Applying Hash-based Indexing in Text-based Information Retrieval

Benno Stein and Martin Potthast

Faculty of Media, Media Systems
Bauhaus University Weimar, 99421 Weimar, Germany,
{benno.stein | martin.potthast}@medien.uni-weimar.de

ABSTRACT

Hash-based indexing is a powerful technology for similarity search in large document collections [13]. Central idea is the interpretation of hash collisions as similarity indication, provided that an appropriate hash function is given. In this paper we identify basic retrieval tasks which can benefit from this new technology, we relate them to well-known applications and discuss how hash-based indexing is applied. Moreover, we present two recently developed hash-based indexing approaches and compare the achieved performance improvements in real-world retrieval settings. This analysis, which has not been conducted in this or a similar form by now, shows the potential of tailored hash-based indexing methods.

Keywords

hash-based indexing, similarity hashing, efficient search, performance evaluation

1. INTRODUCTORY BACKGROUND

Text-based information retrieval in general deals with the search in a large document collection D. In this connection we distinguish between a "real" document $d \in D$, in the form of a paper, a book, or a Web site, and its computer representation $\mathbf{d}$, in the form of a term vector, a suffix tree, or a signature file. Likewise, $\mathbf{D}$ denotes the set of computer representations of the real documents in D.

Typically, a document representation $\mathbf{d}$ is an m-dimensional feature vector, which means that the objects in $\mathbf{D}$ can be considered as points in the m-dimensional vector space. The similarity between two documents, d, d_q, shall be inversely proportional to the distance of their feature vectors $\mathbf{d}, \mathbf{d}_q \in \mathbf{D}$. The similarity is measured by a function $\varphi(\mathbf{d}, \mathbf{d}_q)$ which maps onto $[0; 1]$, with 0 and 1 indicating no and a maximum similarity respectively; φ may rely on the l_1-norm, the l_2-norm, or on the angle between the feature vectors. Obviously the most similar document $d^* \in D$ respecting a query document d_q maximizes $\varphi(\mathbf{d}, \mathbf{d}_q)$.

We refer to the task of finding d^* as *similarity search*, which can be done by a linear scan of $\mathbf{D}$—in fact this task cannot be done better than in $O(|\mathbf{D}|)$ if the dimensionality, m, of the feature space is around 10 or higher [15]. Also the use of an inverted file index cannot improve the runtime complexity since the postlist lengths are in $O(|D|)$. Here the idea of hash-based indexing comes into play: Testing whether or not $\mathbf{d}$ is a member of $\mathbf{D}$ can be done in virtually constant time by means of hashing. This concept can be extended toward similarity search if we are given some kind of similarity hash function, $h_\varphi : \mathbf{D} \to U$, which maps the set $\mathbf{D}$ of document representations to a universe U of keys from the natural numbers, $U \subset \mathbf{N}$, and which fulfills the following property [13]:

$$h_\varphi(\mathbf{d}) = h_\varphi(\mathbf{d}_q) \;\Rightarrow\; \varphi(\mathbf{d}, \mathbf{d}_q) \geq 1 - \varepsilon, \qquad (1)$$

with $\mathbf{d}, \mathbf{d}_q \in \mathbf{D}$ and $0 < \varepsilon \ll 1$.

That is, a hash collision between two elements from $\mathbf{D}$ indicates a high similarity between them, which essentially relates to the concept of precision $prec_\varepsilon$:

$$prec_\varepsilon = \frac{|\{\mathbf{d} \in \mathbf{D}_q : \varphi(\mathbf{d}, \mathbf{d}_q) \geq 1 - \varepsilon\}|}{|\mathbf{D}_q|},$$

with $\mathbf{D}_q = \{\mathbf{d} \in \mathbf{D} : h_\varphi(\mathbf{d}) = h_\varphi(\mathbf{d}_q)\}$.

Though the applicability of hash-based indexing to text-based information retrieval has been demonstrated, there has been no discussion which retrieval tasks can benefit from it; moreover, none of the hash-based indexing approaches have been compared in this connection. Our research aims at closing this gap.

Section 2 discusses retrieval tasks along with applications where hash-based indexing can be applied to improve retrieval performance and result quality. Section 3 introduces two hash-based indexing methods that can be utilized in the field of text-based information retrieval. Section 4 presents results from a comparison of these methods for important text-retrieval tasks.

2. RETRIEVAL TASKS IN TEXT-BASED INFORMATION RETRIEVAL

Let T denote the set of terms that are used within the document representations $\mathbf{d} \in \mathbf{D}$. The most important data structure for searching D is the inverted file [16, 2], which maps a term $t \in T$ onto a so-called postlist that encodes for each occurrence of t the respective documents d_i, $i = 1, \ldots, o$, along with its position in d_i. In the following it is assumed that we are given a (possibly very large) document collection D which can be accessed with an inverted file index, μ_i,

$$\mu_i : T \to \mathcal{D}$$

as well as with a hash index, μ_h,

$$\mu_h : \mathbf{D} \to \mathcal{D}$$

$\mathcal{D}$ denotes the power set of D. The inverted file index μ_i is used to answer term queries; the hash index μ_h is ideally suited to answer

DIR 2007, March 28-29, Leuven, Belgium

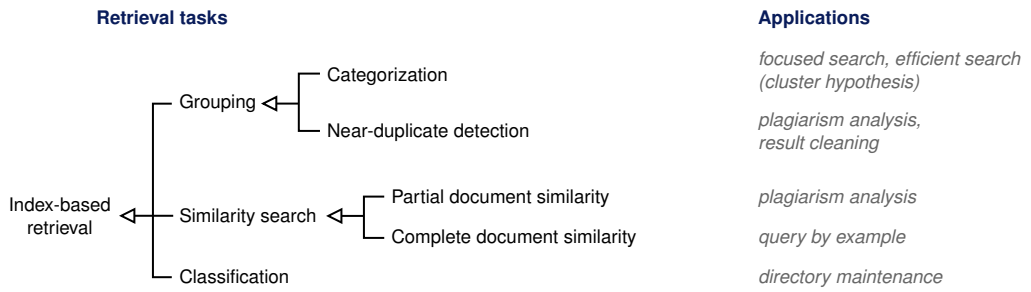

Figure 1: Text-based information retrieval tasks (left) and exemplified applications (right) where hash-based indexing can be applied.

document queries, say, to map a document onto a set of similar documents. Section 3 shows how such a hash index along with a hash function h_φ is constructed.

We have identified three fundamental text retrieval tasks where hash-based indexing can be applied: (*i*) grouping, which further divides into categorization and near-duplicate detection, (*ii*) similarity search, which further divides into partial and complete document comparison, and (*iii*) classification. Figure 1 organizes these tasks and gives examples for respective applications; the following subsections provide a more in-depth characterization of them.

2.1 Retrieval Task: Grouping

Grouping plays an important role in connection with user interaction and search interfaces: Many retrieval tasks return a large result set that needs to be refined, visually prepared, or cleaned from duplicates. Refinement and visual preparation require the identification of categories, which is an unsupervised classification task since reasonable categories are a-priori unknown. Categorizing search engines like Vivísimo and AIsearch address this problem with a cluster analysis [17, 11]; they are well-known examples where such a kind of result set preparation is applied. Duplicate elimination helps in cleaning result sets that are cluttered with many, nearly identical documents [4]. The latter situation is typical for nearly all kinds of product searches and commercial applications on the Web: for the same product various suppliers can be found; likewise, mirror sites can pretend the existence of apparently different documents.

Remarks on Runtime. With hash-based indexing the performance of such kinds of retrieval tasks can be significantly improved: A user formulates his/her information need as a term query for which, in a first step, a result set $D_q \subseteq D$ is compiled by means of an inverted file index μ_i. In a second step the hash index μ_h is applied for the grouping operation, which may be categorization or duplicate elimination. Figure 2 illustrates the strategy. Recall that μ_h helps us to accomplish the grouping operation in linear time in the result set size $|D_q|$. By contrast, if for grouping a vector-based document model is employed we obtain a runtime of $O(|D_q|^2)$, since duplicate elimination or category formation requires a pairwise similarity computation between the elements in D_q.

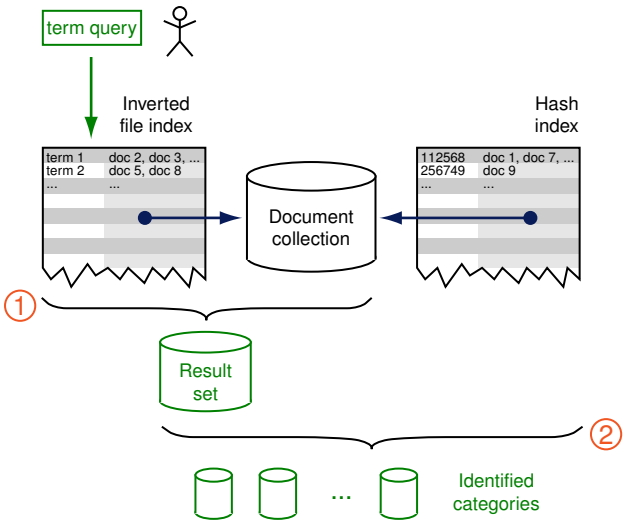

Figure 2: Illustration of a grouping retrieval task. Starting point is an information need formulated as term query, which is satisfied with an inverted file index, yielding a result set (Step ①). The subsequent categorization is realized with a hash index in linear time of the result set size (Step ②).

2.2 Retrieval Task: Similarity Search

The most common retrieval task is a similarity search where a user's information need is formulated as (boolean) term query. If the user is in fact interested in documents that match exactly, say, that contain the query terms literally, the optimum index structure respecting search performance is the inverted file index, μ_i. The major search engine companies like Google, Yahoo, or AltaVista are specialized in satisfying these kinds of information

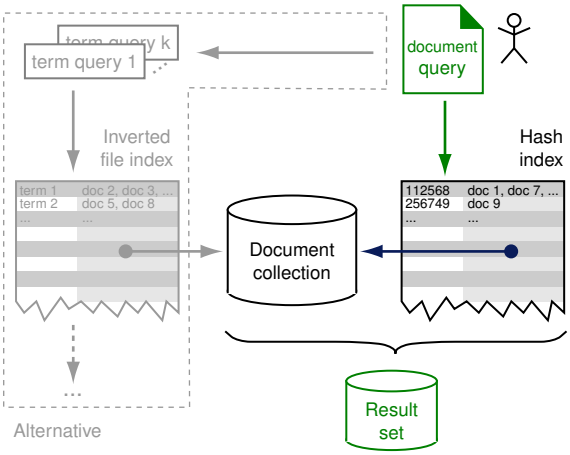

Figure 3: Illustration of a similarity search retrieval task. Starting point is an information need formulated as document query, which is satisfied with a hash index in constant time of the document collection size. Alternatively, keywords may be extracted from the document query to formulate several term queries.

needs. However, if a user specifies his/her information need in the form of a document query, say, in a "more like this" manner, a hash-based index, μ_h, could be employed as index structure. The question is whether a similarity hash function h_φ that fulfills Property (1) can be constructed at all, which in turn depends on the admissible ε-interval demanded by the information need.

Remarks on Runtime. The utilization of a hash-based index, μ_h, can be orders of magnitude faster compared to a classical inverted file index μ_i: To identify similar documents with μ_i, appropriate keywords must be extracted from the document query, a number k of term queries must be formed from these keywords, and the respective result sets $D_{q_1}, \ldots, D_{q_k}$ must be compared to the document query. Figure 3 illustrates both this alternative strategy and the utilization of μ_h. If we assume an effort of $O(1)$ for both inverted file index access and hash index access, the construction of the result set requires $O(|D_{q_1}| + \ldots + |D_{q_k}|)$ with μ_i, and only $O(1)$ with μ_h. Note that the actual runtime difference depends on the quality of the extracted keywords and the finesse when forming the term queries from them; it can assume dramatic proportions. This situation is aggravated if an application like plagiarism analysis requires the segmentation of the query document in order to realize a similarity search for one paragraph at a time [14].

2.3 Retrieval Task: Classification

Classification plays an important role in various text retrieval tasks: genre analysis, spam filtering, category assignment, or author identification, to mention only a few. Text classification is the supervised counterpart of text categorization and, for a small number of classes, it can be successfully addressed with Bayes, discriminant analysis, support vector machines, or neural networks. However, with an increasing number of classes the construction of a classifier with assured statistical properties is rendered nearly impossible.

With hash-based indexing an efficient and robust classifier can be constructed, even if the number of classes, $C_1, \ldots, C_p$, is high and if only a small or irregularly distributed number of training documents is given. The classification approach follows the k-nearest neighbor principle and presumes that all documents of the classes C_i are stored in a hash-index μ_h. Given a document d_q to be classified, its hash value is computed and the documents D_q that are assigned to the same hash bucket are investigated with respect to their distribution in $C_1, \ldots, C_p$. d_q gets assigned that class C_j where the majority of the documents of D_q belongs to:

$$C_j = \underset{i=1,\ldots,p}{\operatorname{argmax}} |C_i \cap D_q|$$

3. REALIZATION OF HASH-BASED INDEXING

Given a similarity hash function $h_\varphi : \mathbf{D} \rightarrow U$, a hash-based index μ_h can be directly constructed by means of a hash table $\mathcal{T}$ along with a standard hash function $h : U \rightarrow \{1, \ldots, |\mathcal{T}|\}$; h maps the universe of keys, U, onto the $|\mathcal{T}|$ storage positions. Indexing a set $\mathbf{D}$ of document representations means to compute for each $\mathbf{d} \in \mathbf{D}$ its hash key $h_\varphi(\mathbf{d})$ and to insert in $\mathcal{T}$ at position $h(h_\varphi(\mathbf{d}))$ a pointer to d. This way $\mathcal{T}$ maintains for each value of h_φ a bucket $D_q \subset D$ that fulfills the following condition:

$$d_1, d_2 \in D_q \;\Rightarrow\; h_\varphi(\mathbf{d_1}) = h_\varphi(\mathbf{d_2}),$$

where $\mathbf{d_1}, \mathbf{d_2}$ denote the computer representations of the documents d_1, d_2. Based on $\mathcal{T}$ and h_φ a document query corresponds to a single hash table lookup. In particular, if h_φ fulfills Property (1) the bucket returned for $\mathbf{d}_q$ defines a set $\mathbf{D}_q$ whose elements are in the ε-neighborhood of $\mathbf{d}_q$ with respect to φ:

$$\mathbf{d} \in \mathbf{D}_q \;\Rightarrow\; \varphi(\mathbf{d}, \mathbf{d}_q) \geq 1 - \varepsilon$$

The crucial part is the choice and the parameterization of a suitable similarity hash function h_φ for text documents. To our knowledge two approaches have been recently proposed, namely fuzzy-fingerprinting and locality-sensitive hashing. Both approaches can be directly applied to the vector space representation $\mathbf{d}$ of a document and are introduced now.

3.1 Fuzzy-Fingerprinting

Fuzzy-fingerprinting is a hashing approach specifically designed for but not limited to text-based information retrieval [13]. It is based on the definition of a small number of k, $k \in [10, 100]$, prefix equivalence classes. A prefix class, for short, contains all terms starting with the same prefix. The computation of $h_\varphi(\mathbf{d})$ happens in the following steps: (*i*) Computation of $\mathbf{pf}$, a k-dimensional vector that quantifies the distribution of the index terms in $\mathbf{d}$ with respect to the prefix classes. (*ii*) Normalization of $\mathbf{pf}$ using a corpus that provides a representative cross-section of the source language, and computation of $\Delta_{\mathbf{pf}} = (\delta_1, \ldots, \delta_k)^T$, the vector of deviations to the expected distribution.[1] (*iii*) Fuzzification of $\Delta_{\mathbf{pf}}$ by projecting the exact deviations according to diverse fuzzification schemes. Figure 4 illustrates the construction process.

Typically, three fuzzification schemes (= linguistic variables) are used whereas each scheme differentiates between one and three deviation intervals. For a fuzzification scheme ρ with r deviation intervals Equation 2 shows how a document's normalized deviation vector $\Delta_{\mathbf{pf}}$ is encoded:

$$h_\varphi^{(\rho)}(\mathbf{d}) = \sum_{i=0}^{k-1} \delta_i^{(\rho)} \cdot r^i, \quad \text{with } \delta_i^{(\rho)} \in \{0, \ldots, r-1\} \qquad (2)$$

$\delta_i^{(\rho)}$ is a document-specific value and represents the fuzzified deviation of $\delta_i \in \Delta_{\mathbf{pf}}$ when applying fuzzification scheme ρ.

A prefix class is not necessarily limited to exactly one prefix. It is also possible to define a combined prefix class as the union of two or more others. There are 26^i possible prefix classes for the Latin alphabet, where i denotes the prefix length. Using a bin packing heuristic the combined prefix classes can be constructed such that each class has the same expected frequency.

Document representations based on prefix classes can be regarded as abstractions of the vector space model. On the one hand they have a weaker performance when directly applied to a retrieval task such as grouping, similarity search, or classification. On the other hand they are orders of magnitude smaller and do not suffer from sparsity.

3.2 Locality-Sensitive Hashing

Locality-sensitive hashing (LSH) is a generic framework for the randomized construction of hash functions [9]. Based on a family H_φ of simple hash functions h, $h : \mathbf{D} \rightarrow U$, a locality-sensitive hash function h_φ is a combination of k functions $h \in H_\varphi$ obtained by an independent and identically distributed random choice. When using summation as combination rule the construction of $h_\varphi(\mathbf{d})$ is defined as follows:

$$h_\varphi(\mathbf{d}) = \sum_{i=1}^{k} h_i(\mathbf{d}), \quad \text{with } \{h_1, \ldots, h_k\} \subset_{rand} H_\varphi$$

[1] In this paper the British National Corpus is used as reference, which is a 100 million word collection of written and spoken language from a wide range of sources, designed to represent a wide cross-section of current British English [1].

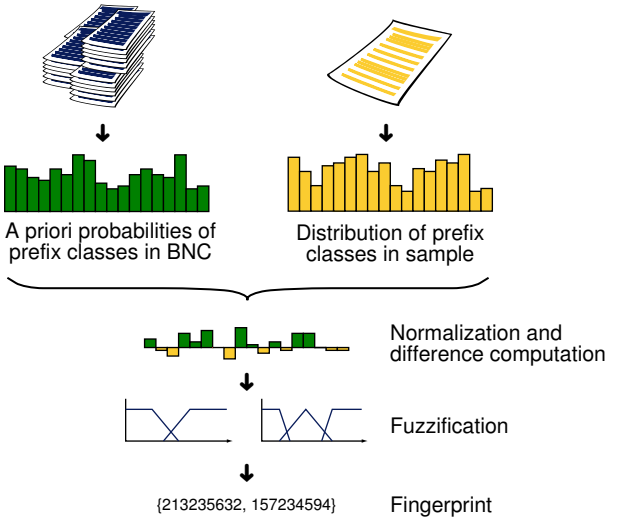

Figure 4: The basic steps of the hash key computation for fuzzy-fingerprinting.

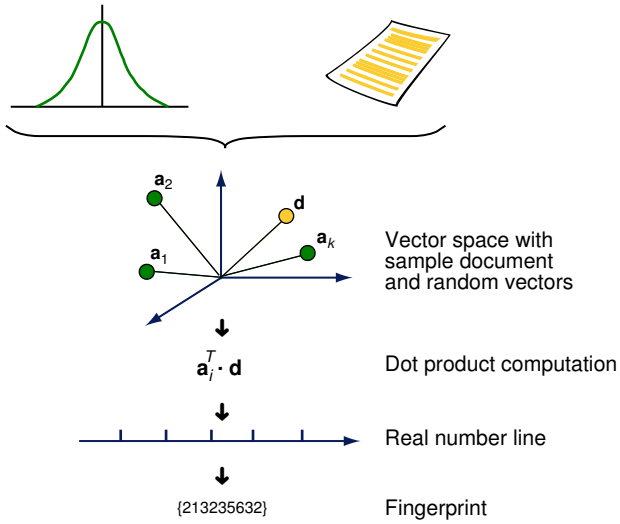

Figure 5: The basic steps of the hash key computation for locality-sensitive hashing.

Several hash families H_φ that are applicable for text-based information retrieval have been proposed [5, 6, 3]; our focus lies on the approach of Datar et al. [6]. The idea of this hash family is to map a document representation $\mathbf{d}$ to a real number by computing the dot product $\mathbf{a}^T \cdot \mathbf{d}$, where $\mathbf{a}$ is a random vector whose vector components are chosen independently from a particular probability distribution. The real number line is divided into equidistant intervals of width r each of which having assigned a unique natural number, and the result of the dot product is identified with the number of its enclosing interval. Under this approach the construction of h_φ for a given set ρ of random vectors $\mathbf{a}_1, \ldots, \mathbf{a}_k$ reads as follows:

$$h_\varphi^{(\rho)}(\mathbf{d}) = \sum_{i=1}^{k} \left\lfloor \frac{\mathbf{a}_i^T \cdot \mathbf{d} + c}{r} \right\rfloor,$$

where $c \in [0, r]$ is chosen randomly to allow any possible segmentation of the real number line. Figure 5 illustrates the construction process.

Note that with locality-sensitive hashing a lower bound for the retrieval quality can be stated: If the average distance of a document to its nearest neighbor is known in advance, the computation of h_φ can be adjusted such that the retrieval probability for the nearest neighbor is above a certain threshold [7]. This fact follows from the locality-sensitivity of a hash family H_φ, which claims that for any $h \in H_\varphi$ the probability of a collision of the hash keys of two documents raises with their similarity.[2]

3.3 Retrieval Properties of Hash Functions

The most salient property of hash-based indexing is the simplification of a fine-grained similarity quantification, operationalized as similarity function φ, toward the binary concept "similar or not similar": Two document representations are considered as similar if their hash keys are equal; otherwise they are considered as not similar. This implication, formalized at the outset as Property (1), is related to the statistical concept of *precision*. The reverse, namely that two hash keys are equal if the similarity of the respective document representations is above a certain threshold $1 - \varepsilon$, is related to the statistical concept of *recall*.

Note that the latter property cannot hold in general for a single hash function h_φ. One hash function computes one key for one document at a time, and, as a consequence, it defines an absolute partitioning of the space of document representations.[3] Hence, the averaged recall of any document query must be smaller than 1. Figure 6 illustrates this connection: Despite their high similarity (= low distance) a hash function h_φ will map some of the document representations onto different hash keys. If a second hash function h'_φ defining a slightly different partitioning is employed, a document query can be answered by the logical disjunction of both functions. Technically this corresponds to the construction of two hash indexes, μ_h, μ'_h, and forming the union of the returned buckets as result set. In reality one can observe a monotonic relationship between the number of hash functions and the achieved recall—but there is no free lunch though: the improved recall is bought with a decrease in precision.

It is interesting to compare the different concepts by which variation is introduced within the hash key computation: fuzzy-fingerprinting employs several fuzzification schemes ρ_i while locality-sensitive hashing employs several sets of random vectors ρ_i. In

[2] In order to guarantee a hash family being locality-sensitive the distribution must be α-stable. An example for such a distribution is the Gaussian distribution. For further theoretical background we refer to [8, 12].

[3] By contrast, a complete similarity graph underlying a set $\mathbf{D}$ of document representations defines for each element its specific partitioning.

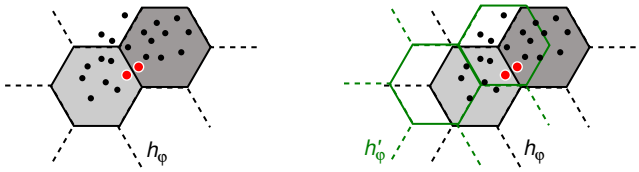

Figure 6: A set of documents projected in the two-dimensional space. A hash function h_φ partitions the space into regions that are characterized by different hash keys. Even if two documents (colored red) are very similar to each other they may be mapped onto different hash keys (left). This threshold-behavior can be alleviated by employing several functions h_φ and h'_φ (right).

both cases a hash index (fingerprint) of a document representation $\mathbf{d}$ is a *set*, comprising a small number l of keys, $\{h_{\varphi}^{(\rho_i)}(\mathbf{d}) \mid i = 1, \ldots, l\}$.

4. FUZZY-FINGERPRINTING VERSUS LOCALITY-SENSITIVE HASHING

This section presents results from large-scale experiments for two retrieval tasks: (*i*) near-duplicate detection, and (*ii*) a similarity search where the case of complete document similarity is analyzed. The experiments were conducted on the basis of two purposefully chosen test collections each of which resembling a realistic retrieval situation for either retrieval task.[4]

Both similarity hashing approaches were employed in order to demonstrate the applicability of this technology in terms of retrieval accuracy and, secondly, to analyze which of both approaches is better suited for the specified tasks and text-based information retrieval in general. Also their runtime performance was evaluated. As a baseline for comparison a linear search in the test collection was used, which is the best exact retrieval approach for the tasks in question.

To measure the retrieval accuracy of the hash functions we set up hash indexes for each test collection using both fuzzy-fingerprinting and locality-sensitive hashing. The values for precision and recall were determined for each document (e. g. by using it as a query) with respect to the similarity thresholds $0.1 \cdot i,\ i = 0, \ldots, 10$, averaging the results. The reference values for precision and recall were computed under the vector space model, employing the term weighting scheme $tf \cdot idf$ along with the cosine similarity measure. Note that for the investigated retrieval tasks this document model is sufficiently competitive compared to a human assessment.

To render the retrieval results comparable the hash functions were parameterized in such a way that, on average, small and equally-sized document sets were returned for a query. As described in Section 3.3, this relates to adjusting the recall of the hash functions, which is done with the number of fuzzification schemes and random vector sets respectively: two or three different fuzzification schemes were employed for fuzzy-fingerprinting; between 10 and 20 different random vector sets were employed for locality-sensitive hashing.

The precision of fuzzy-fingerprinting is controlled by the number k of prefix classes and the number r of deviation intervals per fuzzification scheme. To improve the precision performance either of them or both can be raised. Typical values for k are between 26 and 50; typical values for r range from 1 to 3. The precision of locality-sensitive hashing is controlled by the number k of combined hash functions. For instance, when using the hash family proposed by Datar et al., k corresponds to the number of random vectors per hash function; typical values for k range from 20 to 100.

4.1 Retrieval Task: Near-Duplicate Detection

The experiments in the retrieval task "near-duplicate detection" were conducted using a custom-built plagiarism corpus, compiled from $3,000$ artificially generated documents for the purpose of simulating different kinds of text plagiarism. The corpus relies on selected scientific documents from the ACM Digital Library and employs an algorithm to synthesize plagiarized instances by extracting and recombining paragraphs from the original documents. This collection is used to resemble document collections containing many documents with a high pairwise similarity, which is desirable for the analysis of near-duplicate detection algorithms.

Figure 7 contrasts the performance of the two hashing approaches. The left and middle plot show the recall and the precision performance. In both cases fuzzy-fingerprinting outperforms locality-sensitive hashing significantly. The right plot shows the runtime performance for different sample sizes; both hashing approaches perform significantly better than the linear approach, yet locality-sensitive hashing performs slightly better than fuzzy-fingerprinting. Observe that the runtime of either approach increases linearly if the collection size is raised. This means that the average result set size, $|D_q|$, for a document query is linearly correlated with the collection size, $|D|$, if the kind of documents and their distribution does not change significantly.

4.2 Retrieval Task: Similarity Search

The experiments in the retrieval task "similarity search" were conducted using a collection of $100,000$ documents compiled with the aid of the Web search engines Yahoo, Google, and AltaVista by performing a focused search on a specific topic. As first step in constructing the collection 15 seed documents about this topic were chosen and 100 keywords were extracted from them with a co-occurrence analysis [10]. This methodology ensures an unbiased choice of keywords representing the topic. Within a second step, search engine queries were generated by randomly choosing up to five of the keywords. The highest ranked search results of each query were downloaded and their text content extracted. This collection is used to resemble the analysis of result sets with respect to a Web retrieval system.

Figure 8 contrasts the performance of the two hashing approaches. Again, the left plot shows the recall performance. Observe that either hashing approach is excellent at high similarity thresholds (> 0.8) compared to the recall performance of a linear search which achieves optimal recall for each individual threshold. However, high recall values at *low* similarity thresholds are achievable by chance only—a fact which can be explained with the high number of documents with low pairwise similarity in the test collection. Fuzzy-fingerprinting and locality-sensitive hashing behave similar, the former slightly better.

The middle plot shows the precision performance. Obviously the precision of fuzzy-fingerprinting is significantly higher than the precision of locality-sensitive hashing. I. e., a result set, D_q, returned by fuzzy-fingerprinting is more accurate than a result set returned by locality-sensitive hashing, which directly affects the runtime performance.

The right plot shows the runtime performance for different sample sizes; both hashing approaches perform orders of magnitude faster than the standard linear search.

4.3 Discussion

The results of our experiments provide a comprehensive view of the behavior of hash-based indexes for the retrieval tasks "near-duplicate detection" and "similarity search".

For the detection of near-duplicates the fuzzy-fingerprinting technology outperforms locality-sensitive hashing with respect to both precision and recall, which renders this technology as method of choice for this task. The difference in retrieval accuracy also explains the slight difference in runtime performance between the hashing approaches: lesser retrieval accuracy yields fewer hash collisions which means that, on average, smaller document groups have to be evaluated.

In terms of recall fuzzy-fingerprinting and locality-sensitive hashing achieve the same quality for a similarity search; a slight

[4]Meta files describing the collections, the experiments, and their results have been compiled and are available to other researchers upon request.

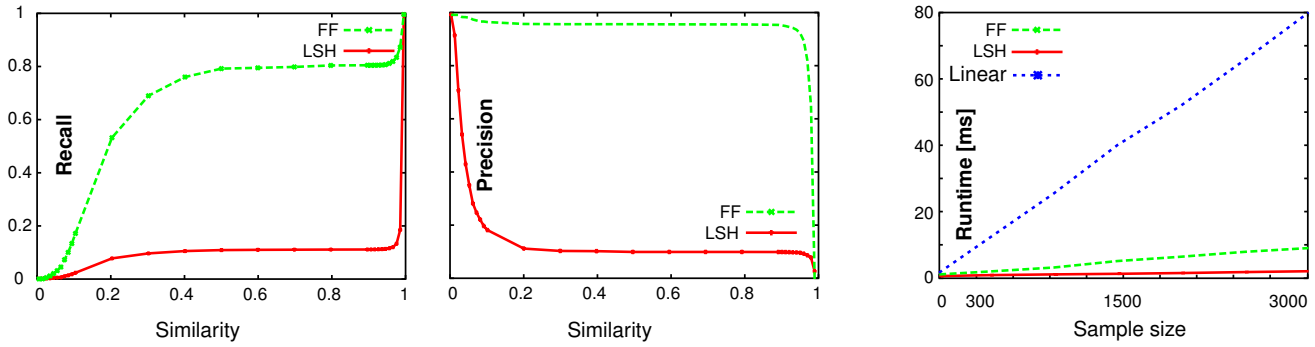

Figure 7: Retrieval task: near-duplicate detection. Application: plagiarism analysis. Collection: plagiarism corpus. The left plot shows recall-at-similarity values, the middle plot precision-at-similarity values, and the right plot runtime-at-sample-size values, using fuzzy-fingerprinting (FF) and locality-sensitive hashing (LSH).

advantage of fuzzy-fingerprinting can be stated yet. In terms of precision fuzzy-fingerprinting outperforms locality-sensitive hashing significantly. However, since the result set size of a document query is typically orders of magnitude smaller compared to the size of the document collection, the precision performance may be negligible and thus locality-sensitive hashing may be applied just as well.

5. SUMMARY

Hash-based indexing is a promising new technology for text-based information retrieval; it provides an efficient and reliable means to tackle different retrieval tasks. We identified three major classes of tasks in which hash-based indexes are applicable, that is, grouping, similarity search, and classification. The paper introduced two quite different construction principles for hash-based indexes, originating from fuzzy-fingerprinting and locality-sensitive hashing respectively. An analysis of both hashing approaches was conducted to demonstrate their applicability for the near-duplicate detection task and the similarity search task and to compare them in terms of precision and recall. The results of our experiments reveal that fuzzy-fingerprinting outperforms locality-sensitive hashing in the task of near-duplicate detection regarding both precision and recall. Within the similarity search task fuzzy-fingerprinting achieves a clearly higher precision compared to locality-sensitive hashing, while only a slight advantage in terms of recall was observed.

Despite our restriction to the domain of text-based information retrieval we emphasize that the presented ideas and algorithms are applicable to retrieval problems for a variety of other domains. Actually, locality-sensitive hashing was designed to handle various kinds of high-dimensional vector-based object representations. Also the principles of fuzzy-fingerprinting can be applied to other domains of interest—provided that the objects of this domain can be characterized with a small set of discriminative features.

Our current research aims at the theoretical analysis of similarity hash functions and the utilization of the gained insights within practical applications. We want to quantify the relation between the determinants of fuzzy-fingerprinting and the achieved retrieval performance in order to construct optimized hash indexes for special purpose retrieval tasks. Finally, we apply fuzzy-fingerprinting as key technology in our tools for text-based plagiarism analysis.

6. REFERENCES

[1] Guy Aston and Lou Burnard. The BNC Handbook. `http://www.natcorp.ox.ac.uk`, 1998.

[2] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999.

[3] Mayank Bawa, Tyson Condie, and Prasanna 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 Press.

[4] Andrei Z. Broder. Identifying and filtering near-duplicate documents. In *COM'00: Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching*, pages 1–10, London, UK, 2000. Springer-Verlag.

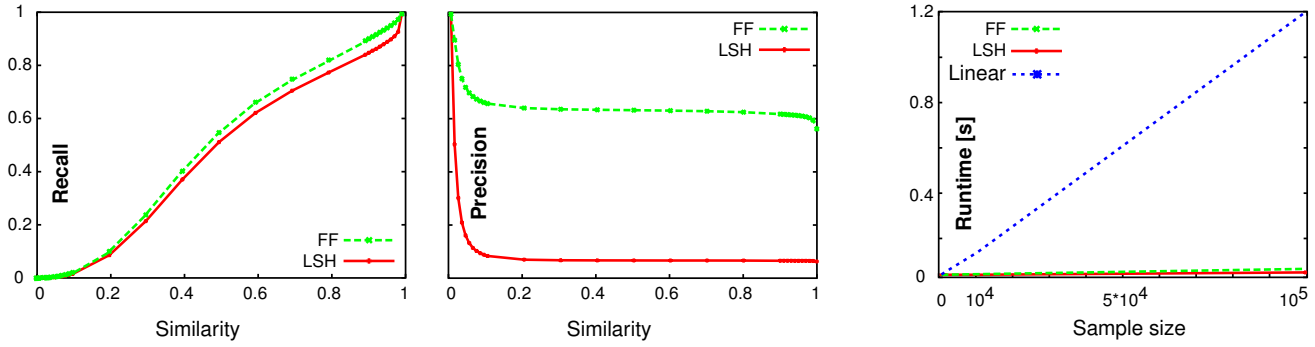

Figure 8: Retrieval task: similarity search. Application: query by example. Collection: Web results. The left plot shows recall-at-similarity values, the middle plot precision-at-similarity values, and the right plot runtime-at-sample-size values, using fuzzy-fingerprinting (FF) and locality-sensitive hashing (LSH).

[5] Moses S. Charikar. Similarity Estimation Techniques from Rounding Algorithms. In *STOC '02: Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*, pages 380–388, New York, NY, USA, 2002. ACM Press.

[6] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab 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 Press.

[7] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity Search in High Dimensions via Hashing. In *Proceedings of the 25th VLDB Conference Edinburgh, Scotland*, pages 518–529, 1999.

[8] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *FOCS '00: Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, page 189, Washington, DC, USA, 2000. IEEE Computer Society.

[9] Piotr Indyk and Rajeev Motwani. Approximate Nearest Neighbor—Towards Removing the Curse of Dimensionality. In *Proceedings of the 30th Symposium on Theory of Computing*, pages 604–613, 1998.

[10] Y. Matsuo and M. Ishizuka. Keyword Extraction from a Single Document using Word Co-ocurrence Statistical Information. *International Journal on Artificial Intelligence Tools*, 13(1):157–169, 2004.

[11] Sven Meyer zu Eißen and Benno Stein. The AISEARCH Meta Search Engine Prototype. In Amit Basu and Soumitra Dutta, editors, *Proceedings of the 12th Workshop on Information Technology and Systems (WITS 02), Barcelona*. Technical University of Barcelona, December 2002.

[12] John P. Nolan. Stable Distributions—Models for Heavy Tailed Data. http://academic2.american.edu/~jpnolan/stable/stable.html, 2005.

[13] Benno Stein. Fuzzy-Fingerprints for Text-Based Information Retrieval. In Klaus Tochtermann and Hermann Maurer, editors, *Proceedings of the 5th International Conference on Knowledge Management (I-KNOW 05), Graz*, Journal of Universal Computer Science, pages 572–579. Know-Center, July 2005.

[14] Benno Stein and Sven Meyer zu Eißen. Near Similarity Search and Plagiarism Analysis. In M. Spiliopoulou, R. Kruse, C. Borgelt, A. Nürnberger, and W. Gaul, editors, *From Data and Information Analysis to Knowledge Engineering*, pages 430–437. Springer, 2006.

[15] Roger Weber, Hans-J. Schek, and Stephen Blott. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *Proceedings of the 24th VLDB Conference New York, USA*, pages 194–205, 1998.

[16] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing gigabytes (2nd ed.): compressing and indexing documents and images*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[17] Oren Zamir and Oren Etzioni. Web Document Clustering: A Feasibility Demonstration. In *SIGIR'98: Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 46–54, University of Washington, Seattle, USA, 1998.