On Placing Skips Optimally in Expectation

Flavio Chierichetti, Silvio Lattanzi, Federico Mari and Alessandro Panconesi *
Department of Computer Science
Sapienza University of Rome
Via Salaria 113 – 00198 Rome
{chierichetti,lattanzi,mari,ale}@di.uniroma1.it

ABSTRACT

We study the problem of optimal skip placement in an inverted list. Assuming the query distribution to be known in advance, we formally prove that an optimal skip placement can be computed quite efficiently. Our best algorithm runs in time $O(n \log n)$, n being the length of the list.
The placement is optimal in the sense that it minimizes the expected time to process a query. Our theoretical results are matched by experiments with a real corpus, showing that substantial savings can be obtained with respect to the traditional skip placement strategy, that of placing consecutive skips, each spanning $\sqrt{n}$ many locations.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; F.2.0 [**Analysis of Algorithms and Problem Complexity**]: General; E.1 [**Data Structures**]: Lists, stacks, and queues

General Terms

Algorithms

Keywords

Inverted Index, Probabilistic Analysis, Skips

1. INTRODUCTION

The inverted index remains to this day one of the basic data structures for query processing in web-search engines. In this paper we study a fundamental problem for query processing– how to place skips in an optimal way in an inverted index. To build an inverted index, documents in the corpus are sorted according to some total order and, for each term, a linked list is created that contains the documents in which the term appears. Crucially, the lists preserve the total order. In this fashion answering a conjunctive query of two or more terms becomes the task of merging the corresponding lists. A skip is a pointer $i \rightarrow j$ between non consecutive documents i and j in a list. Skips are a very effective way to speed up a merge operation.

Broadly speaking, the problem we address in this paper is the following: is it possible to place skips optimally with respect to the query distribution, when this is assumed to be known in advance? Given the fundamental character of this problem it is rather surprising that this has never been tackled before in a systematic way (to the best of our knowledge). The main approach remains that of [7] according to which skips should be placed at regular intervals, i.e. one after the other, each spanning $\sqrt{\ell}$ elements, ℓ being the length of the list. More related work is discussed in a section to follow.

In this paper we formally prove that if the query distribution is known, then an optimal skip placement can be computed efficiently. By "optimal" we mean a skip placement that minimizes the expected processing time of a query, i.e. the expected time to merge the lists of the terms in the query. Crucially, the optimal skip placement can be computed by very efficient polynomial-time algorithms. These theoretical results are complemented by experiments with real corpora showing that our algorithms yield substantial savings in space and query processing time.

We now describe our contributions more precisely. A basic contribution of our paper is to introduce the notion of *useful* document with respect to a query. Informally, a document is useful for a query if it cannot be skipped. The query distribution, assumed to be known, induces another distribution on the documents that gives the probability that a given document is useful for a query, when the query is chosen at random. We shall denote this probability by $p(d^t)$, where d^t is the occurrence of document d in the list of the term t (note that in general it can be $p(d^t) \neq p(d^{t'})$). In practice this distribution on the documents can be approximated quite well and efficiently starting from a sample of the query universe – see the discussion on the experimental results below. Therefore we may assume to have, for every d and t, the probability $p(d^t)$. Now, in general, the events of the form "d is useful for t" are not independent. Working under the simplifying assumption that they are we derive several algorithms. The goal is to place skips in order to minimize the expected query processing time, when a query is presented at random. We say that a skip placement is *optimal* for a term t if it minimizes the expected time to

*This work is partially supported by a grant of Yahoo! Research, by a contract with Enea under project Cresco, and by the European Commission project DELIS

WSDM'08, February 11–12, 2008, Palo Alto, California, USA.

process a query that contains t. To assess the performance of our algorithms at query time we make use of hardware independent measures.

There are several possible skip placement policies. The general case is that of *spaghetti skips* where there are no restrictions (apart from the obvious requirement that if $i \rightarrow j$ is a skip then $i < j$) (see Fig. 1). An interesting special case is that of *simple skips* where if $a \rightarrow b$ and $c \rightarrow d$ are skips and $a < c$ then it must be $b < c$.

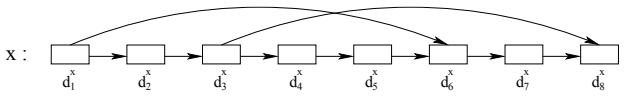

Figure 1: Spaghetti skips

It turns out that best results are obtained under a certain simplifying hypothesis that we now describe. Assume that the last document of every list contains all terms (and is therefore returned for every query). We call dictionaries of this form *doctored*. This is equivalent to pessimistically assuming that while computing a merge we always reach the end of the list. This assumption eliminates low order probability effects while making it possible to exploit a much richer combinatorial structure leading to our best performing algorithm.

Our algorithmic results are as follows. The input to the algorithms is an inverted list of n elements for a term t, where for each document occurrence d^t the probability $p(d^t)$ is known. Our experiments show that the probabilities $p(d^t)$ can be computed quite efficiently by using a small sample of the query universe. The algorithms are meant to be deployed in the preprocessing phase to set up skips.

- An $O(nt)$ time algorithm for optimal skip placement for spaghetti skips, where t is the average length of a skip. While $O(nt) = O(n^2)$, in practice t is much smaller.
- An $O(nt)$ time algorithm for optimal skip placement for spaghetti skips, in the case of doctored dictionaries.
- An $O(n^2)$ time algorithm for optimal simple skip placement.
- An $O(n \log n)$ time algorithm to place simple skips optimally in doctored dictionaries. This is the most important result of this paper in terms of algorithmic sophistication and for its practical value. Our experiments show that this algorithm computes skip placements as good as those of the more general algorithms above, while saving significantly in terms of space, running time, performance at query time and size of the sample to collect statistics on document usefulness.

It is an interesting open problem to determine whether spaghetti skip placement without assuming independence is polynomially solvable.

We now turn to our experimental results. In the experiments we use a web snapshot of the `.uk` domain. In the experiments we assume that a steady stream of queries is presented to the search engine. A small prefix of the query stream is used to collect statistics on the usefulness of documents. We compare the skip placement of our algorithms among themselves and with that of [7]. Although our theoretical results hold for any query distribution, in the experiments we focus on the most relevant case, i.e. we assume that the query distribution follows a power law. In particular we consider the exponents $\alpha = 0.79, 0.9, 1.1, 1.3$. In particular the first and the last are known to occur in practice [1]. Our experiments show that by placing skips with our algorithms yields significant savings in terms of space and query processing time. The algorithm of choice is the fastest $O(n \log n)$ algorithm for simple skips in doctored dictionaries. We consider this as evidence that it does not pay off to consider more sophisticated stochastic modelling such as the general, non independent, case.

Although a more thorough assessment with a larger corpus would be of interest, we believe that our experiments provide good evidence already of the usefulness of our algorithms.

2. RELATED WORKS

Skips in inverted indeces have been studied from a number of different point of views [4, 7]. It seems however that the problem addressed here–how to optimize the placement with respect to the input distribution–has not been studied before.

The classical approach of [7] is to place $\sqrt{n}$ skips, each of approximate length $\sqrt{n}$, one after another, where n is the length of the list. No attempt is made to gear the placement toward the query distribution.

In [8] so-called skip lists were introduced. Although somewhat related to skips, the terminology is somewhat misleading since these data structures, being in essence a binary search trees that use randomness to stay balanced, are actually quite different from inverted lists with skips.

In [3] skips lists are used instead of inverted lists with skips. The experimental results show a good improvement with respect to the classic skip placement strategy in terms of speed and memory occupation. The results do not seem to be directly comparable with ours however. Besides the essential difference between skip lists and inverted lists already remarked, compression issues are the main focus of [3]. Also, the metrics in [3] are not directly comparable since actual I/O time is used instead of a machine independent notion of work. No attempt is made to optimize the data structure with respect to the query distribution. Whether this is at all possible with skip lists is an interesting open question.

The paper [2] explores techniques other than the merge to compute list intersection. The setup is quite different from ours and it does not appear to be directly relevant. In particular no attempt is made to exploit knowledge of the query distribution, which is our main concern here.

3. ALGORITHMS

In this section we develop several algorithms to compute a skip placement that minimizes the expected time to process a query, assuming that the query distribution is known in advance. Our results hold for any distribution.

We need some notation. We assume that the dictionary (i.e. the set of terms) of a given corpus is organized as an inverted index. That is, each document in the corpus has a unique ID and the ID's induce a total ordering. For each term in the corpus, we have a linked list of the documents (posting list) containing the term, sorted by ID. In what follows we will just use the word "document" instead

of "document ID". In the same spirit, we will write $d < d'$ when d's ID is less than that of d'.

When processing a conjunctive query there are three kinds of atomic reads–pointers, skips and ID's–and they can have different unit costs. We distinguish between pointers of the form $i \to i+1$ and skips like $i \to j$ $(i+1 < j)$ because the former are sometimes only logical pointers, for instance when several elements are stored contiguously without explicit pointers. In what follows for the sake of simplicity we will charge a unit cost only for ID and skip reads. The cost of reading a pointer $i \to i+1$ will be absorbed by the read $i+1$. Our algorithms easily extend to the case when different constants are charged for the three kinds of read.

A document (whose ID is) d can belong to many lists. We denote with d^t when d belongs to the list of the term t and refer to this as an *occurrence* of d in t's list. For a query q we will use $t \in q$ to denote that the query q contains the term t. Similarly, we will write $t \in d$ and $t \in d^x$. If a skip goes from an element d^x to another element e^x of x's list we will denote the skip as $d \to e$ and call *tail* the former and *head* the latter.

The starting point of our analysis is the notion of *useful* document. Informally, a document is useful if it cannot be avoided while processing a query. This intuitive notion is captured computationally in the following definition.

DEFINITION 1. *(Useful documents)*

- *Let q be a query and let $q := x_1 \wedge \ldots \wedge x_\ell$. If a document i occurs in the posting list of all x_i then it is* useful *for q (see Figure 2 representing $\ell = 2$).*
- *Let x and y be two terms of a query q. Assume that (a) i is a document occurring in the posting list of x but not in that of y; (b) j and k are documents occurring in y's list; (c) $k < i < j$ in the global ordering defined among documents, and (d) the merge procedure is scanning document i. Then i, j and k are* useful *for q. (see Figure 3)*

A document is useful *if it is useful for some q.*

The first clause of the definition refers to the usual notion of relevance, i.e. a document is relevant if it contains all terms of a query and it therefore must be returned. The interesting thing about the definition is that if one restricts it to relevance, and computes the optimal skip placement accordingly, the result will be of no use! The reason is that one must locate document occurrences that can help during the merge, i.e. those that cannot be skipped.

The notion of usefulness is unambiguous when a query contains at most two terms, but it depends on the merge strategy when the terms are more than two. In general, the merge strategy (i.e. which pointer in which list to advance first) can depend on the query. Our results hold for any strategy whatever, provided that the same strategy is used to generate skips and to answer queries.

The query distribution induces another distribution on the document occurrences, giving the probability that an occurrence d^t is useful when q is drawn at random. We shall denote this probability by $p(d^t)$. In practice, as we shall see in the experiment section, given a sample of the query distribution we can collect statistics to approximate $p(d^t)$ rather efficiently. What we will do is to approximate the probabilities with their frequencies, in a straightforward fashion. It would be interesting to investigate more efficient ways to learn the distribution from the sample.

Henceforth we will assume that $p(d^t)$ is known for every d^t. In general, these probabilities are not independent. *From now on, we will make the simplifying assumption that they are.*

Note an important point. The notion of usefulness is unambiguous when a query contains at most two terms, but it depends on the merge strategy when the terms are more than two. In general, the merge strategy (i.e. which pointer in which list to advance first) can depend on the query. Our results hold for any strategy whatever, provided that the same strategy is used to generate skips and to answer queries.

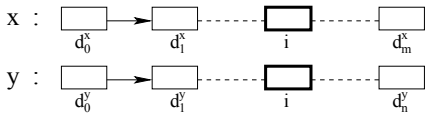

Figure 2: A document that answers a query is useful (in bold)

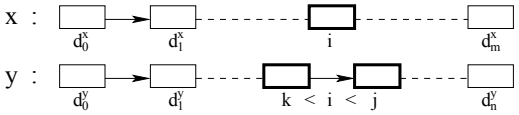

Figure 3: A document that shows where to skip is also useful (in bold)

DEFINITION 2. *A dictionary is* doctored *if the last document of the list of each and every term contains all terms.*

Assuming that a dictionary is doctored introduces an important simplification. In a sense it is equivalent to assuming pessimistically that the list of a term must be scanned in its entirety, for the last document must be returned for every query. In the experiments we will see that ignoring this low order effect is actually beneficial, for it makes the algorithm waste no time in try to mold the skips in too refined a way. At the same time, this assumption gives rise to a rich combinatorial structure that can be exploited by an algorithm.

3.1 Simple Skips in Doctored Dictionaries

We now develop an algorithm for simple skips in doctored dictionaries. As we will see, this algorithm is combinatorially the most interesting while at the same time the best performing. The algorithm will place skips in order to maximize the expected number of reads that are avoided (see Figure 5).

We start with some notation. Let $G(i \to j)$ the expected gain when placing the skip $i \to j$. That is,

$$G(i \to j) := (j - i - 2) \prod_{k=i+1}^{j-1} (1 - p_k) - \left(1 - \prod_{k=i+1}^{j-1} (1 - p_k)\right)$$

The first term is the probability that the skip can be done times the number of reads that are avoided. The second part is the probability that the skip cannot be done times the unit cost of reading the skip. In the first case, although $j-i-1$ documents are jumped, the algorithm needs to read the skip $i \to j$, so that $j - i - 2$ reads are avoided.

If the dictionary was not doctored we would have to add a third term to $G(i \to j)$ taking into account the possibility that we stop at position i because none of the subsequent documents is useful for the query.

In the algorithm to follow we will have to compute $G(i \to j)$. Leaving aside for the moment issues of numerical stability, to which we return when discussing the experiments, we would like to compute this term in constant time. This can be done by precomputing all products of the form $\prod_{k=1}^{j}(1-p_k)$, for all $1 \le j \le n$.

The input to the algorithm is a list $t := d_1 \to \ldots \to d_n$ with the probabilities $p_i := p(d_i)$ that document d_i is useful for a random query q, given that contains $t \in q$. As remarked, we assume these events to be independent. For a given skip placement, let X be the random variable counting the number of reads that are avoided when a query containing the term t is chosen at random. Thus our goal is to maximize $E[X]$ over all possible simple skip placements.

To develop some intuition we first develop an $O(n^2)$ solution by dynamic programming. We then build on top of this a more efficient $O(n \log n)$ algorithm. The algorithm will compute solutions inductively by a forward scan of the list. The value of optimal solution for the prefix $t_k := d_1 \to \ldots \to d_k$ will be denoted by $M(k)$.

DEFINITION 3. *Let H_k be the index i that maximizes the quantity*

$$M(i) + G(i \to k).$$

If the maximum is realized by several values of i we pick the largest.

Thus H_k is the head of the rightmost skip in an optimal placement that is subject to the constraint that a skip must land in position k.

To compute $M(k)$ we have two choices. Either we place the skip $H_k \to k$ or not. By definition of H_k we have,

$$M(k) = \max\{M(k-1), M(H_k) + G(H_k \to k)\}.$$

To actually build a skip placement that realizes $M(k)$ let

$$T_k := \begin{cases} 0 & \text{if } M(k) = M(k-1) \\ 1 & \text{otherwise} \end{cases}$$

i.e. $T_k = 1$ if and only if the skip $H_k \to k$ is placed. The optimal value is given by $M(n)$, while the corresponding skip placement is given by the two vectors $(H_1, \ldots, H_n)$ for the heads and $(T_1, \ldots, T_n)$ for the tails. We start the process by setting $M(1) = 0$, $H_1 = 1$, and $T_1 = 0$. The algorithm takes $n-1$ iterations to compute $H_k, M(k)$ and T_k, $2 \le k \le n$. While computing the last two takes constant time, we need $O(n)$ steps to compute H_k. The resulting complexity is thus $O(n^2)$.

We will show that in fact the body of the loop can be executed in $O(\log n)$ time. The basic intuition is that the H_k's are monotone, i.e. $H_k \le H_{k+1}$. This is the key to determine H_k by binary search. In fact, a more sophisticated version of the H_k is needed.

DEFINITION 4. *Let $H_{j,k}$ be the index i that maximizes the quantity*

$$M(i) + G(i \to j).$$

under the constraint that $i \le k$. If the maximum is realized by several values of i we pick the largest.

We first develop another $O(n^2)$ solution using $H_{j,k}$ and then show how it can actually be implemented in $O(n \log n)$ steps. If we are given

- $M(1), \ldots, M(k-1)$ and
- $H_{j,k-1}$, for all j, $1 \le j \le n$

we can compute M_k and $H_{j,k}$, for all j. To compute $M(k)$, observe that $H_k = H_{k,k-1}$ and thus,

$$\begin{aligned} M(k) &= \max\{M(k-1), M(H_k) + G(H_k \to k)\} \\ &= \max\{M(k-1), M(H_{k,k-1}) + G(H_{k,k-1} \to k)\} \end{aligned}$$

To compute $H_{j,k}$ the only question is whether the new head candidate k is better that the incumbent $H_{j,k-1}$. Therefore, we set $H_{j,k} := k$ if

$$M(k) + G(k \to j) > M(H_{j,k-1}) + G(H_{j,k-1} \to j), \quad (1)$$

and set $H_{j,k} := H_{j,k-1}$ otherwise.

The resulting algorithm is the following:

- $M(1) \leftarrow 0$, $H_{j,1} \leftarrow 1$, for $j = 1, 2, \ldots, n$, and $T_1 \leftarrow 0$
- for $k := 2$ to n do
  - $M(k) \leftarrow \max\{M(k-1), M(H_{k,k-1}) + G(H_{k,k-1} \to k)\}$
  - if $M(k) = M(k-1)$ then $T_k \leftarrow 0$ else $T_k \leftarrow 1$
  - for $j := 1$ to n do: if Equation (1) holds then $H_{j,k} \leftarrow k$ else $H_{j,k} \leftarrow H_{j,k-1}$

The time complexity of this algorithm being the same $O(n^2)$, it would seem that we have not made much progress. The point however is that the inner loop to compute the values $H_{j,k}$ can be computed in $O(\log n)$ steps! The crux of the matter is the following theorem.

THEOREM 5. $\forall j, k \;\; H_{j,k} \le H_{j+1,k}$.

Note that $H_{j,k} \le H_{j,k+1}$ is trivially true. We shall postpone the proof of this theorem to the end of the section and use it to develop the more efficient algorithm.

Let $\hat{j}$ be the smallest index j such that Equation 1 holds. If such a j does not exists let $\hat{j} = \infty$. Thanks to the monotonicity property of Theorem 5, $\hat{j}$ can be determined in $O(\log n)$ steps by binary search. Again by Theorem 5, observe that

- $j < \hat{j} \Rightarrow H_{j,k} = H_{j,k-1}$
- $j \ge \hat{j} \Rightarrow H_{j,k} = k$

We can implement the vector $(H_{1,k}, \ldots, H_{j,k}, \ldots, H_{n,k})$ in such a way that the update takes $O(\log n)$ time. Initially, $(H_{1,1}, \ldots, H_{j,1}, \ldots, H_{n,1}) = (1, \ldots, 1, \ldots, 1)$. This situation can be represented compactly as $(1, [1, n])$. At step k the vector will be represented as a sequence of this form $(1, [1, h_1]), (i_2, [h_1 + 1, h_2]), \ldots (i_\ell, [h_{\ell-1} + 1, h_\ell])$. The entry $(i, [x, y])$ represents the fact that $H_{j,k} = i$ for every $j \in [x, y]$. To compute the representation for $k+1$ we first determine the entry $(i_s, [x_s, y_s])$ such that $\hat{j} \in [x_s, y_s]$ by binary search and then update as follows: (a) each entry before $(i_s, [x_s, y_s])$ stays the same; (b) entry s becomes $(i_s, [x_s, \hat{j} - 1])$, and (c) entry $(k, [\hat{j}, n])$ is added after entry s. All remaining old entries are removed. We shall refer to the procedure just described for the update of $(H_{1,k}, \ldots, H_{j,k}, \ldots, H_{n,k})$ as a *concise update*.

The following algorithm SIMPLETON summarizes the discussion:

- $M(1) \leftarrow 0$, $H_{j,1} \leftarrow 1$, for $j = 1, 2, \ldots, n$, and $T_1 \leftarrow 0$
- for $k := 2$ to n do
  - $M(k) \leftarrow \max\{M(k-1), M(H_{k,k-1}) + G(H_{k,k-1} \rightarrow k)\}$
  - If $M(k) = M(k-1)$ then $T_k \leftarrow 0$ else $T_k \leftarrow 1$
  - Let $\hat{j}$ be the smallest index that satisfies Equation (1), if it exists, or ∞ otherwise. Determine $\hat{j}$ by binary search.
  - Update $H_{j,k}$ concisely.

THEOREM 6. *Algorithm* SIMPLETON *computes an optimal skip placement in time $O(n \log n)$ if the dictionary is doctored.*

We now turn to the proof of Theorem 5 which will follow from the next two lemmas.

DEFINITION 7. *Let $\Delta(i,k,j) := G(k \rightarrow j) - G(i \rightarrow j)$, where $i < k < j$.*

LEMMA 8. *Let $i < k < j$. Then,*

1. $\Delta(i,k,j) < 0 \Rightarrow \Delta(i,k,j+1) \geqslant \Delta(i,k,j)$
2. $\Delta(i,k,j) \geqslant 0 \Rightarrow \Delta(i,k,j+1) \geqslant 0$

PROOF. We start by rewriting $G(k \rightarrow j)$ in a more convenient form. Let

$$P(i,j) := \prod_{t=i}^{j}(1-p_t).$$

Then,

$$\begin{aligned} G(k \rightarrow j) &= \left(\prod_{t=k+1}^{j-1}(1-p_t)\right)(j-k-2) - \left(1 - \prod_{t=k+1}^{j-1}(1-p_t)\right) \\ &= P(k+1,j-1)\,(j-k-2) - (1 - P(k+1,j-1)) \\ &= P(k+1,j-1)(j-k-1) - 1 \end{aligned}$$

Analogously,

$$G(i \rightarrow j) = P(i+1,j-1)(j-i-1) - 1.$$

Thus,

$$\begin{aligned} \Delta(i,k,j) &= P(k+1,j-1)(j-k-1) - P(i+1,j-1)(j-i-1) \\ &= P(k+1,j-1)\,(j-k-1-P(i+1,k)(j-i-1)) \\ &= P(k+1,j-1)Q \end{aligned}$$

Where $Q := (j-k-1-P(i+1,k)(j-i-1))$. Similarly,

$$\begin{aligned} \Delta(i,k,j+1) &= P(k+1,j)(j-k) - P(i+1,j)(j-i) \\ &= P(k+1,j)\,(j-k-P(i+1,k)(j-i)) \\ &= P(k+1,j-1)(1-p_j)Q' \end{aligned}$$

Where $Q' := (j-k-P(i+1,k)(j-i))$. Note that

$$Q' \geq Q. \tag{2}$$

Suppose now that $\Delta(i,k,j) \geqslant 0$. This implies that $Q \geq 0$. Since $Q' \geq Q \geq 0$ it follows that $\Delta(i,k,j+1) \geqslant 0$ and the second part of the claim is proven.

If we assume instead that $\Delta(i,k,j) < 0$ we consider two cases. Note that the assumption implies that $Q < 0$. First, if $Q' \geq 0$ then $\Delta(i,k,j+1) \geqslant 0 > \Delta(i,k,j)$ and the first part of the claim follows. Otherwise, $Q' < 0$. But then

$$(1-p_j)Q' \geq Q' \geq Q$$

which implies

$$P(k+1,j-1)(1-p_j)Q' \geq P(k+1,j-1)Q.$$

And the claim follows again. □

LEMMA 9. *Let $i < k < j$. If*

$$\Delta(i,k,j) \geqslant M(i) - M(k)$$

then

$$M(i) + G(i \rightarrow j) \leqslant M(k) + G(k \rightarrow z)$$

for all $z \geq j$.

PROOF. It is sufficient to prove the lemma for $z = j+1$. Let us then rewrite the claim in an equivalent way: if $\Delta(i,k,j) \geqslant M_i - M_k$ then $\Delta(i,k,j+1) \geqslant M_i - M_k$ for all $z \geq j$. Now, clearly $M(i) \leq M(i+1)$ and therefore $M(i) - M(k) \leq 0$. There are two cases to consider. The first is when $\Delta(i,k,j) < 0$. By Lemma 8 we have that $\Delta(i,k,z) \geq \Delta(i,k,j) \geq M(i) - M(k)$. And the claim follows. The other case is when $\Delta(i,k,j) \geq 0$. Invoking Lemma 8 again we have that $\Delta(i,k,j+1) \geq 0 \geq M(i) - M(k)$. The claim again follows. □

From Lemma 9, Theorem 5 follows. The lemma can be restated as saying that if

$$M(i) + G(i \rightarrow j) \leq M(k) + G(k \rightarrow j)$$

for $i < k < j$ then

$$M(i) + G(i \rightarrow j) \leqslant M(k) + G(k \rightarrow j+1).$$

But this implies that if $H_{jk} = k$ then $H_{j+1,k} = k$. Theorem 5 follows.

3.2 Spaghetti skips

In this section we develop an $O(n^2)$ algorithm to place spaghetti skips. A careful analysis shows that the running time is in fact $O(nt)$ where t is the average length of a skip. Although $O(n^2) = O(nt)$ in the worst case, in practice $O(nt)$ is much smaller. Note also that the algorithm breaks ties by choosing the shorter skip between equivalent alternatives and thus has a tendency to create short skips.

We begin with some notation. Let $t : d_1 \rightarrow \ldots \rightarrow d_n$ be the input list, and let p_i be the probability that d_i is useful. We will assume that these events are independent. Let

$$\overline{P}_s^t = \prod_{i=s}^{t}(1-p_i).$$

This is the probability that none of the documents $s, s+1, \ldots, t$ is useful. And let $P_s^t = 1 - \overline{P}_s^t$. This is the probability that at least one of those documents is useful. The algorithm will build optimal solutions scanning t backwards. At position i the algorithm has already placed skips optimally for the suffix $s_{i+1} := d_{i+1} \rightarrow \ldots \rightarrow d_n$. The algorithm will minimize the expected number of document ID reads.

At position i we have to decide whether to place a skip or not. Assuming that we already have an optimal solution for s_{i+1} we denote by $E_{i \rightarrow j}$ the expected number of reads if the skip $i \rightarrow j$ is placed, and by $E_{i \not\rightarrow}$ the expected number of reads if we place no skip at i. Let

$$E_i := \min\{E_{i \not\rightarrow}, \min_{i<j\leq n} E_{i \rightarrow j}\} \tag{3}$$

(we include $E_{i\to i+1}$ for notational simplicity). In case of equivalent alternatives, the algorithm breaks ties by selecting the shortest skip. The recurrence (3) specifies $O(n^2)$ algorithm provided that we can initialize the process and compute $E_{i\not\to}$ and $E_{i\to j}$ in constant time.

The initialization is $E_n = 1$. Then, for all $i \in \{1,\ldots,n-1\}$,

$$E_{i\not\to} \quad = \quad P_{i+1}^n(1+E_{i+1}) + \overline{P}_{i+1}^n$$

To specify $E_{i\to j}$ we consider three mutually exclusive events: (a) there is a useful document in $[i+1, j-1]$; (b) no document in this interval is useful, but there is a useful document after j; and (c) there is no useful document after i. Then,

$$E_{i\to j} \quad = \quad P_{i+1}^{j-1}(3+E_{i+1}) + \overline{P}_{i+1}^{j-1}P_j^n(3+E_j) + \overline{P}_{i+1}^n$$

For the case $j = i+1 \le n$, the previous equation becomes ill-defined because of P_{i+1}^{j-1}. So let us adopt the convention $P_{i+1}^i = 0$.

This ends the description of the $O(n^2)$ algorithm. A careful analysis reveals its running time to be $O(nt)$, t being the average length of a skip. This is what we do next. The key property is the following. Assume that the skip $i \to j$ is in the optimal solution. Then, for every $k < i$, the skip $k \to \ell$ is such that $\ell \le j$. We call this the *no-leapfrog property*. This property implies the $O(nt)$ bound.

First of all, we prove that the sequence $\{E_i\}_{i=1}^n$ is non-increasing. Although this is intuitively obvious, a rigorous verification seems to require some care.

LEMMA 10. *For all $i \in \{1,\ldots,n-1\}$, $E_i \ge E_{i+1}$.*

PROOF. It can be checked directly that $E_{n-1} \ge E_n$. We want to show that $E_i \ge E_{i+1}$ assuming by induction $E_{i+1} \ge E_j$, for all $j \in \{i+1,\ldots,n-1\}$.

There are two cases to consider. Either $E_i = E_{i\not\to}$ or $E_i = E_{i\to j}$ for some $j > i$. Assuming the former, we proceed by contradiction. If $E_i < E_{i+1}$ then we would have $E_{i\not\to} < E_{i+1\not\to}$, or

$$P_{i+1}^n(1+E_{i+1}) + \overline{P}_{i+1}^n < P_{i+2}^n(1+E_{i+2}) + \overline{P}_{i+2}^n$$

which is equivalent to

$$P_{i+1}^n E_{i+1} < P_{i+2}^n E_{i+2}.$$

But this is impossible because of the inductive property and $P_{i+1}^n \ge P_{i+2}^n$, which trivially holds.

Assume now $E_i = E_{i\to j}$. If $j = i+1$ the claim $E_{i\to i+1} < E_{i\not\to}$ trivially implies a contradiction, so assume $j \ge i+2$.

Let $E(i \to j) := A_{i+1} + B_{i+1}^j + \overline{P}_{i+1}^n$ where

$$A_{i+1} := P_{i+1}^{j-1}(3+E_{i+1})$$

and

$$B_{i+1}^j := \overline{P}_{i+1}^{j-1}P_j^n(3+E_j).$$

If $E_i < E_{i+1}$, then $E_{i\to j} < E_{i+1\to j}$, or

$$A_{i+1} + B_{i+1}^j + \overline{P}_{i+1}^n \quad < \quad A_{i+2} + B_{i+2}^j + \overline{P}_{i+2}^n$$

which is equivalent to

$$A_{i+1} - A_{i+2} \quad < \quad B_{i+2}^j - B_{i+1}^j + \overline{P}_{i+2}^n - \overline{P}_{i+1}^n$$

We want to show that

$$p_{i+1}\overline{P}_{i+2}^{j-1}(3+E_{i+1}) \le A_{i+1} - A_{i+2}$$

and that

$$B_{i+2}^j - B_{i+1}^j + \overline{P}_{i+2}^n - \overline{P}_{i+1}^n < p_{i+1}\overline{P}_{i+2}^{j-1}(3+E_j)$$

From these two inequalities the claim follows since they imply

$$p_{i+1}\overline{P}_{i+2}^{j-1}(3+E_{i+1}) < p_{i+1}\overline{P}_{i+2}^{j-1}(3+E_j).$$

Now, if $p_{i+1}\overline{P}_{i+2}^{j-1} = 0$ we immediately have a contradiction. Otherwise we equivalently have

$$(3+E_{i+1}) < (3+E_j)$$

which is again a contradiction since $E_{i+1} \ge E_j$ by the induction hypothesis. To conclude the proof we verify the inequalities above. Let $p := p_{i+1}$, then

$$\begin{aligned} A_{i+1} - A_{i+2} &= (1-\overline{P}_{i+1}^{j-1})(3+E_{i+1}) - (1-\overline{P}_{i+2}^{j-1})(3+E_{i+2}) \\ &= E_{i+1} - E_{i+2} + \overline{P}_{i+2}^{j-1}((3+E_{i+2}) - (1-p)(3+E_{i+1})) \\ &= (1-\overline{P}_{i+2}^{j-1})(E_{i+1}-E_{i+2}) + p\overline{P}_{i+2}^{j-1}(3+E_{i+1}) \\ &\ge p\overline{P}_{i+2}^{j-1}(3+E_{i+1}) \end{aligned}$$

where the last inequality follows from the inductive hypothesis $E_{i+1} \ge E_{i+2}$. Furthermore,

$$\begin{aligned} &B_{i+2}^j - B_{i+1}^j + \overline{P}_{i+2}^n - \overline{P}_{i+1}^n = \\ &\quad \overline{P}_{i+2}^{j-1}P_j^n(3+E_j) - \overline{P}_{i+1}^{j-1}P_j^n(3+E_j) + \overline{P}_{i+2}^n - \overline{P}_{i+1}^n \\ &= p\overline{P}_{i+2}^{j-1}(P_j^n(3+E_j) + \overline{P}_j^n) \\ &= p\overline{P}_{i+2}^{j-1}((1-\overline{P}_j^n)(3+E_j) + \overline{P}_j^n) \\ &= p\overline{P}_{i+2}^{j-1}((3+E_j) - \overline{P}_j^n(3+E_j) + \overline{P}_j^n) \\ &= p\overline{P}_{i+2}^{j-1}((3+E_j) - \overline{P}_j^n(2+E_j)) \\ &\le p\overline{P}_{i+2}^{j-1}(3+E_j) \end{aligned}$$

□

We now prove the no-leapfrog property.

LEMMA 11. *Let $i \to k$ and $i+1 \to j$ be optimal skips of smallest length. Then, $k \le j$.*

PROOF. Suppose to the contrary that $j < k$. Then the hypotheses are equivalent to $E_{i\to j} > E_{i\to k}$ and $E_{i+1\to j} \le E_{i+1\to k}$.

The first inequality is,

$$\begin{aligned} &P_{i+1}^{j-1}(3+E_{i+1}) + \overline{P}_{i+1}^{j-1}P_j^n(3+E_j) + \overline{P}_{i+1,n} \\ &\quad > \quad P_{i+1}^{k-1}(3+E_{i+1}) + \overline{P}_{i+1}^{k-1}P_{k,n}(3+E_k) + \overline{P}_{i+1}^n \end{aligned}$$

Equivalently,

$$\begin{aligned} &\overline{P}_{i+1,j-1}P_{j,n}(3+E_j) - \overline{P}_{i+1,k-1}P_{k,n}(3+E_k) \\ &\quad > \quad (\overline{P}_{i+1,j-1} - \overline{P}_{i+1,k-1})(3+E_{i+1}) \end{aligned}$$

If $\overline{P}_{i+1,j-1} = 0$ we immediately get a contradiction for both terms becomes zero. Otherwise we factor this term out and get

$$\begin{aligned} M \quad &:= \quad (3+E_j) + \overline{P}_{j,n}(E_k - E_j) - \overline{P}_{j,k-1}(3+E_k) \\ &> \quad (1-\overline{P}_{j,k-1})(3+E_{i+1}). \end{aligned}$$

Analogously for the second hypothesis ($E_{i+1\to j} \le E_{i+1\to k}$), we get

$$\begin{aligned} M &:= (3+E_j) + \overline{P}_{j,n}(E_k - E_j) - \overline{P}_{j,k-1}(3+E_k) \\ &\le (1-\overline{P}_{j,k-1})(3+E_{i+2}). \end{aligned}$$

By combining the inequalities we get

$$(1-\overline{P}_{j,k-1})(3+E_{i+2}) \ge M > (1-\overline{P}_{j,k-1})(3+E_{i+1})$$

which is impossible by the non-negativity of all products' terms and by the inductive hypothesis $E_{i+1} \ge E_{i+2}$. □

The last lemma implies that the running time of the algorithm is $O(nt)$ where t is the average length of a skip. To see this, let t_i be the length of the skip whose head is position i. Then, the running time is

$$O\left(\sum_{i=1}^{n} t_i\right) = O(nt).$$

Remark: using the machinery described in this section we can determine two other algorithms. Algorithm SIMPLE places simple skips optimally in time $O(n^2)$. To derive it we only need to add to the definition of $G(i \to j)$ a term that takes into account the possibility that we stop at position i, as we did in the derivation of algorithm SPAGHETTI. Similarly, we can derive an $O(nt)$ algorithm for spaghetti skips and doctored dictionaries. We refer to this as algorithm DOCTOREDSPAGHETTI. The details of their derivation would not add much to the discussion in terms of new ideas and are therefore omitted from this extended abstract.

4. EXPERIMENTAL EVALUATION

We studied the performances of our algorithms on a partial snapshot of 100 thousands pages taken from the `.uk` domain in crawling order, containing about 700MB of parsed text. This corpus generated 1,358,045 many distinct terms.

We used the APIs supplied by Lucene 1.4.3 [6], among the Apache projects [5], to index the documents on the local disk.

Even if Lucene is an efficient indexer, a user cannot decide to arbitrarily place skips on the lists. So we had to modify Lucene with further new APIs which permit a user to put skips as required.

Finally, we implemented our algorithms running them on a Java Virtual Machine 1.5.

When dealing with products of probabilities, to avoid problems of numerical stability we used the logarithm of the product.

4.1 Query Data Set and Measures

There is strong experimental evidence that the queries on a search engine follow a power law with parameters $\alpha = 0.74$ or $\alpha = 1.3$ [1]. Initially we used a log of real queries kindly provided by `AlltheWeb.com`. Unfortunately this log turned out to be of very limited value because the greatest majority of the queries had no matches in our document corpus. Therefore, to assess the efficiency of our methods, we generated data sets of *conjunctive queries* chosen at random from our corpus, following power laws with parameters $\alpha = 0.74$, $\alpha = 0.9$, $\alpha = 1.1$ and $\alpha = 1.3$. Each query was composed by 2 terms. In this fashion, for each distribution d, we obtained a data set Q_d.

Note an important point. The fact that queries are generated independently does *not* imply that events of the kind "d is useful for t" are also independent. Thus, our corpus is a good benchmark to test the influence of the simplifying independence assumption we made to have a manageable theoretical analysis.

A distribution d on queries induces a distribution on documents usefulness. As discussed previously, we approximate these probabilities by collecting frequencies, using a *sample*. To collect the statistics we used the first $\beta|Q_d|$ queries in Q_d as a sample, where $\beta \in (0,1)$ is the parameter which determines the sample size $\beta|Q_d|$. After the statistics were collected, the experiments were performed on the entire collection Q_d.

In the following, we run experiments on five different algorithms:

- SPAGHETTI is the algorithm in Sect.3.2. This algorithm tries to extract as much information as possible from the query distribution and optimize as much as possible. Its running time is $O(nt) = O(n^2)$.
- DOCTOREDSPAGHETTI is the same as above, for the special case of doctored dictionaries. Its running time is $O(nt) = O(n^2)$.
- SIMPLE is the algorithm that finds the best placement for simple skips. Its running time is $O(n^2)$.
- SIMPLETON is the algorithm of Section 3.1, to place simple skips in doctored dictionaries. Its running time is $O(n \log n)$.
- Finally SQRT is the canonical way of placing skips on a list of length ℓ, one every $\sqrt{\ell}$ documents.

Our interest is in the following measures.

1. *Space overhead*: we counted the number of skips placed by each algorithm. Results described in Fig. 4.
2. *Performances* at run-time: we measured the performances of query answers on lists with skips w.r.t. linear scans on the lists.
3. *Build Time*: the *running time* needed by our algorithms to place the skips, measured in time units.
4. *Sample Size*: we also investigated on the minimum value for the parameter β.

Note that (1), (2) and (4) are machine-independent quantities. We will describe them in next sections.

4.2 Space

Fig. 4 shows the number of skips placed w.r.t. the skips placed by SQRT, as a function of the power law parameter α used to generate the queries. Thus, data in Fig. 4 is normalized to 1 for SQRT. In particular, it depicts the case in which $\beta = 0.25$, the biggest β among our choices.

Each one of our optimal algorithms perform better than the classical SQRT. DOCTOREDSPAGHETTI goes from 60% of the skips placed by SQRT, when $\alpha = 0.74$, to 90% when $\alpha = 1.3$. The same holds for SPAGHETTI. SIMPLETON and SIMPLE come out to be the best solutions here, since they use about 20% of the space for every value of the power-law parameter α.

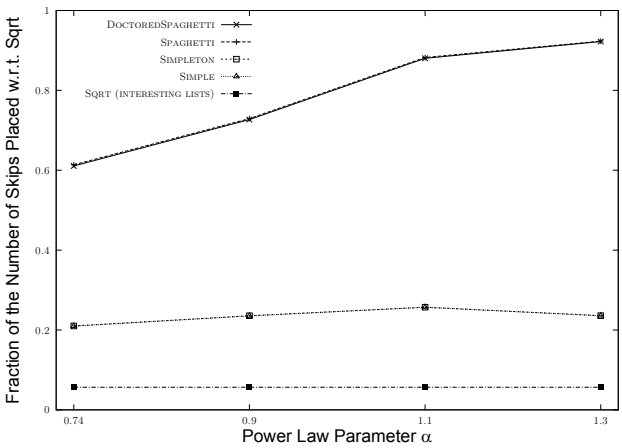

Figure 4: *Space* as the number of skips added to the index measured as fraction of those added by Sqrt. Results for $\beta = 0.25$.

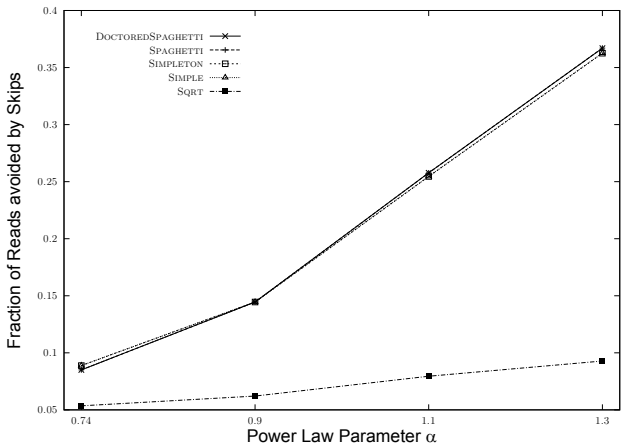

Figure 6: *Performances at run-time* as the fraction of reads avoided by skips w.r.t. the overall number of reads in the benchmark. Results for $\beta = 0.25$.

But if we restrict our attention solely to the lists of terms involved in the data set, Sqrt places the minimum number of skips. These lists are called *interesting* in Fig. 4. We can assume however that in the real world, the data set should approximately contain the entire dictionary. If this is the case, all the lists become interesting and Sqrt suddenly becomes the worst possible choice.

4.3 Performance at Run-Time

We use here as a *benchmark* the index without skips. As in Sect. 3, we assume the decompression time of both a document ID and a skip to be equal to 1, i.e. we assume them to be atomic *reads*.

Fig. 6 shows the fraction of reads avoided by skips w.r.t. the overall number of reads in the benchmark, as a function of the power law parameter α used to create the data set. As before, the case in which $\beta = 0.25$ is shown. Similar results are obtained for different values of β.

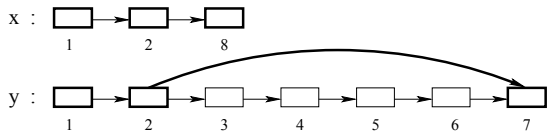

Figure 5: *Measures of Performance*. Bold elements are touched by a query answer using skips. The skip permits to avoid 4 reads minus 1 (the skip) instead of the 10 needed by the benchmark.

As an example, in Fig. 5 the thin documents represents the reads avoided by the skip, while the bold elements are the reads made. The benchmark would force 10 reads, while the skip permits to avoid $4-1$ reads, 4 for IDs minus 1 for the additional skip we pay. Our measure would be $(4-1)/10 = 0,3$, meaning that we perform the 70% of the linear scan work.

Our experimental results show that our optimal algorithms are equivalent in terms of performance, and they all perform better than Sqrt. When $\alpha = 0.74$ our algorithms avoid the 8% of reads w.r.t. the benchmark, while Sqrt avoids only the 5% of reads. When $\alpha = 1.3$, our algorithms gain 37% while Sqrt achieves only the 10%.

4.4 Build Time

We used a machine equipped with two processors *Dual Core AMD Opteron(tm) Processor 275*, with 2GHz each, and 6GB of total RAM. In Fig. 7 we report the time needed by each algorithm to compute the skip placement, where the sample size is determined by parameter $\beta = 0.25$.

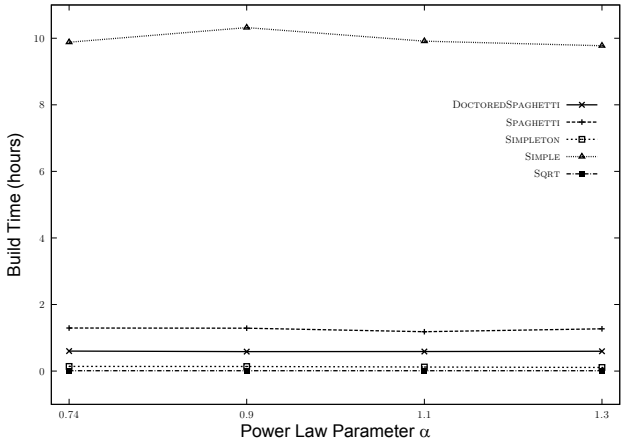

Figure 7: *Time to compute skip placement*. Results for $\beta = 0.25$.

It is worth noting that Simpleton is the fastest algorithm. It takes only half an hour to place skips, while DoctoredSpaghetti and Spaghetti need 1 hour and 1.5 hours respectively. Finally Simple needs almost 10 hours to end its computation. This is evidence that in fact $nt < n^2$.

Not surprisingly, Sqrt is even faster than Simpleton, even if by a small margin.

4.5 Sample Size

Here we investigate the sample size needed to collect reliable enough statistics on document usefulness. That is,

we would like to determine the minimum value for β which makes the sample meaningful.

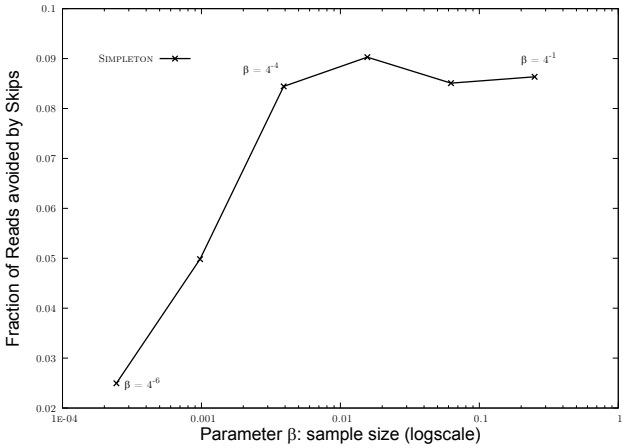

Figure 8: *Performances of* Simpleton *as a function of the sample size* (parameter β). Results for $\beta = 0.25$.

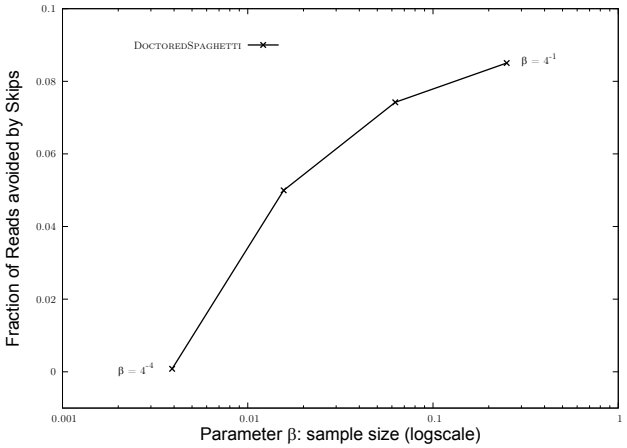

Figure 9: *Performances of* DoctoredSpaghetti *as a function of the sample size* (parameter β). Results for $\beta = 0.25$.

To this aim, we have performed several experiments varying the value for β. In Fig. 8 and 9 the results for Simpleton and DoctoredSpaghetti respectively are shown. We tried $\beta \in [4^{-6}, 4^{-1}]$. For this reason the x-axis are in logarithmic scale.

Fig. 8 shows that for Simpleton the value for β must be at least 4^{-4}. Performance deteriorates for smaller values of β. On the other hand, Fig. 9 shows that DoctoredSpaghetti needs $\beta = 4^{-1}$. Note, in fact, how DoctoredSpaghetti performances dramatically collapse with decreasing values of β.

To summarize, Simpleton needs a sample smaller than the one needed by DoctoredSpaghetti.

5. CONCLUSIONS

Query distributions contain a lot of useful information that can be exploited to improve performance. This paper is the first attempt to do so in a systematic and rigorous way for the important task of answering conjunctive queries.

It is an interesting conclusion that Simpleton is the best performing algorithm in all respects: time, query processing, space and size of the sample to collects statistics. We take this to mean that it does not pay off to consider more refined stochastic models for the problem. In building the skips, Simpleton implicitly assumes that the end of the list is always reached, while we know that this is not the case in practice. Note however that our algorithms Spaghetti and Simple do take into account that a merge can end before reaching the end of a list. Our experiments show that Simpleton performs at least as well, and therefore this is an indication that taking this factor into account has a computational cost that it is not worth in terms of performance.

Future work could consider larger corpora and try to determine the query processing time for the most popular queries. Finally, we feel that further significant improvements can be made following the route traced in this paper. For instance it would be of great interest to see if the approach extends to the case when data structures are compressed.

Acknowledgements

We would like to thank Prabhakar Raghavan for suggesting the problem. We also thank the anonymous referees for many valuable comments.

6. REFERENCES

[1] Ricardo A. Baeza-Yates and Felipe Saint-Jean. A three level search engine index based in query log distribution. In *String Processing and Information Retrieval, SPIRE 2003, Proceedings*, volume 2857 of *Lecture Notes in Computer Science*, pages 56–65. Springer, 2003.

[2] Jérémy Barbay, Alejandro López-Ortiz, and Tyler Lu. Faster adaptive set intersections for text searching. In *Experimental Algorithms, WEA 2006, Proceedings*, volume 4007 of *Lecture Notes in Computer Science*, pages 146–157. Springer, 2006.

[3] Paolo Boldi and Sebastiano Vigna. Compressed perfect embedded skip lists for quick inverted-index lookups. In *String Processing and Information Retrieval, SPIRE 2005, Proceedings*, volume 3772 of *Lecture Notes in Computer Science*, pages 25–28. Springer, 2005.

[4] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems*. The Benjamin/Cummings Publishing Company, Inc., June 1994.

[5] Apache Software Foundation, `http://www.apache.org`, 2007.

[6] Lucene Home Page, `http://lucene.apache.org/java/docs/index.html`, 2007.

[7] Alistair Moffat and Justin Zobel. Self-indexing inverted files for fast text retrieval. *ACM Trans. Inf. Syst.*, 14(4):349–379, 1996.

[8] William Pugh. Skip lists: a probabilistic alternative to balanced trees. *Commun. ACM*, 33(6):668–676, 1990.