Facebrowsing: Search and Navigation through Comparisons

Dominique Tschopp
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne (EPFL)
1015 Lausanne, Switzerland
Email: (dominique.tschopp)@epfl.ch

Suhas Diggavi
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne (EPFL)
1015 Lausanne, Switzerland
Email: (suhas.diggavi)@epfl.ch

***Abstract*—This paper addresses the problem of finding the nearest neighbor (or one of the R-nearest neighbors) of a query object in a database which is only accessible through a comparison oracle. The comparison oracle, given two reference objects and a query object, returns the reference object closest to the query object. The oracle attempts to model the behavior of human users, capable of making statements about similarity, but not of assigning meaningful numerical values to distances between objects. We develop nearest-neighbor search algorithms and analyze its performance for such an oracles.**

Using such a comparison oracle, the best we can hope for is to obtain, for every object in the database, a ranking of the other objects according to their distance to it. The difficulty of searching using such an oracle depends on the non-homogeneities of the underlying space. We introduce the new idea of a rank-sensitive hash (RSH) function which gives same hash value for "similar" objects based on the rank-value of the objects obtained from the similarity oracle. As one application of RSH, we demonstrate that, we can retrieve one of the $(1+\epsilon)r$-nearest neighbor of a query point in time-complexity depending on an underlying property (termed rank-distortion) of the search space.

We use this idea to implement a navigation system for an image database of human faces. In particular, we design a database for images that is organized adaptively based on both baseline comparisons using eigenfaces and refined using selected human input. We present a preliminary implementation of this system which seeks to minimize the number of questions asked to a (human) oracle.

I. Introduction

In this paper, we consider the situation where we want to search and navigate a database, but we do not know the underlying relationships between the objects. This implies, in particular, that distances may be difficult to discern, or may not be well-defined. Such situations are common with objects where human perception may be involved. A collection of pictures of faces, taken from different angles and distances is an illustration of such a dataset. Indeed, the distances between automatically extracted feature vectors might be far from the similarity perceived by humans. Notwithstanding, either with human-assistance or approximate classification, we may be able to determine the relative proximity of an object with respect to a small number of other objects[1]. More precisely, Humans have the ability to compare objects and make statements about which are the most similar ones, though they can probably not assign a meaningful numerical value to similarity. This led to the question of how to design search algorithms based on binary similarity decisions of the type "A looks more like B than C". We address this question in two ways. We formulate and study the theoretical question underlying this setup, and develop a new hashing scheme to address it. We also have a preliminary implementation of a practical image database navigation system that uses this idea of human-assisted search.

More formally, we aim to design an algorithm that given a query object (*e.g.,* the face of a person we are looking for), efficiently returns an object that is similar to that object among the objects in a database. To do so, we have access to a similarity oracle which, given two reference objects and a query object, can tell which of the two reference objects is most similar to the query object. We measure the performance of our algorithms in terms of the number of questions that we need to ask the oracle. This is motivated by the fact that we consider that it is very costly to ask human users to answer questions. Hence, we aim at minimizing the number of such questions. We can pre-process the database during a learning phase, and use the resulting answers to facilitate the search process.

One way to understand this setup is to consider that the objects live in a hidden space, which can only be accessed through the aforementioned oracle. We do *not* make the assumption that the "hidden" space in which the database objects live needs to be a metric space. Using the aforementioned oracle, one can retrieve for every object u in the database a sorted list of the other objects according to their distance to u. We call the position of object v in this list the *rank* of v with respect to u, and denote it by $r_u(v)$. Clearly, this relationship can be asymmetric *i.e.,* $r_u(v) \neq r_v(u)$ in general. This setup raises several new questions and issues, as any space can be described by its ranks relationships. How much does the fact that the rank of some object v w.r.t. some other object u is k, and the rank of w w.r.t. u is k' tell us about the rank of w w.r.t. v? In our work, we introduce the notion of *rank distortion* (see Section III for a rigorous definition). The rank distortion captures how closely $r_v(w)$ is related to the average $\frac{1}{n}\sum_u |r_u(v) - r_u(w)|$. Building on this concept, we introduce

[1]We provide the architecture for a practical implementation of such a system in Section VII-A.

the notion of *rank-sensitive hashing* (RSH) in Section V. Similarly to locality-sensitive hashing, we can retrieve one of the R nearest neighbors of a query point very efficiently. The hash function itself does not require any characterization of the underlying space as an input. However, the smallest value of R we can choose depends on the rank distortion. The criteria (rank distortion) we use to characterize the hidden space seems to capture how "homogeneous" that space is. It appears that the less homogeneous it is, the more difficult it becomes to search. In particular, if the rank relationship is very asymmetric, and some objects are far from every other object, the information contained about those objects in the ranks matrix is very sparse and hard to capture. We apply this idea of RSH to NN search, but we believe that this might be useful in other scenarios as well. In order to illustrate the behavior of RSH, we investigate the implications of RSH for objects randomly placed in $\Re^d$, for large values of d.

In practice, humans are not perfect oracles and consequently a few adjustments need to be made in order to make the RSH scheme practical. In particular, to improve the performance of the system and reduce the number of questions we need to ask human users, we can try and combine perceptual search based on answers given by users and automatic feature extraction based on image processing. The key idea is that we first use an image processing technique, in our case "eigenfaces" (see [1], [2]), to compute approximate similarities between images. This might sound surprising, as we motivated the need for comparison-based search by the fact that image processing performs poorly. Further, we also argued that it's extremely difficult to compute meaningful distances between images. Although image processing algorithms do not perform very well for navigation, we use this process to obtain an initial categorization of which images are similar, and most importantly it helps us isolate the "hard" questions for which we use precious human interventions *i.e.,* the questions we need to ask the human users. In particular, pairs of very similar images or very different images are likely to be mapped to close by, respectively distant, positions. Thus, we intend to use human comparisons to refine the automatically generated similarity measures, for the triples of images we are the least confident about. We investigate those issues in Section VII-A.

In the next Section (II), we briefly review the work related to nearest neighbor search, with a focus on locality-sensitive hashing and comparison-based search. Then, in Section III, we provide the necessary definitions and state the problem. In Section IV, we summarize our contributions. In Section V, we introduce RSH, and explain the theoretical aspects. Finally, we review some image processing results in Section VI, and a practical implementation of RSH in Section VII-A.

II. Related Work

The nearest neighbor (NN) problem, and many variations thereof, have been extensively studied in the literature (see for instance [3] and [4] for surveys). In particular, very efficient algorithms have been developed for specific classes of metric spaces, such as metric spaces with a low intrinsic dimension or a bounded growth factor. In [5], the authors introduce ϵ-nets, a very simple data structure for nearest neighbor search (and many other applications). The complexity of those nets depends on the doubling dimension of the underlying space. In [6], the authors present a random sampling algorithm to produce a data structure for search in growth restricted metrics. The restricted growth guarantees that a random sample will have some nice properties. In particular, by randomly selecting a small number of representatives at different scales for every object in a learning phase, one can zoom in on the nearest neighbor of a query point during the search phase. On the other hand, search when the underlying space is not necessarily a metric space appears to have very little prior work. In some sense, it is a generalization of the above problem, as any dataset can be represented by its rank relationships.

The problem of searching with a similarity oracle was first studied in [7][2], where a random walk algorithm is presented. The main limitation of this algorithm is the fact that all rank relationships need to be known in advance, which amounts to asking the oracle $O(n^2 \log n)$ questions, in a database of n objects. The authors of [8] and [7] work with a combinatorial framework for nearest neighbor search, which defines approximates inequalities for ranks analogous to the triangle inequality for distances. Their bounds depend crucially on the combinatorial disorder, represented by the *disorder constant* D of the database. The value D must be an input to the algorithms. The combinatorial disorder is a notion which captures to what extent the triangle inequality on ranks can be violated (see [7]–[9] for a more rigorous definition). In [8], a data structure similar in spirit to ϵ-nets of [5] is introduced. It is shown that a learning phase with complexity $O(D^7 n \log^2 n)$ questions and a space complexity of $O(D^5 n + Dn \log n)$ allows to retrieve the nearest neighbor in $O(D^4 \log n)$ questions, in a database of n objects. The learning phase builds a hierarchical structure based on coverings of exponentially decreasing radii[3]. We showed (see [9]) that we can improve those bounds by a factor polynomial in D, if we are willing to accept a negligible (smaller than $\frac{1}{n}$) probability of failure. Our algorithm is based on random sampling, and hence can be seen as a form of metric skip list (as introduced in [6]), but applied to a combinatorial (non-metric) framework. However, the fact that we do not have access to distances forces us to use new techniques in order to minimize the number of questions we need to ask (or ranks we need to compute). In particular, we sample the database at different densities, and infer the ranks from the density of the sampling, which we believe is a new technique. We also need to relate samples to each other when building the data structure top down. We also present what we believe is the first lower bound for our problem of searching through comparisons.

A natural question to ask is whether one can develop data structures for NN when a characterization of the underlying

[2]Our interest in this formulation arose independently from these results from an applied viewpoint in the implementation of the facebrowser system (see Section VII-A).

[3]the radius of a ball is defined as the cardinality of that ball.

space is unknown. This has been addressed in the case when the underlying metric space has low "intrinsic" dimension and one has access to metric distances in [5], [10]. In [10], it is shown that one can build a binary tree decomposition of a dataset of points in $\Re^d$, such that the diameter of the sets in the tree is reduced by a constant after a number of level that only depends on the intrinsic dimension of the data, and not d. The term intrinsic dimension either refers to the doubling dimension (also referred to as Assouad dimension) or the local covariance dimension[4]. Therefore, one can similarly ask such a natural question in our framework where we do not have access to metric distances (or they do not exist). We developed, in [9], a binary tree (hierarchical) decomposition, when the characteristics of the underlying space (disorder constant) is unknown. This extends the result of [10] to our framework, where we only have access to the underlying space through comparisons.

The approximate nearest neighbor problem consists in finding an element that is at distance at most $(1+\epsilon)d_{min}$ from the query point q, where $d_{min} = \min_i d(i,q)$. In [11], Indyk and Motwani present two algorithms for this problem. In particular, *locality sensitive hashing*, through which they obtain an algorithm with polynomial learning and query time polynomial in d and $\log n$. For binary vectors, it is remarkable that the performance of the algorithm does not depend on the dimension. A survey of results for LSH can be found in [12]. In [13], Panigrahy shows that instead of using a large number of hash tables as it is the case in the approach above, only a few can be used. These are then hashed to several randomly chosen objects in the neighborhood of the query point, and it is shown shows that at least one of them will fall into the same bucket as the nearest neighbor. The authors of [14] prove a lower bound on the parameter $\rho = \frac{\log 1/p}{\log p/P}$ for (r, cr, p, P)-locality sensitive hashing schemes. We present a new hashing scheme that is *rank-sensitive* (RSH). How efficient the scheme is depends on another property of the hidden space, its *rank-distortion*. The rank-distortion need not be an input to the algorithm, however, the performance will depend on it. We give sufficient conditions for RSH to work and demonstrate its application to NN search. We also evaluate its performance for randomly placed points in $\Re^d$ and show that its performance improves with d.

To the best of our knowledge, the notion of rank-sensitive hashing and approximate (and randomized) nearest neighbor search using similarity oracle is studied for the first time in this work. In [9], we prove a lower bound which demonstrates that our schemes are (almost) efficient.

III. Definitions and Problem Statement

In this section, we define formally the notions that we use in the rest of the paper. We consider a hidden space $\mathcal{K}$ with distance function $d(.,.)$, and a database of objects $\mathcal{T} \subset \mathcal{K}$, with $|\mathcal{T}| = n$. We do not have access to the distances between the objects in $\mathcal{K}$ directly. We can only access this space through a *similarity oracle* which for any point $q \in \mathcal{K}$, and objects $u, v \in \mathcal{T}$ returns:

$$\mathcal{O}(q,u,v) = \begin{cases} u & \text{if } d(u,q) \leq d(v,q) \\ v & \text{else} \end{cases} \quad (1)$$

For the sake of simplicity, we consider that all distances in $\mathcal{K}$ are different. Note that the objects do not need to be in an underlying metric space for this similarity oracle. We now define the notion of *rank*.

Definition 1. *The rank of u in a set $\mathcal{S}$ with respect to v, $r_v(u,\mathcal{S})$ is equal to c, if u is the c^{th} nearest object to v in $\mathcal{S}$.*

To simplify the notation, we only indicate the set if it is unclear from the context *i.e.,* we write $r_v(u)$ instead of $r_v(u,\mathcal{S})$ unless there is an ambiguity. Note that rank need not be a symmetric relationship between objects *i.e.,* $r_u(v) \neq r_v(u)$ in general. Further, note that we can rank m objects w.r.t. an object o by asking the oracle $O(m \log m)$ questions. To do so, create the ranking w.r.t. o by adding one object at a time. Observe that in order to add the $(i+1)^{th}$ object to the list, we need to ask $\log(i)$ questions. More precisely, we need to ask whether the $(i+1)^{th}$ object is closer to o than the object currently at position $i/2$. Then, we can recurse on the set new set of objects (*e.g.*, if the object to insert is closer than the $i/2^{th}$ object, select the $i/4^{th}$ object as the new "pivot"). Summing over i, the total number of questions to be asked to sort m objects is $O(m \log(m))$.

We define a rank-ball of radius r around some point x as $\beta_x(r) = \{i \in S | r_x(i) \leq r\}$. Recall that a "distance" ball is defined as $\mathcal{B}_u(r) = \{i \in S | d(u,i) \leq r\}$. Hence, if $r_x(v) = r$ and $o \in \beta_x(r)$, then $o \in \mathcal{B}_x(d(x,v))$.

We further define the rank matrix $\mathcal{R}$ where $r_{ij} = r_i(j)$, and the matrix $\mathcal{W} = \mathcal{R} + \mathcal{R}'$ (note that the matrix $\mathcal{W}$ is symmetric). For a subset $S \in \mathcal{K}$, we define its diameter $\Delta_S = \max_{i,j \in S} w_{ij}$. Let ρ_i denote the i^{th} column of $\mathcal{R}$ *i.e.,* we associate to every object $o \in \mathcal{T}$ a vector $\rho_o \in \{0, ..., n-1\}^n$, such that the j^{th} coordinate of o is given by $r_j(o)$.

We now define the *rank-distortion* of a set S as follows:

Definition 2. *We say of a set of objects S that its rank distortion function is $f : \mathbb{N}_+ \to \Re_+$, if f is monotonically increasing and if there exists $\gamma > 0$ (the* rank-distortion*) such that $\forall u, v \in S$:*

$$f(r_u(v)) \leq ||\rho_v - \rho_u||_1 \leq \gamma f(r_u(v))$$

Lemma 1. *If the function f is linear* i.e., *$f = cr_u(v)$, then the four approximate triangle inequalities are implied with $D \leq \gamma$.*

For example, for the first inequality, we have $r_x(y,\mathcal{K}) \leq ||\rho_x - \rho_y||_1/c \leq (||\rho_x - \rho_z||_1 + ||\rho_z - \rho_y||_1)/c \leq \gamma(r_z(x,\mathcal{K}) + r_z(y,\mathcal{K}))$. The proof for the other inequalities is similar.

We can define the nearest neighbor problem as follows:

Definition 3 (R-nearest neighbor problem)**.** *Given a set of objects $\mathcal{T}$ and a query point q, return one of the R objects*

[4]Set $S \subset \Re^D$ has local covariance dimension (d, ϵ, r) if its restriction to any ball of radius r has covariance matrix whose largest d eigenvalues satisfy $\sigma_1^2 + ... + \sigma_d^2 \geq (1-\epsilon)\sum_{i=1}^D \sigma_i^2$.

in $\mathcal{T}$ closest to q. In particular, if $R = 1$, return the closest object to q in $\mathcal{T}$.

We say that a hashing scheme is (r, R, p, P)-sensitive if

Definition 4. *We call a hashing scheme h, "(r, R, p, P)-rank-sensitive" if $\forall q \in \mathcal{K}, u \in \mathcal{T}$,*

$$\mathcal{P}\left[h(q) = h(u) | r_q(u, \mathcal{T}) < r\right] > p \quad \text{and}$$
$$\mathcal{P}\left[h(q) = h(u) | r_q(u, \mathcal{T}) > R\right] < P$$

Note that we should have $P < p$.

Finally, we say that a result holds **with high probability (w.h.p.)** if it hold with probability higher than $1 - \frac{1}{n}$.

IV. Contributions

One of the difficulty of searching a hidden space arises from the fact that we cannot know how transitive the rank relationship is *i.e.,* we cannot know whether the fact that A is similar to B, and B is similar to C implies that A is similar to C. This is problematic in the sense that even if the oracle tells us that A is closer to our query point than B, it does not necessarily imply that points close to A are better candidates than points close to B. In metric spaces, such a guarantee is provided by the triangle inequality. Clearly, one of the limitations of the schemes in [7], [8] and the hierarchical scheme in [9] is that we need to know the disorder constant. It might be possible to estimate the value of the disorder constant based on a sample of objects in the database. Limitations of this approach are the fact that we might considerably degrade the performance of the algorithms if the estimator is inaccurate, and that we might run into trouble if the query point does not come from the same distribution as the database points $\mathcal{T}$.

In Section V, we present a new *rank-sensitive* hash function with many potential applications. The idea of rank-sensitive hashing is that by computing many times a hash function drawn at random, similar objects will be assigned the same hash value more frequently than dissimilar objects. The performance of the rank-sensitive hashing scheme depends on the rank-distortion of the hidden space. Instead of capturing how "transitive" the rank relationship is, the rank disorder captures how the rank $r_u(v)$ relates to the average rank *i.e.,* $\mathbb{E}\left[|r_j(v) - r_j(u)|\right]$. In other words, if we picked an object x at random, and sorted all other objects w.r.t. this object, how would $|r_x(v) - r_x(u)|$ relate to $r_u(v)$? If $r_u(v)$ can be approximated by a function f of $\mathbb{E}\left[|r_j(v) - r_j(u)|\right]$), then we can exploit this fact to separate points close to q and points far from q.

Theorem 1. *Given a set of objects S with rank-distortion function f, and rank distortion γ, there exists a function h which is $(r, (1+\epsilon)r, 1 - \frac{f(r)}{n^2}, 1 - \frac{f((1+\epsilon)r)}{n^2\gamma})$-rank-sensitive.*

A special case is when the function f is constant. Then, the behavior of the function is similar to the one observed with locality-sensitive hashing for binary vectors. One of the consequences is that we can retrieve one of the $R = (1+\epsilon)r$ nearest neighbors of a query point q in $n^{O(\frac{\gamma}{\epsilon})}$ questions. By using the output of the hash function in a different way, we can compute an overall ranking of the objects. We can then retrieve "popular" objects *i.e.,* those which are close to many other objects.

The scheme of Section V was based on the answers of a perfect similarity oracle. In practice, humans are not perfect oracles and consequently a few adjustments need to be made. In order to improve the performance of the system and reduce the number of questions we need to ask human users, one might try and combine perceptual search based on answers given by users and automatic feature extraction based on image processing.In Section VII-A, we present preliminary results for the design of comparison-based search engines.

V. Rank-Sensitive Hashing

Intuitively, and by analogy to nearest-neighbor search algorithms for Euclidean spaces, one would expect that by randomly cutting out balls in the hidden space, it is more likely that similar objects will stay together, and dissimilar objects be separated. This should be sufficient, if we can amplify this property, to allow us to efficiently search for similar objects by using an appropriately chosen hash function. Indeed, we now show how we can use this technique to develop a rank sensitive hashing scheme. The rank distortion provides us with a sufficient condition for the scheme to work. Our hash function h selects two objects u.a.r in $\mathcal{T}$ (say x_1 and x_2), and assigns values $h(u) \in \{0, 1\}$ to all objects u as follows

$$h(u) = \begin{cases} 1 & \text{if } \mathcal{O}(x_1, x_2, u) = u \\ 0 & \text{if } \mathcal{O}(x_1, x_2, u) = x_2 \end{cases}$$

Note that computing h requires asking a single question per object, and that the algorithm does not require any characterization of the space as input. The function h is $(r, (1+\epsilon)r, 1 - \frac{f(r)}{n^2}, 1 - \frac{f((1+\epsilon)r)}{n^2\gamma})$-rank-sensitive. This is the result of Theorem 1

Proof of Theorem 1: First, we compute the probability that the hash function h is different for two objects u and q.

$$\begin{aligned} p &= \mathcal{P}\left[h(u) \neq h(q)\right] \\ &= \textstyle\sum_{i,j \in \mathcal{T}} \mathcal{P}\left[h(u) \neq h(q) | x_1 = i, x_2 = j\right] \mathcal{P}\left[x_1 = i, x_2 = j\right] \\ &= \textstyle\sum_{i,j \in \mathcal{T}} \mathbf{1}_{\left\{r_{x_1}(x_2) \in \left[r_{x_1}(q), r_{x_1}(u)\right]\right\}} \mathcal{P}\left[x_1 = i, x_2 = j\right] \\ &= \textstyle\sum_{i \in \mathcal{T}} \mathbb{E}\left[\mathbf{1}_{\left\{r_{x_1}(x_2) \in \left[r_{x_1}(q), r_{x_1}(u)\right]\right\}}\right] \mathcal{P}\left[x_1 = i\right] \\ &= \frac{1}{n} \textstyle\sum_i \mathcal{P}\left[r_{x_1}(x_2) \in \left[r_{x_1}(q), r_{x_1}(u)\right] | x_1 = i\right] \\ &= \frac{1}{n^2} \textstyle\sum_i |r_i(u) - r_i(q)| \\ &= \frac{1}{n^2} ||\rho_u - \rho_q||_1 \end{aligned}$$

Hence, we have

$$\begin{aligned} \mathcal{P}\left[h(u) = h(q) | r_q(u) \leq r\right] &= 1 - \tfrac{1}{n^2} ||\rho_u - \rho_q||_1 \\ &\geq 1 - \tfrac{f(r)}{n^2} \end{aligned}$$

and

$$\begin{aligned} \mathcal{P}\left[h(v) = h(q) | r_q(v) \geq (1+\epsilon)\right] &= 1 - \tfrac{1}{n^2} ||\rho_v - \rho_q||_1 \\ &\leq 1 - \tfrac{f((1+\epsilon)r)}{n^2\gamma}) \end{aligned}$$

Where the two inequalities follow from Definition 2. ∎

A special case is when the function f is linear. Then, we obtain the following result.

Corollary 1. *We can retrieve one of the $(1+\epsilon)r$-nearest neighbors in $\mathcal{T}$ of a query point q, with constant probability, by asking $n^{O(\frac{\gamma}{\epsilon})}$ questions, where γ is the rank distortion of $\mathcal{T}$, when the rank distortion function is linear (*i.e., *$f(r) = cr$).*

Proof: The proof is analogous to the proof for locality-sensitive hashing for binary vectors provided in [11]. More precisely, for an (r, R, p, P)-rank sensitive hashing scheme, retrieving one of the R nearest neighbor of a query point q will requires $O(n^\theta)$ evaluations of the hash function. θ is defined as $\frac{\log \frac{1}{p}}{\log \frac{1}{P}}$. It can be shown that $\theta \leq \frac{1}{\frac{1+\epsilon}{\gamma}-1} = O(\frac{\gamma}{1+\epsilon})$. Indeed, the probabilities p and P take the same form as if we hashed binary vectors of dimension n^2/c, and let $r' = r$, and $(1+\epsilon')r' = (1+\epsilon)r/\gamma$. Then, $\theta \leq \frac{1}{\epsilon'}$ (see [11]). ■

Intuitively, one situation where f is roughly constant is when the underlying space is close to a line in $\Re^d$. Further, our numerics have shown that even for higher dimensions, when the underlying space is homogeneous (*e.g.,* points distributed u.a.r. in a unit box with wrap around distances), the function f is very steep for small values of $r_u(v)$ and then almost linear. An example is given in Figure 1. Note that in order

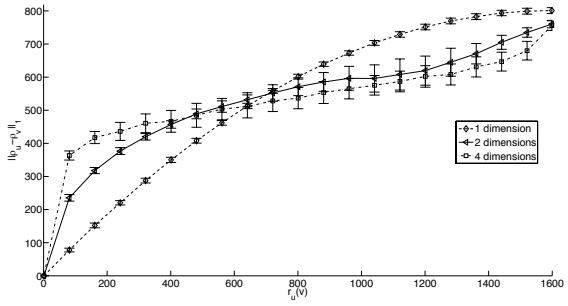

Figure 1. The hidden space consists of 1600 points distributed u.a.r. on $[0,1]^d$, where $d = 1, 2, 4$. To avoid border effects, we compute distances with wrap-around. We plot the $||\rho_u - \rho_v||$ against $r_u(v)$, for a fixed u. The results are averaged over 100 samples and the error bars correspond to the standard deviation. Note that the slope is first steep and then linear. Such a function is appropriate for RSH, as the function f increases monotonically. Further, the fact that we have a steep slope for small values of R make those spaces particularly attractive. Indeed, this implies that P decreases rapidly (so we can search for R-nearest neighbors, even for small R), and p is sufficiently large for small values of r. This example shows that for homogeneous spaces, the rank distortion function is such that we can perform RSH efficiently.

for this scheme to allow us to retrieve the nearest neighbor of a query point q efficiently, it is sufficient to have a constant gap between $f(r_q(1))$ and $f(r_q(j))$, for $j \geq 2$. In the next subsection, we investigate what happens in high-dimensional Euclidean spaces, and show that such a gap exists as long as the query point q is sufficiently close to its nearest neighbor.

A. Rank-Sensitive Hashing in high-dimensional Euclidean Spaces

In order to illustrate and understand the behavior of the RSH algorithm of Section V, we run simulations for normally distributed points in $\Re^d$. More precisely, in our setup we draw the positions of the n database objects according to $N(0, I_d)$, and choose a query point as follows. We choose one of the n points u.a.r, say $x.$, and then select the query point u.a.r. in a box of side δ centered at x. For an (r, R, p, P)-rank sensitive hashing scheme (see definition 4), the number of questions we need to ask in order to retrieve one of the R nearest neighbors grows as n^ρ, where $\rho = \frac{-\ln p}{\ln(p/P)}$ (see [11]). In the case in point, we have no information about the distances in the underlying space. Hence, we can only fix a value for ρ (*i.e.,* fix the maximum number of questions we are willing to ask in the search phase), and "hope for the best". Our aim is to determine numerically, in this setup, the value of R we can expect for a fixed value of ρ as a function of the dimension of the space d.

To do so, for several values of the dimension d, we generated 100 point constellations (standard normal distributed) and placed the query point q as explained above (with $\delta = 0.1$). For each constellations and each point i, we estimated the probability that $h(i) = h(q)$, by sampling 1000 rsh functions. For a fixed ρ, we can compute $P = p^{\frac{1+\rho}{\rho}}$. In order to estimate R, we computed the distance ν between the query point and the furthest point j with $\mathcal{P}[h(j) = h(q)] \geq P$. Then, we estimated R as $R = |\mathcal{B}_q(\nu)|$. That is, for all points i outside $\mathcal{B}_q(\nu)$, we have $\mathcal{P}[h(i) = h(q)] < P$, and for the specified value of ρ, we can retrieve one of the R nearest neighbors.

Somewhat surprisingly, R decreases with increasing dimensionality. In particular, the probability that the query point and its nearest neighbor get the same hash value goes to one, while for all other points the probability that they get the same hash value as the query point remains bounded away from 1 by a constant. In Figure 2, we plot R as a function of the logarithm base 2 of the dimension. It can be seen that for large values of d, R tends to 1. This phenomenon is remarkable, as generally high dimensionality makes search more difficult. The distance oracle seems to allow us to actually search more efficiently in high dimensional spaces than in low-dimensional spaces.

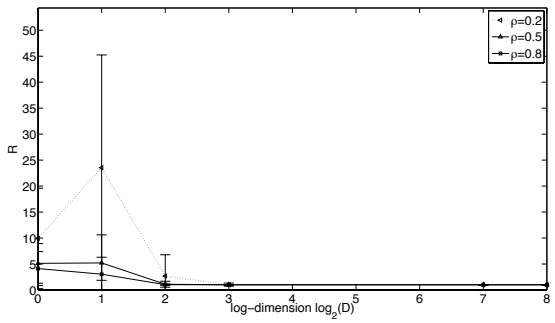

Figure 2. We plot R as a function of the dimension for 1000 points standard normally distributed in $\Re^d$. The query point is selected uniformly at random in a box of side 0.1 centered at a randomly selected point. The value of R decreases as the dimension increases. The results are averaged over 100 constellations of 1000 points. The error bars correspond to the standard deviation.

We now try and get an insight into why this is case. The following theorem has been shown by Demartines (see [15], and also [16]):

Theorem 2 (Demartines)**.** *Let $X \in \Re^d$ be a random vector with i.i.d. components: $X_i \sim \mathcal{F}$. Then,*

$$\mathbb{E}\left[||X||_2\right] = \sqrt{ad-b} + O(1/d) \quad \text{and}$$
$$var\left\{||X||_2\right\} = b + O(1/\sqrt{d})$$

where a and b are constants that do not depend on the dimension.

The theorem is valid whatever the distribution $\mathcal{F}$ of theX_i might be. Different distributions will lead to different values for a and b, but the asymptotic results remain. The theorem proves that the expectation of the euclidean norm of random vectors increases as the square root of the dimension, whereas its variance is constant and independent of the dimension. Therefore, when the dimension is large, the variance of the norm is very small compared with its expected value. Hence, for n points chosen independently and d large enough, if we apply Chebyshev's inequality and take the union bound over all n points, with an arbitrarily high probability all n norms will be $\Theta(\sqrt{d})$. Indeed, we have that $\mathcal{P}\left[|||X|| - \sqrt{ad-b}| > b\alpha\right] < \frac{1}{\alpha^2}$, for some constants a and b independent of d. By letting $\alpha = n$, where $c > 1$, we make sure that the bound holds for all points w.h.p.

We can now state the following theorem:

Theorem 3. *Let $\mathcal{T}$ be a set of n points in $\Re^d$, and let the position $x(i)$ of each object $i \in \mathcal{T}$ be chosen independently from $N(0, I_d)$. Let q be a query point. Let y denote its nearest neighbor in $\mathcal{T}$ and let $\min_{i\in\mathcal{T}} ||x(i) - q|| = ||y - q|| = \delta$. Then, (i) if $\delta = o(\sqrt(d))$, $\mathcal{P}\left[h(q) = h(y)\right] \to 1$, as $d \to \infty$, and (ii) if $\delta = \Theta(\sqrt{d})$, then $\mathcal{P}\left[h(q) = h(y)\right] \to \eta$, as $d \to \infty$, where $\eta < 1$ is a constant independent of d strictly smaller than 1.*

Proof: We have just shown (see Theorem 2) that all points lie very close to the surface of a d-dimensional sphere of radius $\Theta(\sqrt{d})$. By selecting two points x_1 and x_2 randomly for RSH, we hence roughly speaking select two points u.a.r on the surface of this sphere. Observe that for point v, w, t on the surface of the sphere, we have $||v - t||^2 > ||w - t||^2$ implies $v^T t < w^T t$. Consider a query point q at $x(q) \in \Re^d$ and a point $x(y) = x(q) + \Delta$, such that $||\Delta|| = \delta$. Let us denote by $F(q)$, $F(x_2)$ and $F(y)$ the projections of q, x_2 and y on x_1. Thus, $\mathcal{P}\left[h(q) \neq h(y)\right] = \mathcal{P}\left[F(x_2) \in [\, F(q), F(y)\,]\right]$. The normal distribution is a 2-stable distribution (see for instance [17]), and consequently $F(q) - F(y)$ is distributed as $X\delta$, where $X \sim N(0, 1)$. Moreover, by the same argument, $x_2^T x_1$ has distribution $X'\Theta(\sqrt{d})$, where $X' \sim N(0, 1)$.

Let us now consider case (i). The probability that q and y get a different value is the probability that $X'\Theta(\sqrt{d}) \sim N(0, \Theta(d))$ falls in an interval of width $Xo(\sqrt{d})$. Indeed, by Chebyshev's inequality, for d large enough, $Xo(\sqrt{d})$ will be $O(\sqrt{d})$ with an arbitrarily high probability. At the same time, the maximum of the the pdf of $N(0, \Theta(d))$ decreases as $\frac{1}{\sqrt{2\pi d}}$ with d. Hence, we have $\mathcal{P}\left[h(q) = h(y)\right] = 1 - \Theta(\frac{\delta}{\sqrt{d}}) \to 1$.

In case (ii), the probability goes to a constant. Indeed, with constant probability, $|X| > 1$ (about 0.32, according to the "68-95-99.7" rule). Hence, q and y are always at least a standard deviations of X' apart with constant probability. Further, the diameter of the d-dimensional sphere is $\Theta\sqrt{d}$, and the projection of x_2 on x_1 must lie inside this sphere. The probability that X' (which has standard deviation $\Theta\sqrt{d}$) falls in any interval of width $\Theta(\sqrt{d})$ inside the sphere is independent of the dimension. ■

Hence, there is always a constant probability that an arbitrary point in $\mathcal{T}$ and q get separated. On the other hand, the probability that the query point and its neighbor get the same hash value goes to one for very large d, if the query point is sufficiently close (*i.e.*, within $o(\sqrt{d})$) of its nearest neighbor. Consequently, in very high dimensional spaces with normal distribution of points, the nearest neighbor can be retrieved in as little as $O(\log n)$ questions, as long as its distance to the query point is $o(\sqrt{d})$. Conceptually, in high dimensional spaces, all nodes will start looking "similar", except the nearest neighbor, which will lie relatively extremely close to the query point.

VI. FACEBROWSING: NAVIGATING AN IMAGE DATABASE

The design of a system that will return the image most similar to a query image, among all the images in a database, for all users and queries is a central and challenging task for next generation search engines. This is simply due to the fact that humans have different perceptions of similarity, and hence, even with infinite training, we could probably not preprocess the database such that efficient search is possible for all users. Nevertheless, we can try to speed up the search process, such that for most users the queried face will be one of the first answers provided by the system. At least, we can expect to design a system that performs substantially better than exhaustive search. Ideally, our system should present the images to the user sorted by relevance, analogously to what a web search engine does for websites. Hopefully, exhaustive searches will be extremely rare, and most searches will result in a rapid retrieval of the desired image or at least of a similar image. As mentioned above, in a real system, we do not have access to an exact distance oracle, which given a reference pair and a query image, returns the reference image closest to the query image. Humans will sometimes (or even often) disagree. If, for example, we fixed a reference pair (say images A and B) and a query image, and asked 100 humans to act as oracles, it is likely that some fraction f would answer A, and a fraction $1-f$ would answer B. If f were close to 0 or close to 1, we could be pretty confident about the answer another random user would give to this question. On the other hand, if f were close to 0.5, A and B would probably both be as similar to the query image. In this Section, we briefly review some basic image processing and face recognition techniques. These techniques can be used to complement the RSH scheme introduced in Section V, as explained in Section VII-A. Then, in Section VII-A, we propose a practical implementation of RSH.

A. Basics

Typically, images are stored as three $a \times b$ matrices R, G and B, where $P = ab$ is the number of pixels in the images. The matrices R, G and B represent the red, green and blue intensities of every pixel. Each entry in those matrices is an 8 bit vector[5], representing a value between 0 and 255. A pixel with intensities $(0, 0, 0)$ is completely black, while a pixel with intensities $(255, 255, 255)$ is completely white. As suggested by the authors of [18], who provide an extensive comparison of face processing technique, we pre-process images by resizing them if necessary, normalizing intensities and converting them to gray scale. Assume that a pixel i has intensities (r_i, g_i, b_i). Then, intensity normalizations consists in modifying, $\forall i$, the intensities as follows:

$$(r_i, g_i, b_i) \quad \leftarrow (\tfrac{r_i}{r_i+g_i+b_i}, \tfrac{g_i}{r_i+g_i+b_i}, \tfrac{b_i}{r_i+g_i+b_i})$$

A its name indicates, this techniques aims at mitigating the effects of different brightnesses in different colors channels. Conversion to gray scale is simply done by replacing the RGB triple for every pixel by a single value, which is a weighted sum of the three color intensities. Different conversion formulas exist. A common conversion is as follows:

$$I_i = 0.3r_i + 0.59g_i + 0.11b_i$$

Note that the weights are based on experimental results. The gray scale image can now be stored as a single $a \times b$ matrix I. This image can then be converted to a vector Λ, simply by concatenating the rows of the matrix. In order to obtain a low-dimensional representation of the images, we project these vectors on so called eigenfaces. This technique is explained below. That is, we try to represent every image as a combination of "basis vectors" in the image space.

B. Eigenfaces

In this section we give a brief explanation of the eigenface method of face recognition, while referring the reader to Turk and Pentland [1], [2] for more detailed explanations. We chose this approach based on the comparative study in [18], and on the fact that when dealing with images of faces taken for large organizations, we can expect the image format to be fairly standard.

We compute the covariance matrix C, of facial images from a set of M *training* images in vector form $\{\Lambda_1, \Lambda_2, ..., \Lambda_M\}$ as follows:

$$\begin{aligned} \Psi &= \tfrac{1}{M}\textstyle\sum_{i=1}^{M} \Lambda_i \\ \Phi_i &= \Lambda_i - \Psi \\ A &= [\Phi_1 \Phi_2 ... \Phi_M] \\ C &= \tfrac{1}{M}\textstyle\sum_{i=1}^{M} \Phi_i \Phi_i^T \\ &= AA^T \end{aligned}$$

The eigenvectors and eigenvalues of this covariance matrix are calculated using standard linear methods and the M' eigenvectors with the highest eigenvalues chosen to formulate the projection matrix u. A face-key ω *i.e.,* an image vector projected into the face space, can then be produced by the following equation for an image Λ:

$$\omega_j \quad = u_j^T(\Lambda - \Psi) \text{ for } j = 1 \text{ to } M$$

These face-keys can than be compared using the Euclidean distance measure.

Figure 3. The 18 first eigenfaces for a set of 20 training images of size 82×65.

VII. System

Assume that we are given a database $\mathcal{T}$ of n images. We intend to map each of these images to a RSH vector in $[0, 1]^{k_n}$. Intuitively, k_n should be roughly $\lceil \log(n) \rceil$, so that a different binary vector can be assigned to every image. However, each of the entries corresponds to an "approximation" of the value of an RSH function as presented in Section V. Every coordinate should tell us, for a given reference pair A_j, B_j, if the image under consideration is inside or outside the ball $\mathcal{B}_{A_j}(d(A_j, B_j))$. It is an approximation, because neither image processing nor human training can provide us the exact value of the hash function. Rather, we will consider that a value larger to 0.5 indicates that the image is inside the ball, and a value smaller than 0.5 that it is outside. The closer the value is to 0.5, the less confident we are about it. This concept is illustrated in Figure 4.

The *learning* phase works as follows. For every image, as mentioned above, we intend to store a binary vector of length k_n, where each of the k_n entries corresponds to an approximation to the answer which the similarity oracle presented in Section V would give for a given reference pair of images. Hence, we choose k_n pairs of images u.a.r among the $\frac{n(n-1)}{2}$ possible distinct pairs to be the reference pairs for RSH. Obviously, the pairs are the same for all images and remain fixed once chosen. Every image is pre-processed when added to the database. First, an image I is resized, so that all images have the same size. Then, we normalize the intensity, as explained above in Section VI. Finally, the image is converted to grayscale, and its face-key is computed, as explained in Section VI-B. Then, for each of the k_n RSH

[5]the resolution could vary, but for the sake of simplicity, we make the assumption that there are 255 levels of intensity.

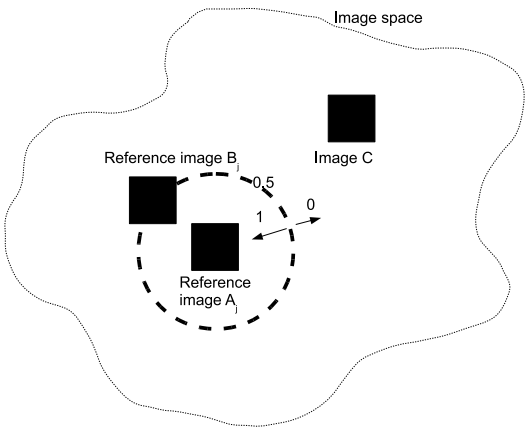

Figure 4. We illustrate how the j^{th} coordinate of the RSH vector for an image C is computed. The j^{th} coordinate corresponds to the reference pair A_j, B_j. If image C is inside the ball $\mathcal{B}_{A_j}(d(A_j, B_j))$, the hash value is 1, otherwise it is 0. However, neither image processing nor human training can tell us exactly where image C lies. Hence, we compute an initial value based on image processing, which lies between 0 and 1. The closer the value is to 0.5, the more uncertain we consider it to be. Later, this value will be refined through human training. If the image lies clearly outside the ball, most human should agree and training will push the value down to 0 (and vice-versa if it is clearly inside the ball).

reference pairs (A_j, B_j) and image I, we compute the ratio

$$r_{I,j} = \frac{||\omega_{A_j} - \omega_{B_j}||_2}{||\omega_{A_j} - \omega_{B_j}||_2 + ||\omega_{A_j} - \omega_I||_2}$$

where ω_x denotes the face key of image x (see Section VI-B). Clearly, if this ratio is close to 0, then the image I is outside the ball $\mathcal{B}_{A_j}(d(\omega_{A_j}, \omega_{B_j}))$, and if it is close to 1, we can be confident that it is inside. In Figure 5, we summarize this process. Human users come into play when a ratio is close

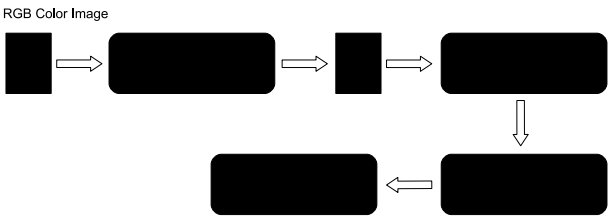

Figure 5. The block diagram for image insertion

to 0.5. For all triples for which this ratio $r_{I,j}$ is such that $\max\{r_{I,j}, 1 - r_{I,j}\} < \theta$, where $\theta \in [0.5, 1]$ is some threshold value, we can ask human users to refine it. In particular, consider a triple A_j, B_j, I, for which $r_{I,j}$ is close to 0.5. Assume that at some point in time, t users have acted as oracle for this triple, and a new user answers the question. Then, we simply recompute the ratio as

$$r_{I,j} \leftarrow \frac{tr_{I,j}}{t+w} + \frac{w}{t+w} r_{I,j}$$

and update $t \leftarrow t + w$. That is, the current value of the ratio is the average of the users opinions given so far. An arbitrary weight *e.g.,* a weight of 1, can be given to the initial ratio obtained with image processing, and an arbitrary weight of w to clicks. Straightforwardly, the higher w, the more importance we give to clicks.

In the search phase, we repeat the same process by asking the user the same k_n questions (corresponding to the k_n reference pairs), but for the query point. That is, for all of the k_n balls, we want to know whether the query point lies inside or outside. This time, we get a binary vector of answers $a \in \{0, 1\}^{k_n}$. This is because in the search phase, a user will be shown every reference pair only once. Depending on which of the reference images the user clicks, a 1 or a 0 is stored. Finally, we sort the objects in the database according to $||a - r_I||_1$, $\forall I \in \mathcal{T}$, where r_I denotes the vector $(r_{I,1}, r_{I,2}, ..., r_{I,k_n})$. The user is presented the results from the most similar image to the least similar image. Potentially, one could improve the results by adding tags to images, and using them to break ties among images at close distances. Further, in order for the system to work well, different users should compare images roughly in the same way. In other words, comparisons should be based on the same criteria, and the weight given to each criteria should be the same. For instance, assume that the reference pair consists of an asian woman, and a Caucasian man, and that the query image is an asian man. Depending on the user, both reference images could be considered to be the most similar one to the query image. In order to mitigate this undesirable effect, we give the users an order on the criteria in our experimental setup, which they must use to break ties. For instance, gender always comes before ethnicity, and consequently the right answer above would be the caucasian man. We now present a few of these experimental results.

A. Experimental Results

We have implemented a full-fledged test system in Matlab, in order to obtain experimental results on perceptual search based on RSH (a screenshot of the training phase is shown in Figure 6). We tested perceptual search based on comparisons on a database of 110 faces, of different gender, ethnicity, age, etc. The images we used to test our implementation come from the Indian Face Database [19], a collection of faces of the University of Essex [20], and a collection of pictures of friends and colleagues. We set $k_n = 10$, $\theta = 0.9$ and w, the weight of a click, to 10. The Eigenfaces were computed based on 20 training images. Seven users were each asked to search for two query images selected u.a.r in the database. This amounts to 20 clicks per user for searches. Additionally, they were asked to make 200 training clicks. We then computed the rank of the query image in the search results, after various amounts of training clicks. That is, we ranked the images with respect to their l_1 distance to the search vector as explained above, and looked at the rank of the query image in the result list. Note that the training clicks of all users were randomly mixed and shuffled. Clearly, if no image processing nor training were

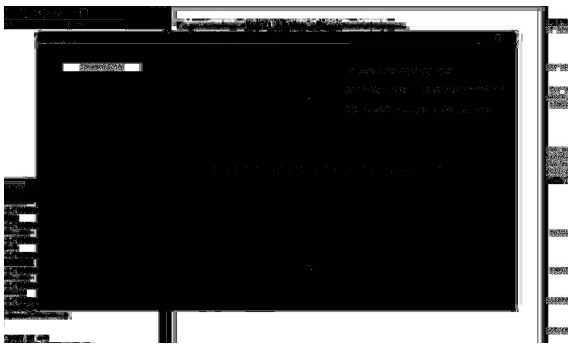

Figure 6. A screen shot of the training phase in the Matlab implementation.

done, the average position of the query image in the result list would be in the middle, so roughly $n/2$. In Figure 7, we plot the average percentage of the database that must be inspected before the query image is found (*i.e.,* the relative rank of the query image in the result list), as a function of the number of training clicks. It can be seen that the curve converges to an average value of roughly 20%. This means that on average the searched image (query image), will appear among the 22 first results presented to the user. Clearly, without any processing, this average would be close to 50% *i.e.,* 55 images. Image processing alone (without training) improves over exhaustive search by roughly 10%.

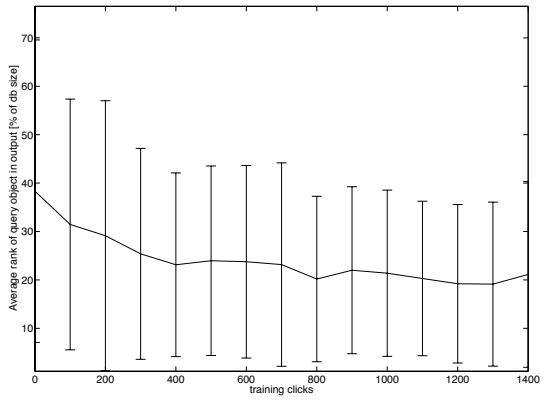

Figure 7. We show the effect of human training on RSH results. The plot shows the average percentage of the database that must be inspected before the query image is found, as a function of the number of clicks. The error bars correspond to the standard deviation

Collecting clicks from users is not necessarily an easy task, which explains the relatively small number of clicks and users. Nevertheless, the results in Figure 7 show that training has a clear impact on the quality of the search. Indeed, training improved the quality of the results by roughly 20% over perceptual search based uniquely on image processing. In itself, this result is interesting as it shows that different users agree more often than not. Indeed, training clicks from different users could have canceled each other out. After observing and talking to users, it appeared that most users find it rather simple to classify faces based on criteria such as ethnicity, age or gender. This fact certainly explains that training led to an improvment, as such distinctions are easy to make for humans, but difficult for computers. We also believe that this is how humans "compute" similarity, by tagging images and matching them whenever possible. On the other hand, when all faces are roughly similar and come from the same "group" of people, classification becomes much more difficult, and seems to be based on the face shape, and criteria specific to each user. Hence, the quality of the search results also depends on the composition of the database. If the faces are very clustered in different group, search will be easier than if they all belong to the same group. Note also that in this experiment we tried to retrieve a specific image, and not any image that is similar to that image. This choice was motivated by the fact that we wanted to obtain sound numerical results. A closer look at the search results after all training clicks (see Figure 8) shows us that roughly 30% of the time, the searched image will be among the 10 first results (in our database of 110 images), and roughly 50% of the time among the 15 best results. On the other hand, 20% of the searches will produce

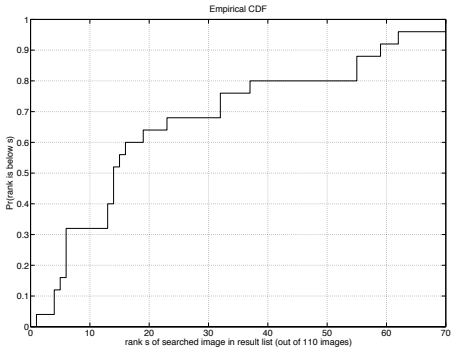

Figure 8. CDF of the quality of search results after training. It can be seen that roughly 50% of images appear among the 15 first search results. Conversely, there is a fraction of the images which are hard to classify and have rank higher than 50.

very bad results *i.e.,* the searched image will not be among the 50 first images shown to the user. This result tends to indicate that there is some inherent slack in the process. In other words, for most of the images the RSH vector is meaningful and corresponds to some extent to human perception. On the other hand, for a part of the images, the process fails and they are very hard to retrieve.

B. Web Platform

In addition to the experimental results exposed above, we have implemented in the framework of several student project, a web platform for comparison based search. The current version of the website can be found at `http://ipg.epfl.ch/~dtschopp/facebrowser/`.

The underlying mechanisms and algorithms are the same as for the matlab implementation. Additionally, we offer the users the possibility to insert images, to tag them, and to add contact details for the person on the images. When new images are inserted, new reference pairs are automatically selected. Users can also navigate the database, either by moving from one image to one of the most similar images, or by moving to a completely different image (we added links to images with exponentially spaced ranks *e.g.,* to image with ranks $2, 4, 8, 16, 32, ...$ w.r.t. to the current image). The website is written in PHP, html and mySQL.

VIII. Conclusion

In this paper, we addressed the nearest-neighbor retrieval problem in an original setup. Indeed, in contrast to most existing formulations, we asked whether the database can be searched efficiently if its distance information can only be accessed through a similarity oracle, and the underlying objects need not be in a metric space. This question was inspired by our project to implement a comparison-based, human assisted search engine for image databases. In particular, the oracle is motivated by a human user who can make comparisons between objects but not assign meaningful numerical values to similarities between objects. This raises interesting questions on what are important properties of the rank relationships and how to design efficient algorithms. Based on the intuition that the average performance must be considerably better than suggested by the results derived in the combinatorial framework introduced in [7], [8], we introduced and designed Rank-sensitive hash functions which enable (approximate) nearest neighbor search in a manner similar to locality sensitive hashing. The performance of RSH depends on an new average characterization, rank distortion, which we introduced in [9]. We believe that ideas of searching through comparisons form a bridge between many well known search techniques in metric spaces to perceptually important (non-metric spaces) situations, and could lead to innovative practical applications. We have also presented the architecture for a systems that implements image search based on rank sensitive hashing (RSH). The system allows users to retrieve an image in a database on average more rapidly than if they had to search the database exhaustively. Every image in the database gets in some sense labeled automatically through image processing. Then, the labels we are most uncertain about are refined by human users. Those labels can then be used to compute distances between images. Search is entirely based on a small sequence of questions, which is processed to output the label of the query image. In contrast to classical labeling schemes, we do not have to explicitly define labels. While these implementations are in early stages, we believe that this approach can be greatly improved. Currently, we are very conservative in the training phase, as a training click only modifies one coordinate of one image. Though this can influence the outcome of a large number of searches, the impact remains small (an image can gain or lose one rank in the search results). Another possible option would be to use the training clicks to learn a distance function, and use this distance function to iteratively recompute the coordinates of the images. Most interestingly maybe, we could use active learning techniques to select the reference pairs for RSH. In particular, as the number of images in the database increases, we need to select new reference pairs. In that case, we could select those pairs based on the clicks collected so far instead of choosing them u.a.r. One possibility might be to select pairs that have not been well discriminated, and consequently lie in a region of the hidden space where objects are not easily classifiable. Obviously most of these ideas can be combined with search based on other tags and features of the images as well.

In summary, we believe that the idea of human-assisted comparison-based search/navigation of image databases is a promising direction for image search engines.

References

[1] M. Turk and A. Pentland, "Eigenfaces for recognition," *Journal of Cognitive Neuroscience*, vol. 3, pp. 72–86, 1991.

[2] ——, "Face recognition using eignefaces," in *CVPR*, 1991, pp. 586–591.

[3] K. Clarkson, "Nearest-neighbor searching and metric space dimensions," in *Nearest-Neighbor Methods for Learning and Vision: Theory and Practice*, G. Shakhnarovich, T. Darrell, and P. Indyk, Eds. MIT Press, 2006, pp. 15–59.

[4] P. Indyk, "Nearest neighbors in high-dimensional spaces," in *Handbook of Discrete and Computational Geometry*, 2nd ed., J. E. Goodman and J. O'Rourke, Eds. CRC Press, 2004.

[5] R. Krauthgamer and J. R. Lee, "Navigating nets: simple algorithms for proximity search," in *SODA*, 2004, pp. 798–807.

[6] D. R. Karger and M. Ruhl, "Finding nearest neighbors in growth-restricted metrics," in *STOC*, 2002, pp. 741–750.

[7] N. Goyal, Y. Lifshits, and H. Schutze, "Disorder inequality: A combinatorial approach to nearest neighbor search," in *WSDM*, 2008, pp. 25–32.

[8] Y. Lifshits and S. Zhang, "Combinatorial algorithms for nearest neighbors, near-duplicates and small-world design," in *SODA*, 2009, pp. 318–326.

[9] D. Tschopp and S. Diggavi, "Approximate nearest neighbor search through comparisons," 2009. [Online]. Available: http://www.citebase.org/abstract?id=oai:arXiv.org:0909.2194

[10] S. Dasgupta and Y. Freund, "Random projection trees and low dimensional manifolds," in *STOC*, 2008, pp. 537–546.

[11] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *STOC*, 1998, pp. 604–613.

[12] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," *Communications of the ACM*, vol. 51, no. 1, pp. 117–122, 2008.

[13] R. Panigrahy, "Entropy based nearest neighbor search in high dimensions," in *SODA*, 2006, pp. 1186–1195.

[14] R. Motwani, A. Naor, and R. Panigrahy, "Lower bounds on locality sensitive hashing," in *SCG*, 2006, pp. 154–157.

[15] P. Demartines, "Analyse de données par réseaux de neurones auto-organisés," 1994.

[16] D. François, V. Wertz, and M. Verleysen, "The concentration of fractional distances," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 7, pp. 873–886, 2007.

[17] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *SCG*, 2004, pp. 253–262.

[18] T. Heseltine, N. Pears, J. Austin, and Z. Chen, "Face recognition: A comparison of appearance-based approaches," in *DICTA*, vol. 1, 2003, pp. 59–68.

[19] V. Jain and A. Mukherjee, "The indian face database," 2002, `http://vis-www.cs.umass.edu/~vidit/IndianFaceDatabase/`.

[20] L. Spacek, "Face recognition data of the university of essex, uk," 2002, `http://cswww.essex.ac.uk/mv/allfaces/index.html`.