Heavy-Tailed Distributions and Multi-Keyword Queries

Surajit Chaudhuri, Kenneth Church, Arnd Christian König, Liying Sui
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
{surajitc, church, chrisko, liying.sui}@microsoft.com

ABSTRACT

Intersecting inverted indexes is a fundamental operation for many applications in information retrieval and databases. Efficient indexing for this operation is known to be a hard problem for arbitrary data distributions. However, text corpora used in Information Retrieval applications often have convenient power-law constraints (also known as Zipf's Law and long tails) that allow us to materialize carefully chosen combinations of multi-keyword indexes, which significantly improve worst-case performance without requiring excessive storage. These multi-keyword indexes limit the number of postings accessed when computing arbitrary index intersections. Our evaluation on an e-commerce collection of 20 million products shows that the indexes of up to four arbitrary keywords can be intersected while accessing less than 20% of the postings in the largest single-keyword index.

Categories and Subject Descriptors

H.3.1 [**Content Analysis and Indexing**]: Indexing Methods

General Terms

Algorithms, Performance, Experimentation

Keywords

information retrieval, partial match, power law, heavy-tailed, index

1. INTRODUCTION

At the core of Information Retrieval performance lies the ability to intersect long lists of postings quickly. Much research has centered on reordering these lists to reduce the fraction of them that is processed (e.g. [11, 3]), and on improving the processing of these intersections. Still, some queries require costly deep traversal into long lists. Consider e-commerce web sites such as Amazon and eBay with large catalogs of products. It is important to these businesses that customers can find what they want quickly. Long latencies increase abandonments and decrease sales and advertising revenue. A few long latencies can be serious, even when the average is not that bad.

The challenge is to reduce the worst-case overhead required to process arbitrary keyword queries. The database literature has studied high-dimensional indexing and partial-match queries, and found this problem to be hard in the general case for unrestricted datasets. Fortunately, datasets of interest to the SIGIR community may be easier than the general case. SIGIR datasets tend to obey power laws, which are common in natural language [4].

SIGIR'07, July 23–27, 2007, Amsterdam, The Netherlands.

1.1 Motivating Scenario

Consider an e-commerce web-site where products are searchable by name, description and category (e.g., 'woman's shoes', 'gold jewelry'. . .). Some terms are more frequent than others. The more frequent terms have relatively long inverted lists. Intersection time typically depends on frequency; intersecting long lists can lead to unacceptably long latencies.

Figure 1 shows that it can take much longer to intersect longer lists than shorter lists. The figure is based on a corpus of more than 20 million products from a large e-commerce portal [1]. We used keywords from three frequency ranges: *F* for frequent (over 900K postings), *M* for middle (about 50K postings), and *L* for low frequency (less than 1K postings). The intersections were performed using the full-text component of a commercial database system.

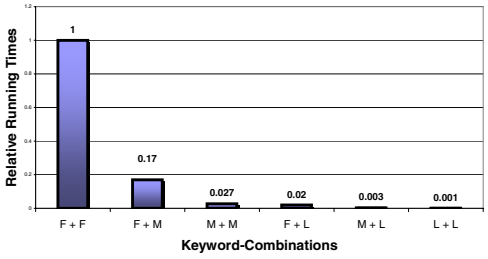

Figure 1: Pairs of frequent terms are relatively slow.

Intersections of long inverted indexes are very slow relative to other queries. Unfortunately, they are not uncommon. We analyzed a search trace of more than 3 million customer searches from the portal and found that many searches (29%) contained a frequent keyword (in the top 0.1% of keywords)[1], 8% of searches contained at least one of the 0.01% most frequent keywords and 1.8% of searches contained at least one of the 0.001% most frequent keywords. The higher the frequency of a keyword, the longer the intersection cost; this means that there is an opportunity for improving the intersection performance, as frequent keywords are commonly queried and even very frequent keywords are not rare.

1.2 Problem Statement

In the remainder of the paper, we address the following problem:

[1]To avoid skewing these results, we eliminated tail end of the vocabulary by removing prices, numbers and IDs.

Given a document collection, propose a set of indexes to materialize, so that the time for intersecting keywords does not exceed a given threshold Δ. The key challenge here is the space requirement. If space were not an issue, we could trivially archive the time requirement by materializing all possible combinations of keywords, but this would take exponential space. The additional indexes should not be larger than k times the size of the original inverted index, for a small factor of k. We will show how to materialize such a set of indexes for reasonable values of Δ (e.g., the time required to scan 20% of the largest inverted index), at least for a collection of short documents distributed by a power law.

1.3 Related Work

To address search performance, the IR community has developed numerous techniques aimed at reducing the amount of data that needs to be processed, by either ordering the postings within each index in a suitable manner, or by proposing approximations of the used scoring methods which may be computed more efficiently. We take a different approach in that our techniques are orthogonal to the specific ranking function used and address the issue of reducing the time needed to compute the intersection *exactly*.

The problem studied in this paper is related to the issue of indexing phrases in Information Retrieval. Similar to our problem scenario, full materialization of indexes of all common phrases entails prohibitive storage costs. The approach adopted in [5, 12] is to use different types of indices – inverted indices for rare words, a variant of *nextword indices* for the commonest words and a phrase index for the commonest phrases; similarly, we use different combinations of access paths depending on keyword frequencies.

The underlying indexing problem can also be phrased as an instance of the *partial-match* problem: lower bounds on the performance of partial-match queries have been studied theoretically in [8] using a cell-probe framework.

Also in the database context, various multidimensional search structures [7] have been proposed. To apply them, each keyword query could either be formulated as a high-dimensional range query over point data or as a high-dimensional point query over heavily overlapping spatial data. Either problem formulation results in an indexing problem over very sparse data in very high dimensionality (>10K dimensions). It is well known [6] that non-redundant space-partitioning techniques suffer from the "curse of dimensionality", meaning that they result in access times exceeding the cost of scanning the full data set for as little as 10 dimensions, rendering them useless for our purposes.

1.4 Our Contributions

In this paper we will present index structures and processing strategies addressing the above problems. Our work makes the following salient contributions:

(1) We introduce index structures and a query processing framework that enforce a given threshold Δ on the overhead of computing conjunctive keyword queries.

(2) We leverage the fact that the frequency distribution of natural-language text follows a power law to model the resulting index size. In particular, we show that while the number of possible l-keyword combinations relevant for indexing grows exponentially with increasing l, the underlying data distribution implies that only a small fraction of these combinations needs to be indexed, when the document sizes are small.

(3) Determining for which keyword-combination to materialize indexes may require significant I/O and main memory. To overcome this limitation, we will describe a probabilistic scheme that significantly reduces this overhead.

(4) We provide an extensive experimental evaluation of the performance on large real-life datasets.

The remainder of this paper is organized as follows: Next, we will describe the query processing strategies we consider and the details of the index structure we propose (Section 2). Then we will describe how to instantiate this structure to ensure a given threshold Δ on the execution cost of conjunctive keyword queries. Subsequently, we describe the relevant properties of the distributions of keywords and keyword-combinations in natural-language text and show how to leverage these to model on the size of the resulting index structure (Section 3). We will describe the index construction in Section 4 and provide an experimental evaluation in Section 5.

2. INDEX STRUCTURE AND USAGE

2.1 System Setup and Notation

For each query Q, we denote the keywords it contains by $words(Q) = \{w_1, \ldots, w_l\}$. In the following, we consider only queries containing up to a threshold k_{max} of keywords (e.g., 7). Each of these keywords comes from a global vocabulary $\mathcal{V}$. Note that the maximum number of keywords in a single query we have to consider for searches in the e-commerce scenario is small; it is well-known that most search queries are short (e.g., according to [14] over 50% of all internet searches contain at most two words, and 75% at most three). We have found similar trends in traces of search queries against the e-commerce site [1].

For all keywords we maintain a global ordering π. We use this ordering for indexing; i.e. when materializing a keyword-combination C containing the words $\{w_1, \ldots, w_l\}$ we let $i_1, \ldots, i_l \in \{1, \ldots, l\}$ be a set of indices such that $\forall j \in 1, \ldots l-1 : w_{i_j} <_\pi w_{i_{j+1}}$, with $<_\pi$ denoting the ordering induced by π and write C as $(w_{i_1}, \ldots, w_{i_l})$ instead. Basically, this ensures that we never distinguish between permutations of the same keyword-combination.

In the following, we sometimes use a query Q and the set of keywords $words(Q)$ interchangeably, with the correct meaning being clear from context. We denote the number of items (=documents) whose text contains all keywords of a query Q by $size(Q)$; similarly, for a single keyword w we denote the number of documents containing w by $size(w)$. Finally, we use the notation $|Q|$ to denote the number of keywords a query Q contains.

2.2 Processing Strategies and Cost Model

To build structures that reduce the maximum latency of keyword queries, we introduce a simple cost model to quantify these latencies. Costs will be expressed as a linear combination of two costs: (1) disk seeks to the beginning of posting lists, plus (2) scanning postings. It is useful to consider these two costs separately because the combination rule is a bit of a moving target. Computational costs have decreased dramatically over time and will continue to do so going forward. However, some costs have decreased more than others. Scanning costs are dropping faster than seek costs. This trend is likely to continue going forward.

For ease of exposition, we normalize the costs s.t. scanning a single posting in an inverted index has unit cost. This normalization allows us to think of Δ as specifying both a cost bound as well as a maximum number of postings that can be scanned.

The cost model assumes only the simplest possible IR-engine, which computes intersections by fully scanning the inverted index of every keyword. However, our approach is equally applicable to more sophisticated engines and hardware configurations (which in turn would lead to different cost models), in particular the case in which the all inverted indexes are read and intersected in parallel (allowing the intersection of the indexes for keywords

$(w_1), \ldots, (w_k)$ in $O(\max_{i=1,\ldots,k} size(w_i)))$ or for engines allowing random access within the indexes (allowing the intersection of two indexes of size n, m, with $n < m$ in $O(n \cdot log_2 m)$ operations). In both cases, fewer keyword-combinations are indexed, which in turn reduces the size of materialized structures significantly.

The cost of a query Q depends on the execution strategy chosen. Initially, there are two access strategies available to us:

ID-intersection: This strategy retrieves all inverted indexes of the queried keywords and intersects them. We model the execution cost as $|Q|$ seek accesses to disk (the cost of one of which we model as a constant $Cost_{seek}$) to retrieve the inverted indices and the cost of reading their contents entirely:

$$Cost_{Int}(Q) := |Q| \cdot Cost_{seek} + \sum_{w \in words(Q)} size(w)$$

Post-filtering: If one of the keywords w_i in Q is very rare, we can process Q by only processing the inverted index of w_i and then retrieving the text of all matching items and verifying the remaining keyword constraints using text itself. The advantage of this strategy is that its processing costs become independent of the number of additional keywords and the lengths of their inverted indices; however, matching the remaining keywords against the text is significantly more expensive than index-intersections for the same number of postings. We model its cost as the cost to retrieve the text associated with $size(w_i)$ items (which is dominated by the seek times) and applying $|Q| - 1$ keyword-filters to the text, which is a function of the text-length for each column. For simplicity, we model the text length of the items as a constant $length$, which we multiply by the cost of applying a single like-predicate: $Cost_{Filter}$. If necessary, we can ensure that this function over-estimates latency in cases with varying text-lengths by choosing this constant large enough; however, for the scenarios we consider, the text lengths tend to be small and not vary very much; hence, the costs for this strategy tend to be dominated by the seek times:

$$Cost_{Probe}(w_i) := size(w_i) \cdot \big(Cost_{seek} + (|Q| - 1) length \cdot Cost_{Filter}\big).$$

2.3 Index Structure

Given a cost model we now describe additional indexes to complement single-keyword inverted indexes which enforce an execution cost of less than a threshold Δ by limiting the maximum number of postings we need to retrieve for an arbitrary query. The structure that we utilize are additional inverted indices that materialize the postings for documents containing *combinations* of keywords; i.e., each such index can be thought of as the materialized result to that particular keyword query. The salient features of these structures are that

(a) We only materialize indexes for a k-keyword combination if the corresponding query result can not be obtained quickly (i.e., with less then $\Delta/2$ overhead) using intersection of inverted indexes for keyword combinations of size $k' < k$.

(b) Part of the query-processing time of a query is allotted to probing the "catalogue" of the materialized structures to discover which relevant keyword-combinations are indexed. We also obtain information on the size of the inverted indexes as part of this probing, allowing us to subsequently chose the best possible execution strategy (as predicted by the cost-model) before the actual processing of the query.

(c) For a small number of keyword combinations simply retrieving the fully pre-computed answer to a search query requires more than the target latency. However – again due to data skew – there will be few such instances; moreover, since these are search results, the user interface needs to initially display only the top-ranked results (ordered by whatever ranking scheme we use) and can use the time required for the user to browse them to retrieve the remainder. Therefore, for the few such keywords or keyword-combinations, we materialize the top-ranked results separately.

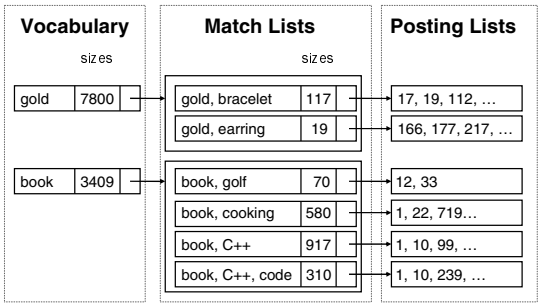

Figure 2: Index Structure

The main structure we use to complement the inverted indexes adds one layer of of indirection to the standard inverted index (Figure 2): instead of pointers from each vocabulary item to the corresponding inverted index, we maintain – for each vocabulary item w – a list of all keyword combinations containing w for which we have materialized the corresponding inverted index, the so-called *match list*. We denote the set of all keyword combinations realized as match list entries by $Match$. Each entry in the match list in turn points to an inverted index containing postings of all items matching all keywords in the entry. In addition, each entry in the match list also stores the number of postings in the corresponding inverted index; we also maintain the number of postings in each single-keyword inverted index together with the vocabulary. The resulting structure is in many ways similar to the *nextword indexes* of [5, 12] and can be implemented in a similar manner.

The physical layout of this structure is as follows: since (as we will describe later) we only materialize combinations of frequent keywords and only a small fraction of them, it is possible to maintain an index with the first two keywords of each combination in main memory[2]. In the following, we will assume for purposes of cost modelling that this layout is in place.

2.4 Query Processing

Once we have this index structure in place, we process a query Q over keywords $w_1, \ldots, w_k$ as follows: if Q contains a keyword w_i sufficiently rare so that the post-filtering strategy becomes sufficiently inexpensive, we use this strategy. Otherwise, we retrieve all match-list entries containing two keywords from Q as their prefix (we assume that the single-keyword vocabulary and sizes are already memory-resident). Using the size-information contained in the match-list entries we can now determine if $size(Q)$ is sufficiently large that we cannot process Q entirely without violating the cost-threshold Δ; if this is the case, we retrieve the top-ranked tuples from the corresponding index. For queries with smaller result sizes, we now determine which combination of inverted indexes covers all keywords in Q (possibly more than once) while minimizing the cost (using our cost model) of intersecting these indexes – note that this covers both multi- and single-keyword inverted indexes. This formulation results in an optimization problem

[2]If the *match list* grows to large, then part of this index can be written to disk, inducing 1 additional seek per keyword.

$$Cost_{Opt}(Q) := \min_{\substack{C \subseteq \mathcal{V} \cup Match: \\ \bigcup C = words(Q)}} \sum_{c \in C} Cost_{Seek} + size(c), \quad (1)$$

which is a variant of the *set cover* problem; however we do not require an exact solution, but only a approximation as long as fulfills two properties:

(A) We require that the algorithm considers – when it chooses a set of inverted indexes to process a query Q over $words(Q) = \{w_1, \ldots, w_k\}$ (among other alternatives) the execution plan formed by intersecting the (sets of) inverted indexes used when processing the queries formed by the keyword sets S_1 and S_2 constructed as follows: let w_1^f, w_2^f be the two most frequent (i.e., occurring most often in the corpus) keywords in $words(Q)$ (ties are broken using the ordering π); now let S_1, S_2 be defined as $S_1 = words(Q) - \{w_1^f\}$, $S_2 = words(Q) - \{w_2^f\}$.

(B) We require that the algorithm considers intersecting the (sets of) inverted indexes used when processing the queries formed by the keyword sets C_1 and C_2 constructed as follows: let w_1^l and w_2^l be the least frequent keywords among $words(Q)$ (ties are broken using the ordering π); now let C_1, C_2 be any two sets for which $C_1 \cup C_2 = words(Q), C_1 \cap C_2 = \emptyset$ and $w_1^l \in C_1$, $w_2^l \in C_2$.

We will illustrate the relevance of these properties later. From now on, we will denote the set of inverted indexes this algorithm selects when processing a query C as $index(C)$; in particular, for any word w, $index(w)$ refers to the "standard" inverted index for a single keyword w. Similarly, we will refer to cost of the solution provided by the algorithm employed as $Cost_{Opt}(Q)$.

Once we have determined a suitable combinations of inverted indices, we compute the query result by retrieving the inverted indexes in the inverse order of their sizes and intersecting them. The total cost of this execution plan is the cost of retrieving all relevant match list entries and the cost of retrieving and intersecting the selected inverted indexes (=$Cost_{Opt}(Q)$). The cost for retrieving the match list entries is dominated by the number of disk seeks required, so we use the disk-seeks alone to model this cost. For a k-keyword query up to $\binom{k}{2}$ entries in the match lists have to be examined; given that the number of keywords in a query is small, this number of seeks can be upper-bounded by the number of keywords multiplied with a small constant (e.g. $k_{max} = 5$, the bound is $3k$). We define $\Delta' = \Delta - \binom{k_{max}}{2} Cost_{Seek}$ as the minimum latency "available" after all relevant match-lists entries have been read. Thus, in order to ensure the overall latency threshold Δ, we have to materialize additional indexes ensuring that $Cost_{Opt}(Q) \leq \Delta'$.

This also means that for any query Q with $size(Q) > \Delta' - Cost_{Seek}$ we may need to explicitly materialize the top-ranked tuples, as we can't process the query with a larger result under the latency-threshold.

3. MODELLING THE INDEX SIZE

In the following, we will first give a general overview of the properties of large corpora that are relevant to this problem setting and show them to be present in a variety of real-life data sets (Section 3.1). Then we will describe in detail for which combinations of keywords we materialize *match list* entries and posting lists (Section 3.2). Finally, we show how we use the properties of the underlying corpora to model the size resulting index structure (Section 3.3). This model makes a number of simplifying assumptions and is not useful for directly predicting the exact sizes of the structures we eventually materialize. Rather, their purpose is to give an intuition as to why the proposed structures do not require prohibitive storage in cases when the underlying text corpora display power law constraints and the document sizes are relatively small.

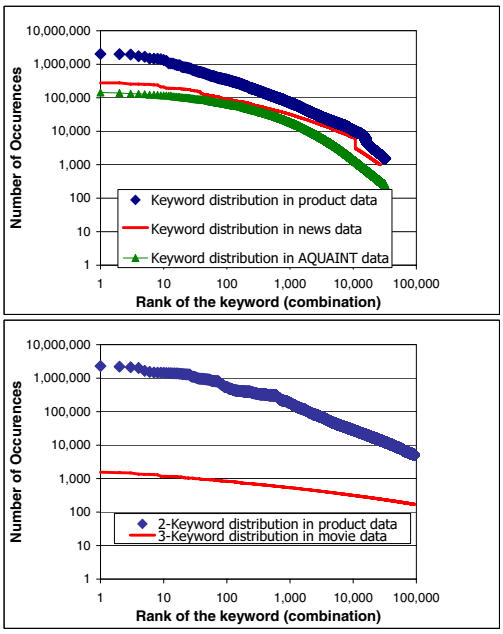

Figure 3: Natural language obeys power laws. This fact has been well-established for keywords taken one at a time, as well as ngrams, sequences of adjacent terms. We find that power-laws also apply to pairs of keywords (and multi-way combinations without the adjacency constraint).

3.1 Properties of Natural Language Corpora

In this section, we give a quick overview and experimental validation of the properties of natural language corpora we leverage.

Word frequency distributions in natural language have been studied extensively (see [4] for a summary) and have been found to be shaped according to a power law. Most of these studies have been conducted on either longer texts by a single author or literary corpora; we have examined the keyword distributions found in a series of natural language datasets selected to capture some of the scenarios found in database-driven web sites: the text corpora underlying (a) the e-commerce database described in Section 1.1, (b) a one-month slice of text from a web news portal as well as two publicly available datasets: (c) a subset of 314K newspaper articles from the AQUAINT Corpus used in the TREC QA track and (d) the polarity V2.0 data containing 2K movie reviews [2]. For every one of these datasets, the distribution of single keywords exhibited power-law behavior, as expected: Figure 3 depicts the frequency distribution of the 26K most frequent keywords for the news data and the 32K most frequent keywords in the e-commerce portal described in Section 1.1 as well as the AQUAINT corpus (after stopword pruning). Note that all the graph displays only the head of the distribution; e.g., less than 3% of the vocabulary of the product data.

Moreover, we found the same property to hold for the frequency distribution over multi-keyword combinations occurring in the data: Figure 3 depicts the frequency distribution of the 90K most frequent 2- and 3-keyword combinations in the product and movie data. To the best of our knowledge, the distribution of keyword-combinations have not been studied (except for distributions of *adjacent* keywords in the context of *language models*).

To give an intuition of how we leverage these properties, our approach essentially performs a "triage" over keywords, assigning them into three categories: (a) low-frequency keywords for which we don't need to materialize additional indexes, (b) medium-frequency

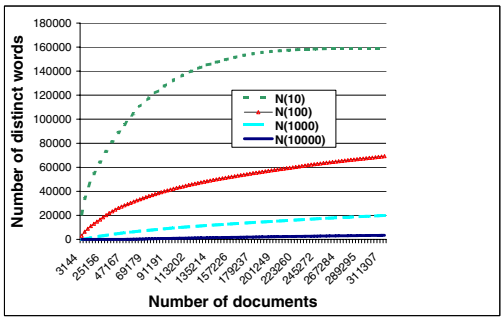

Figure 4: There aren't very many frequent words, even in large document collections.

keywords, at most *one* of which may appear in a match list entry and (c) a small number of high-frequency keywords for which we materialize a number of indexes. We will describe this in detail in Section 3.3. For our method to be scalable, we need to ensure that the number of keywords in the latter two classes does not grow quickly with corpus size.

Fortunately, this appears to be the case, as illustrated in Figure 4. Each line, labeled $N(\alpha)$, shows the growth rate in the AQUAINT corpus for words with document frequency of α or more. Thus, for example, the line for $N(10000)$ shows that there aren't very many words that appear in 10000 documents or more. The number of such words increases slowly with corpus size, much slower than N(1000). We are particularly concerned with large αs. Note that $N(10000) << N(1000) << N(100) << N(10)$. There aren't very many frequent words, even in large document collections.

3.2 Populating the Match List

In this section we will describe the structures materialized to ensure that the cost for processing an query of up to k_{max} keywords does not exceed Δ. To populate the match-lists, we first consider keyword-combinations of size 2 for materialization, and then increase the size until we reach k_{max}. Now, for any size k we materialize all combinations C for which

$$\forall w \in words(C) : Cost_{Probe}(w) > \Delta \quad (2)$$

$$\text{and } Cost_{Opt}(C) \geq \frac{\Delta'}{2} - Cost_{Seek} \text{ using existing indexes} \quad (3)$$

$$\text{and } size(C) \leq \Delta' - Cost_{Seek}. \quad (4)$$

The resulting structures ensure that any query Q for which it holds that $size(Q) \leq \Delta' - Cost_{Seek}$ can be computed using less than Δ cost: If $Cost_{Opt}(Q) \geq \frac{\Delta'}{2} - Cost_{Seek}$ using indexes over combinations of less than $|Q| - 1$ keywords (condition 3) and post-filtering is not an option (condition 2), then we materialize an additional inverted index, as condition 4 must hold.

3.3 Modelling the Size of the Match List

To model the index sizes based on these observations, we will use a relatively simple analytical model of the word-frequency distributions for ease of exposition. The main contribution of the theoretical model will be to show that the potentially exponential growth of possible keyword-combinations is balanced by the power-law behavior of the word-distribution in natural language corpora.

We use the following notation: let N be the total number of words in the text distribution, and $V = |\mathcal{V}|$ be the number of distinct words. Due to the power-law, the frequency of a word of rank z can be expressed as

$$f(z) = \frac{\zeta}{z^{\alpha}} N$$

where ζ is a normalizing constant smaller than 1 ensuring that $\sum_{z=1}^{V} f(z) = N$ and α is a fitting parameter modelling the skew of the distribution. For ease of exposition we set α equal to unity, resulting in the *standard harmonic* probability distribution over words. Under this distribution, the number of words that occur m times, $V(m)$, can be modelled as

$$V(m) = \frac{V}{m(m+1)}.([4], \text{p.17}) \quad (5)$$

We will first show how the power-law distribution and the construction of the previous section lead to the "triage" of keywords we have described: since the cost of the *post-filtering* strategy only depends on the length of the text associated with items and the number of occurrences of the rarest keyword in a query, equation 5 means that the majority of keywords will not occur in any keyword-combination in the match list. Any keyword w for which

$$size(w) \leq \delta_{tail} = \frac{\Delta}{\left(Cost_{seek} + (k-1)length \cdot Cost_{Filter}\right)}$$

cannot lead to execution costs in excess of Δ, and hence no additional indexing is required, eliminating $V - \frac{\zeta \cdot N}{\delta_{tail}}$ keywords from consideration.

Similarly, not more than *one* keyword w with $size(w) \leq \delta_{match} := (\Delta'/2 - Cost_{Seek})$ can occur in an k-keyword entry in the match list. We will prove this by contradiction: consider the case of such a combination being materialized: assume a keyword-entry C consisting of k keywords $words(C) = \{w_1, \ldots, w_k\}$; let w_1, w_2 be the least frequent keywords with $size(w_1) \leq size(w_2) < \delta_{match}$. The (because of the requirements described in Section 2.4) the algorithm must consider an execution strategy that intersects the indexes used when processing two subsets C_1, C_2 of $words(Q)$ sharing no keywords, one of which contains w_1 and the other w_2. Therefore, either C_1 is not materialized, implying that $Cost_{Opt}(C_1) < \Delta'/2 - Cost_{Seek}$, or it is, meaning we can retrieve it using cost $\Delta'/2$. Using a similar argument for C_2, $Cost_{Opt}(C)$ can be at most Δ', meaning there is no need to materialize an entry C, leading to a contradiction.

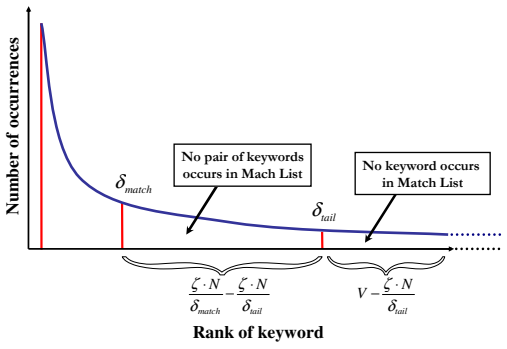

Figure 5: Only words of frequency greater than δ_{match} can occur multiple times in a single match-list entry.

Modelling the number of frequent keyword-combinations: Now we will use this model to model the number of l-keyword combinations that occur in more documents than a threshold χ. We denote this value as $occurrences(l, \chi)$. Subsequently, we will show that

the number of l-keyword entries into the match list can be modelled as a function of $occurrences(\ldots)$. Note that in the target scenario the individual items are associated with relatively small text entries (e.g., a product, a review, or a seller), which we will show to result in a small rate of growth for $occurrences(l, \chi)$ with increasing values of l.

First, we define avg_w as the average numbers of words contained in the text associated with an item. For ease of exposition, we will assume that all items are associated with exactly avg_w words (as opposed to modelling the distribution of this value explicitly). There are necessarily some duplicate words in an item, so we model the number of distinct words $V_e(n)$ in a document of n words as a function of the document size:

$$V_e(n) = R \cdot \sqrt{n} \text{ (see [4], p.25),}$$

for a constant[3] R. Using this model any item will contain $\binom{R\sqrt{avg\_w}}{l}$ distinct l-keyword combinations. Under the simplifying assumption that the power-law distribution governing the l-keyword distribution follows the same skew-parameter as the original keyword-distribution, the number of l-keyword combinations occurring more often than χ can be constrained as

$$occurrences(l, \chi) = \frac{\zeta \cdot N \frac{\binom{R\sqrt{avg\_w}}{l}}{avg\_w}}{\chi} \quad (6)$$

This means that while the number of possible keyword-combinations grows exponentially in the number of keywords, the number of l-keyword combinations larger than a threshold χ grows by a factor of $\frac{R\sqrt{avg\_w}-l}{l}$ with increasing l. Here, the key takeaway is that (a) this factor is a function of the square root of the individual text sizes (which are small for the target scenarios) and independent of the corpus size or the vocabulary size (both of which can become very large in this context), and (b) the factor decreases as l grows, resulting in tractable numbers of combinations to materialize.

This immediately allows us to model the number of keyword-combinations for which we have to explicitly materialize the top results, since their result-sets are too large to be read within Δ cost as $\sum_{l=2}^{k_{max}} occurences(l, \Delta' - Cost_{Seek})$.

Example: To demonstrate the size of the resulting values, consider a data distribution modelled on the product database described in Section 1.1, containing $N/avg\_w = 60 \cdot 10^6$ entities; each entity contains roughly $w = 100$ words, meaning ζ becomes $\approx 1/15$ and there is a total of $N \cdot avg\_w = 6000$ Million postings. We choose $R = 2.5$. Assuming we index for queries containing up to $k = 5$ keywords, and set χ at $50K$ ID-values, we obtain: $occurrences(3, \chi) = 18.4K$, $occurrences(4, \chi) = 101K$ and $occurrences(5, \chi) = 425K$. Even when multiplied with the number of top-ranked postings we materialize for these keyword-combinations, these numbers still are small fraction of the 6 Billion postings in the original index.

Modelling the number of match list entries: Moreover, we can use the above to model a loose constraint the number of l-keyword entries in the match list, of the form $f \cdot occurrences((l-1), \chi)/l$. To show this, consider an arbitrary entry $C = \{w_1, \ldots w_l\}$ in the match-list; let w_{min} be the keyword in C for which $size(w_{min})$ in minimal, $C' = words(C) - w_{min}$. Now one of two conditions must hold:

(a) $\mathbf{size(C') > \delta_{match}}$: in this case, the only statement we can make about $size(w_{min})$ is that it must be larger than δ_{tail}, meaning that there are at most

$$\underbrace{occurrences\big(l-1, \delta_{match}\big)}_{\text{number of combinations for } C'} \cdot \underbrace{\big(\frac{\zeta \cdot N}{\delta_{tail}}\big)}_{\text{possible values for } w_{min}}$$

such combinations possible.

(b) or $\mathbf{size(C') \leq \delta_{match}}$: In this case, let S_1, S_2 be subsets of $words(C)$ as defined in Section 2.4, both containing w_{min}. We know that $size(w_{min}) > \delta_{match}$ (otherwise we could compute C via the intersection of $index(w_{min})$ and $index(C')$ in time less than Δ'). We also know and that either $size(S_1) > \delta_{match}$ or $size(S_2) > \delta_{match}$ (again, otherwise we would not need to index, as we could compute C as the intersection of $index(S_1)$ and $index(S_2)$). The number of such combinations can be no more than

$$\underbrace{occurrences\big(l-2, \delta_{match}\big)}_{\text{number of combinations for } S_1 \cap S_2} \cdot \underbrace{\big(\frac{\zeta \cdot N}{\delta_{match}}\big)^2/2}_{\text{possible combinations of } words(C) - S_1 \cap S_2}$$

– for ease of notation, $occurences(0, \chi)$ is defined as 1.

This means that – with growing number of keywords – we expect the number of entries in the match list to grow more slowly than the number of keyword-combinations occurring more often than a threshold (as δ_{match} grows linearly with l). However, depending on the value of Δ, the factors $\big(\frac{\zeta \cdot N}{\delta_{tail}}\big)$ or $\big(\frac{\zeta \cdot N}{\delta_{match}}\big)^2/2$ may become very large. In these cases, we may have to apply our techniques to a subset of the most frequent keywords only (e.g., only keywords occurring in search logs).

Size-distribution of the resulting inverted indices: While the above calculations allow us to model the *number* of keyword combinations for which we create additional inverted indexes, it does not tell us anything non-trivial about the distribution of posting-sizes of the corresponding inverted indexes. We studied these size-distributions resulting from experiments and found them to be highly skewed as well. Not only does the vast majority of keyword-combinations satisfying conditions 2-4 result in empty intersections[4], but most of the remaining indexes have less than 10 postings. To demonstrate this, Figure 6 shows the size distribution of the 32K largest inverted indexes in an experiment similar to the one described in detail in Section 5.1 (with a different value of Δ). Again, we find a power law governing the distribution of the index sizes.

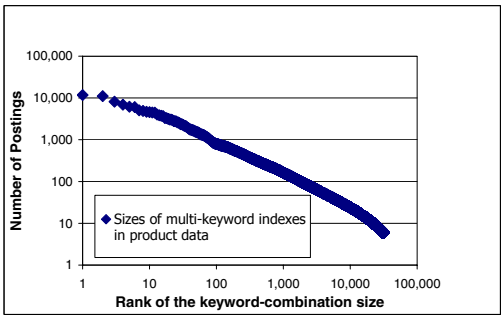

Figure 6: The distribution of inverted index sizes follows a power law

[3]The authors of [4] note that this model is flawed in that R itself varies slightly with large changes in n; however, since in our scenarios the sizes of the text fields do not vary significantly between items, this is an non-issue for our purposes.

[4]In this case we do not have to materialize the corresponding match-list entry; using size-information stored as part of the non-empty match list entries, the execution engine can infer these cases.

4. INDEX CONSTRUCTION

Construction of the proposed structures requires two elementary operations: (a) deciding which additional inverted indices to materialize and (b) building and maintaining the indexes themselves. Part (b) has been studied in great depth already, part (a) however is challenging, as it requires knowledge of the intersection sizes for very large inverted indexes, which are unlikely to fit into main memory at the same time. This may make this part of the computation prohibitively expensive.

As a consequence, we use a probabilistic scheme to *estimate* the required intersection sizes, allowing us to maintain compact representations of the relevant inverted indexes, which fit into main memory. This is made possible by the fact that the cost-thresholds themselves are necessarily large enough so allow the retrieval of tens of thousands of postings without exceeding Δ (a full-text retrieval system that cannot handle these numbers is likely a no-starter in the first place), providing some leeway regarding the accuracy of the probabilistic techniques.

4.1 Approximation of Intersection-Sizes

Computing the size of intersections between lists of postings corresponds to the problem of computing L_1-distances between columns in the indicator matrix A formed using the keywords as one dimension and the item/document ID values as the other. Popular techniques for such distance computations in limited memory are based on *random projections*, which multiply A by a appropriately chosen random matrix R to generate a much smaller data matrix $B = A \cdot R$. However, these estimation methods are typically not applicable to multi-way intersections, which we require. As a consequence, we use a different technique, based on a combination of sketches and sampling introduced in [9, 10]: Let $\mathcal{ID}$ denote the set of identifiers all documents in the corpus. This method then uses a random permutation $\pi_{ID} : \mathcal{ID} \mapsto \{1, \ldots, |\mathcal{ID}|\}$ and – for every inverted index – constructs a sample the first (according to π_{ID}) postings in the index. Now we can estimate intersection sizes between a list of inverted indexes $I_1, \ldots, I_l$, based on these samples as follows: let D_s be the smallest among the maximum (according to π_{ID}) postings in the respective samples. Now we trim the samples from all postings i for which $\pi_{ID}(i) > D_s$. The resulting samples are equivalent to a random sample of D_s rows from across the respective l columns in he indicator matrix A; this sample can now be used to compute a maximum-likelihood estimate of the intersection size. We evaluate the accuracy of this approach in Section 5.3.

4.2 Robustness of the Approximation

Note that the sampling ultimately only affects one condition among the three governing which keyword-combinations to materialize (Section 3.2): Condition (4). Condition (2) depends only on the sizes of single-keyword indexes, which we store together with the vocabulary. Moreover, since we construct the match-list entries and the corresponding indexes in order of the number of keywords they contain (this way, we can use existing indexes, significantly reducing construction-costs), the exact sizes of all *materialized* multi-keyword indexes over $(k-1)$-keyword combinations are known when determining which indexes over k-keyword combinations to construct; this is turn means that condition (3) can also be evaluated exactly and only the size of the new index has to estimated. Note that this means that bad estimates can never cause us to fail to meet the threshold Δ; we just might construct too many indexes.

4.3 Additional Index Compression

In order to further compress the resulting structures, we augmented each posting (which in our experimental setup corresponds to a 32-bit document ID before compression) with an additional 32-bit field, which indicates the presence of certain high-frequency keywords in the document the posting refers to. For example, we can use this field to indicate the presence or absence of one of the 32 most frequent non-stopwords in the corpus. In this case, we often can avoid having to materialize a multi-keyword index over a combination of these high-frequency words and less frequent words $\{w_1, \ldots, w_h\}$, as we can use the index on $\{w_1, \ldots, w_h\}$ (which, however, may be larger) to obtain the same information.

In the experiments on the e-commerce dataset, most frequent keywords correspond to distinct product categories (e.g., 'book') and a few frequent product attributes ('red', 'black', 'pages'), meaning that relatively few combinations of them actually co-occur in product descriptions in the corpus. This allows us to encode all occurring combinations of significantly more than 32 frequent keywords in the 32 bit field. While the additional field doubles the size of each posting before compression (the encoded values are highly skewed and thus should compress well), it can significantly decrease the number of keyword-combinations we materialize.

5. EXPERIMENTS

In this section, we describe the experimental evaluation of our techniques, including the resulting query costs for real-life data (Section 5.1), the total size of the structure on disk (Section 5.2) and the accuracy the probabilistic techniques to estimate the intersection sizes (Section 5.3).

Prototype Implementation: We implemented the match-list data structures using a commercial database system, which we also use to process queries against the match-lists. Due to size restrictions we omit the implementation details. While our experimental results can be used as a proof-of-concept, the absolute performance is not representative of the performance improvement possible when using a full-fledged IR system. To measure the performance of the unmodified IR system, we used a commercial "full-text" extension shipped as part of the database system. In all experiments we pruned the indexed corpora of stop-words and formatting tags.

5.1 Evaluation of Query Cost

To evaluate the effect of our approach on measured query cost, we used the product data set described in Section 1.1, containing 20 million items. For this experiment, we materialized the index structure for a subset of frequent keywords with more than 10K postings that occurred at least once in the query log; we used the parameters $k_{max} = 4$, $Cost_{Seek} = 1000$ (i.e. a disk seek is as expensive as scanning 1000 postings) and set Δ to the cost of scanning 20% of the number of postings of the largest single-keyword inverted index. This means that – once our structure is in place – no query of 4 or fewer keywords accesses more than a fifth of the number of postings in total that would have been read when scanning the inverted index of the most frequent keyword alone.

To measure performance, we generated random 2-word and 4-word queries from the keyword set and compared the running times of the commercial IR engine to our approach, with all inverted indexes residing on disk. We flushed the database caches before every measurement. For individual queries, our approach resulted in speed-ups of up to a factor of 18x (2 keywords) and 14x (4 keywords); averaged over the query set, our approach improved query times by a factor of 6.7x (2 keywords), and 3.1x (4 keywords).

Because we selected the workload as a random combination of keywords, especially the latter results significantly overstate the impact our techniques would have on average query performance. However, they do indicate that our approach can significantly benefit the sub-class of queries that cause user-perceptible latency.

5.2 Evaluation of Index Sizes

The most important aspect of our work is the size of the resulting indexes; if they are overly large, any gains in processing times become immaterial. To evaluate index sizes, we used the e-commerce corpus which contains a total of 899M postings. We use the parameters $k_{max} = 4$, $Cost_{Seek} = 1000$, set Δ to the cost of scanning 20% of the number of postings in the largest inverted index and used $\delta_{tail} = 50$ (i.e. no additional indexes for keywords occurring in less than 50 documents), leaving 141K keywords for indexing. For queries with results of more than $\Delta' - Cost_{Seek}$ postings, we materialize the top 20 postings. The resulting multi-keyword index structures contained 734M postings, i.e. an 81.6% increase, indicating that our approach scales up to large corpora and vocabularies.

Corpora of larger Documents: The significant limitation of our approach is that it does not scale to corpora with larger document lengths. To demonstrate this, we evaluated the index sizes on the 314K document subset of the AQUAINT corpus; the average length of these documents is almost an order of magnitude larger than it is for the e-commerce corpus. In addition, this corpus is challenging for our techniques as a number of keywords occur in 50% or more of the corpus (unlike the e-commerce data where the most frequent keyword occurs in 23% of the documents). For this collection, we used the same parameters as above, but set Δ to the cost of reading 1/3 of the number of postings in the largest inverted index. The resulting multi-keyword index structures contained more than 10x the number of postings of the original index, making straight-forward application of our technique impractical. For such corpora with larger document lengths, depending on the application details, our techniques may still be applicable, if either the number of relevant keyword combinations is reduced in a suitable manner (e.g., by only taking keyword-combinations that appear in query logs into consideration) or the documents are broken down into smaller chunks (e.g., paragraphs). Also note that our cost model is based on the assumption of an overly simplistic processing engine, as we assume that all indexes are processed *entirely* when computing intersections. If an IR engine can compute the intersection between a small and a very large index by scanning the small index and doing random lookups/skips into the large index (e.g. see [13], Chapter 4.3), then match-list entries for such combinations need not be materialized, potentially reducing the size of the materialized structures dramatically.

5.3 Accuracy of Intersection-size Estimation

Finally, we studied the effect of the probabilistic techniques described in Section 4.1. Here, we used sampling rates of 1% and 0.1%, i.e. for each inverted index we maintained the 1% (or 0.1%) smallest elements (according to the π_{ID}) and used these samples to determine which nodes to materialize in the match list. For this experiments, we used the inverted indexes for a subset of 2000 words from the e-commerce corpus, which were evenly distributed across the spectrum of frequency-ranks.

The experimental results underline the robustness of the estimation: for the 1% sampling rate the match list constructed via probabilistic techniques contained 99.3% of the entries in the exact one – here, nearly all missing entries were due to underestimation of "large result" combinations for which we would otherwise materialize only the top-ranked results (a case that we can detect and correct at build-time, but did not for this experiment); these in turn then made materialization of some larger keyword-combinations unnecessary. Compared to the exact list, the probabilistic technique materialized indexes for 0.08% additional keyword combinations. Interestingly, for 0.1% sampling, we achieved almost the same numbers, constructing 99.2% of the entries in the correct match list.

6. SUMMARY

This paper proposed a multi-dimensional index structure that improves latencies for intersecting postings. In the general case, multi-dimensional indexes consume exponential space, which is prohibitive. However, there are some important special cases that include many of the collections of interest to the Information Retrieval community, where multi-dimensional indexes are more promising, especially when appropriate care is taken in deciding which indexes to materialize.

We introduced a cost model to decide what to materialize. The cost model, in junction with various power-law assumptions, led to a triage process, where keywords were assigned to three tiers based on document frequency. The most frequent words require extensive indexing, but fortunately, there are not too many high frequency words. There are more words in the middle tier, at most one of which can occur in a query for which we materialize the result. The vast majority of keywords are assigned to the low frequency tier. No additional indexes beyond standard inverted indexes are required for these low frequency keywords.

The cost model was primarily motivated by latency considerations. We do not have any guarantees on space, though space is not prohibitive for the collections of short documents we have looked at thus far. The experiments with the AQUAINT corpus suggest that space requirements also depend on a few other factors such as document length[5].

7. REFERENCES

[1] `http://shopping.msn.com/`.

[2] Polarity dataset v2.0. http://www.cs.cornell.edu/people/pabo/movie-review-data/.

[3] V. Anh, O. de Kretser, and A. Moffat. Vector-Space Ranking with Effective Early Termination. In *Proceedings of the 24th annual international ACM SIGIR Conference*, pages 35–42, 2001.

[4] H. Baayen. *Word Frequency Distributions*, volume 18 of *Text, Speech and Language Technology*. Kulver Academic Publishers, 2001.

[5] D. Bahle, H. E. Williams, and J. Zobel. Efficient Phrase Querying with an Auxiliary Index. In *ACM SIGIR Conf.*, pages 215–221, 2002.

[6] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? *Lecture Notes in Computer Science*, 1540:217–235, 1999.

[7] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database System Implementation*. Prentice Hall, 2001.

[8] T. Jayram, S. Khot, R. Kumar, and Y. Rabani. Cell-Probe Lower Bounds for the Partial Match Problem. In *STOC*, 2003.

[9] P. Li, T. Hastie, and K. W. Church. Using Sketches to Estimate Two-Way and Multi-Way Associations. Technical report, Microsoft Research, 2006.

[10] P. Li, K. W. Church and T. Hastie. Conditional Random Sampling: A Sketch-based Sampling Technique for Sparse Data. NIPS, 2006.

[11] M. Persin, J. Zobel, and R. Sacks-Davis. Filtered Document Retrieval with Frequency-Sorted Indexes. *J. of the Amercian Society for Information Science*, 47(10):749–764, 1996.

[12] H. E. Williams, D. Bahle, and J. Zobel. Fast Phrase Querying with Combined Indexes. *ACM TOIS*, 22(4):572–594, 2004.

[13] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes - Compressing and Indexing Documents and Images*. Morgan Kaufman Publishers, 1999.

[14] Yahoo. Analysts day report. available at http://yhoo.client.shareholder.com/downloads/2006AnalystsDay.pdf, 2006.

[5] For collections with too many large documents, we suggest various workarounds such as splitting documents up into smaller chunks (e.g., paragraphs), indexing fewer keyword combinations (e.g., the ones present in query logs) as well as using and accounting for more sophisticated list-intersection techniques in the cost model.