On Locality-sensitive Indexing in Generic Metric Spaces

David Novak
Masaryk University
Brno, Czech republic
david.novak@fi.muni.cz

Martin Kyselak
Masaryk University
Brno, Czech republic
xkyselak@fi.muni.cz

Pavel Zezula
Masaryk University
Brno, Czech republic
zezula@fi.muni.cz

ABSTRACT

The concept of Locality-sensitive Hashing (LSH) has been successfully used for searching in high-dimensional data and a number of locality-preserving hash functions have been introduced. In order to extend the applicability of the LSH approach to a general metric space, we focus on a recently presented Metric Index (M-Index), we redefine its hashing and searching process in the terms of LSH, and perform extensive measurements on two datasets to verify that the M-Index fulfills the conditions of the LSH concept. We widely discuss "optimal" properties of LSH functions and the efficiency of a given LSH function with respect to *kNN* queries. The results also indicate that the M-Index hashing and searching is more efficient than the tested standard LSH approach for Euclidean distance.

Categories and Subject Descriptors

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

Keywords

locality-sensitive hashing; metric space; similarity search; approximation; scalability

1. INTRODUCTION

The similarity indexing and searching is a wide research area that gives space to various different paths leading to a solution. A decade ago, Gionis et al. [13] proposed to apply the concept of Locality-sensitive Hashing (LSH) to the search in high-dimensional data and thus started a new research stream. Since then, the concept of LSH-based indexing was further developed and a number of hash functions that preserve the locality of the data have been defined. Current LSH functions are typically defined for specific data types together with given distance functions.

One of other strong streams in this field is indexing based purely on metric properties of the data space [17]. Majority of the structures of this kind are tree-based hierarchical indexes. Recently, M-Index [16] was introduced as a metric-based hash index that can be used both for efficient precise and approximate similarity search. Intuitively, the hashing function defined by the M-Index is locality-preserving and it's approximate algorithm follows the schema of multi-probe LSH searching [14].

In this paper, we focus just on locality-sensitive hashing in generic metric spaces using the M-Index. More specifically, we use the theoretical fundamentals of the LSH concept in order to rigorously study validity of the above-mentioned intuitive statement. The main contributions can be summarized as follows:

- we redefine the indexing and searching process of the M-Index in the terms of the LSH approach;
- we perform extensive measurements on two real-life datasets to verify that the M-Index fulfills the conditions of the LSH concept;
- we put under the same test a standard LSH technique for Euclidean distance and compare it with the M-Index;
- we widely discuss optimality of locality-preserving hash functions and we introduce a new relation that can shed light on the efficiency of a given LSH function with respect to *kNN* queries.

The paper is further organized as follows: In Section 2, we summarize the concept of LSH, as it developed during the last decade. In Section 3, we introduce the M-Index as an LSH technique applicable to a generic metric space. The key Section 4 contains majority of the contribution as specified above, and the work concludes with future work directions in Section 5.

2. LOCALITY SENSITIVE HASHING

The application of Locality Sensitive Hashing (LSH) for indexing and searching high-dimensional data was first proposed by Gionis et al. [13]. The basic idea of this approach is to construct several hash functions to map each (multi-dimensional) point v from domain $\mathcal{U}$ onto a scalar and for each of these hash functions build one hash index on the set of indexed objects $S \subseteq \mathcal{U}$.

The fundamental property of LSH functions $h : \mathcal{U} \rightarrow M$ (where M is a set of buckets) is that the close objects tend to be hashed to the same bucket (they have a higher probability of colliding), whereas objects distant from each other tend to be hashed to different buckets. This property is formally defined as follows [13]:

Definition 1. *Let $d : \mathcal{U} \times \mathcal{U} \to \mathbb{R}$ be a distance function and $\mathcal{B}(u, r) = \{v \in \mathcal{U} | d(u, v) \leq r\}$ for any $u \in \mathcal{U}$. A family $\mathcal{H}$ of functions $h : \mathcal{U} \to M$ is called (r_1, r_2, p_1, p_2)-sensitive for d if for any $u, v \in \mathcal{U}$:*

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

In addition, the LSH function family should satisfy the inequalities $r_1 < r_2$ and $p_1 \gg p_2$.

In practice, achieving high probability of hash collisions for close objects is typically not very difficult; on the other hand, it might be tricky to guarantee that distant objects are hashed to different buckets. Therefore, several hash functions might be grouped to form a new hash function family. Formally, we define [13] a function family $\mathcal{G} = \{g : \mathcal{U} \to M^K\}$ such that $g(u) = (h_1(u), \ldots, h_K(u))$, where $h_i \in \mathcal{H}$. This concatenation technique thus amplifies the gap between p_1 and p_2.

When building an indexing structure, L functions $g_1, \ldots, g_L$ are selected at random from family $\mathcal{G}$. Then, each object $u \in S$ is hashed into $g_j(u)$ buckets, for all $j = 1, \ldots, L$. When processing a k-nearest neighbors query $kNN(q, k)$, buckets $g_1(q), \ldots, g_L(q)$ are accessed and all objects from these buckets form a candidate set C (duplicated objects are removed). For each $u \in C$, distance $d(q, u)$ is computed and the k most similar objects to q are returned.

As hash $g(q)$ consists of components $h_1(q), \ldots, h_K(q)$, the buckets corresponding to the hash keys that differ from $g(q)$ by ± 1 in one or several these K components are also likely to contain objects near the query point q. This *multi-probe* technique [14] can improve the way of forming the candidate set C and, at the same time, reduce the number of hash tables.

Specific LSH functions were proposed, for instance, for the following distance functions: ℓ_p distance [10] (Minkowski distance of order p), Hamming distance [13] (on binary vectors), Jaccard coefficient [6, 7] (for measuring the similarity of sets), or Arccos [8] (the angle between two vectors). In the following, we describe the LSH function for ℓ_p distances.

LSH Function for ℓ_p Distances

A family of LSH functions for real vectors with ℓ_p distance (denoted as $||\boldsymbol{v_1} - \boldsymbol{v_2}||_p$, $\boldsymbol{v_1}, \boldsymbol{v_2} \in \mathbb{R}^d$) for $p \in (0, 2]$ was proposed in [10]. The technique utilizes the properties of p-stable distributions [18]. First, a random vector $\boldsymbol{a} \in \mathbb{R}^d$ is generated such that its each entry is chosen independently from a p-stable distribution. Then, a dot product $\boldsymbol{a} \cdot \boldsymbol{v}$ projects each vector $\boldsymbol{v} \in \mathbb{R}^d$ to the real line. It follows from p-stability that, for two vectors $\boldsymbol{v_1}, \boldsymbol{v_2} \in \mathbb{R}^d$, the distance between their projections is distributed as $||\boldsymbol{v_1} - \boldsymbol{v_2}||_p X$ where X is the p-stable distribution.

If we "chop" the real line into equi-width segments of appropriate size w and assign hash values to vectors based on which segment they are projected onto, then it is intuitively clear that this hash function will be locality preserving in the sense described above.

Formally, each hash function $h_{\boldsymbol{a},b}(\boldsymbol{v}) : \mathbb{R}^d \to M$ maps vector $\boldsymbol{v}$ onto a set of integers. Each hash function in the family is specified by a choice of random vector $\boldsymbol{a} \in \mathbb{R}^d$ and a scalar $b \in \mathbb{R}$, where $\boldsymbol{a}$ has its entries chosen independently from a p-stable distribution and b is chosen uniformly from the range $[0, w)$. For given $\boldsymbol{a}, b$, the hash function $h_{\boldsymbol{a},b}$ is defined as $h_{\boldsymbol{a},b}(\boldsymbol{v}) = \lfloor \frac{\boldsymbol{a} \cdot \boldsymbol{v} + b}{w} \rfloor$.

Specifically, for ℓ_2 (Euclidean distance), the entries of vector $\boldsymbol{a}$ can be chosen, e.g. from the Gaussian (normal) distribution, which is 2-stable.

E²LSH Package

The package E^{2}LSH (Exact Euclidean LSH) [1] utilizes the LSH approach described above to solve the randomized version of the r-near neighbor problem ($r > 0$) defined as follows: For a query point $q \in \mathcal{U}$, find all objects $v \in S$ such that $||q - v||_2 \leq r$. The main aim of this implementation is to propose a self-tuning algorithm to compute the optimal values of parameters K and L and build up a suitable LSH indexing structure.

To minimize the size of the candidate set while preserving a reasonable recall level (percentage of precise answer), the algorithm tries to find the parameter values in order to:

- keep K as high as possible (the larger the value of K, the smaller the number of objects in one bucket),
- keep L as low as possible (high values of L increase the search costs and result in large data replication).

In general, the self-tuning process computes these parameters as a function of the data set S, the radius r, and a set of query points. Therefore, every time one of these properties changes significantly, the parameters K and L should be recomputed and the indexing structure rebuilt in order to maintain the performance and recall level.

LSH Forest

The structure LSH Forest [4] was designed to avoid the need of tuning the L and K parameters and reconstructing the whole indexing structure. Instead of assigning hash values of fixed-length $g(u) = (h_1(u), \ldots, h_K(u))$ to objects $u \in S$, the hashes are of *variable length*; specifically, each object's hash value is made long enough to ensure that every point from S has a distinct value. Then a (logical) prefix tree (called LSH tree) can be constructed on the set of all hash values, with each leaf corresponding to an object. The *LSH forest* then consists of L LSH trees. See the original paper for more details [4].

PP-Index

PP-Index [11] is a recent metric-based technique for approximate search that is based on permutation of pivots (or their prefixes). Its keystone is a main memory tree similar to the M-Index cluster tree as introduced below. In contrast to the M-Index, this approach does not explicitly define any hash function and the author does not study the PP-Index as an LSH technique.

3. METRIC-BASED LSH WITH M-INDEX

The Metric Index (M-Index) [16] is an indexing structure for similarity search that is based on hashing principles. In the following, we introduce the M-Index approach to data indexing and searching in terms of the concept of LSH.

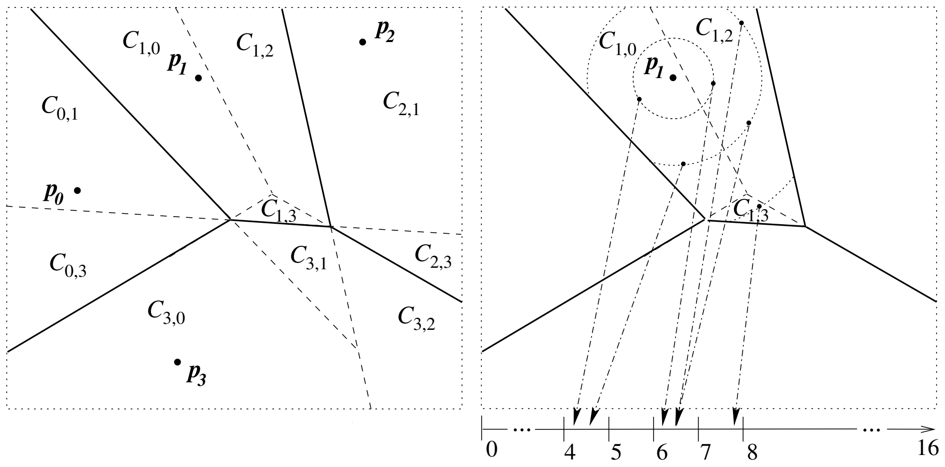

Figure 1: Principles of a two-level M-Index: partitioning (left) and mapping (right).

3.1 Fundamentals

The M-Index treats the data purely as a *metric space* $\mathcal{M} = (\mathcal{U}, d)$, where $\mathcal{U}$ is a *domain* of objects and d is a total *distance function* $d : \mathcal{U} \times \mathcal{U} \longrightarrow \mathbb{R}$ satisfying metric postulates (non-negativity, identity, symmetry, and triangle inequality) [17]. In the following, we assume that this distance function is normalized: $d : \mathcal{U} \times \mathcal{U} \longrightarrow [0, 1)$. In practice, this assumption can be always achieved by dividing the distances by a constant greater than the maximal value of d expected in the data domain.

The M-Index is designed to process standard metric-based similarity queries, in particular the *range query* $R(q, r)$, and *nearest neighbors query* $kNN(q, k)$. These queries can be either processed precisely, which is demanding, or in an approximate manner. In the latter case, the M-Index shares many principles with the locality-sensitive techniques.

3.2 M-Index Mapping

The M-Index defines a *universal mapping schema* from metric space $\mathcal{U}$ to a numeric domain. This mapping uses a fixed set of n reference objects (pivots) $\{p_0, p_1, \ldots, p_{n-1}\}$ and has the ability to preserve locality of the data. In order to formalize this schema, we need a preliminary definition: For an object $v \in \mathcal{U}$, let $(\cdot)_v$ be any permutation of pivot indexes $\{0, 1, \ldots, n-1\}$ such that

$$d(p_{(0)_v}, v) \leq d(p_{(1)_v}, v) \leq \cdots \leq d(p_{(n-1)_v}, v).$$

In other words, sequence $p_{(0)_v}, \ldots, p_{(n-1)_v}$ is ordered with respect to distances between the pivots and object v. The M-Index recursively partitions the data space in a Voronoi-like manner: On the first level, each object $v \in \mathcal{U}$ is assigned to its closest pivot p_i – clusters C_i are formed in this way (in other words, $(0)_v = i$ for all objects $v \in C_i$). On the second level, each cluster C_i is partitioned into $n - 1$ clusters by the same procedure using set of $n - 1$ pivots $\{p_0, \ldots, p_{i-1}, p_{i+1}, \ldots, p_{n-1}\}$ creating clusters $C_{i,j}$ where j is index of the second closest pivot to objects in cluster $C_{i,j}$, i.e. $(1)_v = j$. Figure 1 (left) shows an example of the M-Index partitioning for two levels. This partitioning process is repeated l-times in M-Index with l levels, for any l: $1 \leq l \leq n$.

The l-level M-Index further defines a mapping of the data space to a numeric domain $h : \mathcal{U} \longrightarrow [0, n^l]$ where the integral part of $h(v)$, $v \in \mathcal{U}$ results from a numbering schema of the clusters; specifically, cluster $C_{i_0,i_1,\ldots,i_{l-1}}$ is assigned number:

$$cluster(C_{i_0,i_1,\ldots,i_{l-1}}) = \sum_{j=0}^{l-1} i_j \cdot n^{(l-1-j)} \ . \qquad (1)$$

The fractional part of $h(v)$ is the distance between the object v and its closest pivot $d(p_{(0)_v}, v)$. Altogether:

$$h(v) = cluster(C_{(0)_v,(1)_v,\ldots,(l-1)_v}) + d(p_{(0)_v}, v) \ . \qquad (2)$$

Figure 1 (right) sketches this M-Index mapping principle (only for clusters $C_{1,j}$) – see the M-Index paper [16] for further details. The M-Index as described so far has a *static* partitioning and mapping for a given level l. Since neither the data distribution nor the space partitioning are uniform, the M-Index has a *dynamic* variant that further partitions only clusters that exceed certain data volume limit. In this case, the M-Index maintains a dynamic *cluster tree* to keep track of actual depth for individual clusters. The formula for $h(v)$ calculation is modified to take into account actual tree-depth [16] – the tree has an a priori given maximal level $1 \leq l_{\max} \leq n$. Such key-assignment approach can be considered a variant of *extensible hashing* [12]. The schema of this tree-like structure for $l_{\max} = 3$ is sketched in Figure 2.

Note that the M-Index clusters naturally correspond to LSH buckets. Taking the number of pivots n and the level $l_{\max}$ as parameters, such hash functions h form a family $\mathcal{H}^n_{l_{\max}}$ of M-Index mapping functions. Individual members

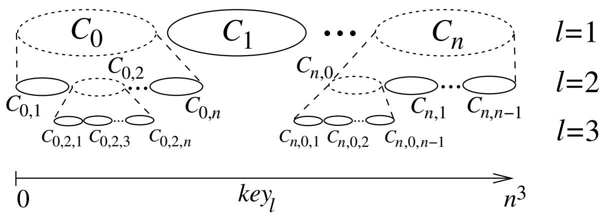

Figure 2: Dynamic cluster tree, $l_{\max} = 3$.

Symbol	Meaning
$\mathcal{U}$	domain of objects
S	set of indexed objects $S \subseteq \mathcal{U}$
M	set of buckets
d	distance function $d : \mathcal{U} \times \mathcal{U} \longrightarrow \mathbb{R}$
h	hash function $h \in \mathcal{H}$, $h : \mathcal{U} \rightarrow M$
K	number of concatenated functions/values
g	concatenated hash f. $g(u) = (h_1(u), \ldots, h_K(u))$
L	number of hash tables/hash functions
k	number of nearest neighbors for $kNN(q, k)$
C	set of candidate objects $C \subseteq S$
n	number of pivots
l	number of M-Index levels, $l \in \mathbb{N}$
$l_{\max}$	maximum level of dynamic M-Index
$\mathcal{H}^n_{l_{\max}}$	family of M-index hash functions

Table 1: Symbols used in this work.

of this family are defined by specific sets of pivots $p_0, \ldots, p_{n-1}$ selected randomly uniformly from the dataset. In the terminology introduced in Section 2, the M-Index does not use concatenation of several h function, i.e. $g(v) = h(v)$. For a better orientation in the symbols used throughout this paper see Table 1.

3.3 M-Index Searching

The M-Index precise evaluation strategies for similarity queries synergically exploit practically all known metric-based principles of data partitioning, pruning and filtering, which makes it relatively very efficient [16]. The approximate evaluation strategy for *kNN*(q, k) queries follows the general schema of the multi-probe LSH with $L = 1$ (see Section 2): A set of query candidate objects is formed by data from cluster corresponding to key $h(q)$ (let us denote this cluster C_q) and an optional number of clusters that should contain objects "close to object q". These partitions are identified by a heuristic based purely on analysis of the $d(p_0, q), \ldots, d(p_{n-1}, q)$ distances. Cluster C_q gets the highest priority – it is assigned *penalty* equal to 0. Cluster C_q has the smallest sum of distances $d(p_{(0)_q}, q) + \cdots + d(p_{(l-1)_q}, q)$ and this sum grows for other clusters. This is reflected by the penalty in order to express the "proximity" of the cluster to q. Specifically:

$$\mathit{penalty}_q(C_{i_0,\ldots,i_{l-1}}) = \\ = \frac{1}{l} \cdot \sum_{j=0}^{l-1} \max\left\{d(p_{i_j}, q) - d(p_{(j)_q}, q), 0\right\}. \quad (3)$$

Note that the penalty is normalized by $1/l$ in order to make its values comparable for clusters on different levels.

By this mechanism, M-Index can construct the set of candidate objects of arbitrary size. In other words, some *virtual buckets* specific to given query object are created at search time. This is a straightforward way to effectively tune the result quality (answer recall) versus the searching costs (I/O or computational).

4. LSH PROPERTIES OF M-INDEX

Current LSH techniques are typically defined for a given data type with a specific distance function. The M-Index has the potential to be a universal LSH technique for *any metric space* with the option of multi-probe bucket access. The objective of this section is to verify that the M-Index matches the LSH theory and to compare its qualities with established LSH methods. We also discuss practical impacts of the LSH definition in context of *kNN* queries.

For LSH functions defined for a specific data space, there typically exists a formula to show that the probability of two points colliding in the hash index is decreasing with their mutual distance (Definition 1). We do not induce such a general rule for the M-Index, as we can hardly assume any properties of the dataset in a generic metric space.

Instead, we rather perform measurements to assess this collision probability on real-life datasets for various M-Index settings and we put a standard LSH technique under the same test.

Experiment Setting

We performed the experiments on two datasets: The *CoPhIR* dataset [5, 3] consists of five MPEG-7 [15] visual descriptors extracted from digital images (280 dimensions altogether). Each of the descriptors is compared by a metric function and these partial distances are aggregated into a single metric space [2]. We also use the set of *Color Structure* (CS) descriptors from CoPhIR as a separate dataset. The CS descriptor is a 64-dimensional vector and we use ℓ_2 metric as the distance function, so it can be indexed by standard LSH techniques. We test various sizes of these datasets up to one million objects.

In order to measure the probability of collisions, we have selected one hundred query objects uniformly at random from the dataset and, for a given structure, we always test hash collisions of these queries with all objects in the database. We report percentage of hash collisions depending on the objects mutual distances (the scale is quantized). In this way, the whole distance scale is covered by the measurements.

Collision Probability Curves

Let us analyze the collision probability of LSH function h using the notation from Definition 1, which defines an (r_1, r_2, p_1, p_2)-sensitive function. Figure 3 schematically depicts the relationship between these four parameters: The collision probability curve of an LSH hash function should run through the grey area.

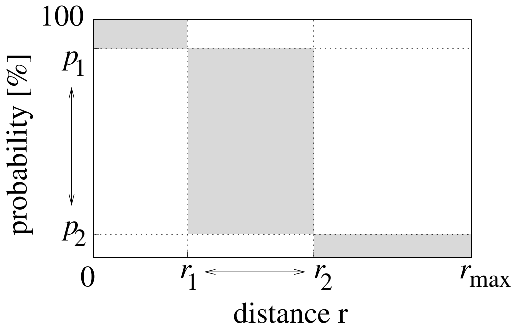

Figure 3: Collision probability case study.

Let us discuss an ideal shape of LSH function h: Probability p_1 should be as high as possible for such distances r_1 that will be used as query radii (distances to the k-th nearest neighbors, for $kNN(q,k)$ queries). Probability p_2 should be as low as possible for distances r_2 that are "not interesting for querying". If the collision probability is high for distances we are not interested in, then bucket $h(q)$ will contain many objects to be added to the candidate set and filtered out later, which increases the search costs. Thus, an ideal LSH function would decrease steeply between distances r_1 and r_2 (their difference would be small).

4.1 Basic LSH Properties

First, let us investigate the collision probability of the standard LSH scheme for ℓ_2 metric [10] (see Section 2). For this purpose, we utilize package E^{2}LSH [1] proposed by the authors. From this implementation, we only use the mechanism to generate member functions from the LSH family and to calculate hash keys for given vectors. We do not let E^{2}LSH set the K and L parameters for specific queries, because we want to study the LSH properties under various conditions and for various settings. Further on, we refer to this technique as E^{2}LSH, and we analyze it for various number of concatenated LSH functions (K) focusing on behavior of individual functions ($L = 1$). Figure 4 shows results for 500,000 objects from the Color Structure dataset.

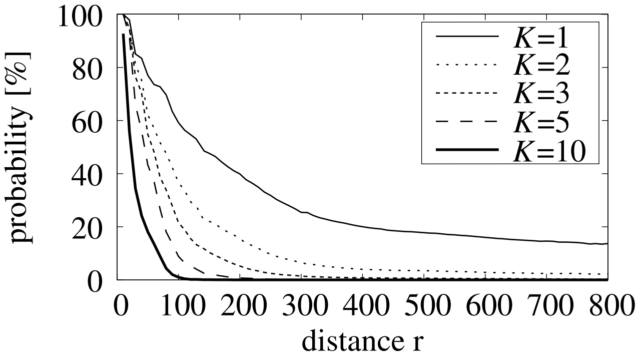

Figure 4: Collision probability of the E^{2}LSH indexes with 500,000 Color Structure objects.

The concatenation of more hash functions (increasing parameter K) results in shrinkage of the collision probability [13] – this effect can be observed well in the graph. We can see that for $K > 3$ the curves are very steep – only objects with distance up to 100 have a notable change to collide. Further in this section, we study what query radii are meaningful for this dataset and we analyze impact the collision probability on the kNN search efficiency. Each E^{2}LSH graph is averaged over five instances from the same LSH family – the results were stable among different instances.

Figure 5 presents results of the same experiment on the dynamic M-Index with $l_{\max} = 6$ for various number of pivots n. We can see that the collision probabilities are strictly monotonic with respect to mutual distance, which classifies the M-Index as an LSH technique according to Definition 1. The graphs are again averaged over several instances from family $\mathcal{H}^n_{l_{\max}}$ defined by specific sets of randomly chosen pivots. The dynamic hashing principle of the M-Index results in buckets of limited sizes – 1,000 objects, in our case. Due to this fact, the differences between individual curves

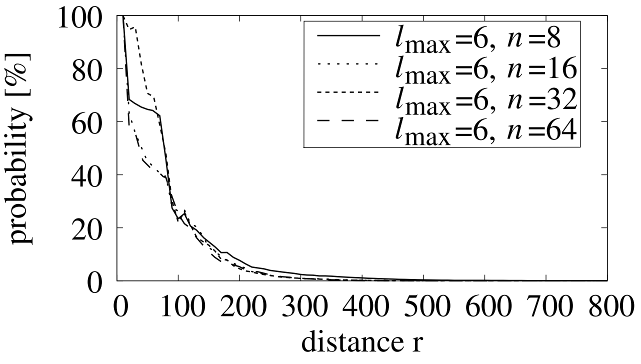

Figure 5: Collision probability of the M-Index with 500,000 Color Structure objects.

in Figure 5 are less significant than in static E^{2}LSH hashing, where higher values of K result in lower numbers of collisions (Figure 4). In the following measurements, we use M-Index functions from family $\mathcal{H}^{32}_6$ ($n = 32$) with cluster capacity of 1,000 objects.

The multi-probe approach, used by the M-Index search mechanism, defines virtual buckets of adjustable sizes (see Section 3.3). In Figure 6, we demonstrate collision probabilities of M-Index considering objects in a virtual bucket as colliding. We can see a significant increase of the probability for smaller mutual distances while the differences are smaller for more distant objects. This trend seems to be more convenient than the trend observed in Figure 4 – we elaborate on this topic more in the following section.

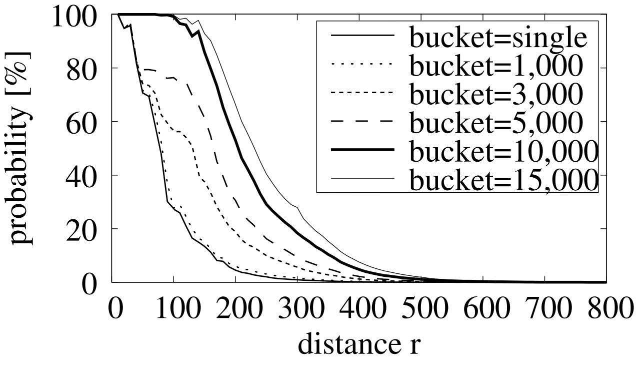

Figure 6: Collision probability of the M-Index with virtual buckets on 500,000 CS objects.

In order to fairly compare results of the collision experiment on the E^{2}LSH and the M-Index, we should put together settings with buckets of similar sizes, because these sizes directly influence the number of objects accessed during query execution. Table 2 shows average sizes of buckets hit by query objects during the measurements on E^{2}LSH indexes. The percentages in this table are stable for all set sizes, because the smaller datasets were created from the 1,000,000 collection as successive subsets by random uniform sampling. For the M-Index, the bucket size is directly tunable by the concept of virtual buckets.

The graph in Figure 7 compares E^{2}LSH with $K = 3$ (accessing 0.57 % of the 500,000 dataset ≈ 2,900 objects) with the M-Index accessing up to 3,000 objects. We can see a

concatenation parameter	$K = 1$	$K = 2$	$K = 3$	$K = 4$	$K = 5$	$K = 10$
average collision size	17.07 %	3.24 %	0.57 %	0.12 %	0.03 %	0.0007 %

Table 2: Average sizes of buckets hit by query objects on E^2LSH indexes (percentage of dataset size).

significant difference in the collision probability for lower distances; this probability for distances over 450 is zero for the M-Index and slightly above the zero level for the E^2LSH (the overall absolute number of collisions is the same for both curves). In the following we discuss the influence of this property on *kNN* search.

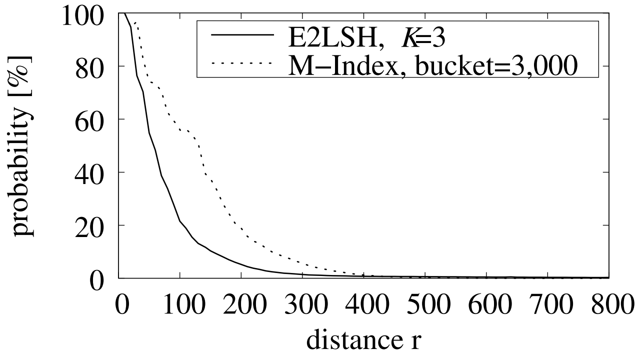

Figure 7: Collision probability of the M-Index and the E^2LSH with similar bucket sizes (3,000 objects) on 500,000 Color Structure objects.

4.2 LSH for Nearest-neighbors Search

This section tries to shed light on influence of the collision probability on the efficiency of $kNN(q, k)$ search using a specific LSH index. As we discussed in the previous section, an "ideal" shape of the collision probability curve depends on query radii – distances between the query objects and k-th nearest neighbors for $k \in \mathbb{N}$ of our interest. Figure 8 depicts average distances to the k-th nearest neighbor in the Color Structure dataset. Note the logarithmic scale on the x axis and the expected trend of the NN distances decreasing as the data space gets denser for larger data collections.

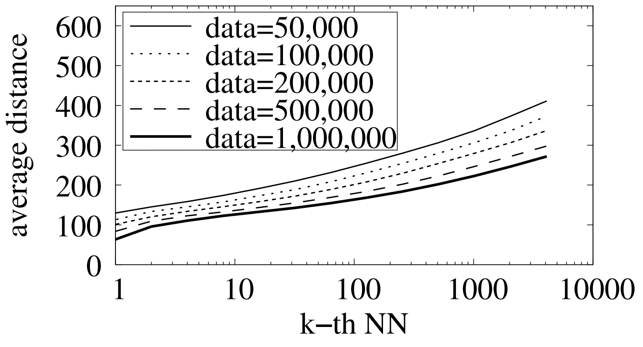

Figure 8: Average distances to k-th nearest neighbor for the Color Structure dataset.

This graph indicates that, roughly speaking, distances between 100 and 200 are interesting from the practical *kNN* point of view (k up to 100), which are exactly the distances for which the E^2LSH and the M-Index differ most significantly according to Figure 7. We would like to quantify this observation precisely, eliminating impreciseness that emerges from averaging. For this purpose, we introduce relation of two objects $p, q \in S$ being *k-th nearest neighbors* (there exist $k - 1$ objects from S that are closer to q than object p, or vice versa). Note that the nearest neighbor rank of an object p from object q is not necessarily the same as of object q from object p.

Definition 2. *Given metric space $\mathcal{M} = (\mathcal{U}, d)$, a finite set of indexed objects $S \subseteq \mathcal{U}$ and two objects $p, q \in S$, let us denote $N_{p(q)} = \{u \in S | d(p, u) < d(p, q)\}$. We say that p, q are k-*th nearest neighbors *for $k \in \mathbb{N}$, iff*

$$\min \{|N_{p(q)}|, |N_{q(p)}|\} = k - 1.$$

In the following test, we measure probability of hash collisions of pairs of objects that are k-th nearest neighbors. In compliance with the previous measurements, we always consider objects from the whole dataset for each of one hundred randomly chosen query objects and, thus, the whole scale of k-th NN is covered. Graph in Figure 9 compares this collision probability on the E^2LSH and the M-Index for variable $k \in \mathbb{N}$. These results confirm the intuition that the E^2LSH and the M-Index collision probabilities differ significantly for distances important for searching in practice.

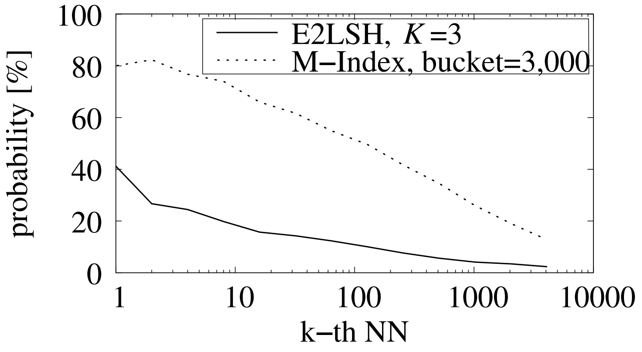

Figure 9: Collision probability for pairs of k-th nearest neighbors in the E^2LSH and the M-Index on 500,000 CS objects.

Let us inspect the scalability of the M-Index approach by fixing the size of the virtual bucket to 5,000 objects and observing the collision probability for dataset of variable sizes – see Figure 10. The probabilities are lower for larger datasets but, as seen from Figure 8, nearest neighbor distances shrink as datasets grow.

The exact influence of this trend is observable in the k-th nearest neighbor variant of this experiment in Figure 11. This graph actually corresponds to recall of $kNN(q, k)$ search with a single LSH index ($L = 1$) with constant search costs. For $k = 100$, the recall would be over 90 % on 50,000 dataset and about 70 % for one million Color Structure dataset.

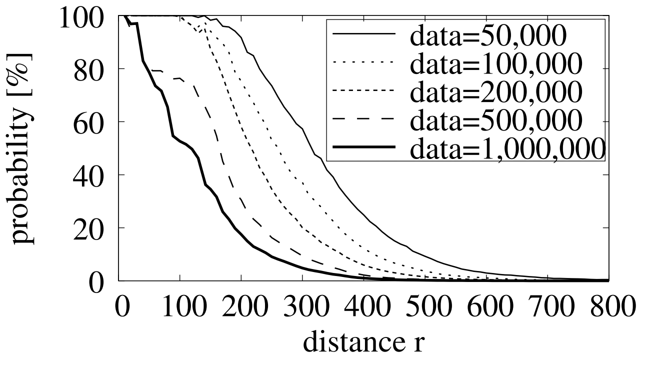

Figure 10: Collision probability of the M-Index with buckets with 5,000 objects on CS dataset.

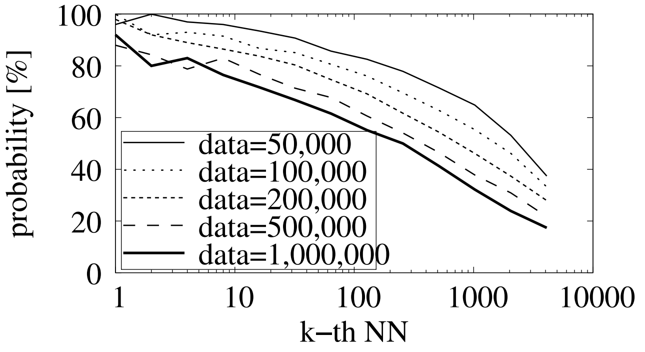

Figure 11: Collision prob. for pairs of k-th NN for CS dataset with virtual buckets of size 5,000.

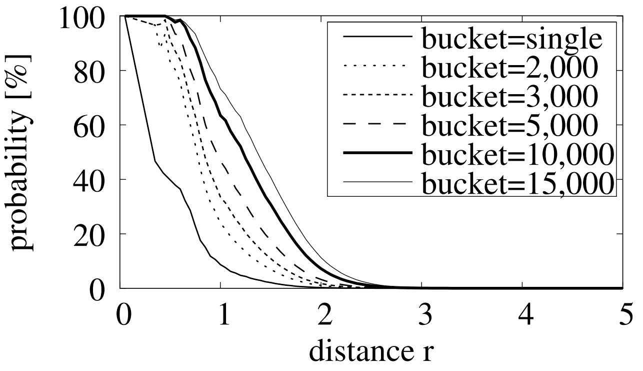

Figure 12: Collision probability for M-Index on CoPhIR dataset with 500,000 objects.

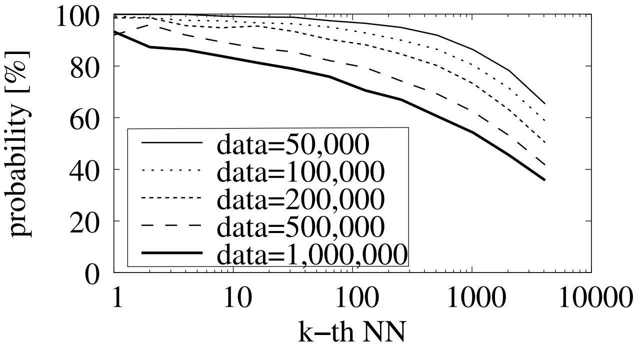

Figure 13: Collision prob. for pairs of k-th NN for CoPhIR dataset with virtual buckets of size 10,000.

4.3 M-Index on Complex Dataset

One of the main advantages of the M-Index in comparison with other LSH approaches is the ability to index any metric space. The distance function used with the CoPhIR dataset [5] is a complex, hierarchical weighted aggregation of several partial ℓ_1 and ℓ_2 functions (or their weighted modifications); such dataset cannot be directly processed by any standard LSH method and its intrinsic dimensionality [9] has been measured as 12.9 [3], which makes it rather difficult to index and search. Let us observe behavior of the M-Index on this data type.

We have tested several families of M-Index hash functions $\mathcal{H}^n_{l_{\max}}$, but we only report results for $\mathcal{H}^{32}_6$ with maximal bucket size of 1,000 objects – the differences between results for various families were only subtle (similarly as for the Color Structure dataset). Figure 12 shows the collision probability for various sizes of virtual buckets. We can see that the basic property of LSH functions according to Definition 1 is fulfilled and, intuitively, the shape of the curves is convenient for search efficiency.

Using Definition 2, we have measured collision probability of random k-th nearest neighbors with respect to $k \in \mathbb{N}$. Figure 13 presents the results using virtual buckets with 10,000 objects. Considering, for instance, the dataset with 1,000,000 CoPhIR objects, these curves correspond to the recall of $kNN(q,k)$ queries probing 1 % of the dataset.

4.4 Search Efficiency for Multiple Indexes

In this section, we study efficiency of the search (costs vs. recall) when multiple M-Indexes are used (parameter L). By means of the virtual buckets, we can precisely tune the number of accessed objects in particular index, thus influence the overall search costs, and observe the tradeoff between these costs and recall for various values of L.

Figure 14 presents the results of $kNN(q,30)$ search in 1,000,000 CoPhIR dataset using one, two, and three M-Indexes ($L = 1, 2, 3$). The x-axis indicates the *overall* number of accessed objects and the y-axis the recall averaged over hundred randomly selected query objects.

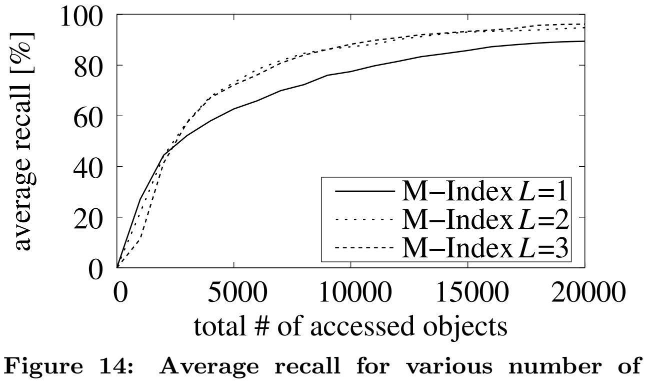

Figure 14: Average recall for various number of M-Indexes for 1,000,000 CoPhIR dataset.

We can see that the usage of multiple indexes started to be superior over one index at the level of approximately 50% recall. On the other hand, usage of more than two indexes does not seem to improve the performance significantly. In the future, we would like to study influence of multiple indexes on recall variance and other statistical properties. Replication of the data in several indexes also offers the option of better fault tolerance and of parallel processing of the query.

5. CONCLUSION

The concept of Locality-sensitive Hashing has been successfully used for similarity indexing. In the last decade, several LSH functions were defined for specific distance measures. In this work, we have shown that the recently proposed structure M-Index [16] has the properties of an LSH technique while it is applicable to any data space with metric properties. The M-Index hashing is dynamic and the search algorithm implements the concept of multi-probe LSH.

We conduct experiments on real-life data in order to measure the collision probability of the M-Index hash function. This property is the basic LSH quality measure and we compare it with a standard LSH function for ℓ_2 distance. We widely discuss the shapes of these probability curves with respect to similarity queries in practice and we measure the collisions of pairs of objects being k-th nearest neighbors. This measure seems to reflect well the search efficiency of given LSH function and it indicates that the quality of the M-Index hashing and its multi-probe approach are superior to the tested ℓ_2 LSH function. Moreover, the M-Index can precisely tune the search costs at query time. When we increase the size of the dataset while keeping the costs constant, the query recall degrades only modestly.

In the future, we would like to study more the relationship between the multi-probe parameter (virtual bucket sizes) and various number of indexes (parameter L). Namely, we plan to focus on the variance and other statistical properties of the query recall with respect to search costs and the value of L. We would also like to compare the M-Index approach with standard multi-probe LSH techniques and with the LSH Forest. Using the distributed version of the M-Index, we plan to investigate the point of fault tolerance gained by data replication in a dynamic distributed system.

6. ACKNOWLEDGMENTS

This work has been supported by national research projects MSMT 1M0545, GACR 201/09/0683, GACR 103/10/0886, and GACR P202/10/P220. The hardware infrastructure was provided by the METACentrum under the research intent MSM6383917201.

7. REFERENCES

[1] A. Andoni and P. Indyk. E2LSH: Exact Euclidean locality-sensitive hashing, 2004. http://web.mit.edu/andoni/www/LSH/.

[2] M. Batko, F. Falchi, C. Lucchese, D. Novak, R. Perego, F. Rabitti, J. Sedmidubsky, and P. Zezula. Building a Web-scale image similarity search system. *Multimedia Tools and Applications*, 2009.

[3] M. Batko, P. Kohoutkova, and D. Novak. CoPhIR image collection under the microscope. In *SISAP '09: Proceedings of the 2009 Second International Workshop on Similarity Search and Applications*, pages 47–54, Washington, DC, USA, 2009. IEEE Computer Society.

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

[5] P. Bolettieri, A. Esuli, F. Falchi, C. Lucchese, R. Perego, T. Piccioli, and F. Rabitti. CoPhIR: A test collection for content-based image retrieval. *CoRR*, abs/0905.4627v2, 2009.

[6] A. Broder. On the resemblance and containment of documents. In *SEQUENCES '97: Proceedings of the Compression and Complexity of Sequences 1997*, page 21, Washington, DC, USA, 1997. IEEE Computer Society.

[7] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. *Comput. Netw. ISDN Syst.*, 29(8-13):1157–1166, 1997.

[8] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC '02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388, NY, USA, 2002. ACM.

[9] E. Chávez and G. Navarro. Measuring the dimensionality of general metric spaces. Technical Report TR/DCC-00-1, Department of Computer Science, University of Chile, 2000.

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

[11] A. Esuli. PP-Index: Using permutation prefixes for efficient and scalable approximate similarity search. In *Proceedings of LSDS-IR'09*, pages 17–24, 2009.

[12] R. Fagin, J. Nievergelt, N. Pippenger, and H. R. Strong. Extendible hashing – A fast access method for dynamic files. *ACM Transactions on Database Systems*, 4(3):315–344, 1979.

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

[14] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-probe LSH: Efficient indexing for high-dimensional similarity search. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 950–961. VLDB Endowment, 2007.

[15] MPEG-7. Multimedia content description interfaces. Part 3: Visual. ISO/IEC 15938-3:2002, 2002.

[16] D. Novak and M. Batko. Metric index: An efficient and scalable solution for similarity search. In *SISAP '09: Proceedings of the 2009 Second International Workshop on Similarity Search and Applications*, pages 65–73, Washington, DC, USA, 2009. IEEE Computer Society.

[17] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search: The Metric Space Approach.* Springer, 2006.

[18] V. M. Zolotarev. One-dimensional stable distributions. In *Vol. 65 of Translations of Mathematical Monographs.* American Mathematical Society, 1986.