Enhancing Locality Sensitive Hashing with Peek-Probing and Nearest Neighbor Links [*]

Aleksandar Stupar
Saarland University
Saarbrücken, Germany
astupar@mmci.uni-saarland.de

Sebastian Michel
Saarland University
Saarbrücken, Germany
smichel@mmci.uni-saarland.de

ABSTRACT

In this work, we consider search in high dimensional data and propose two optimization techniques for Locality Sensitive Hashing (LSH). LSH has been successfully applied to search in multi-media databases or to duplicate detection in large Web or XML collections. LSH maps objects from a high-dimensional feature space to a set of buckets, using a hash function likely to cause hash collisions for *similar* objects. The first enhancement of LSH is based on additionally introduced links for each point in the feature space. These links refer to the exact nearest neighbor. The second approach is coined *Peek-Probing*, where LSH buckets are only fully read if they indicate a certain amount of useful information. The techniques are fully orthogonal and, hence, can be used in a combined way for further improved performance. We study the suitability of our approaches based on a series of experiments using high-dimensional image features of different flavor. We report on performance numbers for our algorithms and baseline competitors when tuned to provide answers of a minimum accuracy.

1. INTRODUCTION

Searching for objects similar to a given sample (query) object is a fundamental problem which occurs in a multitude of scenarios. Among the most prominent examples is the search for similar images or Web documents. Objects are described by features, e.g., their words or word-stems in case of text documents, or features describing the structural content and color distribution of an image. What renders similarity search difficult is the large number of features to consider for semantically meaningful results. The result of a search is a list of objects, ranked by their similarity to the query object, in descending order. In most cases, only the top portion of this result list is of interest and only in this case, there is hope to devise means that inspect only a small fraction of the full database for query answering. Restricted to the best answers, the problem is to find the K Nearest Neighbors (KNN) to a query object. The notion of neighborhood is defined by the multi-dimensional space in which points are aligned based on their feature representation. In many scenarios, returning the approximate results instead of the exact ones, suffices. Often, the approximate version is even more desirable when the data dimensionality is high, as similarity search is very expensive in such domains. Locality Sensitive Hashing (LSH) [1] is such an approximate method which has proven the ability to deal with high-dimensional data in a robust and efficient way. The core idea is to map, using a hash function, data objects to hash buckets. This hash function should have the property of being locality sensitive, that is, similar objects have a higher probability of a hash collision than unrelated ones. At query time, the hash bucket for the query object is determined and all objects in that bucket are ranked according to their distance to the query. To reach a high level of accuracy, i.e., how many of the true K Nearest Neighbors have been identified, multiple hash tables are used. There are dozens of follow up works on LSH that aim at decreasing the runtime of the search process, while still delivering at least as good results as the original work. One of the most prominent extensions is the work in [12] which devises an algorithm that determines a set of additional hash buckets to look into. We use this Multi-Probe LSH approach as the underlying core LSH technique, but also report on performance results of the original LSH algorithm.

In this work we propose two additional ways to accelerate the performance of LSH. The first enhancement of LSH is based on additionally *introduced links* for each point in the feature space. These links refer to the exact nearest neighbor. The second approach is coined *Peek-Probing*, where LSH buckets are only fully read if they indicate a certain amount of useful information. The nice property of our proposed techniques is their orthogonality, hence, they can be jointly applied, and their independence on the underlying LSH method.

The paper is organized as follows. In Section 2 we discuss the related work and give an introduction to the main principles behind Locality Sensitive Hashing, necessary to understand the enhancements proposed. Section 3 presents the enhancement based on the nearest neighbor links, coinded Linked-LSH approach. The Peek-Probing approach is described in Section 4. The description of the experimental setup together with the reported results on baselines and proposed approaches is contained in Section 5. Section 6 concludes the paper.

2. RELATED WORK AND BACKGROUND

Being fundamental to many application areas searching for the K Nearest Neighbors (closest points) in the multi-dimensional space has received a lot of attention by researchers in the past decades (cf., [13] for overview). When the number of dimensions is relatively small, approaches yielding the best results are the ones based on tree structures, such as X-Tree [4] and K-D tree [3]. However, with

[*]This work has been supported by the Excellence Cluster on Multimodal Computing and Interaction (MMCI).

the increasing number of dimensions we fall into a trap commonly known as the "curse of dimensionality" [5]. This has a direct impact on the efficiency of the tree based data structures, rendering them applicable only for a small number of dimensions.

Locality Sensitive Hashing (LSH) [1, 7, 10] is proposed as a solution to this problem, rendering KNN processing efficient in high dimensional space. The basic principle behind LSH is the usage of locality preserving hash functions which map, with high probability, close points from the high dimensional space to the same hash value (i.e., hash bucket). Different parameters of locality preserving functions together with the number of hash function used, render LSH a parametric approach. Studies concerning LSH parameter tuning [8, 2] have been performed providing an insight into LSH parameter tuning for optimal performance.

The idea of looking into, also called probing, neighboring LSH buckets with an aim of improving precision is the key idea behind the Multi-Probe method, proposed in [12]. When the set of previously seen queries is available probing multiple buckets can be done based on the learned probability distributions, as described in [11].

2.1 LSH Basics

The basic idea behind LSH is the usage of *locality sensitive hash* functions for data indexing. A hash function is said to be locality sensitive if it maps, with high probability, neighboring points from the d-dimensional vector space to the same hash value, i.e., related objects are more probable to have the same hash value than distant ones. The LSH index is built of several hash tables with each having multiple hash functions, to increase the probability of collision for close points. At query time, LSH bucket is selected from a hash table based on the hash values of the query point for that table. The distance to all points contained in that bucket is calculated and the closest K points are returned as the final result.

In this work, we consider the family of LSH functions based on p-stable distributions [7] which are most suitable for l_p norms. In this case, for each data point $\mathbf{v}$, the hashing scheme considers k independent hash functions of the form

$$h_{\mathbf{a},B}(\mathbf{v}) = \lfloor \frac{\mathbf{a}\cdot\mathbf{v}+B}{W} \rfloor \tag{1}$$

where $\mathbf{a}$ is a d-dimensional vector whose elements are chosen independently from a p-stable distribution, $W \in \mathbb{R}$, and B is chosen uniformly from $[0, W]$. Each hash function maps a d-dimensional data point onto the set of integers. With k such hash functions, the final result is a vector of length k of the form $g(\mathbf{v}) = (h_{\mathbf{a_1},B_1}(\mathbf{v}), ..., h_{\mathbf{a_k},B_k}(\mathbf{v}))$.

Reaching a higher precision, with the same number of hash tables and the same runtime, has been achieved by probing multiple buckets of the same LSH table, also known as the Multi-Probe [12] method. At query time, several closest buckets are selected for probing, based on the distance between query point and the bucket. The distance between the query point and the neighboring bucket is given as a sum of distances for each locality sensitive function of that hash table, which is given as

$$d(v, L) = (L - h_R)^2 = (L - \frac{\mathbf{a}\cdot\mathbf{v}+B}{W})^2, L > h_R \tag{2}$$

$$d(v, L) = (L+1-h_R)^2 = (L+1-\frac{\mathbf{a}\cdot\mathbf{v}+B}{W})^2, L \leq h_R \tag{3}$$

where L is the label (the integer value) of the bucket, obtained by hashing any point of the bucket with a given LSH function, and h_R is a real value. The other symbols are the same as for the LSH approach above.

Since experiments showed that most of the real KNN are contained in buckets where the distance between individual labels equals one (+1,-1), the Multi-Probe method probes only these buckets. An efficient algorithm based on perturbations is used to find the closest buckets to a query point from this restricted subset.

3. LINKED-LSH

Locality Sensitive Hashing (LSH) indexes data by computing the hash bucket labels of each object separately. That is, indexing of an object is done completely independent from the other data points contained in the same collection. The key idea behind *Linked-LSH* is to use additional information, obtained at indexing time from indexed dataset, with the goal of obtaining an improved query processing performance.

More precisely, we use the (first) nearest neighbor in the indexed collection for all the data points as a global statistics descriptor of that collection.

The intuition behind using the exact first nearest neighbor as a descriptor is given by the triangle inequality $d(q, p_2) \leq d(q, p_1) + d(p_1, p_2)$, where q represents a query point, p_1 is the point indicated by LSH and p_2 is the point missed by LSH and is the exact closest neighbor in the collection for the point p_1. As p_2 is the exact closest neighbor for the p_1, there is a high probability that the distance between these two points $d(p_1, p_2)$ is small. If we make sure that the distance between the query point q and the point indicated by LSH p_1 is small enough, the triangle inequality tells us that distance between the query point q and point p_2 is also small and that there is high probability that the point p_2 is also in the exact top-K results for the given query.

Linked-LSH extends the LSH index by adding a pointer to each indexed data point (feature vector) which points to the closest neighbor in the indexed collection. Pointers for data points are precomputed in the indexing phase and the exact closest neighbor is used which is found through the full scan of all of the data points. It is important to note that this extension of the index results in a negligible increase in index size as each data point contains values for multiple dimensions and only one pointer value in addition. Figure 1 illustrates an LSH index with links introduced by Linked-LSH in two dimensional space. The (red) rectangular data point represents a query point which after hashing to LSH index retrieves a first neighbor. The (green) circular point represents the second exact neighbor to the query point which is missed by LSH due to its approximate nature. We see that following the link from the first retrieved neighbor to its first neighbor results in retrieving the second exact neighbor of the query point which would be lost if only LSH was used.

Answering a query with Linked-LSH is done in two steps. In the first step the query is answered using an existing LSH approach, such as [12]. The obtained query results are then used in the second step, in which the links between data points are used for the retrieval and evaluation of additional points.

As stated above, we need to assure that the distance $d(q, p_1)$ is small enough. That is the reason why we use only the top-$\tilde{K}$ results from the first step as an input to the

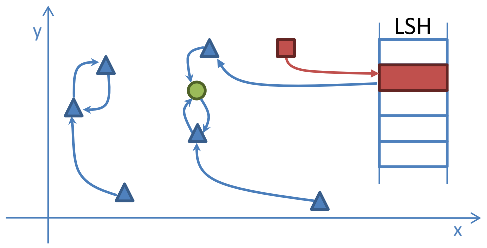

Figure 1: Linked-LSH

second step, as it guarantees that we are using the points that have the smallest distance to the query point out of the points indicated by LSH. The parameter $\tilde{K}$ should depend on the original K as specified in the query and can be experimentally tuned for the best performances. It is important to note, that $\tilde{K}$ as well as the parameter n, introduced in the following paragraph, are parameters that are set at runtime and hence can be adjusted without index re-organization, which is usually needed for the parameters in the raw LSH approaches.

The approximate top-$\tilde{K}$ results from the first step provide a starting point for further retrieval using links in the collection. For each point in the top-$\tilde{K}$ points, we recursively retrieve the closest neighbors up to the depth n by consecutively following closest neighbor links. Depth n of the consecutive retrieval is again a parameter of the approach. The distance to each additionally retrieved data point is calculated, in case it was not already calculated in the previous step, and used to evaluate if the data point is in the top-K result list. The pseudo code of the query answering with Linked-LSH is shown in Algorithm 1.

```
1   intermediate = LSH.eval(query, $\tilde{K}$)
2   knn.add(intermediate)
3   for (point in intermediate):
4       last = point
5       while (steps $\leq$ depth):
6           last = last.getNeighbor()
7           knn.eval(last)
8   return knn.results
```

Algorithm 1: Query answering with Linked-LSH

By design, Linked-LSH can be applied to any of the existing LSH approaches [10, 12] by simply using them in the first step and following the links based on their results in the second step.

4. PEEK-PROBING

The LSH index consists of multiple hash tables such that each of them contains multiple buckets (cf., Section 2.1). Each bucket contains a subset of the data points, assigned by a hash function. In the query answering phase, multiple buckets are selected based on the hash value of the query or based on Multi-Probe techniques [12]. Our *Peek-Probing* assumes that not all of the selected buckets have the same importance to the query answering. We try to determine that importance before evaluating all the data points from all of the buckets. The idea is to use existing LSH techniques to select buckets, and then to peek into each of these buckets and to predict how important it is for answering the given query. After the bucket importance values are approximated we use only the data points from the most important buckets and discard the rest. The key point here is that the importance of the bucket is determined for each specific query.

To approximate the importance of a bucket for the given query, we perform a complete KNN evaluation over the data points received from peeking into all of the buckets. Peeking into a bucket means retrieving a first p elements from the bucket where p is proportional to the bucket size (number of data points in the bucket) and is given by

$$p = 1 + \left\lfloor \frac{b}{f} \right\rfloor$$

where b is the size of the bucket and f is the bucket fraction proportion (we used the value of $f = 8$ in all of the experiments). During the evaluation of the peeked data points, if the data point makes it in the *peeked top-K results*, we remember the bucket that point came from. We show below how we re-organize the bucket content to obtain a meaningful overview of the bucket content, as otherwise the p first points would represent a random sample.

After all the peeked data points are evaluated, we call a bucket important if there is at least a single data point in the top-K peeked results from that bucket. This approach works well in practice as K is usually a small number. In case K is very large number we would judge the importance of the bucket as a total number of data points in the peeked top-K results that come from that bucket, rather then to just make a binary decision. In the following steps only the data points from the important buckets are used for evaluation. The querying algorithm is presented in Algorithm 2.

```
1   buckets = LSH.probe(query)
2   for (bucket in buckets):
3       for (point in bucket.peek):
4           knn.eval(point, bucket)

5   important = knn.getBuckets()
6   for (bucket in important):
7       for (point in bucket.rest):
8           knn.eval(point)
9   return knn.results
```

Algorithm 2: Query Answering with Peek-Probing

It is important to note that as there are by design multiple hash tables in LSH, data points can be contained in multiple buckets originating from different hash tables. This means that we need to store multiple buckets for a data point at runtime, while evaluating peeked points. We have experimented with different number of buckets saved per data point and concluded that the best performances are achieved when only one bucket (the first one encountered) is saved for the data point. This small number is imposed by the large overhead in bookkeeping all of the information when multiple buckets are saved per data point.

Peeking into a bucket is performed to get an idea about the content of that bucket, and as already mentioned is done on the first p data points of the bucket. We can randomly select any p points of the bucket and place them in the beginning of the bucket. However, by doing this we may end up with a bad description of the content based only on the first p data points.

To avoid such situations we select first p points from the bucket by *clustering the bucket data* in p clusters and then selecting the medoid of each cluster to be put in the beginning of the bucket. We use the expectation-maximization

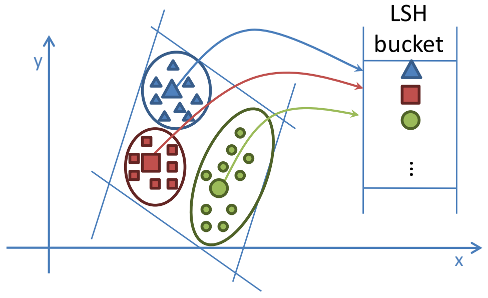

Figure 2: LSH bucket organization

algorithm for k-means clustering to cluster the data contained in the buckets. Figure 2 illustrates this process of data points selection and their placement in the beginning of the bucket.

The motivation behind this idea is given by the k-means optimization criterion $\arg\min_C \sum_{i=1}^{p} \sum_{x_j \in C_i} \|x_j - \mu_i\|^2$, where C_i is a cluster with centroid μ_i and x_j are data points from that cluster. This optimization criterion tells us that the distance between the centroid and the data point in that cluster is minimized, i.e., if the centroid is in the top-K results among the other centroids then there is a high probability that some other points from that cluster are also in the final top-K results. We use medoids, the closest point from the cluster to the centroid, instead of the centroids, as centroids are non existent data points and would incur additional computation.

As we mentioned earlier Linked-LSH and Peek-Probing are orthogonal and can be both applied to any existing LSH approach. We coin the name combined approach for the approach where the Linked-LSH is applied on top of the Peek-Probing approach, which is again applied on top of the Multi-Probe approach from [12].

5. EXPERIMENTAL EVALUATION

We have implemented our two LSH enhancements as well as the baseline approaches in Java 1.6 and use the 64-bit variant of the Java VM to execute the code. The implementation is single threaded. The experiments are conducted on a dual CPU Intel Xeon E55302.4 GHz and 47.9 GB of main memory, running Microsoft Windows Server 2003 Enterprise x64 Edition (Service Pack 2).

We compare the following five approaches.

- **LSH:** This is the implementation of the original work on LSH presented in [10].
- **Multi-Probe**: We have implemented the Multi-Probe algorithm presented in[12] and use it as the underlying LSH method for our approach. While this method outperforms LSH (in most cases), we still include the original LSH method for completeness.
- **Linked-LSH**: Represents the implementation of the enhancement of the LSH using the nearest neighbor links of the data points in the dataset, together with the recursive link traversal, as described in the Section 3.
- **Peek-Probing**: Implementation of the Peek-Probing strategy from Section 4, where LSH buckets are probed completely only if a certain amount of usefulness is indicated.
- **Combined**: This is the implementation of the Peek-Probing together with Linked-LSH, built on top of Multi-Probe LSH buckets selection.

To evaluate the above approaches on real-world data, we have obtained the **CoPhIR dataset** [6] . It consists of MPEG7 feature descriptors extracted from a large collection of images obtained from Flickr.com [9]. Out of all available images in the dataset, we have randomly selected $100,000$ images to index. Additionally $10,000$ images are randomly selected and used as query images.

For each crawled image, the dataset contains MPEG7 feature vectors that are given in an XML based format together with a URL of the source photo. We have transformed the XML format in a convenient binary format before starting the experiments.

We use the following subsets of MPEG7 feature descriptors in the evaluation: color structure, scalable color, and edge histogram. Scalable color and color structure descriptors in CoPhIR are defined by 64 dimensional vectors, while an edge histogram descriptor is a 80 dimensional vector. We use these three different feature representations for deeper insights on the performance of the algorithms under comparison. We will see below that due to their different characteristics, the algorithms can behave quite differently.

Measured Values

As all LSH based approaches are by design approximate methods, i.e., a returned KNN result might or might not differ from the true K nearest neighbors, we measure the *precision* in addition to *runtime* and *inspected data portion*. The precision is measured as the percentage of the returned approximate top-K results that are also found in the exactly computed (using a naive full scan approach) top-K results. The runtime was measured as the number of seconds (with millisecond resolution) needed to answer all of the $10,000$ queries. The measured inspected data portion represents the percentage of the indexed $100,000$ feature vectors for which the distance to the query data point was calculated while answering that query. Clearly, there is a correlation between the inspected data portion and the total runtime, as more distance calculations require more time. We report an average precision and average inspected data portion for $10,000$ queries, while the reported runtime is the total time needed to answer all of 10000 queries.

LSH Setup

Locality Sensitive Hashing is a method that enables us to index and search for close data points with the tradeoff between memory usage (index size) and the time needed to answer a query with a certain precision. Trading off index size against runtime is achieved by changing the number of hash tables used for indexing. The more hash tables are used, the less time is needed to answer a query with the same precision.

LSH is a parametric method with a common practice of tuning the parameters for each individual dataset. Having multiple parameters, yielding multi dimensional parameter space, usual practice is to fix all but one parameter and to vary that parameter until the optimum is found. This procedure is repeated until a global (or local) optimum approximation is found. To achieve the best results, we have performed parameter tuning of each approach independently.

As we are interested in relative improvements, we fix the number of hash tables to 32 for all described LSH approaches.

Achieving a certain precision at fixed number of hash tables depends on the number of data points found in one LSH bucket. This number, in turn, depends on the number of hash functions per hash table as well as on the parameter W of the each of the function, see Equation 1. As W is a continuous variable it gives us more control over the bucket size, so we fix the number of hash functions per table and vary W to achieve certain precision values. Preliminary experiments have shown that the best runtime, with 32 hash tables, is achieved when using 8 hash functions per table. Hence, in the following experiments, we use 8 hash function per table.

Probing multiple buckets from the same hash table requires an additional parameter that describes the number of additional probes. We have experimented with different number of additional probes for each feature descriptor. It turns out that for color structure and scalable color descriptors Multi-Probe is not better than the original LSH and it achieves the best results with only two additional probes per table. However, for edge histogram descriptor Multi-Probe outperforms the original LSH, with the best performances at 30 additional probes per hash table. We have used the same number of probes also for Linked-LSH, Peek-Probing, and the combined approach.

Linked-LSH also introduced depth parameter which defines the depth of the recursive nearest neighbor traversal. Setting this parameter is easy as by intuition it has to have a low value, as we saw in Section 2. By experimenting we found out that best results are achieved when depth is two. For Linked-LSH we also need to determine the value of $\tilde{K}$, we do that by $\tilde{K} = c * K$, where c is determined experimentally for the best performance. We used the value of $c = 3$ when Linked-LSH was tested alone, and $c = 1.1$ when tested in the combined approach.

As the non-deterministic nature of LSH can result in slight deviations of measurements obtained by the same parameter setup, we perform each experiment for each parameter setup 10 times and report average results.

5.1 Experimental Results

For each of the approaches we have performed measurements at three levels of precision: at 80%, 90%, and 95%. As described, the parameter W in Equation 1 is used to tune each of the methods towards a certain precision. As we are not able to ensure the exact precision wanted, the precision is also measured and reported.

Table 1 contains the measurements for all approaches for color structure descriptor. As we can see, using Multi-Probe in the case of color structure descriptor does not result in an improvement over the original LSH. The runtime and inspected data portion for Multi-Probe and original LSH are almost the same, as we used small number (i.e., 2) of additional probes. Increasing the number probes made the results only worse for Multi-Probe in this case. We see that Linked-LSH provides a constant improvement in both runtime and inspected data portion over the baselines. The best improvement in runtime is achieved for the precision at 80%, reducing the runtime by 23.65% while the runtime improvements for 90% and 95% precision are 17.44% and 19.57% respectively. Due to the relation between runtime and inspected data portion, the improvements for inspected portion are proportional to the runtime improvements. Even higher improvement in runtime and inspected data portion is achieved using Peek-Probing, which reduces runtime by 50.21% at 95% precision. Improvements using Peek-Probe

approach	**prec. (%)**	**time (s)**	**insp. (%)**
LSH	80.746	34.138	7.810
	90.384	61.504	13.659
	95.268	103.279	21.949
Multi-Probe	80.530	34.873	7.473
	90.493	69.012	14.496
	95.153	105.864	21.640
Linked-LSH	80.151	26.062	4.915
	90.237	50.773	10.118
	95.155	83.065	16.705
Peek-Probing	80.489	19.646	1.934
	90.245	31.558	3.208
	95.675	51.415	5.127
Combined	80.490	**16.276**	**1.575**
	90.354	**25.490**	**2.617**
	95.215	**40.072**	**4.058**

Table 1: Measurements for color structure descriptor

approach	**prec. (%)**	**time (s)**	**insp. (%)**
LSH	80.617	116.042	23.020
	90.287	183.178	35.243
	95.327	272.212	48.001
Multi-Probe	80.364	125.184	20.996
	90.046	174.275	29.282
	95.200	222.560	36.597
Linked-LSH	80.578	106.495	16.348
	90.135	160.724	25.703
	95.519	223.370	34.787
Peek-Probing	80.468	58.399	6.088
	90.180	**70.624**	7.760
	95.136	**82.843**	9.325
Combined	80.796	**58.109**	**5.765**
	90.301	70.695	**7.459**
	95.044	83.379	**8.901**

Table 2: Measurements for edge histogram descriptor

are seen over all measured precisions, with runtime improvement of 42.45% at 80% precision and 48.68% at 90% precision. Combining Linked-LSH and Peek-Probe (i.e., Combined algorithm) yields the best results, with more than a factor of 2 improvement in runtime and more than a factor of 4 improvement in inspected data portion.

Measurements for the edge histogram descriptor are shown in Table 2. We see that Multi-Probe achieves a significant improvement in runtime over the original LSH method, at 95% precision, but is slightly worse at 80% precision. In the case of the edge histogram descriptor the benefit of using Linked-LSH is quite small at 80% and 90% precision, with an improvement of 8.22% and 7.77% respectively, and non-existent at 95% precision. However, using Peek-Probe results in a significant improvement in runtime, with 62.77% improvement at 95% precision. Improvement for Peek-Probing at 90% precision is 59.47% and at 80% precision is 49.67%. As there was no significant runtime improvement in using Linked-LSH there is no improvement in using combined approach over only using Peek-Probing. The inspected data portion is still best for the combined approach, but values are close the the Peek-Probing approach, showing once again that Linked-LSH has no impact for the edge histogram descriptor.

Table 3 shows measurements for the scalable color descriptor. We can see that relative improvements for Linked-LSH, Peek-Probing, and the combined approach are similar to the improvements for color structure descriptor. This is expected as both descriptors are based on the color distribution of the images. We can see that the combined approach performs best, for both runtime and inspected data portion. The best runtime improvement is achieved at 95% precision,

approach	prec. (%)	time (s)	insp. (%)
LSH	80.647	13.563	2.993
	90.563	25.562	5.741
	95.596	41.651	9.278
Multi-Probe	80.180	15.726	2.978
	90.318	27.254	5.631
	95.032	43.226	9.041
Linked-LSH	80.805	13.620	2.078
	90.849	22.418	4.063
	95.132	34.406	6.685
Peek-Probing	80.283	10.032	0.725
	90.977	14.948	1.264
	95.215	20.107	1.850
Combined	80.294	**8.975**	**0.622**
	90.650	**12.451**	**1.029**
	95.107	**16.809**	**1.477**

Table 3: Measurements for scalable color descriptor

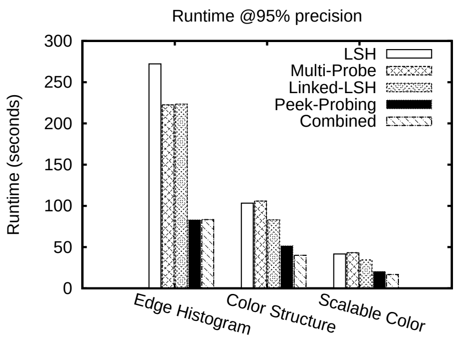

Figure 3: Runtime measurements at 95% precision

reducing runtime for 59.64%.

Runtime measurement for all aproaches and all descriptors at 95% precision are shown in Figure 3. We can see that searching for most similar image based on edge histogram descriptor takes a lot longer than based on the color descriptors. Although the edge histogram descriptor uses higher dimensional feature vectors (80 dimension) than for the color descriptors (64 dimensions), the main runtime difference comes from the fact that more data points are used in calculating the answer for edge histogram descriptor than for color descriptiors, as shown in Figure 4, which illustrates inspected data portion for all approaches and all descriptors at 95% precision. The difference in runtime and inspected data portion between given descriptors shows how all of these approaches are highly depended on the distribution of data points. Figure 3 and Figure 4 show the improvement of proposed approaches over the baselines for all three cases.

6. CONCLUSION

We presented two enhancements to the Locality Sensitive Hashing (LSH) approach. Both approaches carefully inspect potential K Nearest Neighbor (KNN) candidates and, thus, enable the algorithm to achieve high accuracy results with improved runtime performance by fewer data accesses. Our techniques address two crucial points of the general LSH concept. First, we introduced exact nearest neighbor links, for each object in the index. These links help to explore the right region of the high-dimensional space. Second, we introduce a way to peek into an LSH hash bucket without full access to the indexed data. Both approaches show improvements to the underlying LSH approach, for which we used the well-known Multi-Probe method. It is important to stress, however, that the presented techniques are of general nature and can also be applied to other LSH extensions that act as the underlying index infrastructure. Our two enhancements are also completely orthogonal to each other and, hence, can be jointly applied to achieve an even larger performance gain. We reported on the results of an experimental evaluation which evaluated the presented approach and baseline competitors using three different classes of features, all from a real-world image database.

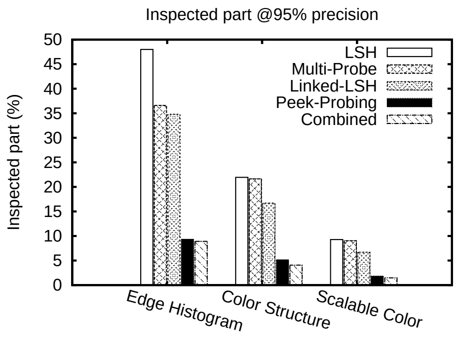

Figure 4: Inspected dataset portion at 95% precision

7. REFERENCES

[1] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *FOCS*, 2006.
[2] M. Bawa, T. Condie, and P. Ganesan. LSH forest: self-tuning indexes for similarity search. *WWW*, 2005.
[3] J. L. Bentley. K-d trees for semidynamic point sets. *Symposium on Computational Geometry*, 1990.
[4] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree : An index structure for high-dimensional data. *VLDB*, 1996.
[5] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? *ICDT*, 1999.
[6] 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*, 2009.
[7] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. *Symposium on Computational Geometry*, 2004.
[8] W. Dong, Z. Wang, W. Josephson, M. Charikar, and K. Li. Modeling lsh for performance tuning. *CIKM*, 2008.
[9] Flickr photosharing – `www.flickr.com`.
[10] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. *VLDB*, 1999.
[11] A. Joly and O. Buisson. A posteriori multi-probe locality sensitive hashing. *ACM Multimedia*, 2008.
[12] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-probe LSH: Efficient indexing for high-dimensional similarity search. *VLDB*, 2007.
[13] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. The Morgan Kaufmann Series in Computer Graphics, 2006.