Comparison of Standard and Zipf-Based Document Retrieval Heuristics

Benjamin Hoffmann
Universität Stuttgart,
Institut für Formale Methoden der Informatik
Universitätsstr. 38, D-70569 Stuttgart, Germany
hoffmann@fmi.uni-stuttgart.de

September 15, 2010

Abstract

Document retrieval is the task to retrieve from a possibly huge collection of documents those which are most similar to a given query document. In this paper, we present a new heuristic for inexact top K retrieval. It is similar to the well-known index elimination heuristic and is based on Zipf's law, a statistical law observable in natural language texts. We compare the two heuristics with regard to retrieval performance and execution time. Therefore, we use a text collection consisting of scientific articles from various computer science conferences and journals. It turns out that our new approach is not better than index elimination. Interestingly, a combination of both heuristics yields the best results.

1 Introduction

Today, information retrieval (IR) gains rapidly increasing interest from researchers in the area of computer science. This is mainly due to the fact that for most people, information retrieval systems (like, for example, web search engines) are the preferred means of information access. Furthermore, technical progress has made it possible to accumulate huge amounts of data. According to the 2008 annual review of Thomson Reuters, every day 15 petabytes of new data are created [2]. Thus, one needs efficient algorithms which are capable of handling huge data sets.

A common task in the context of information retrieval is the following. Given a document collection (corpus) and a query document, we want to determine documents from the collection that are most similar to the query document with respect to some similarity measure (e.g. cosine similarity). We denote this task by *document retrieval*. It appears in a wide range of applications, including for example text clustering, duplicate detection, and search engines (e.g. a "more like this" feature available in the results list) .

In this paper, we present a new document retrieval method which exploits a fact on the probabilities of intersecting sets in the *Zipf model* [9]. The Zipf model is a randomized input model for the maximal intersection problem, which is defined as follows: Given a set $\mathcal{T}$ and a database $\mathcal{D} \subseteq 2^{\mathcal{T}}$ along with a query set $q \subseteq \mathcal{T}$. We ask for a member $d \in \mathcal{D}$ having an intersection of maximal size with the query set q. In general, determining such a member is computationally expensive. It becomes efficiently feasible if the input follows the Zipf model. Loosely speaking, in the Zipf model an almost optimal answer can be found by considering only a relatively small subset of $\mathcal{T}$. In this paper, we transfer this idea to document retrieval in the vector space model and compare it with the well-known index elimination method. Our new approach accomplishes inexact top K retrieval where the selection criterion for relevant terms is the document frequency.

The remainder of the paper is organized as follows: In Section 2 we give an overview of the vector space model and introduce simultaneously the notation used. In Section 3 we state the maximal intersection problem and explain the Zipf model. (Readers familiar with these two models can immediately skip to Section 4.) In Section 4 we show how the Zipf model can be applied to document retrieval and present our new algorithm. Section 5 explains the experimental setting and compares the results obtained by our method with those obtained by index elimination. We give a conclusion in Section 6.

2 The vector space model

The most common information retrieval task is *ad hoc retrieval*. Given a collection of documents, we want from our IR system a method providing documents from within the collection that are relevant to an arbitrary set of (search) terms initiated by the user. This terms are called the *query*. In order to solve the retrieval task by our system, we have to formalize the notion of relevance. (Clearly, the relevance of a document depends on the user's information need. A formal definition tries to capture the information need.) The standard way to do this is the usage of *cosine similarity*. Therefore, we represent each document d as a vector $\vec{V}(d)$ over the *vocabulary* of terms. Usually, the vocabulary does not contain each

word occurring in our collection, but a subset of the words suited for document retrieval. These words are called *terms* (see Section 5.1). The *term frequency* $tf_{t,d}$ is the number of occurrences of term t in document d. The *document frequency* df_t is the number of documents in the collection that contain term t. The components of a document vector $\vec{V}(d)$ are the tf-idf weights of the document's corresponding terms. The tf-$idf_{t,d}$ weight of term t in document d is the product of its term frequency $tf_{t,d}$ and its inverse document frequency $idf_t = \log \frac{N}{df_t}$. Intuitively, this weighting scheme is reasonable since a term gets a high score if it occurs many times in a document and only in a small number of documents (thus, discriminating those documents from the rest). The similarity between two documents is defined as the cosine similarity of their vector representations $\vec{V}(d_1)$ and $\vec{V}(d_2)$,

$$\text{sim}(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{|\vec{V}(d_1)||\vec{V}(d_2)|}.$$

(The numerator represents the dot product of the two vectors.) Clearly, this concept of similarity can easily be transferred to measure similarity between a document and a query by representing the query as a vector over the vocabulary. Consequently, we use the cosine similarity as a measure of relevance of a document to the query. Note that document retrieval differs from ad hoc retrieval since the query itself is a document and not a set of user-initiated search terms.

Now that we have an exact mathematical notion of relevance, we must explain how relevant documents are retrieved. This is done by the usage of an *inverted index*, which is a data structure that stores for each vocabulary term t a so called *postings list*. A postings list is a list of those documents containing term t. In order to answer a query, we calculate for each document that contains at least one search (query) term its cosine similarity with the query. As result, we present the K top-scoring documents. Here, we call this algorithm the *standard* retrieval algorithm. The computation cost of it can be lowered if we do not demand that precisely the K documents with the highest scores are returned. A common heuristic, called *index elimination*, is to consider only the postings lists of terms whose idf exceeds a certain value and to return a ranked list of the documents contained in the union of all such postings (*inexact top K retrieval*). For a more detailed treatment of the vector space model and a comprehensive list of references, we refer the reader to [10, 6].

3 Maximal intersection queries and the Zipf model

Let $\mathcal{T} = \{t_1, t_2, \ldots, t_m\}$. The *maximal intersection (MI) problem* is the following:

Input: A database of n finite sets $\mathcal{D} = \{d_1, \ldots, d_n\} \subseteq 2^{\mathcal{T}}$.

Query: Given a finite query set $q \subseteq \mathcal{T}$, we ask for a set $d \in \mathcal{D}$ having a maximal intersection with q, that is, $|q \cap d'| \leq |q \cap d|$ for all $d' \in \mathcal{D}$.

The k-*approximate* version of this problem asks for a set $d \in \mathcal{D}$ such that $|q \cap d'| \leq k \cdot |q \cap d|$ for all $d' \in \mathcal{D}$, $k > 1$.

The exact version is as hard as the *nearest neighbor search* problem in the Hamming cube (see [7]), for which no preferable solution (i.e., one using $(n \cdot m)^{O(1)}$ storage and having $m^{O(1)}$ search time) is known so far.[1] Moreover, it is believed that no such solution exists at all [3]. However, for many applications like text clustering, recommendation systems, and the distribution of online advertisements it suffices to solve the approximate version.

In real life, most problem inputs are not random, but exhibit a certain underlying structure. An obvious example are natural language texts which exhibits *Zipf's law* [12]. Zipf's law states that the frequency f of a word is inversely proportional to its rank r in the frequency table, that is,

$$f \propto \frac{1}{r}.$$

Or, in other words, there exists a constant c such that $f \cdot r \approx c$. In the next paragraph we will explain a randomized input model for the MI problem based on Zipf's law, called the *Zipf model*.

3.1 The Zipf model

Since the Zipf model is motivated by an empirical law which is mainly observed in natural language texts, we deal in the following with *documents* instead of sets. The set $\mathcal{T}$ can be considered as the vocabulary.

The Zipf model was introduced in [9] and further developed in [8, 7]. It is a probabilistic process for generating a document collection which follows Zipf's law. Each document is generated by choosing terms that will be contained in it. Each term is chosen independently and the term t_i is chosen with probability $\frac{1}{i}$. Every document is also generated independently. (Note that by this process, each term can occur at most once in a document.) Now, if a collection is generated in this way, one can observe that the document frequencies of the terms are distributed according to Zipf's law. For inputs (i.e., document collection *and* query) following the Zipf model, the following holds:

[1] Clearly, for both problems there exist solutions. Namely, a linear scan over the database or the usage of a hash table storing for each possible query the correct answer. However, these solutions do not comply with the complexity constraints of a preferable solution.

Theorem 3.1. *There exists a deterministic algorithm for the MI problem that returns a* $\frac{1+E(\varepsilon,n)}{1-\Delta(\delta,n)}$*-approximate answer with probability* $p(\varepsilon, \delta, n)$ *tending to one as* $n \to \infty$*, where* $\varepsilon, \delta > 0$ *and* $E(\varepsilon, n) \to \varepsilon, \Delta(\delta, n) \to \delta$ *as* $n \to \infty$*. The algorithm has a preprocessing time of* $\tilde{O}(nm)$ *and a query time of* $\tilde{O}(\log m + n)$ *in the average case*[2]*. The space required is* $n^{1+o(1)}$*.*

This algorithm is based on the fact that in the Zipf model a *threshold phenomenon* on the most probable intersection size holds. That is, assume the terms of each document from the collection and the query terms are ordered in descending order according to their document frequency. Then, one can show that up to a certain value s (threshold) a document matching the first $(1-\delta)s$ query terms exists with high probability, where for larger values the probability falls to nearly zero. Surprisingly, this behavior cannot only be observed for a match in the *first* terms, but also for an "arbitrary" match. The crucial observation is that for both kinds of matches the threshold values are close to each other (see Figure 1). Thus, determining a document that has a maximal common *prefix*[3] match with the query yields with high probability an almost optimal answer.

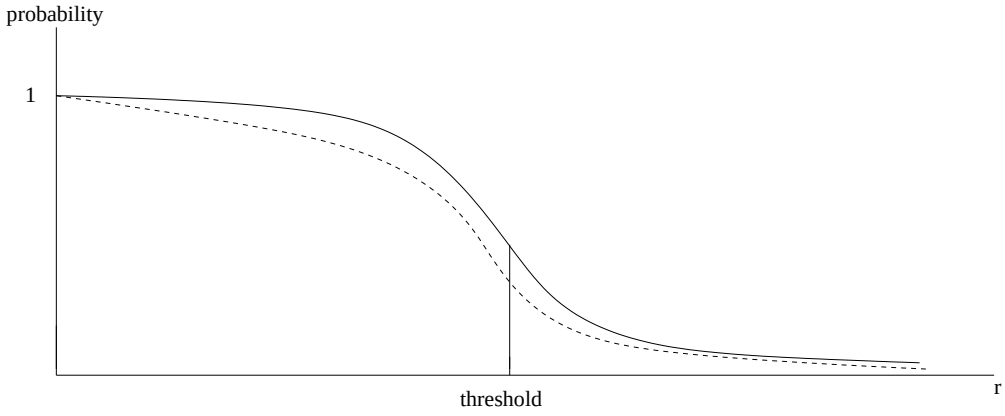

Figure 1: Exemplary probability curves for r-match (the solid line) and r-prefix match (the dashed line)

In [7], the author generalized the Zipf model such that it can cope with stop words. *Stop words* are words which occur very frequently in a text and therefore appear to be of little value in reflecting its content. In the context of the Zipf model, the notion stop word refers to terms that occur in all (or nearly all) documents. Clearly, a term which is contained in every document is irrelevant for the MI problem. It was shown that without stop words, a threshold theorem

[2] $\tilde{O}(f) := \bigcup_{k>0} O(f \log^k f)$

[3] according to the above order

analogously to the one above holds and the same deterministic algorithm can be applied. While the complexity remains the same, the approximation factor k becomes larger by a multiplicative factor depending on the collection. For a full description of the generalized model see [7].

4 Applying the Zipf model to document retrieval

So far we have introduced the "background" we will need in the following. We now explain how the Zipf model and its threshold phenomenon can be applied to document retrieval. Recall that by document retrieval we denote the problem to determine to some given query document a set of documents that are most similar to the query with respect to cosine similarity.

As mentioned in Section 2, in the vector space model index elimination (i.e., consider only the postings of terms whose idf exceeds a certain value) is used in order to speed up query processing. Now, instead of selecting query terms according to idf value, we take the terms with the highest document frequency and intersect the according postings lists. The documents in this intersection constitute the answer set and will be ranked according to their similarity with the query document. To be more precise, we apply the following three-step algorithm:

Preprocessing. Determine the vocabulary of terms (removing stop words, normalization, stemming).

Data structure. Generate an index which is sorted according to document frequency.

Query processing.

1. Preprocess the query (cf. Preprocessing).
2. Determine the answer set by intersecting the postings of the query terms in descending order according to document frequency. Stop if the set size is equal to or for the first time smaller than some predefined value.
3. Calculate the cosine similarity between each answer document and the query document.

The above is a high-level description of the algorithm. In the next section, we explain the different steps in more detail.

We derived this method from the threshold phenomenon mentioned in the last section, where the query terms with the highest document frequency suffice to determine an almost maximal matching. The fact that most documents we deal with in information retrieval are natural language texts (thus, exhibiting Zipf's law) makes this approach obvious and, moreover, justifies it. However, a problem

might be that we want to maximize cosine similarities and not intersection sizes. It is not clear that the set of answer documents chosen according to the ”high frequency” query terms they contain yield high cosine similarities. As we will see in Section 5, for our test collection this is the case.

Other discrepancies to the Zipf model are the facts that documents are usually multisets of words and that Zipf's law makes a statement about the distribution of word frequencies in a single text, and not about document frequencies in a collection. In Section 5.2.1, we discuss these issues in detail.

5 Experiments

In this section, we set out our experimental setting and compare the results of our new approach with the inverted index method.

Our test corpus consists of 1432 freely available[4] scientific articles from various conferences in the area of computer science. The main topics include algorithm theory, automata and language theory, theoretical computer science in general, and combinatorics on words. As queries, we have randomly chosen 120 texts from the corpus; all results are averaged over these queries. The texts were given in PDF format. For further processing, we converted them to the simple TXT format by the tool `pdftotext` using ASCII7 encoding.

5.1 Determining the vocabulary

First, we applied case-folding by reducing all letters to lower case. We then remove stop words since they do not reflect the content of a document and thus appear to be of little value in selecting content-similar documents. (In general, information retrieval systems remove stop words before indexing.) We applied the stop word list provided by the SMART software [1]. This list contains 571 different stop words. Given the nature of our corpus, we extended the list by the following words: abstract, computer, define, defined, definition, denote, denotes, exist, exists, general, introduction, lemma, number, paper, perform, performs, problem, proof, references, result, results, science, section, theorem, theory. In [11] Miller et al. established empirically that the average length of a stop word is 3.13 letters. Hence, in addition to the words of the list, we removed all words whose length is less than 5. We also removed special characters (e.g. @, &, ?). As a final filtering step, we performed some simple *stemming* techniques. To be more precise, we removed each 's' at the ending of a word if the predecessor of this last 's' is not equal to 's', 'i', or 'u'. For words ending

[4]Or available via access provided by the library of the University of Stuttgart.

with ”ies”, we replaced this ending by ’y’. Additionally, we applied the following replacements:

$$\text{vertices} \rightarrow \text{vertex}$$
$$\text{queries} \rightarrow \text{query}$$
$$\text{suffixes} \rightarrow \text{suffix}$$

We list some statistics of our collection in Table 1. Note that both retrieval methods use the vocabulary determined by this linguistic preprocessing.

Statistic	Value
# documents N	1,312
# words (before pp)	137,188
# terms	93,220
avg. # tokens per document (before pp)	3,404
avg. # tokens per document	1,837
avg. # terms per document	551
collection size (before pp)	51 MB
collection size	24 MB

Table 1: Collection statistics. If not explicitly stated otherwise, all values refer to the collection after preprocessing (pp).

5.2 Implementation details

5.2.1 Adapting the data to the Zipf model

As mentioned in Section 3, in a document collection following the Zipf model the terms’ document frequencies are distributed according to Zipf’s law. However, in a collection of real texts, Zipf’s law states that the *collection frequencies* (the total number of times each term appears in the collection) are distributed according to it. To see that this holds, just consider the concatenation of all texts. This means that a real text collection does not necessarily behave according to the Zipf model. And indeed, if we consider the terms from our vocabulary and plot the document frequency as a function of the rank, the resulting graph is not a line with slope -1, as it should be according to Zipf’s law (cf. Figure 2). The shape of this graph is due to the fact that the terms with the highest document frequency occur in the bulk of the documents (see Table 2, the left column) and do not follow Zipf’s law (the most frequent term occurs twice as often as the

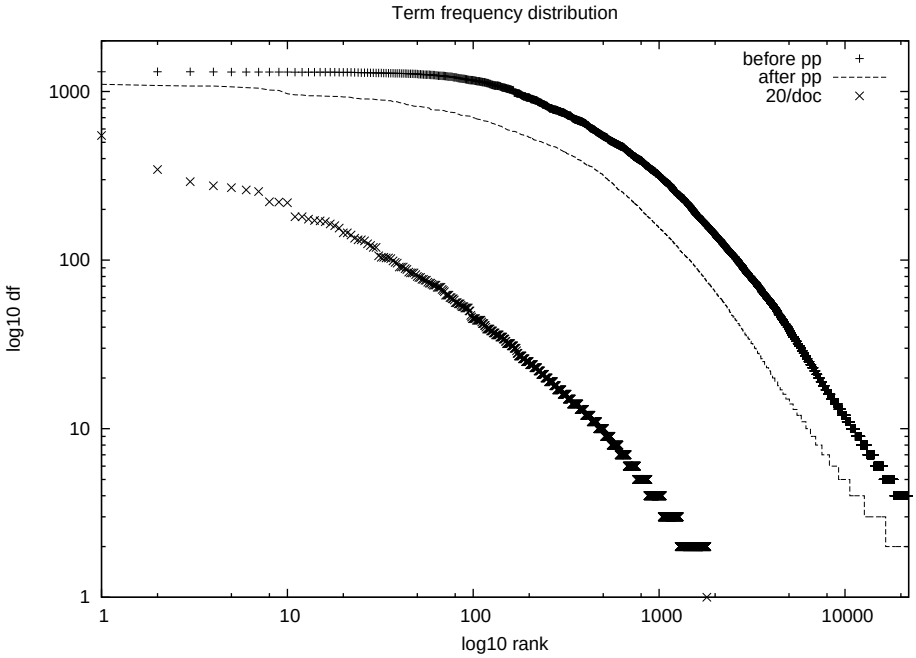

Figure 2: Rank-frequency distribution of our test corpus. Document frequency (df) is plotted as a function of the rank. On both axes, we use logarithmic scales.

second most frequent one and so on). According to our discussion at the end of Section 3, these terms are stop words in the context of the Zipf model. Thus, a possible solution is to remove these most frequent terms. However, they occur not in every document and most of them are relevant regarding the content. Therefore, dropping them might impair retrieval results, see Remark 5.1. Another indicator for this assumption provides the theoretical analysis in [7]. According to it, the approximation factor would become larger by the multiplicative factor of 241 for our corpus. Thus, we applied the following heuristic: We use from each document only the 20 most frequently occurring terms for indexing, which means only the terms which are most important in describing the content of a document. The right column in Table 2 shows the first 10 terms for this case. Note that the document frequencies are closer to Zipf's law. Also, if we plot the document frequency of the resulting term set, we get a graph which fits Zipf's law better than the graph of all terms after preprocessing (see Figure 2). Clearly, this adaption is only applied for the Zipf-based method.

Remark 5.1. *In order to verify the (theoretical) assumption that dropping stop words impairs retrieval results, we applied this approach to the test corpus. To be more precise, we applied the Zipf-based heuristic without further modifications of the vocabulary, with dropping the first 100 terms, and with dropping the first 500*

All terms	20/doc
property (1104)	algorithm (549)
assume (1086)	function (345)
function (1079)	state (292)
order (1078)	language (276)
called (1067)	graph (269)
prove (1055)	bound (261)
algorithm (1048)	finite (255)
application (1022)	probability (222)
university (1015)	system (221)
similar (970)	vertex (219)

Table 2: Most frequently occurring terms and their document frequencies.

terms. In all cases, the results were worse with respect to highest rank and precision/recall than those obtained with adaption. While the decline for the highest ranks was moderate, the decline with respect to precision/recall was significantly (for example, precision decreases by 50 % or more). For the sake of clarity, we do not list the results here explicitly. Note also that without adaption the vocabulary size is larger which results in higher execution times.

5.2.2 Index structure

Using from each document only 20 terms reduces the overall number of terms to 4447. As index structure we build a *term-document incidence matrix*. We sort the terms in decreasing order according to their document frequency. Our experiments have shown that the last matrix row considered was number 326. Therefore, it is likely that we do not need all rows. To use the first 500 rows might be a reasonable number. This observation is also supported by the threshold theorem, which yields implicitly an upper bound for the number of terms among a maximal prefix match occurs with high probability, see [7]. By coding the rows into integers, the whole matrix requires about 0.73 MB. Taking only the first 500 rows reduces the size to 82 KB (we assume that an integer contains 32 bits). Note that instead of using a matrix we could have also used a standard index.

5.2.3 Query processing

In order to process a query, we first apply to it the same preprocessing steps we applied to the corpus. Then, we take the 20 most frequently occurring query terms

and sort them in decreasing order according to their document frequency in the corpus. The answer set is determined by intersecting the matrix rows corresponding to the sorted query term sequence, starting with the terms having the highest document frequency (cf. Section 3: prefix match in the Zipf model). Note that intersecting reduces to a *binary and*. We stop this process once the size of the answer set is equal to or for the first time smaller than $0.05 \cdot N$ (for our corpus, 5% of the corpus size is a good compromise between quality of the results and fast execution time.) Finally, we rank the answer documents according to their cosine similarity with the query document.

5.3 Results

Table 3 shows experimental results the different algorithms yield on our test corpus. We list the results for the standard retrieval algorithm without[5] and with index elimination, where in the latter case we choose as idf threshold the value 2.6. Then, only postings of terms with $df \leq 131$ are considered, which means that the 12711 terms with higher df values are dropped. The threshold 2.6 is chosen so that we get approximately the same number of answer documents as the Zipf-based algorithm retrieves. Considering the last column in the Table 3, we see that

Statistic (measure: **cosine**)	Std. no elimination	Std. $idf > 2.6$	Zipf $\leq 0.05 \cdot N$
avg. sim. best answer	0.410	0.351	0.234
highest rank (avg./med./# rank 1)	1 / 1 / 120	4 / 2 / 59	19 / 6 / 22
avg. size answer set	1304	36	31
avg. # (postings lists traversed)	503	29	2
last postings list (avg./max.)	84409 / 93138		17 / 326
preprocessing time (in μs)	47.7		12.3
avg. query time (in μs)	0.211	0.013	0.008

Table 3: Retrieval results for the standard retrieval algorithm (with and without index elimination) and our new algorithm (Zipf heuristic). Similarity is measured by cosine similarity. The average value over the 120 average similarity values between each query and all documents is 0.022, the average maximal similarity is 0.410.

our new selection heuristic corresponds to high cosine similarities. However, the

[5] We list the results for the algorithm without index elimination mainly for time comparison.

standard algorithm in connection with the quite rigorous index elimination yields even better results. Most notably, in 59 of 120 cases it retrieves the document which is most similar to the query, while for our method this happens in 22 cases only.

The first two entries (cosine) of Table 4 list *precision* (prec.) and *recall* (rec.), which are defined as follows:

$$\text{precision} = \frac{\#(\text{relevant documents retrieved})}{\#(\text{retrieved documents})}$$

$$\text{recall} = \frac{\#(\text{relevant documents retrieved})}{\#(\text{relevant documents})}$$

For our tests, we set the number of relevant documents to be the 20 or 40 highest ranked documents from the corpus. For 40 relevant documents, the precision increases for both heuristics. Simultaneously, the recall decreases. Here, it is interesting that for the Zipf heuristic the decrease in recall is much smaller than for the index elimination heuristic. This means that for the Zipf heuristic the number of relevant documents retrieved scales linearly with respect to the number of relevant documents. Or, in other words, the relative increase in relevant documents is larger. However, in absolute terms index elimination retrieves more relevant documents.

Examining the execution times, we see that our new method is faster, especially preprocessing takes one fifth of the time needed by the standard algorithm. This is due to the fact that the latter method constructs a much larger index structure (93220 instead of 4447). Note that for both methods, we have calculated the weighted document vectors and for each term its idf value once during preprocessing. If space consumption is a concern (note that storing the weights require floating point numbers), the weights could also be computed during query processing at the expense of a longer query time. The difference in query times is mainly indebted to the longer query input time of the standard method (recall that here we read all query terms, while the Zipf method reads the 20 most frequent ones only). To a small amount, this difference comes also from the different number of postings lists traversed. Note that the predecessor step of counting the term frequencies in the query document is done in the same way for both methods, so we do not consider it.

5.4 Jaccard similarity coefficient

The better retrieval results index elimination yields compared to the Zipf method might arise from the fact that in the Zipf model, the objective is to maximize intersection sizes instead of cosine similarities. Originally, cosine similarity is designed to retrieve documents subject to a query consisting of terms expressing a

		Std. $idf > 2.6$		Zipf $\leq 0.05 \cdot N$	
relevant		20	40	20	40
cosine	prec.	0.132	0.200	0.073	0.137
	rec.	0.230	0.170	0.093	0.090
Jaccard	prec.	0.144	0.204	0.083	0.136
	rec.	0.250	0.173	0.110	0.093

Table 4: Average precision and recall values for both selection heuristics and similarity measures.

user's information need. If complete documents are employed as queries, there is another reasonable similarity measure: the size of the common vocabulary of two documents (as mentioned before, stop words do not reflect the content of a text so they are not considered; individual term frequency is not considered, too). Intuitively, this measure captures if two documents have nearly the same content, or at least the same topic. It is derived from the definition of non-identical duplicates stated in [5]. There, two documents are duplicates if they retain much of the same language and if at least 80% of the words in one document are contained in the other (in terms of terminology). Accordingly, we replaced the cosine similarity measure by intersection sizes. Again, the incidence matrix is built by using the 20 most frequent terms of each document. The intersection sizes are calculated using all words. Since a long document can result in a higher score just because it is longer (which increases the probability that it contains more terms from the query document), we use the *Jaccard (similarity) coefficient* $|A \cap B|/|A \cup B|$. This coefficient is a "normalized" form of the intersection size. In [4] the *resemblance* between two documents was also defined by the Jaccard coefficient. However, the author represents each document as a set of so called shingles[6]. To give the reader

Query	Answer
Approx. the cut-norm via Grothendieck's inequ. N. Alon and A. Naor, STOC '04	*Quadratic forms on graphs* Alon et al., STOC '05
Distinct distances in three and higher dimensions Aronov et al., STOC '03	*Cutting triangular cycles of lines in space* Aronov et al., STOC '03

Table 5: Example query and corresponding answer documents (both answers are optimal).

[6] A shingle is a contiguous fixed-length subsequence of tokens

some ideas that this similarity measure is reasonable, we list in Table 5 two query documents for which the Zipf-based algorithm yields the best answer documents with respect to the Jaccard coefficient. The subjects of the first query and answer document are quadratic programming methods and computations on matrices. In both texts, Grothendieck's inequality plays a central role. The query and answer document in the second entry are about geometrical problems. Obviously, each answer is relevant to its corresponding query.

Statistic (measure: **Jaccard**)	Std. $idf > 2.6$	Zipf $\leq 0.05 \cdot N$
avg. sim. best answer	0.244	0.218
highest rank (avg./med./# rank 1)	2 / 1 / 82	14 / 6 / 25
preprocessing time (in μs)	42.0	9.3
avg. query time (in μs)	0.009	0.005

Table 6: Retrieval results for the standard retrieval algorithm with index elimination and our new algorithm (Zipf heuristic). Similarity is measured by the Jaccard coefficient. Note that the average value over the 120 average similarity values between each query and all documents is 0.118, the average maximal similarity is 0.247.

Table 6 shows the results for the Jaccard measure. For both heuristics, the results are better than under cosine similarity, with index elimination showing a significant improvement (cf. table entry "highest rank"). Both heuristics need less preprocessing time since the calculation of weighted document vectors and idf values is not required. The average query time reduces also since we do not need to calculate the tf-idf weights of the query document. Precision and recall behave identical as they do under cosine similarity, whereas under Jaccard similarity most values are slightly higher.

5.5 Combining both heuristics

Examining the obtained results in detail reveals that there exist queries for which the Zipf heuristic yields better results than index elimination, even though the overall performance of the latter one is better. Thus, a natural approach is to combine both heuristics. Here, combining means to determine by each search heuristic an answer set and then taking their union as the final answer set. Compared to the "stand-alone" index elimination method, the additional storage requirement is marginal (cf. Section 5.2.2, storage requirements of the incidence matrix).

Measure		Cosine	Jaccard
avg. sim. best answer		0.364	0.245
highest rank (avg./med./# rank 1)		3 / 1 / 66	1 / 1 / 88
precision	20	0.096	0.104
	40	0.156	0.157
recall	20	0.293	0.315
	40	0.240	0.241
avg. size answer set		65	
avg. query time (in μs)		0.027	

Table 7: Retrieval results for the combination of both search heuristics. Average maximal cosine similarity: 0.410; average maximal Jaccard similarity: 0.247.

In Table 7 we list the results for our test corpus. Compared to index elimination, the improvement regarding the highest rank is quite small due to the already good results obtained by this method. However, recall improves considerably, which means we retrieve a larger number of relevant documents. The decline in precision is due to the fact that on average 65 documents are retrieved instead of 36. The doubling in execution time is also attributed to this larger number of documents.

6 Conclusion

In this paper, we examined a new method for efficient document retrieval. The method is derived from the Zipf model, which is a randomized input model for the maximal intersection problem. We conducted an experimental comparison and analysis of our new approach with the well-known inverted index technique used for computing cosine similarities in the vector space model.

We applied two different document similarity measures: cosine similarity and the Jaccard coefficient. The latter one is a normalized form of the intersection size of two sets, and thus conform to the Zipf model. Our experiments show that for both measures the inverted index technique in connection with the index elimination heuristic yields significantly better results. This is quite interesting, since the index elimination heuristic is designed to maximize cosine similarities, and not intersection sizes. Thus, it seems to be the case that query terms with a low document frequency have a strong influence on the intersection size. With regard to time complexity, our new approach is better. However, the differences

are moderate.

We obtained good results by combining both heuristics. By accepting a slightly higher execution time, this approach outperforms the standard index elimination method.

An important issue in our considerations was the adaption of the data to the Zipf model. We believe that a better adaption could improve the results.

References

[1] SMART ftp site. ftp://ftp.cs.cornell.edu/pub/smart/. Cited August 9, 2010.

[2] Thomson Reuters 2008 annual review. http://ar.thomsonreuters.com/2008/. Cited August 9, 2010.

[3] A. Borodin, R. Ostrovsky, and Y. Rabani. Lower bounds for high dimensional nearest neighbor search and related problems. In *STOC '99: Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 312–321, New York, NY, USA, 1999. ACM.

[4] A. Z. Broder. On the resemblance and containment of documents. In *In Compression and Complexity of Sequences (SEQUENCES97*, pages 21–29. IEEE Computer Society, 1997.

[5] J. G. Conrad, X. S. Guo, and C. P. Schriber. Online duplicate document detection: signature reliability in a dynamic retrieval environment. In *CIKM '03: Proceedings of the twelfth international conference on Information and knowledge management*, pages 443–452, New York, NY, USA, 2003. ACM.

[6] D. A. Grossman and O. Frieder. *Information Retrieval: Algorithms and Heuristics*. Springer, second edition, 2004.

[7] B. Hoffmann. *Similarity Search with Set Intersection as a Distance Measure*. PhD thesis, University of Stuttgart, 2010.

[8] B. Hoffmann, M. Lifshits, Y. Lifshits, and D. Nowotka. Maximal intersection queries in randomized input models. *Theor. Comp. Sys.*, 46(1):104–119, 2010.

[9] B. Hoffmann, Y. Lifshits, and D. Nowotka. Maximal intersection queries in randomized graph models. In *CSR*, pages 227–236, 2007.

[10] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[11] G. A. Miller, E. B. Newman, and E. A. Friedman. Length-frequency statistics for written english. *Information and Control*, 1(4):370–389, 1958.

[12] G. K. Zipf. *Human behavior and the principle of least effort*. Addison-Wesley, 1949.