Result-Size Estimation for Information-Retrieval Subqueries

Guido Sautter
KIT
Am Fasanengarten 5
76128 Karlsruhe
+49-721-608-4066
sautter@ipd.uka.de

Klemens Böhm
KIT
Am Fasanengarten 5
76128 Karlsruhe
+49-721-608-3968
boehm@ipd.uka.de

Andranik Khachatryan
KIT
Am Fasanengarten 5
76128 Karlsruhe
+49-721-608-8131
khachat@ipd.uni-karlsruhe.de

ABSTRACT

Estimating the approximate result size of a query before its execution based on small summary statistics is important for query optimization in database systems and for other facets of query processing. This also holds for queries over text databases. Research on selectivity estimation for such queries has focused on Boolean retrieval, i.e., a document may be relevant for the query or not. But with the coalescence of database and information retrieval (IR) technology, selectivity estimation for other, more sophisticated relevance functions is gaining importance as well. These models generate a query-specific distribution of the documents over the [0, 1]-interval. With document distributions, selectivity estimation means estimating *how many documents* are *how similar* to a given query. The problem is much more complex than selectivity estimation in the Boolean context: Beside document frequency, query results also depend on other characteristics such as term frequencies and document lengths. Selectivity estimation must take them into account as well. This paper proposes and evaluates a technique for estimating the result of retrieval queries with non-Boolean relevance functions. It estimates discretized document distributions over the range of the relevance function. Despite the complexity, compared to Boolean selectivity estimation, it requires little additional data, and the additional data can be stored in existing data structures with little extensions. Our evaluation demonstrates the effectiveness of our technique.

Categories and Subject Descriptors
H.3 INFORMATION STORAGE AND RETRIEVAL: H.3.3 Information Search and Retrieval; Search process

General Terms
Algorithms, Measurement, Experimentation, Theory

Keywords: Selectivity estimation, vector space model, text retrieval

1. INTRODUCTION

Problem statement. Information retrieval (IR) deals with large collections of text objects. There typically are thousands or even millions of documents with some non-zero relevance for a given query. Since this is too much for a user to sift through, IR systems rank documents by their relevance for the query. This is difficult with the so-called Boolean retrieval model. It defines the relevance of a document for a given query only in terms of whether or not the document contains a query term. To facilitate ranking, more sophisticated models like the vector-space retrieval model (VR model; [23, 24]) assign a document a more informative relevance value[1] with regard to a given query, usually out of [0, 1], computed by the *relevance function*. In non-Boolean information retrieval models, the relevance value is also referred to as the *retrieval-status value (RSV) of a document*. Various relevance functions are conceivable. Most of them rely on vector-space proximity measures and make use of the following features of both the query terms and the documents: The **inverted document frequency (idf)** of the query terms gives higher weight to more distinctive terms. The **term frequency (tf)** of a query term per document gives higher importance to documents containing the query term more often. The **length** of the documents is used to normalize the term frequency.

In a database, estimating the selectivity of a query based on small summary statistics right away is important to process it efficiently, as explained below. Existing selectivity-estimation techniques for alphanumeric data, however, only target the Boolean retrieval model. In other words, estimating the selectivity of a query is estimating the number of result documents. In the non-Boolean retrieval models examined here, in turn, it is estimating the document distribution over the range of the relevance function. The concern of this paper is the design of an effective and efficient technique for this estimation, based on small summary statistics. ‘Large' data structures like disk-based indices cannot be used for this purpose – accessing them would be too time-consuming.

Applications. Selectivity estimation is important for IR queries, for the following reasons:

Query planning: The retrieval query may be a sub query of a more complex (database) query which needs to be optimized, i.e., an execution order needs to be determined. Query optimization typically requires selectivity estimates.

***Example:** Consider a Query q against a Document Collection D. It consists of predicates p_1 and p_2, i.e., $q = p_1$ AND p_2. Let p_1 be a ‘conventional' predicate, e.g.,* Author LIKE ‘Smith’, *and let p_2 be a vector-space proximity predicate, e.g.,* proximity(Text, {‘John’, ‘Doe’}) > 0.1. *Suppose query evaluation relies on indi-*

[1] Note that we use the term *relevance* in a sense that probabilistic retrieval models refer to as *score*. Based on the score, probabilistic models compute the probability of relevance.

ces, i.e., the system can either look up all documents satisfying p_1 or those satisfying p_2 as a first step. (The next step is that it evaluates the remaining predicate on each document in the lookup result.) If the lookup result is much smaller in one case, then this predicate should be evaluated first. This is to access and process fewer rows. That is, the sooner we can eliminate irrelevant rows, the more efficient is resource allocation, and system performance increases. Clearly, this decision requires an estimate of the number of documents from D satisfying p_1 and of those satisfying p_2.

User assistance: The idea is to let the user know as early as possible if his query is too broad or too specific. A search engine can then suggest the user to modify his query.

Query expansion: Query expansion aims to improve retrieval results, e.g., by adding synonyms of the query terms to the query. To decide whether query expansion for a particular query is advantageous, selectivity estimation is necessary. If there are already many documents relevant for the original query terms, even more exhaustive query results might not be desirable.

Online query processing: When the document collection is large, processing a retrieval query might take a long time. Knowing the approximate result distribution a-priori allows for determining a threshold RSV_t: a document with an RSV greater than RSV_t is likely to be a top result. It can then be displayed right away, before query processing is completed.

Normalizing relevance values: RSV in vector-space retrieval and other retrieval models tend to be small; the result distribution is skewed towards the 0 end of the [0, 1]-interval. With other search techniques such as geographic or temporal proximity, RSV may be more evenly distributed over the [0, 1]-interval. When combining VR with such other kinds of search in a complex query, the RSV need to be normalized [8]. Having an estimate of the result distribution before query execution allows estimating the normalization factor a priori, and thus online query processing.

Example: *Consider again a Query q against a Document Collection D. Let q consist of predicates p_1 and p_2, i.e., $q = p_1$ AND p_2. Let p_1 be a numerical proximity predicate, e.g.,* Date NEAR 'Dec 25th'. *Let p_2 be a sub-query in the vector-space model, e.g.,* RSV(Text, {'Santa', 'Claus'}) > 0.1. *Let the document collection be a newspaper archive. Then the documents in the result of p_1 (R_1) will be distributed close to uniformly over [0, 1], while the ones in the result of p_2 (R_2) will be skewed towards the 0 end of this interval. To compute the final result from R_1 and R_2, the relevance values of the documents in these sets have to be adjusted to have roughly the same distribution. Otherwise, the overall relevance value will more or less depend on p_1 alone.*

All these applications require selectivity estimation data structures to reside in main memory so they are accessible without disc reads in the query optimization phase.

Contributions. Existing selectivity-estimation techniques only estimate the number of relevant documents (result cardinality), but not the distribution of RSV values. This paper presents a technique for estimating this distribution for a given query in the vector-space model. It is independent of the actual relevance function. A key insight of ours is that mechanisms that incur only very little overhead suffice to estimate selectivity reliably. More specifically, our contributions are as follows:

Prediction of Result Distribution. We show that – for common relevance functions, namely the cosine measure, the Dice and the Jaccard coefficient – the result distribution for a query can be computed from the distributions for the individual query terms. This is not trivial, since one cannot compute the proximity values of two vectors from the proximity values of the individual vector dimensions in a straightforward way. In line with existing work on selectivity estimation [5, 12, 16], our computation assumes that the distributions for the individual query terms are independent. However, [13] does consider term correlations when estimating the selectivity of queries in the Boolean model. We expect that [13] can be generalized to document distributions as well, similar to the way this current paper leverages existing techniques that rely on the independence assumption.

Memory-Efficient Representation of Statistics Data. Our technique discretizes the range of the relevance function and then estimates the number of documents per interval. Compared to existing estimation algorithms for Boolean queries, our technique requires only little additional data. One additional counter per index term is sufficient for accurate estimates of discrete document distributions with many intervals, compared to plain document frequencies. We achieve this by describing the document distributions with approximation functions, i.e., curve fitting or interpolation. We say how to upgrade the data structures used with existing selectivity-estimation techniques to store the few extra counters required for our technique. The nodes of a count suffix tree, for instance, can easily store two or three counters instead of one. Thus, our technique has the same scalability characteristics as the techniques for selectivity estimation in the Boolean model. See Section 4. All these techniques have a small memory footprint, and the data required can permanently reside in main memory, which is necessary for the scenarios outlined before.

Choice of Optimal Parameter Values. Our estimation technique for document distributions has various *endogenous parameters*: the definition of the intervals we discretize the document distributions to, the choice of the approximation function, and the allocation of the counters that can be stored per index term. The optimal choice of these parameters is not obvious. In particular, it depends on *exogenous parameters*: the proximity measure serving as the relevance function, and the number of counters that can be stored per index term. We conduct systematic experiments to determine the optimal choice of the endogenous parameters for given exogenous ones. Our experiments result in the important insight that, with a good choice of parameters, simple approximation functions like parabolas yield good estimates. In particular, the average estimation errors are in the range of the ones reported for estimation techniques for Boolean queries [5, 6, 16].

This paper presents our technique based on the vector-space retrieval model. Discussing the technique for other non-Boolean retrieval models would exceed the scope of this paper. Nevertheless, our work is helpful for these other retrieval models as well. In particular, this is because many more advanced retrieval models use the RSV from VR as a basis for their computations.

Paper outline. Section 2 presents related work. In Section 3, we analyze the vector-space retrieval model, derive a mathematical model for selectivity estimation and specify which statistical data it requires. Section 4 discusses how this additional data may be kept in existing data structures with little overhead. Section 5 is a thorough evaluation of our new technique. Section 6 concludes.

2. RELATED WORK

In this section, we discuss related work on selectivity estimation, on ranked query results, and on other retrieval models.

Parametric methods [19, 27] are the simplest approach to selectivity estimation: Assuming a certain statistical distribution of the values of an attribute, they estimate the selectivity using the parameters of the distribution. Such methods are not applicable to alphanumeric data. Thus, they cannot be used for estimating the selectivity of queries in the Boolean retrieval model. But, as we will see later, for the documents containing a given term, the distribution of these documents according to their individual term frequencies for that term are close to exponential functions. Given the document frequency of a term and the total number of its occurrences in the document collection, we can exploit this to estimate the number of documents per term frequency.

Histograms [11, 23] are statistical representations of the frequency distributions of attribute values over their domain. They split the latter into buckets, and each bucket counts the actual values in it. In an equi-width histogram, all buckets cover an equal portion of the domain. The handling is easy, but accuracy is rather poor if the distribution is skewed. Equi-depth histograms counter this effect by allowing the buckets to have different widths, and enforce the buckets to have similar levels. There also are more sophisticated types of histograms to deal with special cases: Compressed histograms [24] reduce memory requirements, and wavelet histograms [21] deal with attribute correlations. For alphanumeric data, however, classical histograms may not always be the ideal choice, given Count Suffix Trees (see below). This is because alphanumeric data is difficult to discretize without loosing information [14]. In addition, the distribution is skewed, regarding the power set of the alphabet as the data domain.

Originally, **Suffix Trees** were developed as indexing structures for wildcard searches over alphanumeric data [28]. [16] was first to use them for selectivity estimation of alphanumeric predicates. They added a frequency counter and created the Count Suffix Tree (CST). Today, CST are widely used for this purpose, with many algorithms based on them, e.g., [5, 12, 13].

However, the data structures as presented so far can only estimate individual selectivity values and counts. They require some adaptation to be applicable to the estimation of distributions. We propose the necessary extensions in Section 4.

Sampling [22] is an approach to selectivity estimation that does not require any statistics data behind it. Instead, it randomly retrieves a representative sample of data items and extrapolates the data computed from it to the size of the entire collection. It works in any case, for every possible a-priori information need, but also induces high effort if many data items have to be taken into account before the probability of an error is sufficiently low. Variants like Block Sampling [17, 18] reduce the effort for retrieving the sample of items. Text documents, however, are very big data items, compared to tuples in a typical relational database. This results in very high computational effort for obtaining a sufficiently large sample: Chaudhuri [4] assumes a disc block size of 8 KByte, which cannot hold more than two average text documents. In the worst case, documents occupy more than one disc block, which further increases the effort. If the data distribution is very skewed, even more data items need to be sampled to obtain a good estimation result [2]. Even if blocks are larger today (up to 2 MByte), the picture is the same: First, the number of documents per block would still be too low to draw significant benefit from block sampling. Second, document collections grow rapidly as well. One would have to sample more documents for the sample to yield accurate estimates. So the number of blocks to read would still be too large.

Top-K queries [3] deal with ranked query results and the presence of a score function, which is similar to the relevance function in VR. But they focus on multi-attribute relations in databases, not on a-priori estimation of the distribution of results.

Probabilistic retrieval models [15] rank documents based on their probabilities of being relevant to a given query. These probabilities are related to, but not identical with the scores computed in the VR model. In particular, probabilistic models start from scores and compute probabilities by weighting them, based on other statistical data, for instance, inverse document frequencies, more complex quantitative characteristics of the document collection, and relevance feedback from users. [20] has shown that a combination of a Gaussian and an exponential function can approximate the relevance distribution of a query result in probabilistic retrieval models, given the score distribution as input. In contrast to this, our technique estimates the score distribution before the actual query execution, solely based on the query terms and small summary statistics.

3. SELECTIVITY-ESTIMATION MODEL

In order to estimate the distribution of RSV for a query in the vector space retrieval model, we first analyze how such a distribution is computed (Section 3.1). We then introduce some notation and give examples (Section 3.2). Finally, we explain which parts of the input to the computation need to be estimated in order to estimate the RSV distribution (Section 3.3).

3.1 RSV Computation

The vector space retrieval model [25, 26] represents both documents and queries as vectors in an m-dimensional vector space, where m is the number of distinct index terms in the document collection. The elements of the vectors are the term frequencies of the individual index terms in the documents, or the frequencies of the query terms in the queries, respectively. Based on this representation, the relevance r of a document d with regard to a query q is defined as the value according to some vector space proximity measure. More formally, the proximity measure in use defines a relevance function as follows:

$$r(d,q) := \begin{pmatrix} tf(d,t_1) \\ \dots \\ tf(d,t_m) \end{pmatrix} \otimes \begin{pmatrix} tf(q,t_1) \\ \dots \\ tf(q,t_m) \end{pmatrix}$$

where $\otimes$ is some proximity measure. Commonly used proximity measures are the inner scalar product SP, the cosine measure CM, the Dice coefficient DC, and the Jaccard coefficient JC:

$$\vec{d} := \begin{pmatrix} d_1 \\ \dots \\ d_m \end{pmatrix} := \begin{pmatrix} tf(d,t_1) \\ \dots \\ tf(d,t_m) \end{pmatrix}, \vec{q} := \begin{pmatrix} q_1 \\ \dots \\ q_m \end{pmatrix} := \begin{pmatrix} tf(q,t_1) \\ \dots \\ tf(q,t_m) \end{pmatrix}$$

$$SP(\vec{d},\vec{q}) := \vec{d} \times \vec{q} := \sum_{i=1}^{n} d_i \cdot q_i , \quad CM(\vec{d},\vec{q}) := \frac{\vec{d} \times \vec{q}}{\| \vec{d} \| \cdot \| \vec{q} \|},$$

$$\mathrm{DC}(\vec{d},\vec{q}) := \frac{2\cdot\vec{d}\times\vec{q}}{\vec{d}\times\vec{d}\cdot\vec{q}\times\vec{q}}, \qquad \mathrm{JC}(\vec{d},\vec{q}) := \frac{\vec{d}\times\vec{q}}{\vec{d}\times\vec{d}+\vec{q}\times\vec{q}-\vec{d}\times\vec{q}}$$

When processing a query, a retrieval system computes the relevance function for the query and all documents in the collection and then sorts the documents according to their relevance value. This results in a distribution of the documents over the range of the relevance function. In case of the scalar product, this range is the natural numbers. The range of the other proximity measures is the interval [0, 1] over the real numbers.

3.2 Discretized Document Distributions

To formalize the computation, we first introduce some notation. We refer to the document collection as D and to an individual document as d. Let RSV be the relevance function, e.g., one of the proximity measures presented in Section 3.1. RSV(d, q) denotes the retrieval status value for a given Document d and a given Query q. The execution of q assigns a (query-specific) RSV to each document. I.e., it distributes the documents over the range of the relevance function according to their individual relevance values. We refer to such a distribution as a *document distribution* from now on. In order to reduce the number of distinct relevance values to handle, we bucketize the document distributions, similarly to [1]. Let n be the number of buckets we discretize the document distributions to. Let $b_0, \ldots, b_n$ be the boundaries between the buckets, with $b_0 := 0$ and $b_{b-1} < b_b$. Further, $b_n = 1$ for all measures except SP. Then we refer to the individual buckets as B_b, $b \in \{1, \ldots, n\}$. Each bucket B_b contains the documents whose RSV lies in $(b_{b-1}, b_b]$. In addition, we fix B_0 to [0, 0].

We define function buck(d, q), which assigns Document d to a bucket in the result distribution for Query q:

$$\mathrm{buck}(d,q) = b :\Leftrightarrow \mathrm{RSV}(d,q) \in B_b$$

We also define function level(b, q), which yields the level of Bucket B_b (the number of documents in it) in the document distribution resulting from Query q:

$$\mathrm{level}(b,q) := \left|\{d \in D \mid \mathrm{buck}(d,q) = b\}\right| \text{ or}$$

$$\mathrm{level}(b,q) := \left|\{d \in D \mid \mathrm{RSV}(d,q) \in B_b\}\right|$$

Now we can formally describe the minimum value $\mathrm{RSV}_k(q)$ of buck(d, q) for a document in the top-k result as:

$$\mathrm{RSV}_k(q) := \max\left\{b \in \{1,\ldots,n\} \mid \sum_{i=b}^{n} \mathrm{level}(i,q) \geq k\right\}$$

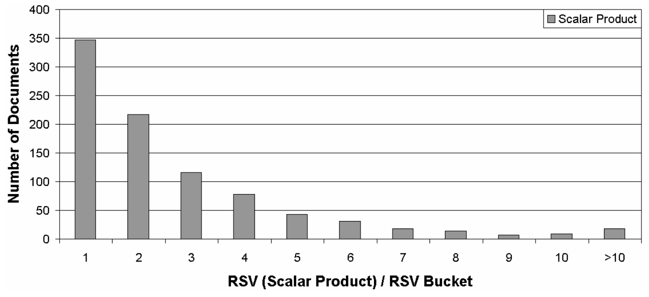

Figure 1: Distribution of documents by their RSV

To illustrate the notation, we now discuss the result distribution of an example query q. We first take a look at the un-normalized case, where the scalar product is used as the relevance function.

Figure 1 graphs level(b,q) for our example query. For $b \in \{1, \ldots, 10\}$, it shows level(b,q) individually while for clarity we have conflated the mostly empty buckets 11 to n into one bucket >10.

Given such a distribution, it is easy to see that RSV_{20} is 10 for our example query: level(10,q) + level(>10,q) = 25 (≥20), while level(>10,q) = 18 (<20).

We now illustrate the normalized case, where the relevance function yields values from [0, 1] instead of natural numbers.

Figure 2 graphs the bucketized distributions resulting from our example query for the CM and DC, and for different numbers of buckets (n). We use 50, 20, and 10 equi-width buckets here. In addition, we have conflated the mostly empty rightmost buckets to give more space to the leftmost ones, which contain more documents. Note that the ordinate scale in the three figures is logarithmic. Using the same method as above, we can find out RSV_{20} for our example query (Table 1).

Table 1. RSV_{20} values for a given query

n		10	20	50
RSV_{20}	CM	0.6	0.65	0.68
	DC	0.2	0.25	0.26

3.3 Estimating Document Distributions

Using the notations and the bucketization from the previous section, estimating the document distribution resulting from a Query q in the vector space model reduces to estimating level(b, q) for each $b \in \{1, \ldots, n\}$. In other words, we estimate the number of documents d for which $\mathrm{RSV}(d, q) \in B_b$.

To base our estimation on a query-independent summary of the document collection, we leverage the following idea: We estimate the selectivity of a multi-term query as the product of the selectivity values of the individual query terms stored in a data structure. We strive to estimate the result distribution of a multi-term query based on the distributions for the individual query terms. We then convolute the document distributions for the individual query terms to obtain the document distribution for the complete query. This approach relies on the independence assumption. Besides generalizing [13], an option to deal with term correlations is to introduce correlation factors to the convolution. We plan to address this issue in future work.

The decomposition of the document distribution for a multi-term query into document distributions for individual query terms is correct if RSV(d, q) can be computed from the individual $\mathrm{RSV}(d, t_i)$, where the t_i are the individual query terms. Computing the vector space proximity measures from the individual elements of their input vectors is not straightforward, compared to Boolean retrieval. There, a document simply contains all terms of a query if it contains each one of them. In order to facilitate term-by-term estimation for vector-space retrieval, we transform the proximity measures into a form that allows for computing RSV(d, q) from the individual $\mathrm{RSV}(d, t_i)$. To do so, we first decompose and convolute the non-normalized scalar product. Think of a two-term query q comprising two terms t_1 and t_2. Then the only elements of the vector representation of a document d that contribute to the RSV are $\mathrm{tf}(d,t_1)$ and $\mathrm{tf}(d,t_2)$. This is because all elements of the vector representation of q are 0, except for $\mathrm{tf}(q,t_1)$ and $\mathrm{tf}(q,t_2)$, which are 1. The level of the B_b in the resulting document distribution then becomes

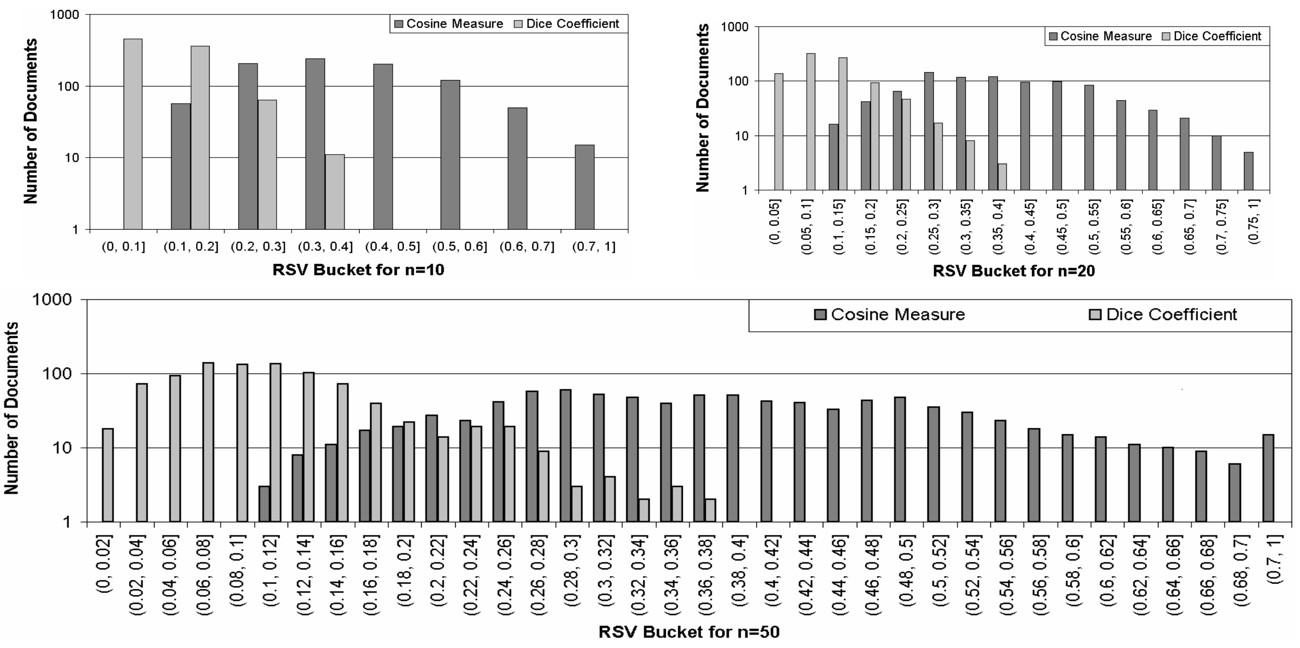

Figure 2: Bucketized distribution of documents by their RSV

$$\text{level}(b,q) := |\{d \mid d \in D, \vec{d} \times \vec{q} \in B_b\}|$$

Inserting the actual vector representation of q simplifies this to

$$\text{level}(b,q) := |\{d \mid d \in D, \vec{d} \times \begin{pmatrix}1\\1\end{pmatrix} \in B_b\}|$$

and then to

$$\text{level}(b,q) := |\{d \mid d \in D, (\text{tf}(d,t_1) + \text{tf}(d,t_2)) \in B_b\}|$$

Suppose that we have generated the estimate distributions for t_1 and t_2 from statistics data independently. We then convolute these two distributions to obtain the estimate of the complete result distribution for both terms (level(b,t) is the level of B_b in the distribution for term t):

$$\text{level}(b,q) = \sum_{i=0}^{b} \frac{\text{level}((b-i),t_1) \cdot \text{level}(i,t_2)}{|D|}$$

We now carry out the decomposition and the convolution for the normalized relevance functions (CM, DC, and JC). The estimation works similarly to the scalar product, except that we have to re-normalize the distribution by the size of q after the convolution step. For a two-term query q with the terms t_1 and t_2, and the Cosine Measure as the relevance function, the RSV of a document d is computed as follows:

$$\text{RSV}(d,q) = \frac{\vec{d} \times \vec{q}}{\|\vec{d}\| \cdot \|\vec{q}\|} = \frac{\begin{pmatrix}\text{tf}(d,t_1)\\ \text{tf}(d,t_2)\end{pmatrix} \times \begin{pmatrix}1\\1\end{pmatrix}}{\|\vec{d}\| \cdot \|\begin{pmatrix}1\\1\end{pmatrix}\|}$$

$$= \frac{\frac{\text{tf}(d,t_1)}{\|\vec{d}\|} + \frac{\text{tf}(d,t_2)}{\|\vec{d}\|}}{\sqrt{2}} = \frac{\text{RSV}(d,t_1) + \text{RSV}(d,t_2)}{\|q\|}$$

Consequently, we can again estimate the distributions for t_1 and t_2 independently and convolute them afterwards. We only have to normalize the distributions with ||q|| after the convolution step.

If the Dice Coefficient is the relevance function, the only difference is the normalization: The denominator is q×q instead of ||q||.

$$\text{RSV}(d,q) = \frac{2 \cdot \vec{d} \times \vec{q}}{\vec{d} \times \vec{d} \cdot \vec{q} \times \vec{q}} = \frac{\begin{pmatrix}\text{tf}(d,t_1)\\ \text{tf}(d,t_2)\end{pmatrix} \times 2 \cdot \begin{pmatrix}1\\1\end{pmatrix}}{\vec{d} \times \vec{d} \cdot \begin{pmatrix}1\\1\end{pmatrix} \times \begin{pmatrix}1\\1\end{pmatrix}}$$

$$= \frac{2 \cdot \left(\frac{\text{tf}(d,t_1)}{\vec{d} \times \vec{d}}\right) + 2 \cdot \left(\frac{\text{tf}(d,t_2)}{\vec{d} \times \vec{d}}\right)}{2} = \frac{\text{RSV}(d,t_1) + \text{RSV}(d,t_2)}{\vec{q} \times \vec{q}}$$

Only for the JC, the estimation has to be different. This is because the sum in the denominator of the Jaccard Coefficient prohibits decomposition. But exploiting the fact that ||d||>>||q||, i.e., the document is much bigger than the query, which holds for almost all cases of information retrieval, we can estimate the Jaccard Coefficient as below. This is because the q×q and d×q parts of the denominator are very small in comparison to d×d:

$$\text{JC}(\vec{d},\vec{q}) := \frac{\vec{d} \times \vec{q}}{\vec{d} \times \vec{d} + \vec{q} \times \vec{q} - \vec{d} \times \vec{q}} \approx \frac{\vec{d} \times \vec{q}}{\vec{d} \times \vec{d}} = \left(\frac{1}{\vec{d} \times \vec{d}} \cdot \vec{d}\right) \times \vec{q}$$

Using this approximation for the JC, the only difference to the DC is the factor 2 in the nominator, which we can consider in the normalization after the convolution step. Note that the convolution step is the best point to include (possibly query-dependent) term weights in the estimation. This is because in most non-Boolean retrieval models, the computation of the relevance of a document for an entire query from its relevance values for the individual query terms is when term weights are taken into account.

As a result so far, we have reduced the estimation of the bucketized result distribution of a query in the VR model to estimating the number of documents in the individual buckets for one-term queries. Given those numbers, we can compute the distribution for a query by convoluting the estimate distributions for the individual query terms and normalizing the result.

4. STORING THE BASE DATA

In the previous section, we have shown how to estimate the document distribution for a retrieval query in the VR model. The basis of our estimation technique is estimating the document distribution resulting from a one-term query. In this section, we discuss how to store the underlying statistical data.

Estimating a bucketized document distribution is the same as estimating the number of documents for each of its buckets. If we divide the relevance interval into 20 buckets (plus the zero bucket), for instance, we have to estimate 20 numbers per index term, as opposed to the one number with selectivity estimation in the Boolean model.

Even estimation techniques for the Boolean model struggle with the size of the underlying statistics: CST have to be pruned to fit in the data dictionary because database systems typically only have a small portion of main memory available for it. Sophisticated algorithms like Maximum Overlap [5] are in use to estimate entries which have been pruned. Given these memory problems with one counter per node already, it is hardly possible to store the counters for 20 buckets: The tree would have to be pruned too heavily to yield good estimates. To reduce memory requirements, this section discusses how to estimate the number of documents in the individual buckets based on very few counters.

However, storing only few counters and estimating the other ones, e.g., based on averages, is likely to induce errors: Even though the numbers of documents in the individual buckets have some regularity in the average case, we have to treat them as independent of each other. This is because for a Query q, knowing $\text{level}(2,q)$ and $\text{level}(4,q)$, for instance, does not allow to reliably infer $\text{level}(3,q)$. Using equi-width buckets, Figure 2 shows that the leftmost buckets tend to have relatively high levels, compared to the rightmost ones. But there can be significant exceptions from this trend, which induce severe errors.

Aggregation. In order to obtain upper and lower bounds for the counters we do not store explicitly, we aggregate the counters. Instead of $\text{level}(b,t)$, we therefore store the cumulative distribution $\text{Level}(b,t)$ – the number of documents in bucket b and the buckets to the right of it. If the plain scalar product is the relevance function and thus the buckets represent the natural numbers, for instance, $\text{Level}(b,t)$ is the number of documents containing term t **at least** b times.

$$\text{Level}(b,t) := \sum_{i=b}^{n} \text{level}(i,t)$$

$\text{level}(b,t)$ then is

$$\text{level}(b,t) = \text{Level}(b,t) - \text{Level}(b+1,t)$$

Regardless of the actual proximity measure serving as the relevance function, $\text{Level}(1,t)$ is the number of documents with a non-zero relevance with regard to term t. These are exactly those documents containing t at least once. In other words, $\text{Level}(1,t)$ is the document frequency $\text{df}(t)$ of term t. The latter is important in the VR model because the inverse document frequency is commonly used to weight query terms. Furthermore, it is the selectivity of term t in the Boolean retrieval model. In the following, we discuss how to use interpolation or parametric methods to approximate $\text{Level}(b,t)$ based on few counters. With interpolative approximation, the counters are the data points. With parametric methods, we fit curves to the aggregated distributions and use the counters to store the parameters of these curves.

Interpolation. To reduce the number of counters stored, we approximate $\text{Level}(b,t)$ using interpolation methods based on j data points $\{(p_1, \text{Level}(p_1, t)), \ldots, (p_j, \text{Level}(p_j, t))\}$, $p_{(i-1)} < p_i$ for all $1 < i < j$. I.e., we store $\text{Level}(p_1, t), \ldots, \text{Level}(p_j, t)$. Since $\text{Level}(1, t)$ is very important, we fix p_1 to 1 to include it in all choices of data points. In our evaluation (Section 5), we assess different interpolation methods and choices of data points.

Parametric Approximation. We have investigated the RSV distributions for the 200 most frequent index terms of the Reuters Corpus [29]. Most of the distributions are surprisingly close to exponential functions of the form $v \cdot e^{-w \cdot x}$, $v, w > 0$. Because these functions take only the two parameters v and w, using them as an approximation for the actual distribution would require only two counters to be stored.

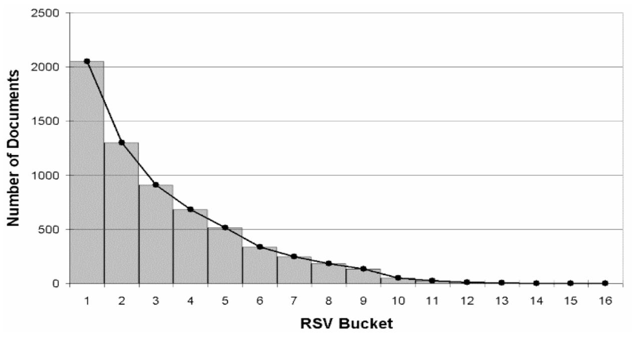

Figure 3: Level (b,"loss"), graph and sub sum

To illustrate our considerations that exploit this effect, Figure 3 displays the graph for the index term "loss" The sum of the gray columns $T(t)$ corresponds to the sub sum of the graph, i.e.,

$$T(t) = \sum_{b=1}^{n} \text{Level}(b,t)$$

If the plain scalar product is the relevance function, and thus the buckets represent the natural numbers, the sum of the columns is the number of occurrences of index term t in the collection. The formula reflects that, if a term is contained in the document more than once (say, z times), the document is counted z times.

To approximate the value of the sub sum closer to the value of the (discrete) integral, we shift the graph left by half a bucket. Now, with appropriate values for v and w, function f defined as follows is a close approximation of the graph in Figure 3.

$$f_{v,w}(x) = ve^{-w(x-\frac{1}{2})}, x \in N$$

We now derive closed forms for parameters v and w from two constraints f has to fulfill:

$$f_{v,w}(1) = \text{Level}(1,t) \text{ and } \sum_{n=1}^{\infty} f_{v,w}(n) = T(t)$$

The right side of the first constraint is the document frequency of index term t, i.e., its selectivity in the Boolean retrieval model. Using the integral of f instead of the discrete sum, the second constraint can be approximated as follows:

$$\int_1^\infty f_{v,w}(x)dx = T(t)$$

Using

$\int ve^{-w(x-\frac{1}{2})} = F_{v,w}(x) = -\frac{v}{w}e^{-w(x-\frac{1}{2})}$ and $\lim_{x\to\infty} F_{v,w}(x) = 0$,

this results in a more concise form of the same constraint:

$$F_{v,w}(1) = T(t)$$

Given these two constraints, we can now describe the parameters v and w using $Level(1,t)$ and $T(t)$:

$$v = Level(1,t)e^{\frac{Level(1,t)}{2T(t)}}, \quad w = \frac{Level(1,t)}{T(t)}$$

Our evaluation will show that this parametric method yields good approximations. Further, it requires only one additional value besides the Boolean selectivity $Level(1,t)$, namely $T(t)$. We do not further investigate how to estimate $Level(1,t)$ and $T(t)$ because existing selectivity-estimation techniques for Boolean retrieval address this problem [5].

Storing the Counters. The additional one to three counters required to estimate document distributions can be stored beside the document-frequency counter in nearly all data structures used for selectivity estimation in modern database systems. For the most frequent index terms, these data structures usually store the result cardinality for queries in the Boolean model. We leverage these data structures to store approximations of histograms that represent the document distributions resulting from queries in the VR model. These approximations require two to three counters, compared to the one counter used in the well-established techniques for the Boolean model, most notably pruned Count Suffix Trees [5, 16]. One to three extra counters per CST node are acceptable, considering the inherent structural overhead (child pointers, etc.) of the CST. In particular, the individual nodes of a pruned CST require an average of 8.5 bytes in the currently optimal implementation [9]. Then an extra 4 byte for the $T(t)$ is acceptable. Even the 8 or 12 byte for two or three extra counters fit within memory limitation if we increment the pruning threshold a little.

Summary. This section has shown how to estimate bucketized document distributions using two to four counters per index term: When using the exponential approximation, we need one extra counter for the $T(t)$, besides the one for the document frequency. For the interpolative methods, we need two to three counters in addition to the document frequency. Established data structures like Count Suffix Trees are easily extended to store these additional counters.

5. EVALUATION

In this section, we evaluate our estimation technique for document distributions, in the VR model. Note that estimating the total number of relevant documents (i.e., those with RSV > 0) is identical to selectivity estimation in Boolean retrieval. Since this issue has been investigated before [5, 14], we will not address it again.

5.1 Experimental Setup

Test Bed. As our test bed, we use the Aquaint corpus [10], which comprises roughly a million articles from three different newspapers. Its total size is around 3GB. Currently, it serves as the document base for the TREC tasks [7]. Following [5], we choose as test queries those index terms that appear in at least one percent of the corpus documents. This results in over 4000 index terms we used as test queries.

Error Metrics. In line with [5], we use the average relative error as our error metric, i.e., the average of the relative errors for the individual index terms. It quantifies the overall accuracy of an estimator as the ratio

$$|estimate - actual| \,/\, actual,$$

where $estimate$ is the selectivity estimate and $actual$ is the exact selectivity value. We also apply the correction technique proposed in [5]. It overcomes the problem that the average relative error over-penalizes errors with small actual selectivities: Specifically, if the actual selectivity is less than 100/|D|, we divide the absolute selectivity error by 100/|D|, rather than by the actual selectivity. As opposed to [5], we have several estimates per term (one for each bucket). We therefore compute the average relative error for each bucket.

5.2 Configurations Tested

Parameters. There are two exogenous parameters our estimation technique has to deal with, namely the proximity measure in use which serves as the relevance function and the number of counters which can be stored per index term. Depending on these two parameters, three endogenous parameters can be tuned: the method used to approximate the document distributions (investigated in Subsections 5.3 and 5.4), the allocation of the counters available (Subsections 5.3 and 5.4), and the definition of the buckets we discretize the document distributions to (Subsection 5.4).

Approximation Methods. Which method to use to approximate the document distributions largely depends on the number of counters we can store per index term. In particular, parametric approximation is the only option if we have only two counters per index term, since interpolation typically does not yield good results with two data points: The interpolation of two data points always yields a straight line, which is likely to result in high estimation errors, given the shape of the distributions discussed in Section 4. If three or more counters are available in turn, we can use interpolation; see below.

Allocation of Counters. For the interpolative method of estimating document distributions, we evaluate different interpolation functions and different choices of the data points to store.

The simplest approach is using equidistant data points for, e.g., {1, 3, 5, 7, …} or {1, 4, 7, 10, …}, formally

$$p_i := 1 + q \cdot (i-1), i \in \{1,\dots,j\}, q \in N$$

We do not expect this choice to be very useful, however. This is because we either (1) have to store many data points, or (2) have no data points for higher values of b, or (3) the data points for smaller b are too far apart to yield good estimates. The latter is because the $Level(b,t)$ have the highest mutual differences for small values of b. Any super-linear function can produce a choice of data points that is dense for small values of b and yields high values after few points. We have evaluated the following ones, for different values of j:

- The Fibonacci sequence, {1, 2, 3, 5, 8, 13, …}:
  $p_0 = 1, p_1 = 2, p_i = p_{(i-2)} + p_{(i-1)}, i \in \{3,\dots,j\}$
- Exponential sequences, {1, 2, 4, 8, …} or {1, 3, 9, …}
  $p_i = q^{(i-1)}, i \in \{1,\dots,j\}, q \ge 2$

- The factorial sequence, {1, 2, 6, 24, …}
  $p_i = (i-1)!, i \in \{1,...,j\}$

Interpolation Methods. For interpolating the data points explicitly stored, we have evaluated several functions. The choice of the function depends on two issues. First, the function should yield accurate estimates. Second, it should not be complex, so that its evaluation is fast.

From the complexity point of view, linear interpolation is the method of choice. It approximates $Level(b,t)$ as

$$Level(b,t) = \frac{Level(p_{(i+1)},t) \times (b - p_i) + Level(p_i,t) \times (p_{(i+1)} - n)}{p_{(i+1)} - p_i}$$

$$b \notin \{p_1,...,pj\}, p_i < b < p_{(i+1)}, i \in \{1,...,(j-1)\}$$

If b is greater than the greatest data point explicitly stored, we use auxiliary values for the interpolation:

$$p_{(j+1)} := \min(n,256), Level(p_{(j+1)},t) := 0$$

Polynomial interpolation methods (e.g., Lagrange and Newton interpolation) yield good approximations. However, their computational complexity is relatively high, so we did not consider them any further.

Since we do not need to interpolate all data points if we only want to approximate $Level(b,t)$ for a particular value of b, we also try fitting a lower-degree polynomial to a subset of our data points. For subsets consisting of two points, this is linear interpolation. For three points, the curve is a parabola. The computation of the three parameters a_0, a_1 and a_2 in the formula $f(x) = a_2x^2 + a_1x + a_0$ is a standard procedure.

Splines in general and cubic splines in particular are good approximations for arbitrary functions and are widely used. They yield better results than polynomial methods, and their computational complexity is lower. As splines are constructed of individual pieces, in theory we need only construct a single piece of the spline to approximate $Level(b,t)$ for a given n. In practice, however, a spline through two data points is always a straight line, regardless of its degree, and thus equivalent to the linear method. Therefore, spline interpolation in our setting makes sense only if $j \geq 3$, and it is likely that we will have to compute the complete spline for every estimation. This is relatively complex, compared to fitting a parabola. Therefore computation would take too long for selectivity estimation, which has to be fast. Consequently, we do not consider spline interpolation any further.

5.3 Un-Normalized Relevance Functions

In order to assess the applicability of the interpolation methods and the choices of data points, we first evaluate the estimation for the case that the un-normalized scalar product serves as the relevance function. Therefore, we have a theoretically unlimited number of buckets in the document distribution: There is no upper bound for the term frequency of a term in a document. Figure 4 shows the average relative error of the estimated numbers of documents per bucket. For interpolative methods, the graphs are labeled with the choice of data points: the numbers of the buckets whose levels were stored explicitly are in brackets.

The relative errors of most of the data point sets yield good estimates. It turns out that the relative errors are less dependent on the number of data points than on the bucket numbers actually selected for explicit storage. Choices with the data points close in lower buckets and further apart in higher buckets perform best. Regarding the interpolation method, there are hardly any differences. Both linear and parabolic interpolation yield approximately equal estimation quality. We therefore omit the graphs for the linear interpolation. The parametric approach yields acceptable estimation results as well. As mentioned, its additional advantage is that it requires only one additional counter, as opposed to the interpolation methods.

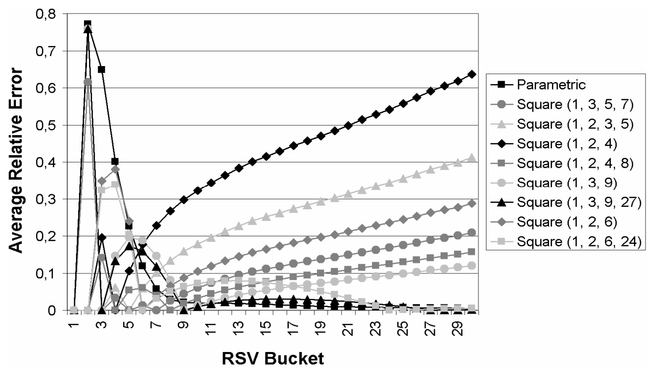

Figure 4: Average relative error for non-normalized term frequency

5.4 Normalized Relevance Functions

We now report on experimental results of our technique applied in cases where the relevance function is normalized, namely the Cosine Measure and the Dice and Jaccard Coefficients. In these cases, we have to estimate the distribution of normalized term frequencies. We evaluate different definitions of the buckets. The results for linear interpolation are slightly worse than the ones for parabolic interpolation, about 1% - 3% in every bucket. We omit them in favor of the readability of the figures.

Equi-width Buckets. Figure 6 displays the average relative error per bucket for term frequencies normalized for the cosine measure, and the quartile distribution of the relative error. The buckets in this experiment are equi-width:

$B_0 := [0,0]$, $B_b := (\frac{(b-1)}{n}, \frac{b}{n}], b \in \{1,...,n\}$, n=20

As Figure 5 shows, the number of data points is not as important as the actual choice of the buckets to store explicitly. The data point triplet (1, 2, 6), for instance, yields better estimates than the 4-tuple (1, 3, 5, 7). The parametric exponential approximation yields acceptable results as well, but they are not as good as for un-normalized term frequencies.

With the same bucket definition and the term frequencies normalized for the Dice and Jaccard coefficient, errors in the leftmost buckets are higher. The rightmost buckets in turn are mostly empty, since the distribution of these normalized term frequencies is highly skewed towards 0. Parametric estimation does not work well in this setting: It tends to overestimate severely. The more of the leftmost buckets are stored explicitly, the better the interpolative methods. The lack of data on the rightmost buckets hardly has any effect due to their low levels. In particular, *any* choice of data points storing the second bucket explicitly has resulted in comparatively low estimation errors.

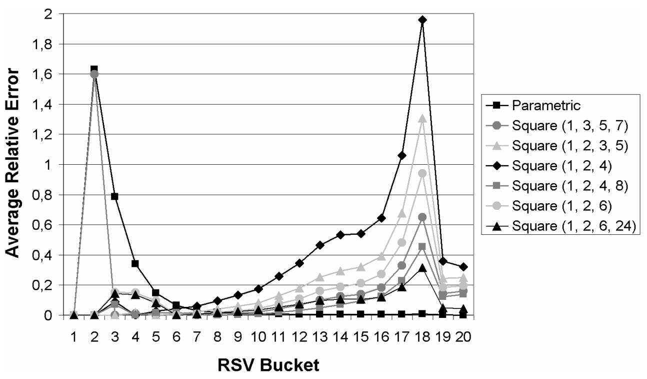

Figure 5: Average relative error for equi-width buckets, cosine normalized term frequency

Widening Buckets. To mitigate the skew of the bucket levels, we have used an alternative definition of the buckets (see below). It uses a square function for the borders. Figure 6 graphs the performance of the estimators with this definition of the buckets for term frequencies normalized for the cosine measure.

$$B_0 := [0,0],\ B_b := (\left(\frac{(b-1)}{n}\right)^2, \left(\frac{b}{n}\right)^2], b \in \{1,...,n\},\ n=20$$

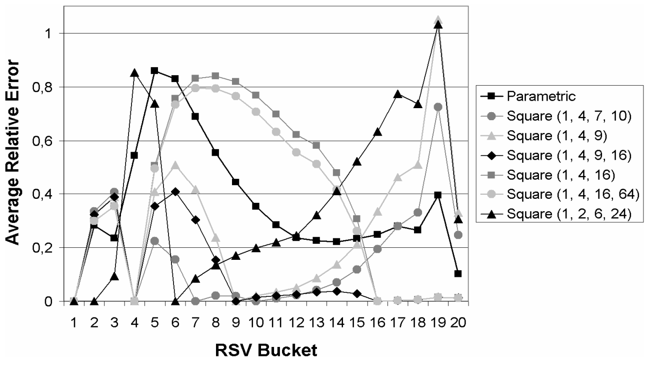

Figure 6: Average relative error for widening buckets, cosine normalized term frequency

When the cosine measure serves as the relevance function, this alternative definition of the buckets does not yield any improvement, compared to the equi-width one. This holds for both interpolative and parametric approximation.

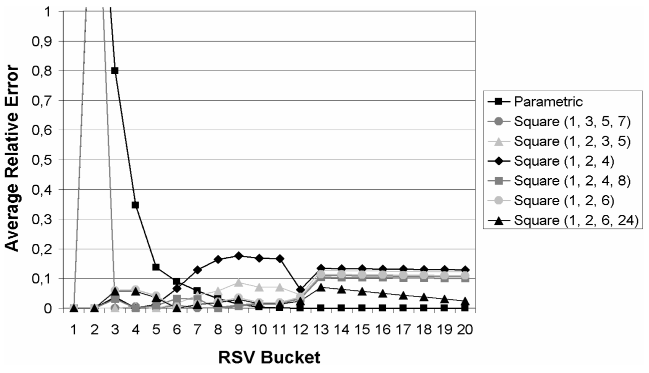

Figure 7: Average relative error for widening buckets, Dice / Jaccard normalized term frequency

If the Dice or Jaccard coefficient is used as the relevance function, however, the widening bucket definition is clearly superior to the equi-width one. This is because it distributes the documents conflated in the leftmost buckets. Figure 7 visualizes the results. Again, the accuracy of the interpolative method depends more on the actual choice of the data points than on their number. The accuracy of the parametric estimation is worse than for term frequencies normalized for the cosine measure. In particular, this holds for choices of data points that are dense in the leftmost buckets and further apart in the rightmost ones.

5.5 Summary

Table 2 summarizes the results of our evaluation. In particular, estimation accuracy depends on both the exogenous parameters given, and the choice of the endogenous parameters derived from them. The former are the relevance function in use and the number of counters that can be stored. The latter are the definition of the buckets, the choice of buckets whose level to store explicitly, and the approximation function.

The optimal definition of the buckets depends on the relevance function in use; see Table 2. The optimal choice of the approximation function depends on the amount of data that can be stored explicitly. In particular, the exponential function is the only viable option if we have only two counters per term. If we have more than two, parabolic interpolation is the method of choice. In all experiments, it has been slightly better than linear interpolation.

The choice of buckets to store in turn depends on both the approximation function and the number of counters available per term: The first one stores the level of the first bucket, the document frequency of the term. In case of exponential approximation, the second counter stores the sum of all buckets (indicated by Σ in Table 2). In case of parabolic interpolation, it is advantageous to store the level of the second bucket explicitly. The allocation of further counters depends on the definition of the buckets.

Table 2. Summary of experimental results

Exogenous Parameters		Optimal Choice of Endogenous Parameters			Fig.
Proximity Measure	Available Counters	Bucket Def.	Buckets to Store	Approx. Function	
SP	2	Natural Numbers	1, Σ	Exponential	4
	3		1, 2, 6	Parabolic	
	4		1, 2, 6, 24	Parabolic	
CM	2	Equi-Width	1, Σ	Exponential	5
	3		1, 2, 4 / 1, 2, 6	Parabolic	
	4		1, 2, 4, 8	Parabolic	
DC / JC	2	Widening	1, Σ	Exponential	7
	3		1, 2, 4	Parabolic	
	4		1, 2, 3, 5	Parabolic	

6. CONCLUSIONS

Quickly estimating the result of queries in non-Boolean retrieval models before actual query execution (i.e., based on small summary statistics) is useful in many situations. But as opposed to estimating the selectivity of a query in the Boolean model, i.e., the document frequency of the query terms, one has to estimate a distribution over the range of the relevance function. In this paper, we have presented a new approach for estimating such a distribution for a given query, with our presentation based on the vector-space retrieval model. Our approach describes the distributions for individual terms by curve fitting. The estimation technique for multi-term queries convolutes the distributions for the individual query terms. Our evaluation shows that relatively simple curves (exponential functions, parabolas) fit to few data points (two to four) are sufficient for obtaining good estimates. This holds for many common vector space proximity measures.

7. REFERENCES

[1] V.N. Anh, A. Moffat, *Impact Transformation: Effective and Efficient Web Retrieval*, in Proceedings of SIGIR 2002, Tampere, Finland, 2002

[2] S. Chaudhuri, R. Motwani, V. Narasayya, *Random Sampling for Histogram Construction: How much is enough?*, in Proceedings of SIGMOD 1998 Seattle, WA, USA, 1998

[3] S. Chaudhuri, L. Gravano, *Evaluating Top-k Selection Queries*, in Proceedings of VLDB 1999, Edinburgh, Scotland, UK, 1999

[4] S. Chaudhuri, G. Das, U. Srivastava, *Effective use of block-level sampling in statistics estimation*, in Proceedings of SIGMOD 2004, Paris, France, 2004

[5] S. Chaudhuri, V. Ganti, L. Gravano, *Selectivity Estimation for String Predicates: Overcoming the Underestimation Problem*, in Proceedings of IDCE 2004, Washington, DC, USA, 2004

[6] Z. Chen, F. Korn, N. Koudas, and S. Muthukrishnan, *Selectivity Estimation for Boolean Queries*, in Proceedings of SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pages 216- 225, Dallas, TX, USA, 2000

[7] H. T. Dang, J. Lin, D. Kelly, Overview of the TREC 2006 question answering track, in Proceedings of TREC 2006, USA, 2006

[8] R. Fagin, *Fuzzy Queries In Multimedia Database Systems*, in Proceedings of PODS 1998, Seattle, WA, USA, 1998

[9] R. Giegerich, S. Kurtz, J. Stoye, *Efficient Implementation of Lazy Suffix Trees*, Software: Practice and Experience, Volume 33, No 11, John Wiley & Sons Ltd., 2003

[10] D. Graff. The aquaint corpus of english news text. Linguistic Data Consortium, Philadelphia, 2002.

[11] Y. Ioannidis, *The History of Histograms (abridged)*, in Proceedings of VLDB 2003, Berlin, Germany, 2003

[12] H. V. Jagadish, R. T. Ng, D. Srivastava, *Substring selectivity estimation*, in Proceedings of PODS 1999, Philadelphia, PA, USA, 1999

[13] H. V. Jagadish, O. Kapitskaia, et al., *One dimensional and multi-dimensional substring selectivity estimation*. The VLDB Journal (2000) 9, pages 214-230, 2000

[14] L. Jin, C. Li, *Selectivity Estimation for Fuzzy String Predicates in Large Data Sets*, in Proceedings of VLDB '05, Trondheim, Norway, 2005

[15] K. S. Jones, S. Walker, S. E. Robertson, *A Probabilistic Model of Information Retrieval: Development and Comparative Experiments*, Information Processing & Management 36, 6 (Nov. 2000), pp 779-808

[16] P. Krishnan, J. S. Vitter, B. Iyer, *Estimating alphanumeric selectivity in the presence of wildcards*, in Proceedings of SIGMOD 1996, Montreal, Canada, 1996

[17] R. J. Lipton, J. F. Naughton, D. A. Schneider, *Practical selectivity estimation through adaptive sampling*, in Proceedings of SIGMOD 1990, Atlantic City, NJ, USA, 1990

[18] R. J. Lipton, J.F. Naughton, D.A. Schneider, S. Seshadri, *Efficient Sampling Strategies for Relational Database Operations*, Theoretical Computer Science, 116(1993), pages 195-226, Berlin, Germany, 1993

[19] C. Lynch, *Selectivity estimation and query optimization in large databases with highly skewed distributions of column values*, in Proceedings of VLDB 1988, Los Angeles, CA, USA, 1988

[20] R. Manmatha, T. Rath, F. Feng, *Modeling Score Distributions for Combining the Outputs of Search Engines*, in Proceedings of SIGIR 2001, New Orleans, LA, USA, 2001

[21] Y. Matias, M. Hill, J. S. Vitter, M. Wang, *Wavelet-based histograms for selectivity estimation*, in Proceedings of SIGMOD '98 Seattle, WA, USA, 1998

[22] F. Olken, D. Rotem, *Simple Random Sampling from Relational Databases*, in Proceedings of VLDB 1986, Kyoto, Japan, 1986

[23] G. Piatetsky-Shapiro and C. Connell, *Accurate estimation of the number of tuples satisfying a condition*, in Proceedings of SIGMOD 1984, New York, NY, USA, 1984

[24] V. Poosala, P. J. Haas, Y. E. Ioannidis, E. J. Shekita, *Improved histograms for selectivity estimation of range predicates*, in Proceedings of SIGMOD 1996, Montreal, Canada, 1996

[25] V. V. Raghavan, S. K. M. Wong, *A Critical Analysis of Vector Space Model for Information Retrieval*, Journal of the American Society for Information Science, 37(5):279-287, 1986

[26] G. Salton, M. J. McGill, *Introduction to Modern Information Retrieval*, McGraw-Hill, 1983

[27] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price, *Access path selection in a relational database management system*, in Proceedings of SIGMOD 1979, Boston, MA, USA, 1979

[28] P. Weiner, *Linear Pattern Matching Algorithms*, in Proceedings of IEEE Symposium on Switching and Automata Theory 1973, pages 1-11, 1973

[29] D. D. Lewis. Reuters-21578. http://www.daviddlewis.com/resources/testcollections/reuters21578/.