# Reordering an index to speed query processing without loss of effectiveness.

David Hawking
Funnelback Pty Ltd.
Canberra ACT 2602, Australia, and
RsCS, Australian National University
david.hawking@acm.org

Timothy Jones
Funnelback Pty Ltd.
Melbourne VIC 3066, Australia
tjones@funnelback.com

## ABSTRACT

Following Long and Suel, we empirically investigate the importance of document order in search engines which rank documents using a combination of dynamic (query-dependent) and static (query-independent) scores, and use document-at-a-time (DAAT) processing. When inverted file postings are in collection order, assigning document numbers in order of descending static score supports lossless early termination while maintaining good compression.

Since static scores may not be available until all documents have been gathered and indexed, we build a tool for reordering an existing index and show that it operates in less than 20% of the original indexing time. We note that this additional cost is easily recouped by savings at query processing time. We compare best early-termination points for several different index orders on three enterprise search collections (a whole-of-government index with two very different query sets, and a collection from a UK university). We also present results for the same orders for `ClueWeb09-CatB` . Our evaluation focuses on finding results likely to be clicked on by users of Web or website search engines — *Nav* and *Key* results in the TREC 2011 Web Track judging scheme.

The orderings tested are Original, Reverse, Random, and QIE (descending order of static score). For three enterprise search test sets we find that QIE order can achieve close-to-maximal search effectiveness with much lower computational cost than for other orderings. Additionally, reordering has negligible impact on compressed index size for indexes that contain position information. Our results for an artificial query set against the TREC ClueWeb09 Category B collection are much more equivocal and we canvass possible explanations for future investigation.

## Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Information Storage and Retrieval—*Information Search and Retrieval*; H.3.4 [**Information Systems**]: Information Storage and Retrieval—*Systems and Software*

## Keywords

Enterprise search; inverted files; efficiency and effectiveness; information retrieval.


*Australasian Document Computing Symposium* '12 Dunedin, NZ


## 1. INTRODUCTION

Commercial search engines are subject to strong incentives to deliver fast query response, while using as little hardware as possible. Slow response degrades user experience [3] and negates the value of "instant search"[1], while inefficient retrieval algorithms increase infrastructure costs and lower profitability. If the time taken to run a query over an index of a certain size can be reduced by a factor of $f$ while keeping the hardware constant, then the number of machines needed to support a large query load is also reduced by a factor of $f$. Not only is the cost of hardware reduced but so too is the cost of data centre hosting, system administration, and electricity. Even better, for most countries, greenhouse gas emissions fall in line with reduced electricity demand. For major world wide web search engines, a factor of two in query processing efficiency could translate to billions of dollars per annum and, depending upon energy mix, to savings of the order of a million tonnes of $CO_2$ emissions.

To illustrate the importance of this point, a European Energy Agency graph[2] shows that, worldwide, approximately 500gm $CO_2$ is emitted per kWh of electricity *generated*. A typical rack-mounted server consumes a constant 250W and in the course of a year consumes $365 \times \frac{24}{4} = 2190$ kWh. Transmission losses and data centre overheads such as airconditioning push that up to a generating requirement of around 3000 kWh p.a, corresponding to about 1.5 tonnes of $CO_2$ per server. Guessing a deployment of a million servers for a hyothetical major search engine, leads to an emission estimate above a million tonnes. (In Australia, where electricity generation is heavily reliant on coal, $CO_2$ emissions in 2009 were about 80% higher at 928gm per kWh generated.[3]) These arguments also apply in smaller scale commercial search engine deployments, such as intranets in large organisations, high-volume e-commerce sites, whole-of-government search, multi-tenancy hosted website search (Sofware-as-a-Service, or SaaS), and search "in a cloud". The largest of such deployments may index of the order of a billion documents and/or process thousands of queries per second. An increase in query efficiency allowing for smaller numbers of servers clearly has a large environmental impact.

We investigate the hypothesis that smarter ordering of documents in an index can significantly improve the efficiency of query processing without reducing quality of results. We present a tool to efficiently re-order an existing index according to a permutation file and report its cost relative to indexing time. We then take two example enterprise web indexes for which we have appropriate

[1]When a search engine takes a partially submitted query, guesses a likely completion and runs it, presenting one or more result sets before the user has finished typing.
[2]`http://www.eea.europa.eu/data-and-maps/figures/co2-emissions-per-kwh-of`, accessed 26 Oct 2012.
[3]ABB *Australia: Energy efficiency report*, updated Jan 2011.

query test sets and plot query response and effectiveness against different DAAT termination points for Original, Reverse, Random, and QIE (Query Independent Evidence, i.e. descending order of static score) permutations. We conduct a similar comparison for `ClueWeb09-CatB` using an artificial query set.

# 2. BACKGROUND AND CONTEXT

## 2.1 Search engine architecture

A simple model which is adopted in essence by many search companies is to split a large collection of documents into $m$ partitions comprising roughly equal numbers of documents. This may be done by hashing on URLs. Each partition is indexed separately and the index is replicated across the $n$ query processing servers assigned to that partition. Thus we have an $m \times n$ matrix of servers. The incoming query stream is load-balanced across the $n$ rows of that matrix. Each row supports a broker function which multicasts an incoming query to all the query processors in the row and merges the results.

The optimal partition size depends upon the hardware configuration of the query processing servers, the efficiency of the ranking algorithm, the size of documents, and the design goals for response time. With 2012-era hardware, and an average response time goal of say $r_{ave} = 100$msec, a reasonable partition size might be up to around 100 million web documents. That number could be increased significantly for shorter documents such as microblog posts or database records.

The above architecture is oversimplified and takes no account of real-life complications such as query caching, real time indexing, load balancing, efficient result merging and fault tolerance. However, it is sufficiently accurate to place the present work in context.

Here we focus on efficient ranking on a single query processing server – one node in the matrix – with an index of up to 100 million web pages. If the response time goals are achieved and the server has $c$ processor cores, the whole matrix will be able to handle $\frac{c \times n}{r_{ave}}$ and each individual server will be able to handle $c/r_{ave}$ queries per second. For a 12-core server that would give a total throughput of 120 queries per second, assuming that query processing is CPU-bound and that the full benefit of parallelism can be achieved across the cores.

## 2.2 Queries and information needs

Although law firms, patent departments, investment advisers, medical reviewers and intelligence agencies have genuine needs for complex, high-recall search, here we focus on the set of queries which comprise a small number of words and no operators, and which result from an information need which can be satisfied by a small number of documents. The vast majority of queries submitted to the commercial search engines we have been discussing belong to this set.

For example, a student at a university searches for 'exam timetable' and clicks only on the link which takes them to the relevant timetable site; A computer scientist searches on the Web for 'RFC 3261' and clicks only on the Session Initiation Protocol document from IETF; A citizen searches for 'tax', and selects the home page for the national taxation office. In other words, although a typical query may find many matching documents, the searcher is most often satisfied by a very small number of them.

Queries without operators are often described as "bag of words" queries. However, modern commercial ranking functions assign higher scores to documents which match the sequence of words in the query or subsequences of it. In the following example queries from the TREC-2009 Web Track set, it seems intuitive that documents matching the query as a phrase are more likely to be useful than those containing unassociated occurrences of the query words: "mitchell college", "rick warren", "orange county convention center", "pampered chef". Accordingly, it may be better to think of "sequence of words" queries.

Since 2010, the TREC Web Track [6] has adopted a six-point relevance judging scale: {Nav, Key, HRel, Rel, Non, Junk}. *Nav* and *Key* correspond to the desired answers to homepage finding and topic distillation tasks respectively. These tasks were investigated in earlier Web Track campaigns.

Documents receiving clicks in the categories of search represented by the examples above are likely to be Nav or Key documents. Consequently, we focus our evaluations on common Web-style queries and these Nav or Key answers.

## 2.3 Ranking and query matching methods

We assume that documents are ranked by a combination of query-dependent (dynamic) and query-independent (static) scores, as is believed to be the case in most commercial search engines. Query independent scores may include link-graph measures [15, 4], access frequency [12, 13], document quality scores [16, 1], non-spam score [8], recency, and so on. The overall ranking function is typically machine-learned [23] and may combine more than 1000 features.

Sequence-of-words queries which contain more than one word $(W_1, W_2 \ldots W_n)$ may be processed either term-at-a-time (TAAT) or document-at-a-time (DAAT) [24]. In unoptimised TAAT, all documents containing $W_1$ receive a partial score, then those containing $W_2$, have their scores augmented and so on up to $W_n$. Finally, all documents with non-zero scores are sorted in descending score order. Until the last term is being processed, it cannot be known which documents satisfy the AND of the query terms. An obvious optimisation is to store postings in impact order and truncate low-impact postings, but this is potentially lossy, as truncated postings for one term may have combined with high-impact scores for another, and AND matches or even phrase matches, may be missed. Long and Suel [14] claim that the TAAT model is infeasible at large scale.

In DAAT, postings lists for terms are scanned in "parallel" and are assumed to be in document number order. It is easy to determine whether candidate documents match the AND of the query terms and whether they match the query as a phrase[4].

The DAAT model avoids the need for a large accumulator set, supports more efficient AND or Weak-AND (WAND) processing [2], allows for easier upweighting of phrases and terms in proximity, delivers higher quality results in the event of a forced timeout, and, as we shall see, supports principled early-termination. Svore et al [22] claim retrieval effectiveness gains of up to 13% through use of proximity features in Web search.

Here we restrict our discussion to the DAAT model.

## 2.4 Index formats

According to Ding and Suel [11], methods for early termination of query processing use one of three inverted file structures:

- **Document-sorted Indexes:** in which the postings in each list are sorted by document ID. This is the usual representation for DAAT-based early termination techniques.

- **Impact-sorted Indexes:** in which the documents in each postings list are sorted by their impact on ranking under e.g. Cosine or BM25. This is the usual approach for TAAT-based early termination techniques

[4] Assuming that the index includes term positions

- **Impact-layered Indexes:** in which the postings are divided in to layers according to impact, but those layers are sorted by document ID. This allows some of the benefit of impact sorting in DAAT techniques, although it comes at a compression cost due to larger gaps between document IDs.

Ding and Suel note that document-sorted indexes are less studied for early termination strategies [11]. In this paper we assume a document-sorted index, since it fits the DAAT model which is feasible at web scale.

Compression increases the size of index which can be accommodated in a given memory size, or read in a single read from disk. Much work in query processing assumes that indexes cannot be fully resident. More recently, this assumption has been revisited [9, 21], and according to [10] Google's web search indexes have been memory-resident since 2003.

In this paper we make no assumption about whether indexes are fully resident or not. The index ordering method investigated here will bring benefit in the fully resident case and also increase the locality of reference in the partially resident case. In the experiments reported here the small indexes are fully resident and the `ClueWeb09-CatB` index is not.

## 2.5 Optimising DAAT processing when static scores are used

Long and Suel [14] were the first to illustrate that early termination of DAAT processing is possible when postings are arranged in order of decreasing static score. In their work, PageRank [15] was used for the static score, and a modified cosine measure for the dynamic component. Several different pruning strategies were evaluated in terms of number of disk blocks accessed during query processing, query throughput per second, and error rate – defined as the percentage deviation from a full ranking for some $k$ documents. The strategies that best balanced a high throughput with a low error rate used two postings lists per term – the first list contains the $h$ postings that contain the highest term value according to the cosine measure, and the second list contains the remaining postings. Single-list based strategies had high throughput, but were also associated with a higher error rate.

To illustrate lossless early termination, let us assume that the final score of document $d$ is $F_d = \alpha \times D_d + (1-\alpha) \times S_d$, where all scores are normalised to $0 \ldots 1$. Thus, for example, $\alpha = 0.3$, scores are determined 70% by static factors and 30% by the extent to which the document matches the query. If, during DAAT processing, we are looking at a document $d$ whose static score is $S_d$ and the $k$-th best document found so far has a final score of $F_k$, then processing can stop if $F_k > \alpha + (1-\alpha) \times S_d$. I.e. even if a document not yet encountered achieves a dynamic score of 1.0, it will not make it into the final top-$k$.

Additional optimisation can be performed based on the dynamic score component. In the Weak-AND technique proposed by [2], an upper bound $UB_t$ for contribution to the final ranking score is recorded alongside each postings list. In the simple case, $UB_t$ is equal to the maximum score contribution from any document in the postings list for term $t$. Due to the difficulty of calculating this upper bound for phrase terms or other complex query-time combinations of postings lists, $UB_t$ can be estimated. Document scores are fully evaluated each time a match is found, and a heap of the $h$ highest scoring documents is maintained. Once the heap is full, the score of the lowest scoring document in the heap can be used in combination with $UB_t$ to faciliate early advancement or termination in postings lists where that term's score contribution cannot push the document over the score required to join the heap. A further optimisation is to record the upper bound for blocks of postings [11], which allows skipping more documents without loss of quality.

In practice, processing can terminate quite a bit earlier than the lossless point with very low deterioration of scores on evaluation measures. Our experiments explore the effect on search quality of varying the cutoff point in a basic DAAT approach, similar to the naive approach in [14]. We use this naive approach as it does not depend on the composition of the dynamic ranking function, which allows for the query time customisation and flexibility desirable in enterprise search.

In this context, lossless early termination can apply only when the static part of the ranking function used at query processing time corresponds to that used when the index order was determined. In enterprise search this is often not the case, because search profiles associated with different classes of users may assign different weightings to the static variables.

In the equation $F_d = \alpha \times D_d + (1-\alpha) \times S_d$, low values of alpha are common, but this doesn't mean that matching the query is unimportant. In the model we assume, candidate documents must generally satisfy the AND of the query terms[5]

We note that in some search engines alpha is a function of the length of the query – the longer the query the higher the value of alpha.

## 2.6 Reordering document IDs

Other studies have investigated the effect of reordering document IDs on compression. [20] investigated several different orders, and found that sorting documents by URL is cheap and results in effective compression. This is because documents with similar URLs are often topically similar, since they are located in similar directories on the same server.

Further compression is possible when context information for term frequencies or position is carried across between documents. [25] introduced a scheme using this context to achieve additional compression when indexes were sorted by URL (although they note their improvement is also likely to apply to other orderings).

## 2.7 Aims

The aims of the present study are:

- to confirm the findings of Long and Suel on enterprise scale web collections, using enterprise search test sets and a combination of static features found to be useful in enterprise search,
- to confirm the findings of Long and Suel for fully resident and partially resident indexes,
- to quantify the in-practice time costs of reordering all components of an existing index.
- to quantify the impact on compression of reordering an index by QIE instead of crawl order.

# 3. METHOD

## 3.1 Retrieval system

We use an experimental variant of a commercial retrieval system whose inverted file indexes include word position information. Postings lists are stored as variable-byte [18] encoded differences. Skip blocks are not included in the inverted files used in the experiments reported here. The index comprises many files in addition to

[5]In practice, stopwords may be removed, very long queries may be truncated and the strict AND requirement may sometimes be weakened, e.g. when there are few full-AND matches.

**Table 1: Datasets used in the experiments. Index size includes only the index files needed in the present experiments. Other files used in query suggestion, summary generation etc are not included. Note that the `ClueWeb09-CatB` dataset was indexed with options designed to limit the size of the index. For example the `gov-Whole` index includes a great deal more metadata. Average query length is in words.**

| Test | No. doc.s | index size (GB) | no. of queries | ave query length |
|---|---|---|---|---|
| `University` | 386325 | 1.2 | 134 | 1.37 |
| `gov-Popular` | 2294156 | 9.1 | 91 | 1.25 |
| `gov-Agencies` | 2294156 | 9.1 | 100 | 3.84 |
| `ClueWeb09-CatB` | 50217545 | 89.6 | 100 | 3.10 |

the basic inverted file: sorted term dictionary, files to support snippet generation, a document table recording document properties such as length, spam features, recency, URL length, inlink scores, query-independent content feature scores, web host feature scores and so on.

The results of experiments reported here are dependent on the value of $\alpha$. If it were close to one, there would be little or no value in reordering the index. If it were close to zero the value of reordering would be increased. However, arbitrary settings of $\alpha$ make no sense since they would lead to poor result quality. We used a configuration of the ranking function in which $\alpha = 0.39$ which has been shown to produce good results across eleven query test sets involving eight different enterprise document collections.

Precise details of the ranking function are not important, since it is constant across all experiments. Dynamic scores are computed in BM25-like [17] fashion, taking into account document fields and referring anchortext with additional weight for implicit-phrase and proximity features. The static score is a fixed weighting of features derived from the link graph, document URL, document recency, document quality classifiers, the host domain and the host graph.

## 3.2 DAAT cutoff

It is possible to specify a stopping condition in terms of the number of full AND matches found for the query. I.e. stop after $z \geq k$ full matches have been found. We explored how response time and result quality varied with changes in $z$ for different index orders.

In all our experiments the number of results displayed was $k = 10$.

## 3.3 Infrastructure

All experiments were run on a laptop computer with a quad-core CPU (Intel Core i7-2720QM) with a clock-speed of 2.20GHz. It was equipped with 16GB of RAM and a single 7200 RPM SATA disk drive. No advantage was taken of the four CPU cores as index reordering and query processing were run single-threaded. Larger RAM would be recommended for a production deployment but this configuration gave us the potential to explore the value of index reordering on a non-resident index.

## 3.4 Timing

Timing results reported below represent the elapsed (wall-clock) time taken to run the query test set in a single execution of the query processor. Times are elapsed times and include query preprocessing, query matching, and ranking only. Snippet generation, spelling suggestion, faceting and related query suggestion functions are not included.

## 3.5 Datasets

The datasets used are summarized in Table 1. The query / answer sets for `University` and `gov-Popular` were generated by webmasters for those organisations. Queries correspond to popular and/or business critical queries.

We had intended to use the `ClueWeb09-CatB` queries and answers from the 2011 TREC webtrack [7]. Unfortunately, once they were filtered down to exclude answers outside the Category B set, there were only ten queries with Nav or Key answers. This was considered too small to support meaningful experiments.

Instead we accessed lists of universities in the US, Canada, New Zealand, Australia, United Kingdom and Ireland. For each university whose homepage and Wikipedia entry were in the `ClueWeb09-CatB` collection, we used the name of the university as the query and recorded the homepage as a Nav answer (grade 4) and the Wikipedia page as a Key answer (grade 3). In some cases we added multiple Wikipedia URLs or multiple homepage URLs but treated them as equivalents – no extra credit for retrieving more than one of them. Many of the university homepages and some of the Wikipedia entries are not present in the category B collection and those universities were rejected. However with the addition of a handful of additional universities from China, Singapore and Denmark we reached our target of 100 queries. As shown in the table, the queries are quite long (average: 3.10 words).

Example queries (showing only one of multiple equivalent answers):

| |
|---|
| University of Otago<br>`www.otago.ac.nz/`<br>`en.wikipedia.org/wiki/University_of_Otago` |
| Stony Brook University - State University of New York<br>`www.stonybrook.edu/`<br>`en.wikipedia.org/wiki/SUNY_Stony_Brook` |
| University of Cambridge<br>`www.cam.ac.uk`<br>`en.wikipedia.org/wiki/University_of_Cambridge` |

We would be very happy to make this test set available to other researchers on request.

The `gov-Agencies` query set is another artificially constructed set in which the queries comprise the names of agencies within the government and the answers (Nav only) are the homepages of those agencies. Query lengths in this set are even longer on average (3.8 words).

# 4. EFFECTIVENESS MEASURE

For consistency with the TREC Web Track, and because the arguments in [5] are quite persuasive, we used the Expected Reciprocal Rank (ERR) measure. Our relevance grades were converted to be consistent with a grade of 4 for a Nav answer and 3 for a Key answer, and we used the same function as the TREC Web Track for maping assessor grades to gain values. Our measurement software explicitly avoids giving extra credit when multiple equivalent URLs are returned. For example, a university homepage may have multiple URLs which lead to the same content. E.g. `www.uni.edu/` and `uni.edu/`.

# 5. INDEX REORDERING

We built a tool which takes each of the many files comprising an index and makes a reordered copy of them. The new order is defined by a text file containing a permutation of the document numbers $1, \ldots, |C|$, where $|C|$ is the number of documents in the collection. The reorder command takes three arguments: old index, new index, and permutation file.

Small files which are subject to permutation are read entirely into memory. Then each record is accessed in the new order and written

sequentially to the copy. Naturally, memory space is freed after each file is permuted.

Some large files (such as the data from which snippets are generated) may be processed in windows to avoid random disk I/O. In this case, the output file is divided into windows whose size corresponds to the available physical memory. The input file is scanned sequentially once for each window. During a scan, records in the current window are read into memory and reordered. At the end of the scan, those records are written sequentially to the output file.

The order of the term dictionary does not change, but pointers from dictionary entries to the inverted file may do so. As each dictionary entry is processed, its postings list is decompressed, then permuted, then compressed and written to the output file. Finally, the pointer from the dictionary entry to the inverted file is updated.

Simple tools were used to create permutation files from the original index. The three permutations we investigated were: Reverse, Random, and QIE (Query Independent Evidence, i.e. descending order of static score).

# 6. RESULTS

We first of all report the times taken to reorder the indexes and then report the results of query processing experiments in which the DAAT cutoff value is varied.

## 6.1 Time to reorder collections

Table 2 shows that the time taken to reorder an index is a small fraction of the original indexing time. Note that times reported include the reordering of all non-temporary index files, not only the ones needed to support the stripped-down results presentation used in our query processing experiments.

## 6.2 Index size

Unlike [20, 25], we did not observe a change in index size under different document orders. Observed sizes differed by at most one or two percent. This is likely to be due to the inclusion of term position information, which does not change between orderings, and makes up the bulk of the index. Possibly using context sensitive compression for posittion information [25], or alternative index structures for matching phrases [19] would yield changes in index size after reordering.

## 6.3 Varying the DAAT cutoff parameter

Figures 1, 2, 3 and 4 show the effect of varying the DAAT cutoff value on search effectiveness (as measured by ERR on the test sets) and on computational effort. Elapsed times are the time to process the full query batch and are the median of five observations.

We used a perl script to determine for each index order, the DAAT cutoff at which a criterion level of 95% of overall maximum ERR score was achieved and to report the elapsed time corresponding to that cutoff. The results for `University` are shown in the following table, with the time ratio expressed relative to the lowest value in Column 3.

| Order | Cutoff | Elapsed time @ cutoff | Time ratio |
|---|---|---|---|
| Original | 40 | 0.132 | 1.11 |
| QIE | 10 | 0.119 | 1 |
| Random | 640 | 0.227 | 1.91 |
| Reverse | 1280 | 0.292 | 2.45 |

We see a big effect for the order of the index. Near-maximal performance is achieved at cutoff 10 for QIE order but not until 640 or 1280 for Random and Reverse orders. Indeed Reverse doesn't ever match the maximum QIE effectiveness level. Interestingly, the original index order (corresponding to approximately breadth-first crawl-order) performs relatively well, attaining the criterion at a cutoff of 40. The time ratio column shows that achieving the criterion perfermance level requires twice as much computation for the Reverse and Random orders.

For `gov-Popular` we see even larger differences between the best order and the worst:

| Order | Cutoff | Elapsed time @ cutoff | Time ratio |
|---|---|---|---|
| Original | 2560 | 0.421 | 3.53 |
| QIE | 160 | 0.141 | 1.18 |
| Random | 2560 | 0.455 | 3.82 |
| Reverse | 80 | 0.119 | 1 |

The original order for this collection was not crawl order and it seems to be particularly perverse. By reversing the order we can reduce the cost of achieving criterion performance by a factor of 3.5! In this case Reverse order slightly outperforms QIE, but the difference is only small.

The results for `gov-Agencies` shown in Figure 3 and in the table below, show a similar picture despite the fact that queries are very much longer and artificially created. Criterion performance is achieved with a much lower DAAT cutoff for QIE order than for the other orders and Random and Reverse orders require roughly twice as much computational effort to reach criterion performance.

| Order | Cutoff | Elapsed time @ cutoff | Time ratio |
|---|---|---|---|
| Original | 80 | 0.541 | 1.47 |
| QIE | 20 | 0.369 | 1 |
| Random | 160 | 0.875 | 2.37 |
| Reverse | 80 | 0.634 | 1.72 |

For `ClueWeb09-CatB` the picture is rather different to that of the other three experiments. Figure 4 shows that at very low cutoffs the QIE and Original orders achieve substantially better effectiveness than Random or Reverse. However the lines converge more quickly than for the other data sets and the performance of the QIE order relative to that of the Original falls away. Results in the following table appear to show that Original and Reverse orders can achieve criterion (95% of maximum) effectiveness with only 40% of the computational effort. The fact that the best order and its reverse achieve similar performance levels after a while seems interesting.

| Order | Cutoff | Elapsed time @ cutoff | Time ratio |
|---|---|---|---|
| Original | 2560 | 12.72 | 1.50 |
| QIE | 5120 | 20.98 | 2.47 |
| Random | 5120 | 23.56 | 2.78 |
| Reverse | 1280 | 8.49 | 1 |

In all four experiments, very close to 100% of maximum effectiveness is achieved by cutoff 5120, regardless of index order.

# 7. DISCUSSION

On the basis of the `University` experiment, it seems that there is a very clear advantage to be had from creating a favourable ordering of the index and choosing a low DAAT cutoff value. The `gov-Popular` and `gov-Agencies` experiments seem to confirm the value of this approach. However, the results for the `ClueWeb09-CatB` experiment do not show anywhere near as strong an effect for index ordering.

The QIE order does not perform as well on `ClueWeb09-CatB` as in the other experiments. A possible explanation is that the `University` , `gov-Popular` and `gov-Agencies` test sets are included among the eleven test sets used to set the weights for the static variables in the ranking function, but that `ClueWeb09-CatB` is not. Including `ClueWeb09-CatB` data among the tuning testsets

**Table 2: Elapsed time taken to reorder the indexes as a percentage of the original indexing time, including the time taken to determine the permutation. Reorder times used were the median of 3 observations.**

| Test | Original | qie | Reverse | Random | notes |
|---|---|---|---|---|---|
| `University` | 100 | 12.0 | 8.8 | 11.8 | index originally in crawl order |
| `gov-Whole` | 100 | 16.1 | 14.5 | 19.5 | 1index not in crawl order |
| `ClueWeb09-CatB` | 100 | 18.2 | 15.6 | 18.3 | index not in crawl order |

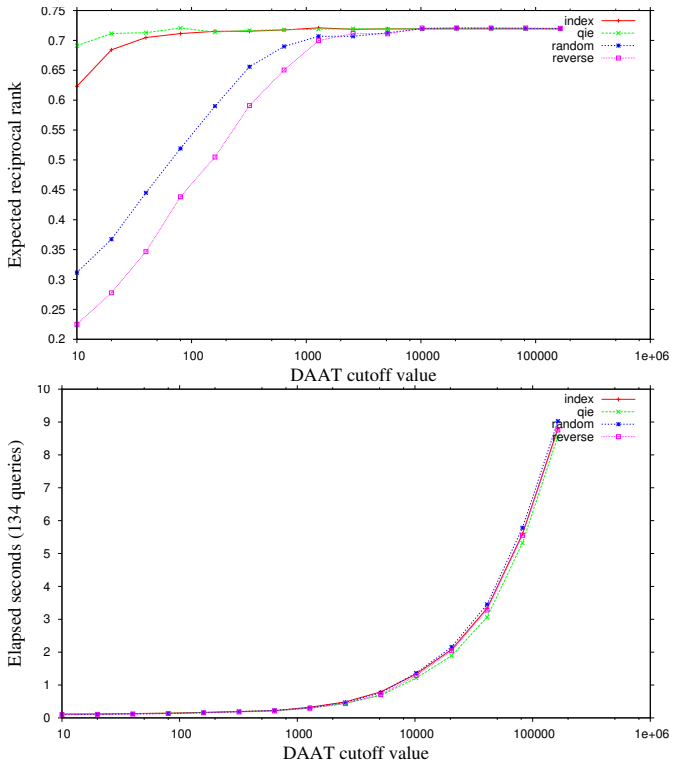


**Figure 1: Results for `University` plotted against DAAT cutoff. The upper plot shows expected reciprocal rank while the lower one shows time to process the batch of queries.**

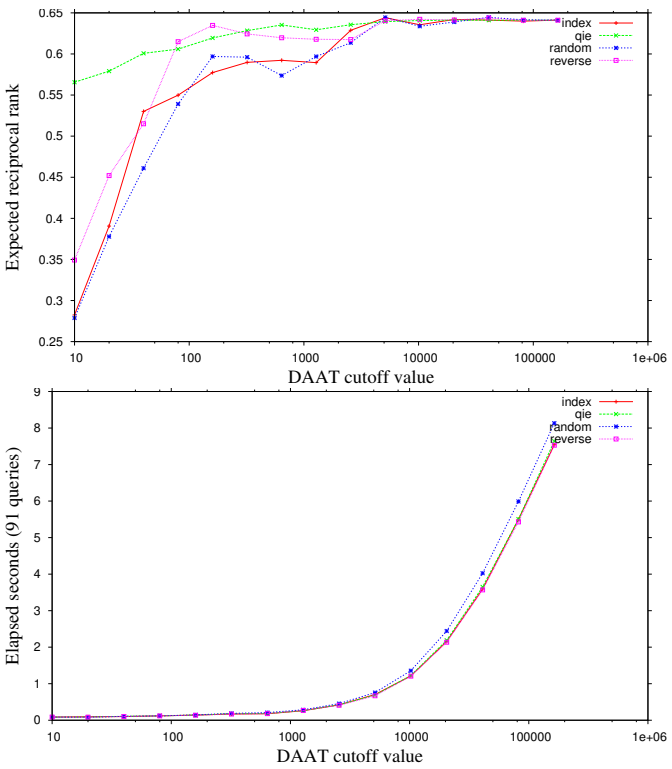


**Figure 2: Results for `gov-Popular` plotted against DAAT cutoff. The upper plot shows expected reciprocal rank while the lower one shows time to process the batch of queries.**

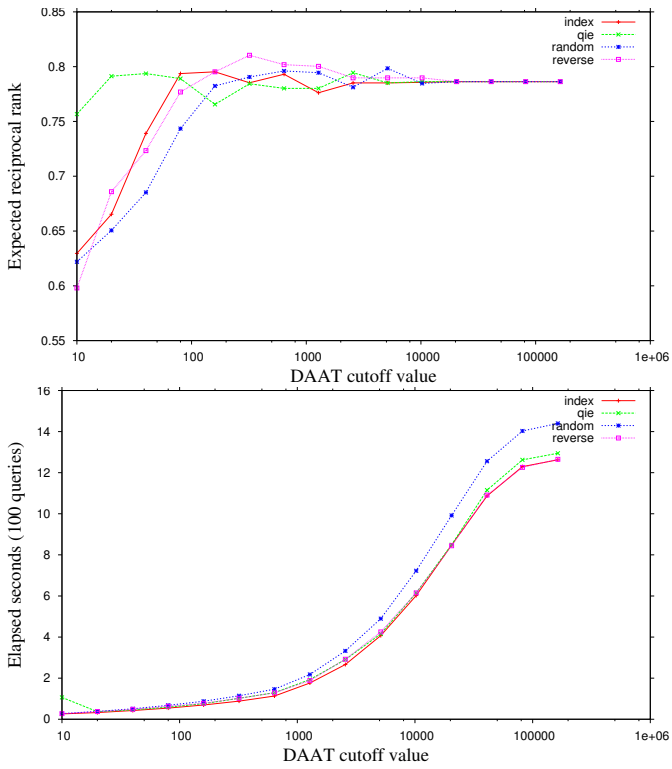


**Figure 3: Results for `gov-Agencies` plotted against DAAT cutoff. The upper plot shows expected reciprocal rank while the lower one shows time to process the batch of queries.**

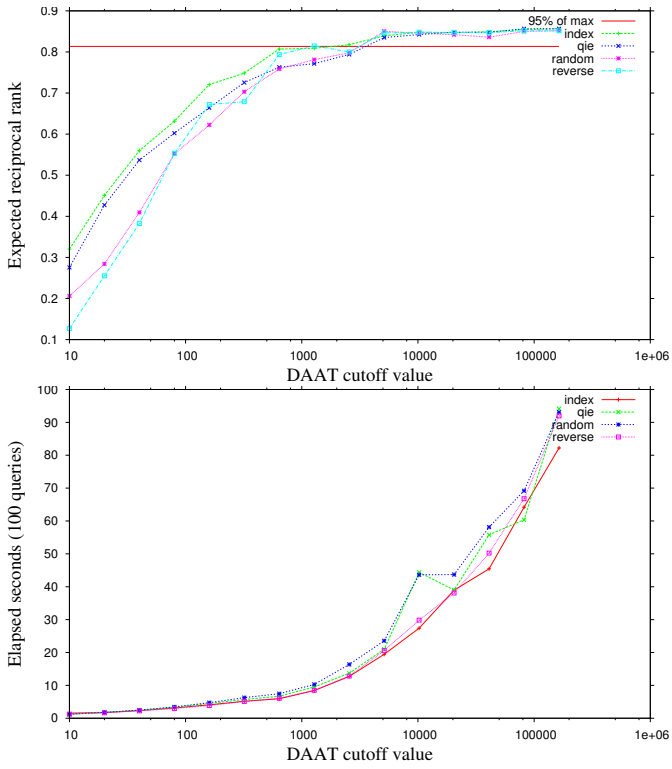


**Figure 4: Results for `ClueWeb09-CatB` plotted against DAAT cutoff. The upper plot shows expected reciprocal rank while the lower one shows time to process the batch of queries.**

may have resulted in different tunings and better performance of the QIE order in the present experiment.

Without a clear effectiveness result on the `ClueWeb09-CatB` data, we are unable to conclude anything useful about differences between resident and only partially resident indexes.

# 8. CONCLUSIONS AND FUTURE WORK

We have shown that reordering of an existing index can be achieved in a small fraction of the original indexing time, even when the size of the index is much larger than the available RAM configuration. Such reordering is of practical importance in collections where documents are gathered in an unfavourable order and where the optimal order is only determined during indexing.

Based on three of our experiments it appears that reordering an unfavourably ordered index allows near-full effectiveness to be achieved with only a fraction of the computational effort needed to fully index the collection. These findings essentially confirm the results of Long and Suel, in a range of different conditions, using a combination of features found to be effective across a range of enterprise search collections.

Further work is clearly needed to understand the different outcome of our fourth experiment. Useful follow up experiments include using a different query set or learning a better QIE order for the `ClueWeb09-CatB` data.

Crucially, we found that reordering indexes that include term position information does not affect the compressed index size. This allows for arbitrary orderings without affecting memory residency.

# 9. REFERENCES


[1] M. Bendersky, W. B. Croft, and Y. Diao. Quality-biased ranking of web documents. In *Proceedings of WSDM 2011*, pages 95–104, New York, NY, USA, 2011. ACM.

[2] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien. Efficient query evaluation using a two-level retrieval process. In *Proceedings of CIKM 2003*, pages 426–434, New York, NY, USA, 2003. ACM.

[3] J. Brutlag. Speed matters for Google web search. Technical report, Google, June 2009. `services.google.com/fh/files/blogs/google_delayexp.pdf`.

[4] P. Calado, B. Ribeiro-Neto, N. Ziviani, E. Moura, and I. Silva. Local versus global link information in the web. *ACM TOIS*, 21:42–63, January 2003.

[5] O. Chapelle, D. Metzler, Y. Zhang, and P. Grinspan. Expected reciprocal rank for graded relevance. In *Proceedings of CIKM 2009*. ACM, 2009.

[6] C. Clarke, N. Craswell, and E. Voorhees. TREC 2012 web track guidelines, 2012. `http://plg.uwaterloo.ca/~trecweb/2012.html`.

[7] C. L. Clarke, N. Craswell, I. Soboroff, and E. Voorhees. Overview of the TREC 2011 Web Track. In *Proceedings of TREC 2011*. NIST, 2011.

[8] G. V. Cormack, M. D. Smucker, and C. L. Clarke. Efficient and effective spam filtering and re-ranking for large web datasets. *Inf. Retr.*, 14(5):441–465, Oct. 2011.

[9] J. S. Culpepper, M. Petri, and F. Scholer. Efficient in-memory top-k document retrieval. In *Proceedings of SIGIR 2012*, pages 225–234, New York, NY, USA, 2012. ACM.

[10] J. Dean. Challenges in building large-scale information retrieval systems: invited talk. In *Proceedings of WSDM 2009*, pages 1–1, New York, NY, USA, 2009. ACM.

[11] S. Ding and T. Suel. Faster top-k document retrieval using block-max indexes. In *Proceedings of SIGIR 2011*, pages 993–1002, New York, NY, USA, 2011. ACM.

[12] S. Garcia and A. Turpin. Efficient query evaluation through access-reordering. In *Proceedings of the Third Asia conference on Information Retrieval Technology, AIRS 2006*, pages 106–118, Berlin, Heidelberg, 2006. Springer-Verlag.

[13] Y. Liu, B. Gao, T.-Y. Liu, Y. Zhang, Z. Ma, S. He, and H. Li. BrowseRank: letting web users vote for page importance. In *Proceedings of SIGIR 2008*, pages 451–458, 2008.

[14] X. Long and T. Suel. Optimized query execution in large search engines with global page ordering. In *Proceedings of VLDB 2003*, pages 129–140, 2003.

[15] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford, January 1998. `dbpubs.stanford.edu:8090/pub/1999-66`.

[16] M. Richardson, A. Prakash, and E. Brill. Beyond PageRank: machine learning for static ranking. In *Proceedings of WWW 2006*, 2006.

[17] S. E. Robertson, S. Walker, M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. In *Proceedings of TREC-3*, pages 109–126, November 1994. NIST special publication 500-225.

[18] F. Scholer, H. E. Williams, J. Yiannis, and J. Zobel. Compression of inverted indexes for fast query evaluation. In *Proceedings of SIGIR 2002*, pages 222–229. ACM Press, 2002.

[19] D. Shan, W. X. Zhao, J. He, R. Yan, H. Yan, and X. Li. Efficient phrase querying with flat position index. In *Proceedings of CIKM 2011*, pages 2001–2004, New York, NY, USA, 2011. ACM.

[20] F. Silvestri. Sorting out the document identifier assignment problem. In *Proceedings of ECIR 2007*, pages 101–112, Berlin, Heidelberg, 2007. Springer-Verlag.

[21] T. Strohman and W. B. Croft. Efficient document retrieval in main memory. In *Proceedings of SIGIR 2007*, pages 175–182, New York, NY, USA, 2007. ACM.

[22] K. M. Svore, P. H. Kanani, and N. Khan. How good is a span of terms?: exploiting proximity to improve web retrieval. In *Proceedings of SIGIR 2010*, pages 154–161, New York, NY, USA, 2010. ACM.

[23] K. M. Svore, M. N. Volkovs, and C. J. Burges. Learning to rank with multiple objective functions. In *Proceedings of WWW 2011*, pages 367–376, New York, NY, USA, 2011. ACM.

[24] H. Turtle and J. Flood. Query evaluation: strategies and optimizations. *Inf. Process. Manage.*, 31(6):831–850, Nov. 1995.

[25] H. Yan, S. Ding, and T. Suel. Inverted index compression and query processing with optimized document ordering. In *Proceedings of WWW 2009*, pages 401–410, New York, NY, USA, 2009. ACM.