Term-Frequency Surrogates in Text Similarity Computations

Stefan Pohl *Alistair Moffat*
NICTA Victoria Research Laboratory,
Department of Computer Science and Software Engineering
The University of Melbourne
Victoria 3010, Australia
{*spohl,alistair*}@*csse.unimelb.edu.au*

Abstract *Inverted indexes on external storage perform best when accesses are ordered and data is read sequentially, so that seek times are minimized. As a consequence, the various items required to compute Boolean, ranked and phrase queries are often interleaved in the inverted lists. While suitable for query types in which all items are required, this arrangement has the drawback that other query types – notably pure ranked queries and conjunctive Boolean queries – do not require access to word position information, and that component of each posting must be bypassed when these queries are being handled. In this paper we show that the term frequency component of each posting can be completely replaced by a surrogate that allows skipping of positional information interleaved in inverted lists, and obtain significant speedups in ranked query execution without increasing the index size, and without harming retrieval effectiveness. We also explore two methods of reconstituting approximations to the original term frequencies that can be employed if use of the surrogates is deemed too risky. Our simple improvement can thus be used with all ranking functions that make use of term frequencies.*

Keywords Information retrieval, inverted index, skip pointer, proximity query, efficiency, effectiveness.

1 Introduction

Web search is an expensive operation, on which many millions of dollars are spent each year in terms of computing and energy costs. Small improvements in search efficiency can thus yield significant monetary savings, and are of considerable interest to the web search industry. Users of web search services are fickle, and if they do not get their results quickly, no matter if their query is common or rare, and easy or complex, they will switch their allegiance to a different product.

A significant fraction of the queries in a typical query log contain phrases. But even without explicitly specifying phrases in queries, documents are expected to be ranked higher if keywords occur in close proximity. To support these options, the inverted index has to store positional information for every word in every document. To minimize random access costs, it is common practise to compactly store them in the inverted lists along with the document numbers and document statistics in an interleaved fashion.

To rank documents in response to a query, similarity measures use heuristic computations based on a range of document statistics, including the overall document frequency f_t of each term t that appears in the collection, and the number of times $f_{d,t}$ that term t appears in document d. In this paper we show that the $f_{d,t}$ component in each posting in the inverted index can be replaced by a surrogate that allows skipping of the positional pointers in query modalities in which they are not required. This substitution significantly improves query execution speed, and compared to the alternative option of inserting an additional per-posting skip into each pointer, both reduces the cost of storing the index and also the cost of processing it. Because the surrogate is highly correlated with term frequency, it has only a marginal effect on retrieval quality, and in some of our trials, actually improved it. Nevertheless, we have also evaluated two approximation methods that allow reconstitution of a good approximation to the original term frequencies, to reduce the risk of effectiveness being eroded. Our simple improvement can thus be used with all ranking functions that make use of term frequencies.

2 Background

This section provides background information and introduces the relevant literature.

2.1 Inverted indexes

The inverted index is the most efficient access structure for fast document retrieval [22]. It consists of a vocabulary, storing the distinct terms t that occur in the document collection, including information such as f_t, the number of documents containing this term; and a set of inverted lists, one per term. Each term t in the vocabulary links to its inverted list, which stores a set of postings, each of which in turn reflects the occurrence of t in a document d. The term frequency $f_{d,t}$ –

Proceedings of the 13th Australasian Document Computing Symposium, Hobart, Australia, December 8, 2008.

the number of times term t occurs in document d – is usually also stored in each posting, which in simplest form has the structure

$$\langle d, f_{d,t} \rangle .$$

To evaluate a query, each term's entry is located in the vocabulary, which is held either completely in memory, or with the most frequently used portions in memory. The corresponding inverted lists are then retrieved, and depending on the query type, combined in some manner that yields a set of answers.

Different methods can be employed to evaluate inverted lists: *term-at-a-time*, which reduces the number of random accesses to disk [13]; or *document-* or *impact-at-a-time* [2, 17], which are appropriate if only the correct (or even an approximate) ranking of the top k documents is required. In the latter two approaches the inverted lists do not necessarily have to be completely processed, and the risk of additional disk seeks is mitigated by reduced processing costs. Document-level indexes are typically very compact, and when suitably compressed require just 5–10% of the space of the text they index [22].

If more complex queries are to be supported, and similarity measures employed that are based on the proximity of query terms, word positional information must also be stored in the index. A direct extension to the previous posting layout is to include the positions $p_1, \ldots, p_{f_{d,t}}$ of term t in document d in each posting:

$$\langle d, f_{d,t}, p_1, \ldots, p_{f_{d,t}} \rangle .$$

Having the positional information immediately available allows arbitrary position-based operators to be implemented, independently of the processing strategy, and can be also advantageous for snippet generation [18].

2.2 Compression

The space required by an index can be significantly reduced through compression. If document identifiers (respectively, term positions) are sorted in increasing order, the difference between consecutive entries can be stored instead. These are commonly referred to as d-gaps for documents, and p-gaps for positions. Because most d- and p-gaps are smaller than the original values, they can be stored in fewer bits, saving space. The drawback of the gap transformation is that the lists of document and position numbers need to be accessed sequentially, and it also makes it more difficult to bypass the positional component of each pointer if it is not required.

Different coding schemes for integer values have been proposed which are able to greatly reduce index size [22]. Reduction in the amount of data transferred from disk can also speed up query processing. The most effective integer coding schemes are bit-based, and unless care is taken with their implementation, add a non-trivial overhead to the computational cost of query processing. As a tradeoff between efficiency in space and time, the use of byte-aligned codes has also been suggested. Byte codes typically outperform bit-aligned compression schemes in terms of decoding speed, at the cost of a modest loss in compression effectiveness [15, 20]. In the simplest of byte code arrangements, one bit is spent to indicate whether or not the next byte is also part of the codeword for the current integer. The other seven bits in each byte store the actual integer value. More complex byte-aligned schemes have also been proposed [6, 7, 10] that reduce the amount of compression "leakage" that occurs compared to bit-aligned codes. Other fast-decoding methods (including word-aligned codes [3]) have also been devised, but typically require that the elements in the stream being compressed be drawn from the same distribution, which is not the situation when the components in each posting are interleaved.

Because not all query modes require access to the position lists, it is desirable to be able to bypass them in any given posting, and move immediately to the next posting. If positions are stored uncompressed, bypass is readily accomplished by stepping over the next $f_{d,t}$ stored values. But when the p-gaps are compressed using a variable length code, the natural ability to forwards seek over $f_{d,t}$ values is lost, a combination that means that if the positional components are not required during querying, each p-gap must be explicitly counted off. In the simplest byte code, this can be done by examining the top bit of each byte looking for stopper bytes, but even this degree of processing is a cost that is better avoided.

2.3 Index organization

The posting index layout indicated above is not the only way to store the inverted lists. An important design decision is the choice of where and how to store position information. There are three distinct alternatives.

Pointer interleaving This straightforward approach, suggested above, minimizes random accesses costs, as the lists for the terms in a query can be sequentially processed using the term-at-a-time strategy. However, in the case of pure ranked queries, in which the position information is not needed, the positions are of necessity still read from disk and accessed in memory, and have to be bypassed as every pointer is processed. On the other hand, the storage requirements are minimal, as there is no overhead incurred through storage of additional control information.

Term interleaving The positional information can also be placed into a separate part of each inverted list, or possibly even into a separate part of the inverted index. Extra space will then be required to allow coordination between the postings in the inverted lists, and their corresponding positions lists, and the space required by the overall index might approach that of having separate document-level and word-level indexes side by side. With term interleaving, performance on ranked queries should be close to that obtainable using a strictly document-level index; but phrase queries may execute more slowly than is the case with pointer interleaving, because of the need to access additional disk locations to retrieve positional information.

Phrase indexes If phrase queries are the dominant operation to be supported, additional indexes for term-pairs can be built [21]. A document-level index combined with a word-pair index has good performance for both ranked and phrase queries, but is limited to these because no position information is explicitly stored. Another approach is to index only common phrases [9].

Augmentations There have been several approaches described for augmented index organizations, in which internal structures are added within each inverted list to accelerate either the search for pointers, or to bypass blocks of pointers.

In the context of document-level indexes, and conjunctive Boolean and pure ranked queries, Moffat and Zobel showed that their "skipping" approach allows significant time savings to be made [14]. The key part of their proposal was to enlarge the index, and to at regular intervals insert into each inverted list a forward pointer, expressed as a bit or byte offset. To allow decoding to resume after one or more blocks of postings have been bypassed, the document number at the destination of each forward pointer is stored as a gap relative to the document at the start of the skip, rather than relative to the immediately preceding posting. A range of similar techniques have been proposed for other situations, including when the entire index is being held in main memory [16].

An upper bound for performance improvements through skipping can be found if the skipped data is simply not stored. Of course, doing this in regards to whole postings results in loss of generality, and means that some queries might not be properly answerable; nevertheless, this is what is done by static pruning methods, and is approximated by approaches that separate the index into two disjoint parts and seek to resolve queries using only one part [8]. Similarly, skipping the position lists in a pointer interleaved index corresponds (at best) to the use of a document-level index; and the performance differences between document- and word-level indexes can be large (for example, see Hawking [12]). In particular, we use a document-level index as one of the baselines in our experiments with the pointer-skipping approach that is described in Section 3.

Horses for courses Systems employing term-at-a-time evaluation have the advantage that disk accesses are sequential, and the number of disk seeks is minimized. On the other hand, processing the inverted lists of all terms in parallel in pursuing the document-at-a-time approach simplifies the implementation of operators such as word adjacency and proximity. Term-at-a-time systems can also resolve mandatory phrases in queries in a pre-processing step, and then only score the documents that pass the phrase or proximity constraint. This approach avoids having to store position information temporarily in the term-at-a-time accumulators for later use. Either way, if the position information is not pointer-interleaved in the inverted lists, additional linkages into the index are necessary.

Our contribution Anh and Moffat discuss the relationship between index structure and query modes [5], and conclude that pointer-interleaved indexes are slower than term-interleaved ones. However, their experiments did not allow for the possibility that each set of word positions might be able to be rapidly bypassed, and it is that opportunity that we consider in this paper. But rather than simply add skipping information to the index to allow position list bypass, we *replace* the $f_{d,t}$ value (which is the length of the positions list, counted in pointers) by a surrogate, namely the length $b_{d,t}$ (counted in bytes) of the compressed positions list. When a byte code is used, the value of $b_{d,t}$ is never smaller than $f_{d,t}$ and is usually greater; nevertheless, our proposal is that $b_{d,t}$ then be used as a surrogate for $f_{d,t}$ in the ranking formulation. As we shall see, the result of this simple substitution is that index size is largely unaffected; retrieval effectiveness is largely unaffected; and processing speed for ranked queries (assuming a pointer interleaved index) is greatly improved.

2.4 Effectiveness-efficiency tradeoffs

Important factors determining the cost of a query evaluation are the size of document collections and the length and difficulty of queries. Consequently, improvements are promising to achieve if the number of (full or even partial) document evaluations can be reduced. A constant theme has thus been methods for trading (a hopeful small loss of) retrieval effectiveness for (a hopefully large gain in) retrieval efficiency.

Heuristic ranking algorithms can be categorized by their effect on the results, compared to full, exhaustive, evaluation. The least intrusive optimizations are those that guarantee to produce identical results, that is, the same set of k top scoring documents, along with the same final similarity scores. Documents not in the top k might be scored differently.

The next fidelity level is a requirement that the top k documents be correct, and that they be in the correct score ordering, but that the scores not be faithful compared with exhaustive evaluation. This level of approximation is completely acceptable if, for example, the users of a system see the answer rankings, but not the scores. In this case even top-ranked documents may not need to be fully evaluated.

The third category includes methods that guarantee that the right documents will appear in the top k, but not that they appear in the right order. That is, the split between "shown to the user" and "not shown to the user" is correct, but the ranking might not be. Weakening the fidelity criterion allows additional short-circuiting of the similarity computation.

The last category consists of algorithms that produce rankings that are experimentally similar to those of the underlying exhaustive computation, but cannot be guaranteed in any way. This kind of approach is also of interest to system developers, since similarity functions are themselves heuristics, and there is no guarantee that slavishly executing any particular computation in full detail does in fact give the best possible retrieval effectiveness. A simpler and more succinct computation might do just as well in practice.

The restricted accumulator methods that can be used with term-at-a-time evaluation of ranked queries are examples of this final approach. The accumulators store intermediate results for each document, and represent partially evaluated similarity scores. Moffat and Zobel [14] proposed methods for limiting the number of active accumulators, saving on per-query memory costs, and also allowing the skipping pointers to gain additional traction. Many similar dynamic pruning approaches have been developed, including ones that make use of impact-sorted indexes in which processed and quantized $f_{d,t}$ are stored, rather than raw $f_{d,t}$ values. Impact-ordering also allows the effectiveness-efficiency tradeoff to be controlled on a per-query basis, and allows query evaluation to be terminated even before the end of any of the inverted lists has been reached [4]. Another recent proposal uses the similarity and dissimilarity between documents to build document clusters which can be represented in an index structure [1]. The parallel scoring of clusters and documents within those leads to speedups, but also requires new scoring functions.

3 Term-frequency surrogates

Skips that bypass groups of consecutive postings are useful in ranked querying optimizations that restrict the number of accumulators, or in conjunctive Boolean queries. The key operation in this case is to "forward search" for a specified document number, bypassing all pointers in which the document identifier is less. In this case, there is a balance to be struck between short and long skip groups – too short, and access gets slowed by the large number of skip pointers that themselves must be handled, as well as the index becoming enlarged; and too long, and it is likely that the pointer being sought appears in the very next block anyway. That is, any additional control information not only adds to the space requirement of the index, but also potentially introduces additional costs during query processing, if stored interleaved. Each augmentation value in a pointer-interleaved inverted list has to be read and at least inspected, even if it is not used to evaluate the query.

3.1 Skipping positions

In evaluating pure ranked queries, position information is not used, and it is natural to consider adding a skip to every pointer so that the positional components can be bypassed. But positional components are typically very fine-grained units, and adding another value to every posting is potentially expensive. Given that the typical posting in a text index contains around 5–10 values (a d-gap, an $f_{d,t}$ value, and then 3–8 positions on average), the overhead might be 10–20% in terms of space.

Hence, instead of adding a skip element to the posting and (hopefully) trading execution time for storage space, we fold the desired control information (the skip amount) into the information that must be read and decoded anyway (in particular, the $f_{d,t}$ value). With this small adjustment, skipping of positional lists suddenly becomes feasible. At risk, of course, is the fidelity of the similarity scoring process for ranked queries, since this is why $f_{d,t}$ values are included in the index.

The key observation that allows the substitution to take place is that these two values are reasonable well correlated – because of the monotonic relationship between integer values and the byte length of their byte-coded representations, lists that contain many p-gaps are almost always longer than lists that contain a smaller number of p-gaps. That is, we hypothesize that ranked querying retrieval effectiveness should not change dramatically if $b_{d,t}$, the byte length of the positions list in document d for term t, is used in place of $f_{d,t}$, the actual frequency of t in d. In particular, we note again that the similarity function is itself a heuristic, and should be resistant to small changes in document statistics.

3.2 Correlation

The skip pointer for a position list is just the number of bytes $b_{d,t}$ it takes to code the $f_{d,t}$-long position list of document d and term t, decremented by one if it is given that at least one position must appear in every posting. The compressed position list length in bytes using a byte code is at least as large as the term frequency, because at least one byte is necessary for every coded value. Perfect correlation would be achieved if each position gap could be coded in exactly one byte, which might in fact hold in domains with very short documents (containing $\leq$ 128 indexed words). Figure 1 illustrates the relationship between $f_{d,t}$ and $b_{d,t}$, and shows the least and greatest $f_{d,t}$ value associated with each $b_{d,t}$ value over the approximately six billion postings present in a full inverted index for the 426 GB TREC gov2 collection. The strength of the relationship is clear.

3.3 Space overhead

The number of bytes needed to store an integer using the simple byte code is a monotonic function of its value, and because $b_{d,t} \geq f_{d,t}$, storing the surrogate instead of term frequency gives rise to a slight increase

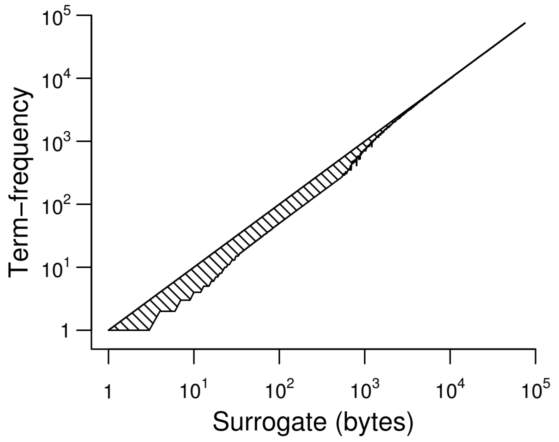

Figure 1: Term-frequency $f_{d,t}$ as a function of compressed byte length $b_{d,t}$, when positional p-gap lists are stored using the simple byte coder. The upper and lower lines record the extremes measured on the TREC gov2 collection. Except for relatively short lists, the two are very closely correlated.

<table>
</table>

System	Space	
	GB	%
Base	44.418	100.0
Full skips	50.174	113.0
Part skips, $f_{d,t} > 10$	44.673	100.6
Surrogate skips	44.421	100.0

Table 1: Space requirements of different word-level index arrangements for the gov2 collection. Row "Part skips, $f_{d,t} > 10$" provides for retention of the $f_{d,t}$ value included in rows "Base" and "Full skips", but adds skip information only when there are more than ten p-gaps in the position list. Row "Surrogate skips" replaces the $f_{d,t}$ values by $b_{d,t}$ values.

in index size. Table 1 shows space overheads for different alternative indexes relative to storing term frequency alone for the gov2 collection described in more detail in Section 4.2, including for a compromise arrangement that adds positional skip information only when there are more than ten p-gaps to be bypassed. Use of full positional skips adds 13% to the index, whereas storage of $b_{d,t}$ in place of $f_{d,t}$ translates into an increase in index size of an inconsequential 3 MB over a 44 GB index.

3.4 Reconstructing frequencies

Most similarity functions are based on $f_{d,t}$ in some way. To facilitate the incorporation of our surrogate with different similarity functions in a transparent manner, it is important to find ways to reconstitute original term frequencies. This way, the surrogate can be plugged into any search engine and is independent of the similarity metric being used.

One simple arrangement is for a lookup table to be employed, in which for each posting, a combination of context variable settings (such as F_t, the overall collection frequency of the term; f_t, the document frequency of the term; and $b_{d,t}$) is mapped to the average term frequency observed for that combination of conditioning values. This table can then be used to convert a given $b_{d,t}$ value, in a given context, back to an estimated $f_{d,t}$ that can be used in the similarity computation.

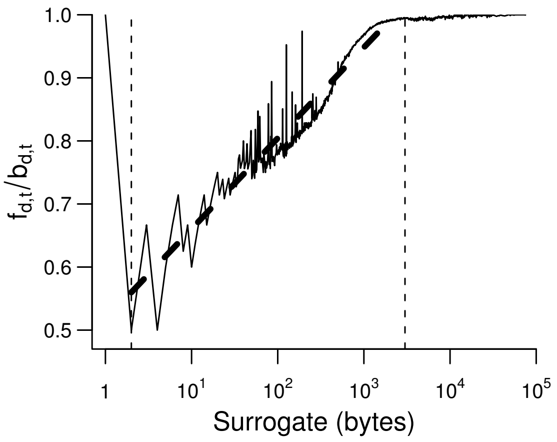

Figure 2: Average observed ratio $f_{d,t}/b_{d,t}$ as a function of the compressed byte length $b_{d,t}$, for the TREC gov2 collection.

The two most interesting variables in search engines are the document-frequency f_t, indicating the number of documents in which a term occurs, and the correlated term frequency surrogate itself.

It turns out (based on experiments not reported here) that the term-document frequency f_t has a very low correlation with the ratio $f_{d,t}/b_{d,t}$, and the best single predictor of $f_{d,t}$ is $b_{d,t}$ alone, with no other context information. Calculating the average value of $f_{d,t}/b_{d,t}$ for each distinct $b_{d,t}$ value yields the relationship plotted in Figure 2. As expected, $b_{d,t} = 1$ uniquely indicates $f_{d,t} = 1$, and for large byte-lengths ($b_{d,t} >$ 3,000), strongly indicates $f_{d,t} = b_{d,t}$. Between these two extremes, a small table serves to capture the observed average relationship.

An even more compact representation is to derive a formula to reconstitute term frequencies. As can be seen from the mid-section of Figure 2, a logarithmic function is a good fit for the indicated interval, and can be pre-calculated at indexing time. Then, during query evaluation, either the direct mapping or the formula is employed, depending on the value of $b_{d,t}$. The dashed line fitted to Figure 2 shows the relationship $f_{d,t} \approx b_{d,t}(0.5 + (1/7)\log_{10} b_{d,t})$.

Finally, note that – with a relatively small amount of computation – exact $f_{d,t}$ frequencies are still available if they are required. All that is necessary is that the next $b_{d,t}$ bytes be processed to count (in the case of the simple byte code and the (S, C)-byte code) the number of stopper bytes. This option could be employed conditionally when exact reproduction of a similarity formula is required. For example, the correct ordering of the top k ranked documents generated by the surrogate approach might then be fully scored, to place them into final presentation order.

4 Experimental results

We adapted version 0.9.3 of the freely available research search engine `zettair`.[1] To measure the impact of using $f_{d,t}$ surrogates, we performed experiments in terms of both efficiency and effectiveness.

4.1 Experimental arrangement

The TREC `gov2` collection is currently the biggest available research dataset, and was the main resource used in the efficiency experiments. It consists of more than 25 million documents and 426 GB of data, and represents a large portion of the available (crawlable) `.gov` part of the web, as of early 2004. A word-level byte coded index for `gov2` occupies 44 GB, and contains more than 6 billion postings.

We used the first 1,000 queries of the query log provided for the efficiency task of the TREC 2006 Terabyte Track. The queries are a mixture of different length ranked queries and do not contain phrases. This is not an issue, because the use of a surrogate does not affect query execution times for phrase queries. We performed neither stemming nor stopping of the index or queries. Timings are for production of a ranking of the top 20 result documents, without lookup of documents or generation of snippets, using a Linux server running Ubuntu 7.10 and kernel version 2.6.22, with dual quad-core Xeon E5345, 2.33 GHz, 64-bit processors and 4 GB of main-memory. The machine was otherwise under light load during experiments and memory was flushed between the runs for different systems and data sizes.

4.2 Efficiency

To measure the extent to which skipping position lists speeds up ranked query processing, we measured timings for the two limiting cases: when all of the inverted lists for each query have to be fetched from disk; and when all of the necessary data is readily available in main memory as a result of a recent execution of the same query. The first case is representative for search in large collections that do not fit into memory, while the latter measures the improvement for in-memory search. A real-world search engine under steady load will have performance somewhere between these extremes, because the index data for at least some of the terms in some of the queries is likely to be available in main memory via standard caching mechanisms.

To achieve these measurements, we executed the query stream with each query immediately re-evaluated after its first evaluation, and kept separate records of the two execution times. This way, the second query is likely to be executed without accesses to disk, because the index data required will probably be served from the disk-cache managed by the operating system. We

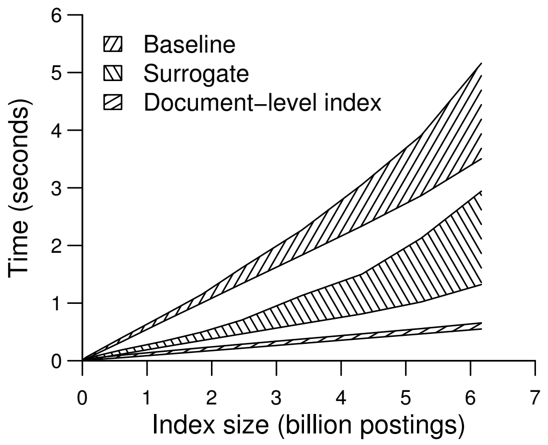

Figure 3: Average per query execution time of the first 1,000 efficiency queries of the TREC 2006 Terabyte Track. The upper and lower ranges for each system and index size show the cost of executing that query with and without disk accesses.

measured timings for the same set of queries over a range of collection sizes, with random sampling from the whole of `gov2` used to form the sub-collections.

Figure 3 gives the results of these experiments, where the vertical axis denotes average query time (in seconds), and the horizontal axis shows the size of the index, measured in pointers. Three different systems were tested: the original `zettair` word-position index, with pointer-interleaved positional lists; a modified `zettair` in which $f_{d,t}$ is replaced in each pointer by $b_{d,t}$, and all (pointer interleaved) positional lists are bypassed during ranked query execution; and a document-level index in which no positional information is stored at all.

The word-level index using the surrogate is demonstrably faster than the original, and when disk access costs are also factored out, is close to the execution times of the much smaller document-level index. That is, the in-memory case appears to be CPU-bound and exhibits a linear growth in cost with increasing index size; on the other hand, the two upper limits show a distinctive curve as the index size grows beyond 40 GB, and it becomes harder for the operating system to exploit inter-query caching effects.

Other experiments not included here have shown that stopping the queries leads to much smaller execution times, but that the surrogate system keeps its advantage over the baseline; and that executing queries a third (or even fourth) time always leads to timings consistent with the second execution.

4.3 Effectiveness

To see how the use of surrogate $f_{d,t}$ values affected the quality of ranked retrieval, we compared four versions of `zettair` over three different sets of TREC topics and data: Topics 701–750 on the `gov2` collection; Topics 401–450 on the `wt2g` collection; and Topics 451–500 on `wt10g`. The title of the topic was always

[1]`http://www.seg.rmit.edu.au/zettair/`

System	TREC gov2		TREC wt2g		TREC wt10g	
	MAP	σ	MAP	σ	MAP	σ
Baseline	0.237[†]	0.184	0.294	0.219	0.203	0.220
Surrogate	0.250[*]	0.182	0.294	0.214	0.199	0.215
Surrogate $b_{d,t}$	0.243[*†]	0.184	0.299	0.212	0.197	0.213
Surrogate Formula	0.243[*†]	0.184	0.298	0.214	0.198	0.214

Table 2: Effectiveness comparison using TREC topics 701–850 on the gov2 collection; topics 401–450 on the wt2g collection; and topics 451–500 on the wt10g collection. The MAP values were tested for significance at the 0.05 level, with * denoting significant relative to the Baseline, and † denoting significant relative to Surrogate.

taken as the query. We compared the performance of the unmodified baseline `zettair` system with our plain surrogate that uses $b_{d,t}$ instead of $f_{d,t}$; with a version that computed an approximate $f'_{d,t}$ using a lookup array indexed by $b_{d,t}$, as depicted by the plotted line in Figure 2; and with a version that computed an approximate $f''_{d,t}$ using a formula (but still implemented as a lookup table indexed by $b_{d,t}$), as depicted by the dashed fitted line in Figure 2.

Retrieval effectiveness was quantified using Mean Average Precision (MAP) and the standard TREC methodology, which measures MAP on the first 1,000 returned documents, and assumes that unjudged documents are irrelevant. We then performed pairwise t-tests at the 0.05 level to gauge significance. Within `zettair` we used the default Dirichlet-smoothed language modeling similarity function (with $\mu = 1{,}500$) that has been found to perform best across these datasets [11].

As can be seen from the results shown in Table 2, the influence of the surrogate is mixed, but always small. On the gov2 collection, use of surrogates gave rise to a significant *gain* in measured effectiveness, and the two reconstitution approaches then shifted effectiveness back towards that of the baseline `zettair`. On the other hand, on both wt2g and wt10g, no significant differences in MAP were recorded, perhaps partly because of the smaller number of topics in these two collections, but primarily because surrogates just didn't seem to make much of a difference.

Figure 4 shows, for the gov2 collection and Topics 701–850, the effect of replacing $f_{d,t}$ by $b_{d,t}$ on a topic by topic basis, and confirms that the use of surrogates perturbs almost all MAP scores up or down by a small amount, rather than create any large shifts, or move them all consistently in the same direction.

Rather than converting pairs of rankings to effectiveness scores and then comparing the scores, it is also possible to directly compare the relative fidelity of the two rankings, in terms of whether they retrieve the same documents in the same order, regardless of relevance. The *Rank-Biased Overlap* (RBO) computation of Webber et al. [19] provides a way of calculating the difference between two indefinite rankings, with a bias towards early positions in the ranking, and the ability to deal with rankings over disjoint sets of objects. A parameter $0 \le p < 1$ controls the extent of the top-weightedness of the comparison, with p interpreted as being the *persistence* of a user who is side-by-side comparing overlap between the two rankings, and steps from rank r to rank $r+1$ with probability p. A useful property of the RBO formulation is that the influence of the (unranked) tail sections of the lists is bounded, and the RBO score for any pair of rankings is a range that can be calculated without looking at all documents.

Table 3 lists RBO scores for the gov2 collection at two different values of p. Broadly speaking, $p = 0.9$

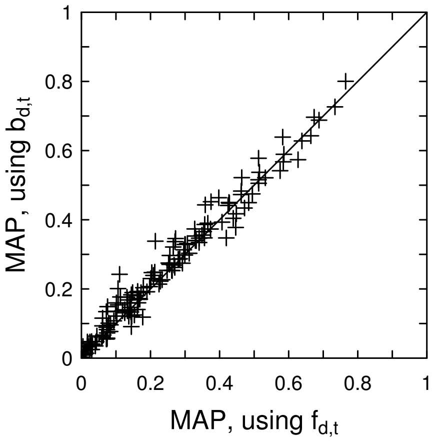

Figure 4: Surrogate versus Baseline, with MAP scores for Topics 701–850 using the $b_{d,t}$-based computation plotted as a function of the MAP scores attained using the original $f_{d,t}$-based computation.

System	RBO	
	$p = 0.9$	$p = 0.99$
Baseline	1.000	1.000
Surrogate	0.768	0.796
Surrogate $b_{d,t}$	0.782	0.839
Surrogate Formula	0.779	0.838

Table 3: Rank-biased overlap (RBO) for two different values of p using TREC Topics 701–850 on the gov2 collection.

places the bulk of the emphasis on the top ten documents in the ranking, and an RBO of 0.8 suggests that around eight of the top-10 determined by the Baseline computation appear in the same top-10 positions as in the Surrogate one. Similarly, $p = 0.99$ spreads the emphasis to depth 100 and beyond, and a score of 0.8 suggests that around 80% of the elements are in, or not too far from, their original rank positions. Use of RBO in this experiment neatly indicates the quality of the reconstitution methods – both give RBO scores higher than the plain Surrogate one.

5 Conclusion

Interleaving position information in inverted indexes allows processing of queries with only one disk seek per term in term-at-a-time systems, and reduces the number of parallel pointers required in document- and score-at-a-time systems. We proposed a term frequency surrogate as a way of speeding up query processing in such indexes, without adding to the space required by the index. We have shown that the length of the compressed p-gaps is highly correlated with term frequency, and allows the use of $b_{d,t}$ instead of $f_{d,t}$ in similarity computations. We also used approximation functions of differing quality and efficiency to reconstitute the original term frequencies. The surrogate method is demonstrably faster than the baseline approach, and approaches the speed on a document-level index for ranked queries. Effectiveness is largely unchanged by the substitution.

Acknowledgements Justin Zobel took part in a number of fruitful discussions, and William Webber provided the RBO implementation. National ICT Australia (NICTA) is funded by the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council.

References

[1] I. S. Altingovde, E. Demir, F. Can and Ö. Ulusoy. Incremental cluster-based retrieval using compressed cluster-skipping inverted files. *ACM Trans. Information Systems*, Volume 26, Number 3, pages 1–36, 2008.

[2] V. N. Anh, O. de Kretser and A. Moffat. Vector-space ranking with effective early termination. In *Proc. 24th Ann. Int. ACM SIGIR Conf. Research and Development in Information Retrieval*, pages 35–42, New Orleans, Louisiana, United States, 2001. ACM.

[3] V. N. Anh and A. Moffat. Improved word-aligned binary compression for text indexing. *IEEE Trans. Knowledge and Data Engineering*, Volume 18, Number 6, pages 857–861, June 2006.

[4] V. N. Anh and A. Moffat. Pruned query evaluation using pre-computed impacts. In *Proc. 29th Ann. Int. ACM SIGIR Conf. Research and Development in Information Retrieval*, pages 372–379, Seattle, Washington, USA, 2006. ACM.

[5] V. N. Anh and A. Moffat. Structured index organizations for high-throughput text querying. In *Proc. String Processing and Information Retrieval Symposium*, pages 304–315, Glasgow, Scotland, October 2006. LNCS 4209, Springer.

[6] N. R. Brisaboa, A. Fariña, G. Navarro and M. F. Esteller. (S, C)-dense coding: An optimized compression code for natural language text databases. In *Proc. String Processing and Information Retrieval Symposium*, pages 122–136, Manaus, Brazil, October 2003. LNCS Volume 2857.

[7] N. R. Brisaboa, A. Fariña, G. Navarro and J. R. Paramá. Simple, fast, and efficient natural language adaptive compression. In *Proc. String Processing and Information Retrieval Symposium*, pages 230–241, Padova, Italy, October 2004. LNCS Volume 3246.

[8] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer and J. Y. Zien. Efficient query evaluation using a two-level retrieval process. In *Proc. 2003 ACM CIKM Int. Conf. Information and Knowledge Management*, pages 426–434, New Orleans, Louisiana, November 2005. ACM Press, New York.

[9] M. Chang and C. K. Poon. Efficient phrase querying with common phrase index. *Information Processing & Management*, Volume 44, Number 2, pages 756–769, 2008.

[10] J. S. Culpepper and A. Moffat. Enhanced byte codes with restricted prefix properties. In *Proc. String Processing and Information Retrieval Symposium*, pages 1–12, Buenos Aires, November 2005. LNCS Volume 3772.

[11] S. Garcia, N. Lester, F. Scholer and M. Shokouhi. RMIT University at TREC 2006: Terabyte track. In *Proc. 15th Text REtrieval Conference (TREC)*, Gaithersburg, MD, 2007. National Institute of Standards and Technology.

[12] D. Hawking. Efficiency/effectiveness trade-offs in query processing (from theory into practice workshop, 1998 SIGIR conf.). *SIGIR Forum*, Volume 32, Number 2, pages 16–22, 1998.

[13] M. Kaszkiel, J. Zobel and R. Sacks-Davis. Efficient passage ranking for document databases. *ACM Trans. Information Systems*, Volume 17, Number 4, pages 406–439, 1999.

[14] A. Moffat and J. Zobel. Self-indexing inverted files for fast text retrieval. *ACM Trans. Information Systems*, Volume 14, Number 4, pages 349–379, 1996.

[15] F. Scholer, H. E. Williams, J. Yiannis and J. Zobel. Compression of inverted indexes for fast query evaluation. In *Proc. 25th Ann. Int. ACM SIGIR Conf. Research and Development in Information Retrieval*, pages 222–229, Tampere, Finland, 2002. ACM.

[16] T. Strohman and W. B. Croft. Efficient document retrieval in main memory. In C. L. A. Clarke, N. Fuhr, N. Kando, W. Kraaij and A. P. de Vries (editors), *Proc. 30th Ann. Int. ACM SIGIR Conf. Research and Development in Information Retrieval*, pages 175–182, Amsterdam, The Netherlands, July 2007. ACM Press, New York.

[17] T. Strohman, H. Turtle and W. B. Croft. Optimization strategies for complex queries. In *Proc. 28th Ann. Int. ACM SIGIR Conf. Research and Development in Information Retrieval*, pages 219–225, Salvador, Brazil, 2005. ACM.

[18] A. Turpin, Y. Tsegay, D. Hawking and H. E. Williams. Fast generation of result snippets in web search. In *Proc. 30th Ann. Int. ACM SIGIR Conf. Research and Development in Information Retrieval*, pages 127–134, Amsterdam, The Netherlands, 2007. ACM.

[19] W. Webber, A. Moffat and J. Zobel. A similarity measure for indefinite rankings. Manuscript, November 2008.

[20] H. E. Williams and J. Zobel. Compressing integers for fast file access. *Computer Journal*, Volume 42, pages 193–201, 1999.

[21] H. E. Williams, J. Zobel and D. Bahle. Fast phrase querying with combined indexes. *ACM Trans. Information Systems*, Volume 22, Number 4, pages 573–594, 2004.

[22] I. H. Witten, A. Moffat and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, second edition, May 1999.