of the Approximate Nearest Neighbor problem. Although in many applications solving the decision version is good enough, one can also reduce the approximate NN problem to approximate PLEB via binary-search-like approach. In particular, it is known [IM98, HP01] that the $\epsilon$-approximate NN problem reduces to $O(\log(n/\epsilon))$ instances of $(R,\epsilon)$-PLEB. Then, the complexity of $\epsilon$-approximate NN is the same (within log factor) as that of the $(R,\epsilon)$-PLEB problem.

## 3.2 Using LSH to solve $(R,c)$-PLEB problem

To solve the $(R,c)$-PLEB problem we employ the Locality Sensitive Hashing or LSH [IM98]. Its main idea is use hash functions such the probability of collision is higher for points that are "close" to each other than for those that are "far apart". Formally, for a domain $S$ of the points set with distance measure $D$, an LSH family is defined as:

**Definition 2** *A family $\mathcal{H} = \{h : S \to U\}$ is called $(r_1, r_2, p_1, p_2)$-*sensitive *for $D$ if for any $v, q \in S$*

- *if $v \in B(q, r_1)$ then $\Pr_{\mathcal{H}}[h(q) = h(v)] \geq p_1$,*
- *if $v \notin B(q, r_2)$ then $\Pr_{\mathcal{H}}[h(q) = h(v)] \leq p_2$.*

In order for a locality-sensitive hash (LSH) family to be useful, it has to satisfy inequalities $p_1 > p_2$ and $r_1 < r_2$.

We will briefly describe, from [IM98], how a LSH family can be used to solve the $(R,c)$-PLEB problem: We choose $r_1 = R$ and $r_2 = c \cdot R$. Given a family $\mathcal{H}$ of hash functions with parameters $(r_1, r_2, p_1, p_2)$ as in Definition 2, we amplify the gap between the "high" probability $p_1$ and "low" probability $p_2$ by concatenating several functions. In particular, for $k$ specified later, define a function family $\mathcal{G} = \{g : S \to U^k\}$ such that $g(v) = (h_1(v), \ldots, h_k(v))$, where $h_i \in \mathcal{H}$. For an integer $L$ we choose $L$ functions $g_1, \ldots, g_L$ from $\mathcal{G}$, independently and uniformly at random. During preprocessing, we store each $v \in P$ (input point set) in the bucket $g_j(v)$, for $j = 1, \ldots, L$. Since the total number of buckets may be large, we retain only the non-empty buckets by resorting to hashing. To process a query $q$, we search all buckets $g_1(q), \ldots, g_L(q)$; as it is possible (though unlikely) that the total number of points stored in those buckets is large, we interrupt search after finding first $3L$ points (including duplicates). Let $v_1, \ldots, v_t$ be the points encountered therein. For each $v_j$, if $v_j \in B(q, r_2)$ then we return YES and $v_j$, else we return NO.

The parameters $k$ and $L$ are chosen so as to ensure that with a constant probability the following two properties hold: (1) a point within distance $r_1$ from $q$ (if exists) collides with $q$ under *some* hash function, and (2) the number of points that are further than $r_2$ from $q$ and that collide with $q$, under all hash functions, is $< 3L$. Observe that if (1) and (2) hold, then the algorithm is correct. It follows (see [IM98] Theorem 5 for details) that if we set $k = \log_{1/p_2} n$, and $L = n^\rho$ where $\rho = \frac{\ln 1/p_1}{\ln 1/p_2}$ then (1) and (2) hold with a constant probability. Thus, we get following theorem (slightly different version of Theorem 5 in [IM98]), which relates the efficiency of solving $(R, c)$-PLEB problem to the sensitivity parameters of the LSH.

**Theorem 1** *Suppose there is a $(R, cR, p_1, p_2)$-sensitive family $\mathcal{H}$ for a distance measure $D$. Then there exists an algorithm for $(R, c)$-PLEB under measure $D$ which uses $O(dn + n^{1+\rho})$ space, with query time dominated by $O(n^\rho)$ distance computations, and $O(n^\rho \log_{1/p_2} n)$ evaluations of hash functions from $\mathcal{H}$, where $\rho = \frac{\ln 1/p_1}{\ln 1/p_2}$.*

In this paper, we use a novel LSH scheme based on $p$-stable distributions, first proposed in [DIIM03].

## 3.3 LSH using $p$-stable distributions

A distribution $\mathcal{D}$ over $\Re$ is called *$p$-stable*, if there exists $p \geq 0$ such that for any $n$ real numbers $v_1 \ldots v_n$ and i.i.d. variables $X_1 \ldots X_n$ with distribution $\mathcal{D}$, the random variable $\sum_i v_i X_i$ has the same distribution as the variable $(\sum_i |v_i|^p)^{1/p} X$, where $X$ is a random variable with distribution $\mathcal{D}$. It is known [Zol86] that stable distributions exist for any $p \in (0, 2]$. In particular, *Cauchy distribution* $\mathcal{D}_C$, defined by the density function $c(x) = \frac{1}{\pi} \frac{1}{1+x^2}$, is 1-stable.

Given a vector $\boldsymbol{v}$ of dimension $d$, the dot product $\boldsymbol{a}.\boldsymbol{v}$ is a random variable which is distributed as $(\sum_i |v_i|^p)^{1/p} X$ (i.e., $||\boldsymbol{v}||_p X$), where $X$ is a random variable with $p$-stable distribution. The dot products $(\boldsymbol{a}.\boldsymbol{v})$ can be used to assign a hash value to each vector $\boldsymbol{v}$. Intuitively, the hash function family should be locality sensitive, i.e. if two vectors $(\boldsymbol{v_1}, \boldsymbol{v_2})$ are close (small $||\boldsymbol{v_1} - \boldsymbol{v_2}||_p$) then they should collide (hash to the same value) with high probability and if they are far they should collide with small probability. The dot product $\boldsymbol{a}.\boldsymbol{v}$ projects each vector to the real line; It follows from $p$-stability that for two vectors $(\boldsymbol{v_1}, \boldsymbol{v_2})$ the distance between their projections $(\boldsymbol{a}.\boldsymbol{v_1} - \boldsymbol{a}.\boldsymbol{v_2})$ is distributed as $||\boldsymbol{v_1} - \boldsymbol{v_2}||_p X$ where $X$ is a $p$-stable distribution. If we "chop" the real line into equi-width segments of appropriate size $r$ and assign hash values to vectors based on which segment they project onto, then it is intuitively clear that this hash function will be locality preserving in the sense described above.

Formally, each hash function $h_{\boldsymbol{a},b}(\boldsymbol{v}) : \mathcal{R}^d \rightarrow \mathcal{N}$ maps a $d$ dimensional vector $\boldsymbol{v}$ onto the set of integers. Each hash function in the family is indexed by a choice of random $\boldsymbol{a}$ and $b$ where $\boldsymbol{a}$ is, as before, a $d$ dimensional vector with entries chosen independently from a $p$-stable distribution and $b$ is a real number chosen uniformly from the range $[0, r]$. For a fixed $\boldsymbol{a}, b$ the hash function $h_{\boldsymbol{a},b}$ is given by $h_{\boldsymbol{a},b}(\boldsymbol{v}) = \lfloor \frac{\boldsymbol{a} \cdot \boldsymbol{v} + b}{r} \rfloor$

The probability that two vectors $\boldsymbol{v_1}, \boldsymbol{v_2}$ collide under a hash function drawn uniformly at random from this family, can be computed as follows. Let $f_p(t)$ denote the probability density function of the absolute value of the $p$-stable distribution. We may drop the subscript $p$ whenever it is clear from the context.

For the two vectors $\boldsymbol{v_1}, \boldsymbol{v_2}$, let $c = ||\boldsymbol{v_1} - \boldsymbol{v_2}||_p$. For a random vector $\boldsymbol{a}$ whose entries are drawn from a $p$-stable distribution, $\boldsymbol{a}.\boldsymbol{v_1} - \boldsymbol{a}.\boldsymbol{v_2}$ is distributed as $cX$ where $X$ is a random variable drawn from a $p$-stable distribution. Since $b$ is drawn uniformly from $[0, r]$ it is easy to see that

$$Pr_{\boldsymbol{a},b}[h_{\boldsymbol{a},b}(\boldsymbol{v_1}) = h_{\boldsymbol{a},b}(\boldsymbol{v_2})] = \int_0^r \frac{1}{c} f_p(\frac{t}{c})(1 - \frac{t}{r})dt$$

For a fixed parameter $r$ the probability of collision decreases monotonically with $c = ||\boldsymbol{v_1} - \boldsymbol{v_2}||_p$. Thus, as per Definition 2 the family of hash functions above is $(r_1, r_2, p_1, p_2)$-sensitive for $p_1 = \int_0^r f_p(t)(1 - \frac{t}{r})dt$ and $p_2 = \int_0^r \frac{1}{c} f_p(\frac{t}{c})(1 - \frac{t}{r})dt$ for $r_2/r_1 = c$.

Henceforth, we shall restrict our attention to the $p = 1$ case since we are only concerned with LSH in $l_1$ space. For this case, it was shown in [DIIM03] that the ratio $\rho = \min_r \frac{\ln 1/p_1}{\ln 1/p_2}$ (which, as discussed earlier, corresponds to the exponent in the query time) is close to $1/c$. Thus, we expect sublinear query time from this hash family.

# 4 Experimental Evaluation

In this section we present an experimental evaluation of our image retrieval scheme. We discuss some of the issues pertaining to the implementation of our technique and then report some preliminary performance results.

## 4.1 Implementation details

In this section we describe the issues that arise when implementing the embedding and LSH algorithms, as well as our solutions.

### 4.1.1 $l_1$ Embedding

Within each embedding, the signature of each image is mapped into a series of vectors, one for each grid, and concatenated into one vector. The resulting vector can still be stored compactly owing to it's (very) sparse nature.

Since our embedding process is randomized, and the low-distortion guarantee (in particular, the upper bound) holds only in the expectation, the embedding process needs to be replicated a few times to guarantee good results. In our implementation, we replicate the embedding process 5 times for the entire image database.

It should be noted that the above replication process increases the query time as well as the overall memory used by a factor of 5. However, the increase in the memory usage is secondary to our main goal of achieving fast retrieval time. As far as the running time is concerned, note that the experimental query times reported in later sections account for the 5-fold slowdown caused by the replication; despite the slowdown, the resulting algorithm is still order(s) of magnitude faster than the linear scan.

### 4.1.2 Nearest neighbor vs PLEB

Our goal was to design a fast algorithm for solving (approximately) the nearest neighbor problem. However, as mentioned earlier, the LSH approach solves only its *decision* version. In principle, we could use the aforementioned reductions of [IM98, HP01] to reduce the former problem to logarithmically many instances of the latter one. However, instead, we decided to use only *one* LSH data structure to solve the nearest neighbor problem. This is due to the fact that, in practice, an LSH data structure works well for a range of radii $R$, not just for one value. For data sets in which the distance to the nearest neighbor varies a lot, several instances of LSH data structure (for different values of $R$) could be used.

### 4.1.3 $p$-stable LSH

It might appear from the way we have described the hash family that it requires a large number of random bits to represent each function from this family: Each hash function is specified by a vector of Cauchy random variables, $\boldsymbol{a}$ (of length equal to the dimension of the embedded space) and a scalar $b$. This obstacle can be avoided by using Nisan's pseudorandom number generator for space bounded computation [Nis90], which enables reducing the number of required random bits to $O(\log^2 d)$. However, in our implementation, we use a much simpler approach to generate the random Cauchy distributed variable $a_i$. We store one random variable, $r$ per hash function, and use it to generate a uniform random variable, $u_i \in [0, 1)$ for coordinate $i$ as follows: $u_i = i\, r \bmod p/p$ for some large prime $p$. Given the uniform random variable $u_i$, the corresponding Cauchy random variable $a_i$ is generated simply by $a_i = \tan(\pi(u_i - 1/2))$. It might be pertinent to mention here that we use a simple table lookup to approximate the *tan* function, thus avoiding the overhead of a potentially expensive floating point trigonometric routine.

As the total number of buckets for each each hash function $g_j = (h_{j1}, h_{j2}, \ldots h_{jk})$ may be large (potentially unbounded), we compress the buckets by resorting to standard hashing. Thus, we use two levels of hashing: the LSH function maps a point $p$ to bucket $g_j(p)$ and a standard hash function maps the contents of these buckets into a hash table of size $M$, using chaining to handle overflows.

**Parameters and Performance Tradeoffs:** The three main parameters that affect the performance of the LSH algorithm are: number of projections per hash value ($k$), number of hash tables ($l$) and the width of the projection ($r$). In general, one could also introduce another parameter (say $T$), such that the query procedure stops after retrieving $T$ points. In our analysis, $T$ was set to $3l$. In our experiments, however, the query procedure retrieved *all* points colliding with the query (i.e., we used $T = \infty$). This reduces the number of parameters and simplifies the choice of the optimal.

As we increase the value of $k$ the probability of two points colliding decreases exponentially. This decreases the number of false positives per hash tables, i.e. points that are not near neighbors but hash into the same bucket as the query point. At the same time, increasing $k$ also increases the number of false negatives, i.e. points that are true near neighbors but do not hash into the same bucket as the query point. If we increase $k$, we also have to increase $l$ in order to maintain the fraction of false negatives below a certain threshold. As a result, not only does the time to compute a single hash function increase, but we also need to evaluate more hash functions per query point. However, the total number of data points that collide with the query point over **all** the hash functions, i.e. candidate near neighbors, is reduced. Effectively, we spend more time doing hashing and less time checking distances with the candidate near neighbors.

Decreasing the width of the projection ($r$) decreases the probability of collision for any two points. Thus, it has the same effect as increasing $k$. As a result, we would like to set $r$ as small as possible and in this way decrease the number of projections we need to make. However, decreasing $r$ below a certain threshold increases the quantity $\rho$, thereby requiring us to increase $l$. Thus we cannot decrease $r$ by too much.

For a given value of $k$, it is easy to find the optimal value of $l$ which will guarantee that the fraction of false negatives are no more than a user specified threshold. This process is exactly the same as in earlier work on LSH.

In our experiments we tried different values of $k$ and $r$ and found that $k = 6$ and $r = 5.0$ semed to provide the best tradeoff between fast query time and quality of solution. With this set of parameters, we found that the query processing time for our algorithm is orders of magnitude faster (median speedup $\approx 60$, average speedup $\approx 90$) than the linear scan. Before we report our performance numbers we describe the data set that we used for testing.

## 4.2 Experiments

**Data Set:** We performed our image retrieval experiments on a collection of 20,000 color images from the Corel Stock Photo Library. The data set is identical to the one used in [RTG00]. Each color image was first transformed into the CIE-Lab color space (after appropriate preprocessing to remove dithering artifacts). The distribution of points was then coalesced into clusters of similar colors, using a two-stage clustering algorithm based on a *k-d* tree. In this database, the average image signature turned out to have 8.8 clusters.

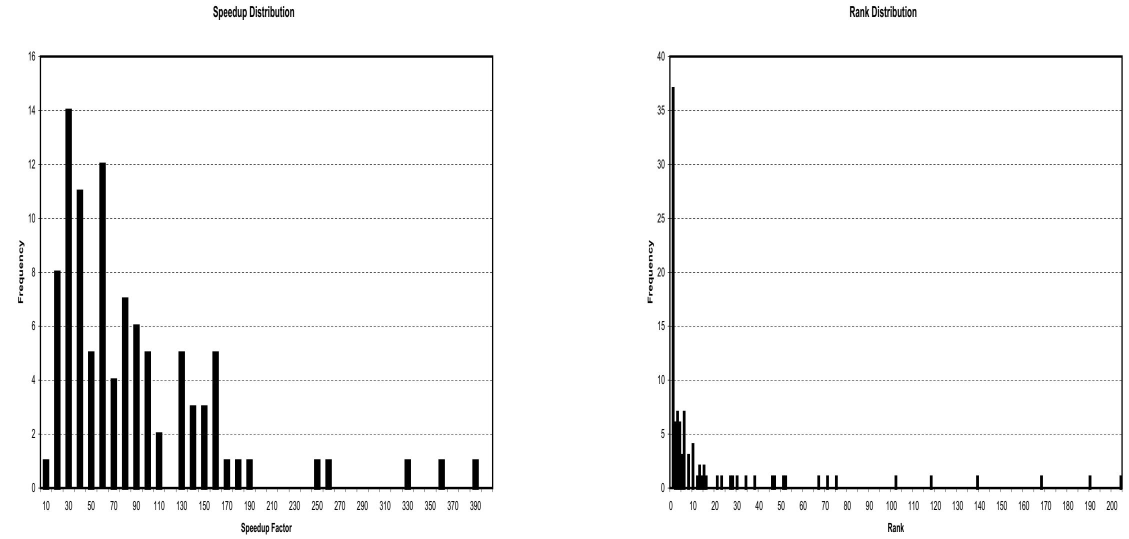
**Experimental Results:** There are two main parameters of interest in our approach, namely the retrieval time and the accuracy of the answer. In this section, we experimentally evaluate both parameters for our proposed algorithm.

For all our experiments we set the parameters $k = 6$ and $r = 5.0$. Moreover, we set the percentage of false negatives that we can tolerate to 10%. For this choice of parameters, $\ell$ evaluated to 25. All experiments were performed on a dual-processor Intel machine (Pentium II, 300 Mhz) with 256 Mb main memory and 512 Kb cache on each processor, running Redhat Linux 6.2.

The first set of experiments measured the accuracy and retrieval time for 100 randomly selected query images against the naive EMD computation technique. The median speedup achieved by the algorithm was 59 while the average speedup was 90. The median rank of the retrieved image was 3. Figures 2(a) and 2(b) show the distribution of speedups and ranks of retrieved images for the 100 queries.

There are a few outlier query images for which the rank of the retrieved images is high. A closer look at one such query image (with rank 204) reveals the reason for the somewhat poor performance. A *distance profile* of a query is the number of its $c$-approximate nearest neighbors as a function of $c$. The LSH technique for finding near neighbors works best for smooth profiles. As Figure 3 illustrates, the distance profile curve for the outlier image is much steeper than the average.

For the second set of experiments, we varied the number of images in the database from 5000 to 20000 and observed the retrieval times for the query images. Figures 4(a) and 4(b) show the




(a) speedup (b) rank

Figure 2: Distribution of speedup and accuracy

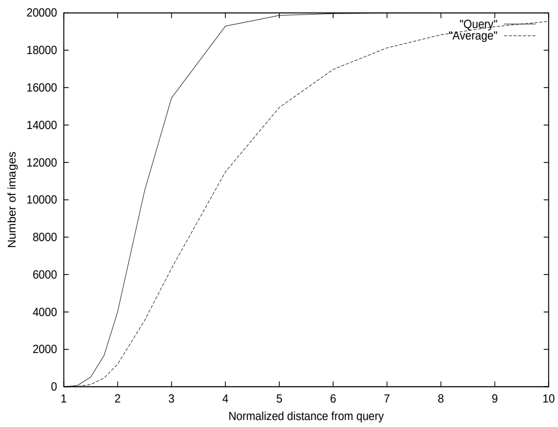
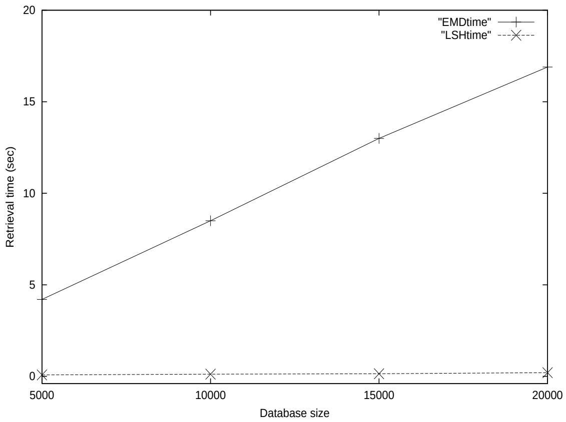
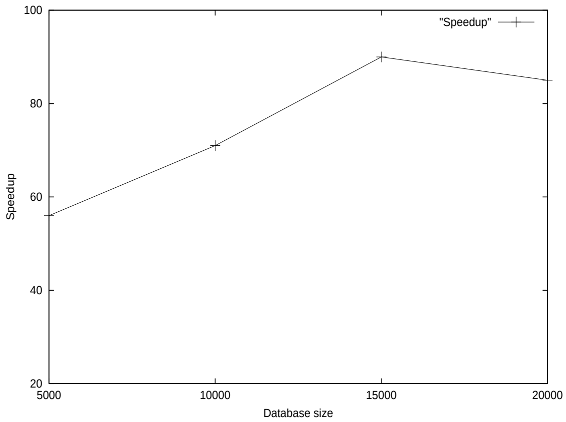



Figure 3: Distance profile for outlier query




(a) query time vs $n$




(b) speedup vs $n$

Figure 4: Improvement in speedup with data size

average processing times and speedups respectively as $n$ is varied. As we see from the Figures, the running time of our technique is sublinear in $n$. Thus, our algorithm scales much better than the brute force EMD computation algorithm.

# 5 Conclusions

In this paper we present a self-contained and direct method for embedding Earth Mover Distance into $l_1$ with provably bounded distortion. By combining it with the recent algorithm for approximate nearest neighbor in $l_1$, we obtain fast approximate nearest neighbor algorithm for the EMD metric.

The technique used for the embedding can be modifed to work as well for the following metric (we call it a *Connectivity Metric*, or *CM*). Let $P, Q$ be two sets of points in $\Re^d$. Then $CM(P,Q)$ is equal to the minimum value of $\sum_{(p,q)\in E} ||p-q||$ over all graphs $G = (P \cup Q, E)$ with the property that every point in $P$ is connected (in $G$) to some point in $Q$, and vice-versa. Note that two vertices $p$ and $q$ in $G$ are connected if $G$ contains a *path* (not necessarily an edge) from $p$ to $q$. The embedding of CM is very similar to the one for EMD; the only difference is that each coordinate in $f(P)$ is equal to 1 if the corresponding cell contains any point from $P$, and is equal to 0 otherwise. We are not aware of any previous work on (and therefore motivation for) the Connectivity Metric. However, we plan to investigate applications of this metric to visual information retrieval in the future.

# References


[ADG+03] A. Andoni, M. Deza, A. Gupta, P. Indyk, and S. Raskhodnikova. Lower bounds for embedding of edit distance into normed spaces. *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, 2003.

[Bar96] Y. Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. *Proceedings of the Symposium on Foundations of Computer Science*, 1996.

[CCG+98] M. Charikar, C. Chekuri, A. Goel, S. Guha, and S. Plotkin. Approximating a finite metric by a small number of tree metrics. *Proceedings of the Symposium on Foundations of Computer Science*, 1998.

[Cha02] M. Charikar. Similarity estimation techniques from rounding. *Proceedings of the Symposium on Theory of Computing*, 2002.

[CMS01] G. Cormode, M. Muthukrishnan, and C. Sahinalp. Permutation editing and matching via embeddings. *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP)*, 2001.

[CPSV00] G. Cormode, M. Paterson, C. Sahinalp, and U. Vishkin. Communication complexity of document exchange. *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, 2000.

[DIIM03] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. *DIMACS Workshop on Streaming Data Analysis and Mining*, 2003.

[FCI99] M. Farach-Colton and P. Indyk. Approximate nearest neighbor algorithms for hausdorff metrics via embeddings. *Proceedings of the Symposium on Foundations of Computer Science*, 1999.

[GIM99] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB)*, 1999.

[HP01] S. Har-Peled. A replacement for voronoi diagrams of near linear size. *Proceedings of the Symposium on Foundations of Computer Science*, 2001.

[IM98] P. Indyk and R. Motwani. Approximate nearest neighbor: towards removing the curse of dimensionality. *Proceedings of the Symposium on Theory of Computing*, 1998.

[IM03] P. Indyk and J. Matoušek. Low distortion embeddings of finite metric spaces. *CRC Handbook of Discrete and Computational Geometry*, 2003.

[Ind03] P. Indyk. Nearest neighbors in high-dimensional spaces. *CRC Handbook of Discrete and Computational Geometry*, 2003.

[KT99] J. M. Kleinberg and E. Tardos. Approximation algorithms for classification problems with pairwise relationships: Metric labeling and markov random fields. *Proceedings of the Symposium on Foundations of Computer Science*, 1999.

[MS00] S. Muthukrishnan and C. Sahinalp. Approximate nearest neighbors and sequence comparison with block operations. *Proceedings of the Symposium on Theory of Computing*, 2000.

[Nis90] N. Nisan. Pseudorandom generators for space-bounded computation. *Proceedings of the Symposium on Theory of Computing*, pages 204–212, 1990.

[RTG00] Y. Rubner, C. Tomassi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 40(2):99–121, 2000.

[Zol86] V.M. Zolotarev. *One-Dimensional Stable Distributions.* Vol. 65 of Translations of Mathematical Monographs, American Mathematical Society, 1986.