Pruned Query Evaluation Using Pre-Computed Impacts

Vo Ngoc Anh
Computer Science and Software Engineering
The University of Melbourne
Victoria 3010, Australia
vo@csse.unimelb.edu.au

Alistair Moffat
Computer Science and Software Engineering
The University of Melbourne
Victoria 3010, Australia
alistair@csse.unimelb.edu.au

ABSTRACT

Exhaustive evaluation of ranked queries can be expensive, particularly when only a small subset of the overall ranking is required, or when queries contain common terms. This concern gives rise to techniques for dynamic query pruning, that is, methods for eliminating redundant parts of the usual exhaustive evaluation, yet still generating a demonstrably "good enough" set of answers to the query. In this work we propose new pruning methods that make use of impact-sorted indexes. Compared to exhaustive evaluation, the new methods reduce the amount of computation performed, reduce the amount of memory required for accumulators, reduce the amount of data transferred from disk, and at the same time allow performance guarantees in terms of precision and mean average precision. These strong claims are backed by experiments using the TREC Terabyte collection and queries.

Categories and Subject Descriptors

H.3.1 [Information Storage and Retrieval]: Content analysis and indexing – *indexing methods*; H.3.2 [Information Storage and Retrieval]: Information storage – *file organization*; H.3.3 [Information Storage and Retrieval]: Information search and retrieval – *search process*; H.3.4 [Information Storage and Retrieval]: Systems and software – *performance evaluation*.

General Terms

Experimentation, performance, algorithms.

1. INTRODUCTION

Exhaustive evaluation of ranked queries can be expensive, particularly when only a small subset of the overall ranking is required, or when queries contain common terms. For example, typical two or three word web-style queries tend to contain at least one word that is not rare, meaning that a non-trivial fraction of the documents in the collection might need to be scored before the query is resolved. But if only a small number of highly scoring documents are required for presentation to the user, exhaustive query processing might be an extravagant expense.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGIR'06, August 6–11, 2006, Seattle, Washington, USA.
Copyright 2006 ACM 1-59593-369-7/06/0008 ...$5.00.

The concern for speedy query evaluation gives rise to techniques for dynamic query pruning, that is, methods for eliminating redundant parts of the usual exhaustive evaluation, yet still generating a demonstrably "good enough" set of answers to the query. Given the objective of reducing the time spent on each query, only a subset of the index pointers associated with the query term can be processed. The first goal of any pruning strategy is thus to identify how best to isolate a suitable subset of each term's pointers, while still retaining confidence in the quality of the resultant ranking. Consequent to the goal of identification is a second goal of application – when only a minority subset of the pointers are to be used, appropriate data structures and index organizations need to be employed so as to capitalize on the possibility of saving execution time.

In this work we propose new pruning methods that make use of impact-sorted indexes. Compared to exhaustive evaluation, the new methods reduce the amount of computation performed, reduce the amount of memory required for accumulators, reduce the amount of data transferred from disk, and at the same time allow performance guarantees in terms of precision and mean average precision. These strong claims are backed by experiments using the TREC Terabyte collection and queries.

2. SIMILARITY COMPUTATIONS

Ranked query evaluation is based on the notion of a *similarity heuristic*, a function that combines observed statistical properties of a document (in the context of a collection) and a query, and computes a numeric score indicating the likelihood that the document is an answer to the query. There have been many such functions devised over more than forty years of research (see, for example, Voorhees and Harman [2005]). In this paper we focus on the implementation of the term-frequency based impact-oriented computation described by Anh and Moffat [2005]. This section briefly summarizes that computation, and explains why it is of interest.

The key idea of the impact-based scoring regime described by Anh and Moffat is that documents are treated as independent self-modelling entities, and used holistically to assign *impact* scores to the terms that appear within them. To do this, the queryable terms that appear within each document are sorted by decreasing within-document frequency $f_{d,t}$. Based on that ordering, each term is assigned an impact varying from k down to 1, with exponentially growing numbers of terms assigned to the lower-valued buckets. The exception is a set of 600 common stop words, which, whenever they occur, are always assigned an impact score of 1 [Anh and Moffat, 2005]. The document-term impact associated with term t in document d is denoted $\omega_{d,t}$. The upper limit k is decided at index construction time, and is typically a value such as $k = 8$.

For example, consider a document in which $n_d = 55$ distinct terms appear, of which $n_s = 10$ are stop words. The base B of the exponentially growing set of bucket sizes is calculated via $B = (n_d - n_s + 1)^{1/k}$, and the impact buckets are allocated to contain $(B-1)B^i$ items, where $i \in 0 \ldots 7$, items. When rounded off to integers, these values correspond to $1, 1, 1, 3, 4, 7, 11, 17$ items. (The Fibonacci-ish sequence of impact bucket sizes that results from this computation is, of course, no coincidence.) That is, the most frequent non-stop word in the document is assigned an impact of 8 (in this document); the next most frequent non-stop word an impact of 7; and so on; until the 17 most frequent non-stop words are all assigned an impact of 1, along with the 10 stop words.

There is a slight complication that arises when terms have equal frequency within the document. In order to avoid discriminating between terms of equal frequency, and so as to avoid any requirement for a secondary sort key, groups of terms with equal $f_{d,t}$ are treated identically, and are all assigned the impact that would accrue to the middle term in the cluster. In the example, if there were 20 non-stop word terms with $f_{d,t} = 1$ then all 20 would be assigned to the impact-1 bucket, even though nominally only 17 terms should have that impact score.

In a classical TF·IDF interpretation, these document-term impacts account for both the TF component, and, because they are relative to the number of terms in the document, document length normalization. The IDF component is supplied when the corresponding query-term impacts are computed. Following Anh and Moffat [2005], we compute query-term weights as

$$(1 + \log_e f_{q,t}) \times (\log_e(1 + f^m/f_t)) ,$$

where f^m is the maximum value of f_t over the collection, and $f_{q,t}$ is the number of times term t appears in the query. The set of query-term weights is then transformed to a set of integer query-term impacts by linear scaling and integer quantization so that the maximum query-term impact is exactly k. The query-term impact associated with term t in query q is denoted $\omega_{q,t}$.

Finally, the similarity of document d and query q is calculated as the inner-product of the two integer impact vectors:

$$S_{d,q} = \sum_{t \in d \cap q} \omega_{d,t} \times \omega_{q,t} ,$$

and is a number between 0 and $k^2|q|$.

Anh and Moffat [2005] give retrieval effectiveness results that show that this formulation of similarity provides highly competitive retrieval performance when evaluated with reference to human relevance judgments on standard sets of queries. Because it is based on integer computations, it also allows for fast implementation; but at the cost of non-trivial amounts of memory. A key objective of our investigation was to preserve the retrieval effectiveness of the impact-based approach, while reducing the resources required to implement it.

3. INDEX REPRESENTATIONS

The conventional arrangement for storing the pointer lists in an inverted index is in *document order*, see, for example, Witten et al. [1999]. This ordering is simple, and offers compact storage, because the ascending set of document numbers is amenable to a range of compression techniques. Document-sorted indexes can be processed in either *document-at-a-time* mode or *term-at-a-time* mode. In document-at-a-time operations, the index lists for all terms are processed in tandem, and a merge/intersection operation performed. Each document is assigned a final score before the next document is considered, and a running set R of the top r highest scoring documents is maintained at all times. If other query features require checking while documents are being selected – for example, Boolean constraints, phrase or proximity restrictions, or structural constraints – that checking can be included in the same document-at-a-time pass.

The alternative, term-at-a-time processing, requires an *accumulator* variable per document, and an operational regime in which each term's inverted list is processed, and partial score contributions are added to the accumulators of the corresponding documents. Once all the terms' lists have been processed and any necessary normalizations undertaken (to account for document length, for example), the r documents with the largest accumulators are identified, and returned to the user as answers.

The advantage of term-at-a-time processing is that the merge is implicit, and is based on radix-type operations rather than the more expensive (particularly if there are many query terms) comparison-based merging. The disadvantage of term-at-a-time processing is the need for the accumulators, in an array (or some other data structure) indexed by document number. In a collection of 25 million documents, an array-based accumulator structure requires 100 MB for a simple ranked query; and potentially even more if complex constraints such as phrase or proximity restrictions must be checked.

A third processing mode that is described in more detail in the next section is the *score-at-a-time* approach, in which a suitably structured index is created, and pointers are applied in an order that is neither strictly term-based nor strictly document-based.

Finally in this section, we note that another way in which indexes can be categorized is by the type of information contained in each pointer. In a *document-level* index each pointer contains a single document number, and a weighting factor (such as an impact score $\omega_{d,t}$, or an $f_{d,t}$ count) to allow ranked queries to be evaluated. The alternative is a *word-level* index, in which each pointer also includes a list of ordinal offsets at which that term appears in the document. In the experiments in this paper the focus is on the evaluation of ranked queries using a document-level index; Anh and Moffat [2006] examine how best to structure the word-level indexes necessary for more complex query modalities.

4. PRUNING TECHNIQUES

To reduce similarity computation effort, the notion of *pruning* was introduced by Buckley and Lewit [1985] for term-at-a-time processing, and by Turtle and Flood [1995] for document-at-a-time processing. These authors reasoned that a system that correctly identifies the top r documents is no less useful than one that completely scores the whole collection. In the case of the term-at-a-time approach of Buckley and Lewit, whole query terms might be dropped as a result of pruning, and when they are, both processing time and disk transfer time can be saved. On the other hand, in the case of the document-at-a-time approach of Turtle and Flood, some pointers might be dispensed with after just a cursory amount of processing, saving overall processing costs; but all inverted lists must be fetched.

Moffat and Zobel [1996] describe a mechanism for inserting additional information called "skips" into document-sorted compressed inverted lists in order to support a term-at-a-time processing strategy that they called CONTINUE. Skips are forward pointers within a compressed inverted list, and allow unnecessary sections to be passed over with minimal effort, and then decoding resumed. The other key aspect of the CONTINUE approach is the notion of OR-mode and AND-mode processing of index pointers. If a pointer to some document is processed in OR-mode, then it has the authority to nominate this document as being a potential answer, and have it considered by subsequent processing steps, even if no other

Processing mode	Document-sorted index organization	Impact- or frequency-sorted index organization
Document-at-a-time	Reasonably fast unless query has many terms. Non-ranked query components (Boolean, phrases, proximity, structure) can be incorporated. Pruning allows some pointers to be processed quickly, but all of each list must be fetched, and dropping of whole terms is not practical. No accumulators are required, and memory footprint is small. Pruning effectiveness can be guaranteed in terms of precision, see Turtle and Flood [1995] and Strohman et al. [2005].	*Reasonably fast. See Section 8.*
Term-at-a-time	Fast. Accumulator structure required. Pruning reduces memory requirement, and allows faster processing if skips inserted into inverted lists, see Moffat and Zobel [1996]. Unless whole lists are dropped, all of each list must be fetched. Pruning effectiveness can be guaranteed in terms of precision, see Buckley and Lewit [1985].	Fast. Accumulator structure required. Pruning possible by dropping tail of any list, or by dropping whole lists, saving all of memory space, processing time, and (if lists are read one block at a time) disk transfer time.
Score-at-a-time	Not practical.	Fast. Accumulator structure required. Pruning possible by dropping tail of any list, or by dropping whole lists, saving all of memory space, processing time, and (if lists are read one block at a time) disk transfer time, see Persin et al. [1996], Hawking [1998]. *With an impact-sorted index, pruning effectiveness can be guaranteed, in terms of precision and/or average precision, see Section 5.*

Table 1: Combinations of index organization and processing mode, and options for dynamic pruning.

query terms appear in it. Every document that is eventually scored and ranked must have been nominated by a pointer processed in OR-mode. On the other hand, pointers processed in AND-mode are permitted to boost the scores of previously nominated documents, but are not allowed of themselves to nominate documents. If all of the index pointers corresponding to some document are processed in AND-mode, then that document will not be scored, and will not be considered as a candidate answer.

Moffat and Zobel showed that term-at-a-time processing of selected query terms in OR-mode, and of the remaining terms in AND-mode, resulted in considerable reduction in the space required by accumulators compared to an exhaustive evaluation; faster query processing; and no significant effect on retrieval effectiveness.

A third mode of processing is IGNORE-mode, in which pointers are simply not considered. With these three modes available, the term-at-a-time pruning methods explored by Moffat and Zobel [1996] can be described as follows:

- Exhaustive: all pointers are processed in OR-mode.
- QUIT: a subset of the terms have all of their pointers processed in OR-mode, while the remaining terms have their pointers processed in IGNORE-mode.
- CONTINUE: a subset of the terms have all of their pointers processed in OR-mode, while the remaining terms have their pointers processed in AND-mode.

The benefit of having a significant number of pointers ignored is clear – there will be a saving in execution cost. In particular, if the IGNORE-mode pointers in each inverted list form a contiguous group, they can be discarded *en masse*. For example, the QUIT strategy of Moffat and Zobel (see also Lester et al. [2005]) potentially ignores whole inverted lists, as does the pruning technique of Buckley and Lewit [1985]. Fast query processing is the result.

There is also a tangible benefit in terms of memory consumption if the number of pointers processed in OR-mode can be kept small. As was noted above, the memory space required for accumulators can be significant, and minimizing the number of OR-mode pointers allows that cost to be controlled.

Even with skips inserted, it is, however, still necessary for the whole of each term's document-sorted inverted list to be read from disk. Persin et al. [1996] recognized that the transfer costs were also an issue (and one that would become more important as the relative costs of disk access and processing time continued to shift) and suggested storing the pointers in the inverted lists in decreasing within-document frequency order, rather than in increasing document number. Their *frequency-sorted* indexes naturally group an attractive set of OR-mode pointers close the front of each inverted list, and, equally importantly, group the unattractive IGNORE-mode pointers together in the tail section of each inverted list; where "attractive" is defined as "having a high within-document frequency compared to other pointers for that term".

Anh et al. [2001] (see also Hawking [1998]) further extended the frequency-sorted approach, and suggested that the index be stored in *impact-sorted* form, where the index pointers in each list are ordered according to their overall contribution to the similarity score, as described in Section 2. Impact-sorted indexes can be used for term-at-a-time processing or for score-at-a-time processing, in both cases making use of a set of accumulators. In the latter case, all of the inverted lists are simultaneously open, with the merge by decreasing contribution $\omega_{d,t} \times \omega_{q,t}$, so that pointers that make big contributions to document scores are processed first. This processing mode provides good support for dynamic pruning – the front part of each inverted list can be processed in OR-mode; then a section processed in AND-mode; and then the rest of the list ignored. Provided that long lists are fetched in fixed-length blocks, rather than all at once, disk transfer savings are likely.

Table 1 summarizes these various options for index organization, processing mode, and pruning. The italicized statement in the last cell shows the main purpose of this paper – we describe a pruning technique suitable for score-at-a-time processing using an impact-sorted index, and show that it is possible to obtain guaranteed performance in terms of precision and mean average precision. The new method has low memory consumption, and is also significantly faster in execution than exhaustive evaluation, because only the required part of each inverted list need be transferred from disk.

To set a baseline for those experiments, the following pseudocode shows the use of a score-at-a-time evaluation strategy, without any pruning. The set of accumulators is denoted A; and R is a set of r highly-scoring documents, all of which have a similarity score greater than or equal to R_{min}. When document-term impacts are in the range $k \ldots 1$ and query-term impacts are in the range $k \ldots 1$, the maximum similarity contribution from any single pointer is k^2. The main loop or the algorithm counts through exactly these k^2 possible values. Note also that to aid the explanation, each pointer is presumed to be of the form $\langle t, d, \omega_{d,t} \rangle$, but that the actual index has common t and $\omega_{d,t}$ values factored out so as to save space.

```
[**Baseline – Exhaustive computation**]
set $A \leftarrow \{\}$ and $R \leftarrow \{\}$
**for** $I \leftarrow k^2$ **downto** $1$ **do**
    **for** each pointer $\langle t, d, \omega_{d,t} \rangle$ for which $I = \omega_{d,t} \times \omega_{q,t}$ **do**
        set $A_d \leftarrow A_d + I$
        if necessary, adjust $R_{min}$ and the membership of set $R$
    **end for**
**end for**
sort set $R$ and return it to the user
```

Other researchers have also considered techniques for accelerating query processing by suppressing – at either index construction time or query evaluation time – some subset of the pointers associated with each term [Harman and Candela, 1990, Brown, 1995, Soffer et al., 2001, Broder et al., 2005, Theobold et al., 2005, Strohman et al., 2005, de Moura et al., 2005], but space restrictions preclude a detailed discussion of that work. However it is worth noting that the "top docs" approach of Brown [1995] and Strohman et al. [2005] is in some ways a rudimentary form of a frequency- or impact-ordered index, with a subset of the pointers identified as being "important" and extracted into a first block of pointers at the start of the list. But where Brown and Strohman et al. use two levels of pointer importance, we are able to use k.

5. IMPACT-BASED PRUNING

We first introduce yet another mode of pointer processing, which we denote REFINE-mode. Pointers encountered in REFINE-mode are processed only if they relate to a document that is a member of R, the set of documents that have the highest scores. This stage of processing is done once the set of answers has been determined, but before their final ordering is fixed. If we seek only to guarantee precision at r documents retrieved, we can skip any REFINE-mode processing; but if we seek to guarantee mean average precision, or some other metric that factors in document ranks, then the top r documents must be ordered correctly as well as selected correctly.

Figure 1 shows the relationship between REFINE-mode and the other modes, and the way that the size of the set A of interesting documents varies. The most influential pointers – those that should be processed in OR-mode – appear at the front of each inverted list. Partway through the processing of each inverted list, processing changes to AND-mode, only using each pointer if it augments the score of a candidate that has already been nominated. The transition is made once it is certain that no document that has not already been nominated can be a member of the final answer set.

At a second transition point, also discussed below, a set of r answers has been generated, and all that remains is to place them into their final rank order for presentation to the user. At this transition, all other non-zero accumulators can be dispensed with, and processing restricted to the set R of documents d for which $A_d \geq R_{min}$, where R_{min} is the score of the rth highest document. That is, $|R| \geq r$, and ties in scores are properly accounted for.

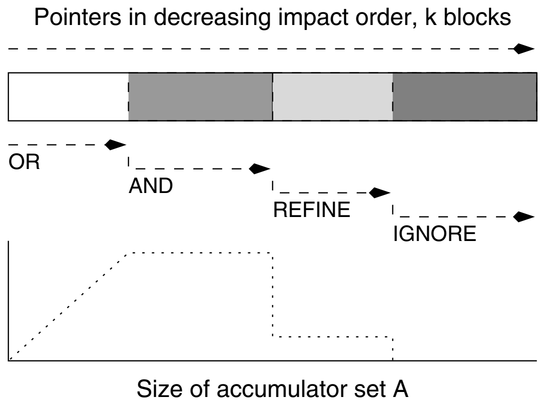

Pointers in decreasing impact order, k blocks
OR
AND
REFINE
IGNORE
Size of accumulator set A

Figure 1: Modes used when processing each impact-ordered inverted list. Each inverted list contains k blocks of equal-impact pointers. Each block is processed in one of four modes, with only the highest impact pointers handled in OR-mode. The graph underneath shows the growth, stabilization, and then reduction in the number of accumulators that are active as the query is processed.

Finally, at some third transition point, it can be known that the ordering of the top r documents is final, even if not their exact scores, and pointer processing is halted. The remainder of each of the inverted lists is ignored, and r of the documents currently in R (or all of them, if $|R| = r$) are presented to the user.

To facilitate computation of the three transition points, each accumulator A_d becomes a more complex structure. As well as recording the current score of the document d, it also tracks (as the quantity T_d) which terms contributed to that score. In addition, the quantity $next_t$ is maintained for each term t, to indicate the I value (starting at k^2 and counting down to 1) at which term t will supply its next block of pointers. If all blocks in the impact-sorted list for term t have been processed, then $next_t$ is set to zero. Each accumulator A_d can then also have associated with it an inferred quantity M_d, the maximum score that document d can aspire to based on the evidence encountered so far. This set of observations leads to the following processing logic:

```
[**Method A – Optimized exact computation**]
set $A \leftarrow \{\}$ and $T \leftarrow \{\}$ and $R \leftarrow \{\}$
set *mode* $\leftarrow$ OR-mode
initialize $next_t$ for each term $t$
**for** $I \leftarrow k^2$ **downto** $1$ **do**
    **for** each pointer $\langle t, d, \omega_{d,t} \rangle$ for which $I = \omega_{d,t} \times \omega_{q,t}$ **do**
        **if** *mode* $=$ OR-mode **or** $A_d > 0$ **then**
            set $A_d \leftarrow A_d + I$
            set $T_d \leftarrow T_d \cup \{t\}$
            if necessary, adjust $R_{min}$ and the membership of set $R$
        **end if**
    **end for**
    **for** each term $t$ for which $I = next_t$ **do**
        update $next_t$ to the next block in $t$'s inverted list
    **end for**
    **if** *condition1* **then**
        set *mode* $\leftarrow$ AND-mode
    **end if**
    **if** *condition2* **then**
        set *mode* $\leftarrow$ REFINE-mode
        **for** each document $d$ for which $A_d > 0$ and $d \notin R$ **do**
```

$\lvert q\rvert$	#	Exhaustive		Method A, $r = 20$				Method A, $r = 1{,}000$			
		OR ('000)	$\lvert A\rvert$ ('000)	OR (%)	AND (%)	REFINE	$\lvert A\rvert$ ('000)	OR (%)	AND (%)	REFINE	$\lvert A\rvert$ ('000)
1	11,410	277.3	277.3	38.7	0.0	0.0	107.2	39.0	0.0	0.0	108.0
2	15,200	1,859.9	1,732.9	3.3	19.5	37.1	56.1	6.9	36.6	41.7	122.0
3	10,258	5,866.8	4,856.8	1.6	47.2	38.0	88.6	3.8	57.5	29.8	219.2
4	6,248	12,003.4	7,459.1	1.2	53.1	21.2	136.8	2.9	61.5	12.0	331.9
5	3,361	17,960.8	9,217.3	1.0	56.8	10.7	171.0	2.4	63.5	2.8	407.4
6+	3,513	33,488.3	14,263.6	0.7	64.5	3.9	226.4	1.7	66.9	0.6	519.0
All	49,990	6,893.9	4,141.2	1.6	53.4	17.5	104.2	3.2	60.4	11.9	212.1

Table 2: Average number of pointers processed per query in OR-mode, AND-mode, and REFINE-mode, and number of non-zero accumulators required, for exhaustive "every-pointer" processing as described by Anh and Moffat [2005], and for our Method A, broken down by query length, for 49,990 queries, retrieving the top $r = 20$ documents and the top $r = 1{,}000$ documents for each query from the GOV2 collection. Pointer counts for the "Exhaustive" method, and all three sets of accumulator counts, are in thousands; pointer counts for "Method A" are expressed as percentages relative to the pointer count numbers for the "Exhaustive" method.

```
            set $A_d \leftarrow 0$
        **end for**
    **end if**
    **if** *condition3* **then**
        processing can be halted, since the top $r$ are final
    **end if**
**end for**
sort set $R$ and return it to the user
```

Now consider the three conditional expressions that govern the timing of the mode transitions. The first, *condition1*, must be false until the rth highest similarity score among the documents that do have accumulators is at least as large as any score M_d that can be attained by a document d which still has not yet been nominated by any of the query terms. But if document d has not yet been struck by any query terms, then M_d is simply the sum of the *next* quantities, and *condition1* can be expressed as:

$$R_{min} \geq \sum \{next_t \mid t \in q\} \ ,$$

where q is the set of terms that comprise the query. That is, no document that has not yet been scored can enter the top r once the sum of the *next* values is less than the score already assigned to the rth largest accumulator.

Once AND-mode has been commenced, accumulator values A_d continue to be non-decreasing, while at the same time, the M_d values are non-increasing. The transition to REFINE-mode can thus be triggered when all M_d values for documents that are in A but not in R are less than R_{min}. That is, *condition2* can be expressed as:

$$R_{min} \geq \max\{M_d \mid d \in A, d \notin R\} \, .$$

For a document d and an accumulator A_d, M_d is calculated as

$$M_d = A_d + \sum \{next_t \mid t \in q, t \notin T_d\} \ .$$

The final transition is governed by *condition3*, a constraint that is established if

$$T_d = q, \forall d \in R \, .$$

But situations in which not all of the top r answers contain all of the query terms can also arise. Suppose that R is ordered by decreasing accumulator score, so that $A_{R_1} \geq A_{R_2}$, and so on; then processing can be stopped when

$$A_{R_{i-1}} \geq M_{R_i}, \forall R_i \in R \, .$$

This latter expression is the *condition3* used in Method A.

$\lvert q\rvert$	#	$r = 20$		$r = 1{,}000$	
		AND	REFINE	AND	REFINE
1	11,410	0.00	0.00	0.00	0.00
2	15,200	0.26	<0.01	0.58	0.01
3	10,258	0.67	<0.01	1.60	0.01
4	6,248	1.05	<0.01	2.19	0.01
5	3,361	1.33	<0.01	2.68	0.01
6+	3,513	1.45	<0.01	3.09	0.01
All	49,990	1.19	<0.01	2.44	0.01

Table 3: Average number of accumulator updates during AND-mode and REFINE-mode processing in Method A, as a percentage of the total number of pointers processed while operating in each mode. Experimental details are as for Table 2.

Table 2 shows the effect of this method compared to the exhaustive evaluation strategy used by Anh and Moffat [2005]. A set of approximately 50,000 queries are broken into groups according to their length $|q|$. Two different retrieval depths are used, $r = 20$ and $r = 1{,}000$, and applied to the 25 million documents in the 426 GB TREC GOV2 test collection (the cost of exhaustive evaluation does not depend on r).

As can be seen from the table, the number of pointers needing to be processed in OR-mode is a very small fraction of the number of pointers indicated by the query terms, and averages just 1.6% when $r = 20$, and 3.2% when $r = 1{,}000$. This translates into a dramatic saving in the amount of space required for accumulators, and even when $r = 1{,}000$ answers are required, the number of accumulators averages around 200,000, or approximately 1% of the size of the collection. We reiterate that with this number of accumulators the ranking of the top r documents can be *guaranteed* to match the ranking that is delivered by an exhaustive computation.

The number of pointers processed in AND-mode is also bounded in Method A. Table 2 shows that for typical queries a further 50–60% of pointers need to be processed if precision is to be guaranteed, and then another 10–20% if the exact rank ordering (and thus mean average precision) must be protected. Conversely, approximately 25–30% of all pointers can be ignored without there being any effect on the ranking. In addition, if the effort required to process a pointer in AND-mode (and hence REFINE-mode) is less that the effort required to process an OR-mode pointer, then a non-trivial saving in execution cost arises.

Q	Topics 701–750		Topics 751–800	
	P@20	MAP	P@20	MAP
0	0.3622	0.1943	0.3973	0.2324
10	0.4888	0.2332	0.5396	0.2813
30	0.5122	0.2542	0.5648	0.3107
50	0.5173	0.2571	0.5635	0.3145
100	0.5163	0.2620	0.5626	0.3199

Table 4: Retrieval effectiveness obtained on two different sets of recent TREC topics using the `GOV2` collection and Method B processing. The two columns "P@20" note precision at $r = 20$ documents retrieved, while the two columns "MAP" list mean average precision at $r = 1{,}000$ documents retrieved. The variable Q corresponds to the target percentage of the available post-OR-mode pointers that are processed in AND-mode.

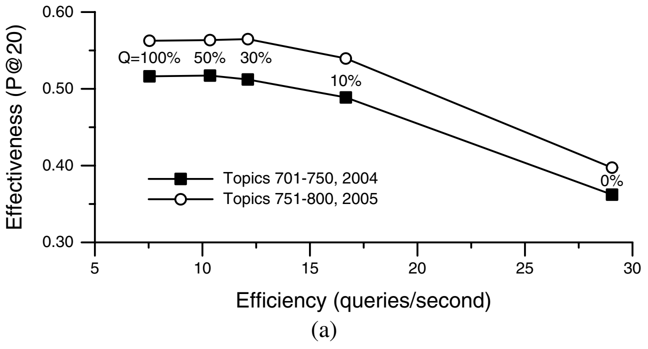
Table 3 shows further detail of the Method A computation. It lists, for the same set of queries, the fraction of AND-mode and REFINE-mode pointers that actually result in an increase to an accumulator value A_d. What is quite remarkable is that only a tiny fraction – around 2% or so – of the pointers processed in AND-mode actually strike candidates, and almost no pointers at all of the ones processed in REFINE-mode make tangible contributions. A clear conclusion is that the key component of processing during AND-mode and REFINE-mode is testing document numbers d for membership of (respectively) the sets A and R. The actual cost of changing A or R is not critical, since it happens so infrequently. Our implementation exploits this observation, and once the OR-mode pointers have been exhausted, it flattens the accumulator search structure (a hash table) into a dense sorted array so that document-ordered merge operations against document-sorted equal-impact inverted list blocks are efficient.

Even though accumulator updates are rare, we still wish to avoid the CPU cost associated with computing *condition2* and *condition3*, and the space associated with the sets T_d. (Computing *condition1* is easy.) Table 3 also suggests that while *condition2* and *condition3* give suitable guarantees, they may be unnecessarily pessimistic. To address these two concerns, we introduce a "fidelity control knob" Q, which controls the percentage of the post-OR-mode pointers that are to be processed for any given query. We call this heuristic approach *Method B*; space limits preclude the inclusion of full pseudo-code. In Method B, pointers are processed in the same manner as described in Method A in OR-mode until *condition1* is satisfied. Then the total number of remaining pointers is multiplied by Q, and processing resumed in AND-mode until that many more pointers have been processed. That is, when $Q = 0\%$, the final ranking is based on the OR-mode pointers alone; and when $Q = 100\%$, the all pointers are considered and the ranking is guaranteed. Method B has the same space requirement as Method A.

Table 4 gives effectiveness results for Method B for two different sets of TREC Terabyte Track topics, those used in 2004 and 2005, and for two different retrieval effectiveness metrics. When $Q = 0$ retrieval effectiveness is relatively poor. But by the time $Q = 30\%$, both effectiveness metrics indicate good retrieval performance, on both sets of queries, and there seems little value in processing further pointers. Note that even when the scores of the top r documents are guaranteed by using a large value of Q, the need for tie-breaking in the similarity scoring regime, and in the effectiveness evaluation metric that then gets fed those scores, can lead to small variations in effectiveness.

Q	Exhaustive		Pruned	
	MB	queries/sec	MB	queries/sec
0	—	—	1.5	29.0
30	—	—	1.5	12.1
100	95.0	4.2	1.5	7.5

Table 5: Average memory footprint per query, in megabytes, and query throughput rates, in queries per second, using the TREC Terabyte Efficiency Track protocols. For each of 49,990 queries, the TREC document identifiers for the top-ranked 20 documents are output, with all processing on each query completed before the next query is initiated. The hardware used is a 2.8 GHz Intel Pentium IV with 1 GB of RAM and 250 GB local SATA disk.



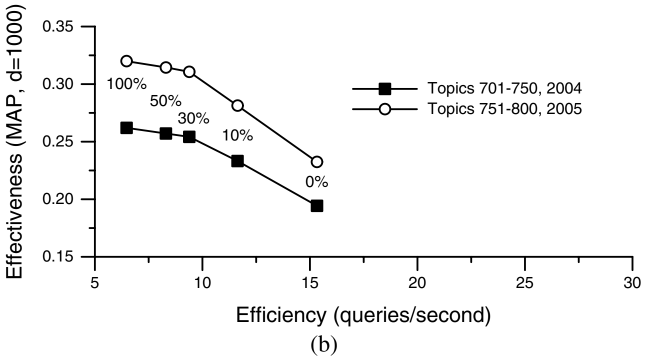


Figure 2: Tradeoff as Q is varied in Method B, using collection `GOV2`: (a) with effectiveness measured using precision at 20, and a retrieval depth of $r = 20$; and (b) with effectiveness measured using mean average precision at a retrieval depth of $r = 1{,}000$.

6. QUERY THROUGHPUT

Table 5 shows query throughput rates and memory requirements for Method B using three different values of Q, on a stream of nearly 50,000 queries. The reference point for this table is the Exhaustive approach; because it processes all pointers in OR-mode, the only sensible accumulator data structure is an array, and the memory space required is high. Method B has much more compact memory requirements, and in part because of this, executes more rapidly, even when $Q = 100\%$. The other reason for the increase in throughput is that the revised implementation processes AND-mode pointers faster than OR-mode pointers, and in the column "Exhaustive" all pointers are regarded as being OR-mode.

Figure 2 combines Tables 4 and 5, to show how Q serves as a tradeoff knob controlling execution time. Note that the throughput rates are derived from the 49,990 Efficiency Track queries (10 of the original 50,000 TREC queries have no matching documents in our system); whereas the effectiveness results are from the TREC topics 701–750, and 751–800, respectively, so in this sense the graph does not represent a single set of data. It does, however, show in a convincing manner that use of $Q = 30\%$, and only processing around a third of the post-OR-mode pointers (and thus completely ignoring around two thirds of the pointers for any given query) gives excellent retrieval effectiveness, at close to triple the query throughput rates of the exhaustive computation.

7. COMPARATIVE EVALUATION

Section 6 showed that the new impact-sorted pruning regime greatly speeds query processing. This section draws on two sets of comparable results to further illustrate the gains achieved.

The first reference point we use was established by Strohman et al. [2005], who implemented and tested a document-at-a-time pruning mechanism based on the `max_score` mechanism of Turtle and Flood [1995]. Strohman et al. did not have access to the 50,000 TREC queries at the time they carried out their experiments. Nevertheless, their results represent a useful yardstick. Compared to the requirements of the 2005 TREC Efficiency Track, Strohman et al. have retrieved $r = 10$ rather than $r = 20$ documents; have worked with 50 rather than 50,000 queries; and have carried out a "warm-up" run that was not part of the timing. On the other hand, their index is a word-level one that includes word positional information. These pointers are unused during ranked querying, and if stored with the document pointers in a fully interleaved manner, can markedly slow ranked query evaluation [Anh and Moffat, 2006]. Strohman et al. have also designed their implementation to allow future support of complex queries involving positional and/or structural constraints, and the extent to which this flexibility affects operational speed on simple ranked queries is unknown.

Nevertheless, the fundamental comparison remains: on a 2.6 GHz machine, and with pruning enabled, Strohman et al. require 1.73 seconds to evaluate each query. Even taking into account the difference in processor clock speeds, this is more than 10 times slower than the speeds attained in this paper.

As a second reference point, we draw on the set of data submitted by the participants of the 2005 TREC Terabyte Efficiency Track. Figure 3 shows the tradeoff between effectiveness and economy (measured in units of "queries per second per thousand American dollars"), using data from Table 4 of Clarke and Scholer [2005]. Hardware costs are listed in that table in terms of 2005-value US dollars; the single processor system used in our experiments was purchased mid-2004 and has a 2005-equivalent hardware cost of approximately $US750. In each case the top $r = 20$ documents for each of 50,000 web queries are identified, and their TREC document identifiers retrieved. Figure 3 shows in no uncertain terms the magnitude of the gain in performance that has been achieved by our pruning approach, and provided that $Q \geq 30\%$, that gain is not in any way at the expense of retrieval effectiveness.

To allay possible concerns about the validity of the comparison made in Figure 3, we note that the same hardware was also used to build the index (with the compressed source data and all temporary and permanent index files stored on the local 250 GB disk), in a construction time of 6.5 hours.

8. DOCUMENT-ORDERED EVALUATION

The evaluation of the previous two sections has presumed that only pure ranked queries are being handled. But there are also situations in which Boolean filters – perhaps expressed as term proximity or document structure relationships – must be checked prior to any similarity evaluation, and only documents that pass the filter

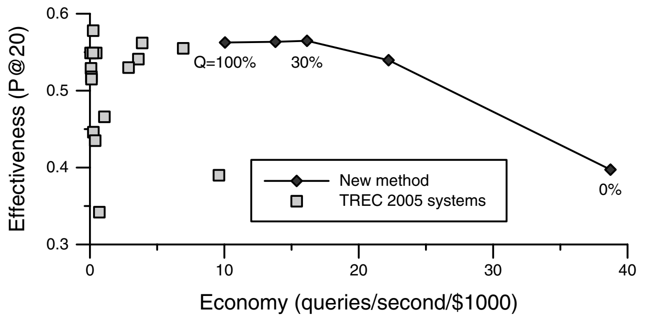


Figure 3: Performance of retrieval systems, comparing effectiveness (P@20) and economy, the latter defined in units of queries per second per thousand dollars of hardware. In all cases a retrieval depth of $r = 20$ is assumed. Data for the 2005 TREC systems is from Table 4 of Clarke and Scholer [2005], and covers two submitted runs from eight retrieval systems. The runs for the new system use the same methodology as in Table 5 and Figure 2(a).

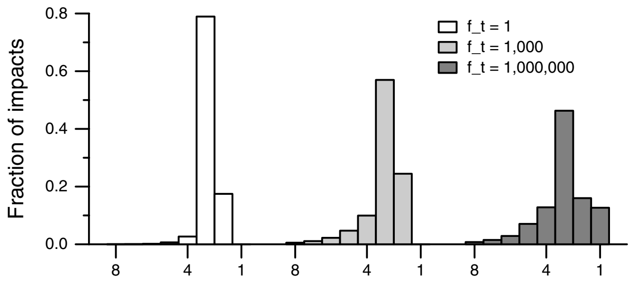


Figure 4: Distribution of impact values within inverted lists. The three bar graphs represent the average impact distribution over terms in the GOV2 test collection with $f_t = 1$ (10,725,138 terms); $1{,}000 \leq f_t < 2{,}000$ (22,390 terms); and $1{,}000{,}000 \leq f_t < 2{,}000{,}000$ (641 terms).

should be considered for ranking. When complex queries of this sort are being handled, document-at-a-time processing is the most practical approach [Strohman et al., 2005].

Processing in document-at-a-time order does not, however, mean that the inverted lists in the index must be document-sorted. All that needs to be done is that the document-based merge operation take place at the impact block level rather than the term level. For example, if the query has $|q|$ terms, then the merge operates on $k|q|$ input cursors, rather than $|q|$. In addition, the difference is not as great as it might seem, because of the highly variable lengths of the impact blocks, and of the fact that to a certain extent they are mutually exclusive – factors that mean that a heap-based merge is not the best document-at-a-time strategy for impact-sorted indexes.

As evidence of these features of impact blocks, Figure 4 shows the distributions of impact values within inverted lists, averaged over three different groups of f_t values, for the collection GOV2. Rare terms (those with $f_t = 1$) tend to only occur sparingly in documents in which they do appear, and are assigned impact scores of 3 or 2. Common terms, shown in the rightmost bargraph of the figure, are more likely to be assigned high impacts. (Note that this counter-intuitive bias is reversed when the query impacts are introduced, since they include an IDF factor.) But even so, within any given inverted list, the majority of the pointers are in the $\omega_{d,t} = 3$ block, and the impact blocks for other values are much shorter.

Index organization and processing mode	Throughput (q/sec)
Document-sorted, document-at-a-time	2.4
Impact-sorted, document-at-a-time	1.7
Impact-sorted, document-at-a-time (improved)	3.2

Table 6: Ranked query throughput rates for document-at-a-time processing using an impact-sorted index, assuming exhaustive processing and no pruning, using 50,000 TREC Efficiency Track queries, and identifying $r = 20$ answers for each query.

Table 6 gives query throughput rates for exhaustive document-at-a-time processing. The first row shows the speed obtained using a document-sorted index. Relative to the speed shown in Table 5 for "Exhaustive" processing, use of a document-sorted index involves more decoding steps out of the index, and a more complex processing logic.

The second row of Table 6 shows the throughput of the "merging all the impact blocks" approach; and the final row shows throughput for an improved mechanism that is sensitive to the lengths of the impact blocks, and uses a length-biased merge rather than an egalitarian heap. These processing times may also be compared with those shown in the last row of Table 5; pure impact-sorted processing is faster, but document-at-a-time processing using an impact-sorted index is also perfectly sensible, and is faster than using a document-sorted index for these typical queries. That is, there is no reason to prefer a document-sorted index, even if a filtering step is required as a precursor to ranked querying. The impact-sorted index also provides flexible support for the "top docs" approach to the `max_score` pruning regime for complex queries that is proposed by Strohman et al. [2005].

Our final comment in this regard concerns index size – the impact-sorted index is slightly more compact than the document-sorted one, and (including all associated vocabulary information) occupies 6.12 GB for the `GOV2` collection, compared to 6.77 GB for a document-ordered index.

9. CONCLUSION

We have described a new method for dynamic query pruning using an impact-sorted index, including compelling experimental evidence of the versatility and speed of the new system. The same index arrangement can also be used for fast document-at-a-time querying in support of complex queries.

Acknowledgment. This work was supported by the Australian Research Council, by the ARC Center for Perceptive and Intelligent Machines in Complex Environments, and by the NICTA Victoria Laboratory. We also thank the referees for their careful comments.

References

V. N. Anh, O. de Kretser, and A. Moffat. Vector-space ranking with effective early termination. In W. B. Croft, D. J. Harper, D. H. Kraft, and J. Zobel, editors, *Proc. 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 35–42, New Orleans, Louisiana, September 2001. ACM Press, New York.

V. N. Anh and A. Moffat. Simplified similarity scoring using term ranks. In G. Marchionini, A. Moffat, J. Tait, R. Baeza-Yates, and N. Ziviani, editors, *Proc. 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 226–233, Salvador, Brazil, August 2005. ACM Press, New York.

V. N. Anh and A. Moffat. Structured index organizations for high-throughput text querying. April 2006. Submitted for publication.

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 CIKM Int. Conf. Information and Knowledge Management*, pages 426–434, New Orleans, Louisiana, November 2005. ACM Press, New York.

E. W. Brown. Fast evaluation of structured queries for information retrieval. In E. A. Fox, P. Ingwersen, and R. Fidel, editors, *Proc. 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 30–38. ACM Press, New York, July 1995.

C. Buckley and A. F. Lewit. Optimization of inverted vector searches. In *Proc. 8th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 97–110, Montreal, Canada, June 1985. ACM Press, New York.

C. L. A. Clarke and F. Scholer. The TREC 2005 Terabyte Track. In *The Fourthteenth Text REtrieval Conference (TREC 2005) Notebook*, Gaithersburg, MD, November 2005. National Institute of Standards and Technology. `http://trec.nist.gov/act_part/t14_notebook/t14.notebook.html`.

E. S. de Moura, C. F. dos Santos, D. R. Fernandes, A. S. Silva, P. Calado, and M. A. Nascimento. Improving web serach efficiency via a locality based static pruning method. In *Proc. 14th International World Wide Web Conference*, pages 235–244, Chiba, Japan, May 2005.

D. K. Harman and G. Candela. Retrieving records from a gigabyte of text on a minicomputer using statistical ranking. *Journal of the American Society for Information Science*, 41(8):581–589, August 1990.

D. Hawking. Efficiency/effectiveness trade-offs in query processing. *ACM SIGIR Forum*, 32(2):16–22, September 1998.

N. Lester, A. Moffat, W. Webber, and J. Zobel. Space-limited ranked query evaluation using adaptive pruning. In A. H. H. Ngu, M. Kitsuregawa, E. J. Neuhold, J.-Y. Chung, and Q. Z. Sheng, editors, *Proc. 6th International Conference on Web Information Systems Engineering*, pages 470–477, New York, November 2005. LNCS 3806, Springer.

A. Moffat and J. Zobel. Self-indexing inverted files for fast text retrieval. *ACM Transactions on Information Systems*, 14(4):349–379, October 1996.

M. Persin, J. Zobel, and R. Sacks-Davis. Filtered document retrieval with frequency-sorted indexes. *Journal of the American Society for Information Science*, 47(10):749–764, October 1996.

A. Soffer, D. Carmel, D. Cohen, R. Fagin, E. Farchi, M. Herscovici, and Y. S. Maarek. Static index pruning for information retrieval systems. In W. B. Croft, D. J. Harper, D. H. Kraft, and J. Zobel, editors, *Proc. 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 43–50, New Orleans, Louisiana, September 2001. ACM Press, New York.

T. Strohman, H. Turtle, and W. B. Croft. Optimization strategies for complex queries. In G. Marchionini, A. Moffat, J. Tait, R. Baeza-Yates, and N. Ziviani, editors, *Proc. 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 219–225, Salvador, Brazil, August 2005. ACM Press, New York.

M. Theobold, R. Schenkel, and G. Weikum. Efficient and self-tuning incremental query expansion for top-k query processing. In G. Marchionini, A. Moffat, J. Tait, R. Baeza-Yates, and N. Ziviani, editors, *Proc. 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 242–249, Salvador, Brazil, August 2005. ACM Press, New York.

H. Turtle and J. Flood. Query evaluation: strategies and optimizations. *Information Processing & Management*, 31(1):831–850, November 1995.

E. M. Voorhees and D. K. Harman. *TREC: Experiment and Evaluation in Information Retrieval*. MIT Press, 2005. ISBN 0262220733.

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