Efficient Extended Boolean Retrieval

Stefan Pohl, Alistair Moffat, and Justin Zobel

Abstract—Extended Boolean retrieval (EBR) models were proposed nearly three decades ago, but have had little practical impact, despite their significant advantages compared to either ranked keyword or pure Boolean retrieval. In particular, EBR models produce meaningful rankings; their query model allows the representation of complex concepts in an and-or format; and they are scrutable, in that the score assigned to a document depends solely on the content of that document, unaffected by any collection statistics or other external factors. These characteristics make EBR models attractive in domains typified by medical and legal searching, where the emphasis is on iterative development of reproducible complex queries of dozens or even hundreds of terms. However, EBR is much more computationally expensive than the alternatives. We consider the implementation of the p-norm approach to EBR, and demonstrate that ideas used in the `max-score` and `wand` exact optimization techniques for ranked keyword retrieval can be adapted to allow selective bypass of documents via a low-cost screening process for this and similar retrieval models. We also propose term-independent bounds that are able to further reduce the number of score calculations for short, simple queries under the extended Boolean retrieval model. Together, these methods yield an overall saving from 50 to 80 percent of the evaluation cost on test queries drawn from biomedical search.

Index Terms—Document-at-a-time, efficiency, extended Boolean retrieval, p-norm, query processing.

1 Introduction

Search service providers have an interest in delivering competitive effectiveness levels within the smallest possible resource cost. This is no less true in specialized search services dedicated to medical and legal literature, which are called upon to support complex queries by professional searchers, possibly with significant commercial or societal outcomes resting on the results of the search. In particular, although the number of queries submitted per day to biomedical search engines is orders of magnitude less than the number submitted to web-scale search systems (millions per day for PUBMED,[1] rather than billions per day for free web search), such services are typically funded as public services rather than by advertising; the queries are often much more complex, involving dozens or even hundreds of terms; there is a great deal of reformulation and reevaluation; and the user evaluation process typically involves hundreds or thousands of answer documents rather than a mere handful.

Ranked retrieval has been successfully deployed in a wide range of applications. The main advantages of ranking are the simplicity of querying, and that results are ordered by estimated relevance, so that query quality can quickly be assessed once the top few results have been inspected. Having the answers returned as a ranked list also gives users the ability to consciously choose the amount of effort they are willing (or able) to invest in inspecting result documents.

1. http://www.ncbi.nlm.nih.gov/pubmed/.

• *The authors are with the NICTA Victoria Research Laboratory, Department of Computer Science and Software Engineering, The University of Melbourne, Victoria 3010, Australia.*
E-mail: {spohl, alistair, jz}@csse.unimelb.edu.au.

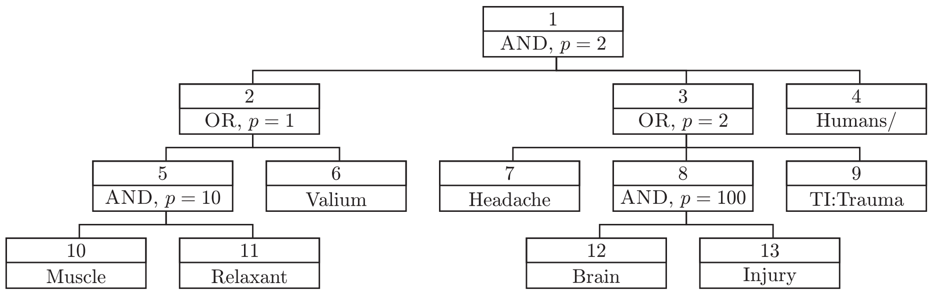
Manuscript received 10 May 2010; revised 8 Sept. 2010; accepted 10 Dec. 2010; published online 7 Mar. 2011.
Recommended for acceptance by B.C. Ooi.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2010-05-0276.
Digital Object Identifier no. 10.1109/TKDE.2011.63.

However, Boolean retrieval has not been superseded, and is still the preferred method in domains such as legal and medical search. Advantages of Boolean retrieval include:

- *Complex information need descriptions*: Boolean queries can be used to express complex concepts;
- *Composability and Reuse*: Boolean filters and concepts can be recombined into larger query tree structures;
- *Reproducibility*: Scoring of a document only depends on the document itself, not statistics of the whole collection, and can be reproduced with knowledge of the query;
- *Scrutability*: Properties of retrieved documents can be understood simply by inspection of the query; and
- *Strictness*: Strict inclusion and exclusion criteria are inherently supported, for instance, based on meta-data.

For these reasons, Boolean retrieval—and the extended Boolean variant of it that we pursue in this paper—remains a critically important retrieval mechanism. For carefully formulated information needs, particularly when there are exclusion criteria as well as inclusion criteria, ranking over bags of words is not appropriate. As one particular example, recent results suggest that ranked keyword queries are not able to outperform complex Boolean queries in the medical domain [1].

Boolean queries have the disadvantage of being harder to formulate than ranked queries, and, regardless of the level of expertise of the user, have the drawback of generating answer lists of unpredictable length. In particular, changes in the query that appear to be small might result in disproportionately large changes in the size of the result set. This is a problem that even expert searchers struggle with, adding and removing terms and operators until a reasonably sized answer set is retrieved, potentially even at the expense of retrieval effectiveness. Only when the answer set is of a manageable size can the searcher begin to invest time in examining its contents.



Id.	Term	Frequency
4	Humans/	9,912,283
6	Valium	603
7	Headache	41,671
9	TI:Trauma	36,313
10	Muscle	461,254
11	Relaxant	7,244
12	Brain	733,025
13	Injury	274,994

Fig. 1. Example query tree with assigned node identifiers, p-values, terms, and their document frequencies.

Extended Boolean retrieval (EBR) models, such as the p-norm model, seek to rank on the basis of Boolean query specifications [2], [3]. They generate a list of top-k answers that can be extended if required, without necessarily sacrificing detailed control over inclusion and exclusion of terms and concepts. But EBR queries are slow to evaluate, because of their complex scoring functions [4]; and none of the computational optimizations available for ranked keyword retrieval have been applied to EBR. In particular, approaches that involve nonexact methods, such as quantized impact-ordered indexes or index pruning [5], do not satisfy all of the requirements listed above.

Our contributions in this paper are threefold:

- We present a scoring method for EBR models that decouples document scoring from the inverted list evaluation strategy, allowing free optimization of the latter. The method incurs partial sorting overhead, but, at the same time, reduces the number of query nodes that have to be considered in order to score a document. We show experimentally that overall the gains are greater than the costs.
- We adopt ideas from the `max-score` and `wand` algorithms and generalize them to be applicable in the context of models with hierarchical query specifications and monotonic score aggregation functions. Further, we show that the p-norm EBR model is an instance of such models and that performance gains can be attained that are similar to the ones available when evaluating ranked queries.
- Term-independent bounds are proposed, which complement the bounds obtained from `max-score`. Taken alone, term-independent bounds can be employed in the `wand` algorithm, also reducing the number of score evaluations. Further, in conjunction with the adaption of `max-score`, this novel heuristic is able to short-circuit the scoring of documents.

We evaluate the efficiency of these methods on a large collection of biomedical literature using queries and results derived from real searches. Taken together, the optimizations greatly reduce query evaluation times for the p-norm EBR model on both short and complex queries, making EBR a competitive and viable choice for retrieval situations where such models are required. The results generalize to other models with hierarchical query specification and monotonic score functions.

2 Background

Our work is motivated by a commonly performed task in the biomedical domain, that of constructing a *systematic review*. Authors of systematic reviews seek to identify as much as possible of the relevant literature in connection with some aspect of medical practice, typically a highly specific clinical question. The review's authors assess, select, and synthesize the evidence contained in a set of identified documents, to provide a "best currently known" summary of knowledge and practice in that field. A variety of organizations provide central points of call for systematic reviews, including the Cochrane Collaboration,[2] the largest of these efforts, and the Agency for Healthcare Research and Quality, AHRQ.[3] The collections used as the source material are already large, and continue to grow. For example, as at end of 2009, MEDLINE, the largest of the available collections, contained more than 19 million entries, with more than 700,000 citations having been added during the year.

To construct each systematic review, a complex Boolean search query is used to retrieve a set of possibly relevant documents (typically in the order of one to three thousand), which are then comprehensively triaged by multiple assessors [6]. The Boolean query might consist of as many as several dozen query lines, each describing a concept with fielded keywords, MESH headings (a hierarchical taxonomy of medical terms), metadata, free-text term expansions, and Boolean operators aggregating and balancing the concepts. Fig. 1 shows the structure of one such query; this expression would be one small component of a typical complex query of (say) 50 clauses.

Reproducibility and scrutability are key requirements for the document selection process, and to this end the queries used are usually included verbatim in the review document. That requirement allows readers to study the query and, if necessary, to retrospectively analyze why particular documents have (and also have not) been included in the review.

Much research has been dedicated to the finding of good queries, for example, that of Zhang et al. [7]. Common concept descriptions should be reusable and composable to leverage such results. The complexity of queries is then further increased through the use of wildcard expansion terms based on either syntactic similarities, or inclusions based on taxonomy-based expansions. Queries might

2. http://www.cochrane.org/reviews/.
3. http://www.ahrq.gov/clinic/.

TABLE 1
Effectiveness of Different Retrieval Systems, Measured in Terms of the Fraction of Known Relevant Documents Identified, Averaged over a Query Set

System	Effectiveness at rank			
	$0.25B_q$	$0.5B_q$	B_q	$2B_q$
p-norm$_{\mathrm{BIN}}$, $p=9$	0.28	0.44	0.61	0.69
p-norm$_{\mathrm{TF\cdot IDF}}$, $p=9$	0.22	0.43	0.59	0.63
Boolean	0.16	0.34	0.59	n/a
Keyword queries (BM25)	0.20	0.29	0.41	0.58

The various rank cutoffs are expressed as multiples of B_q, the per-query Boolean result set sizes. This table is reproduced from Pohl et al. [10].

become as large as a thousand terms; leading to the inescapable conclusion that efficient evaluation techniques are required.

The strict nature of the conjunctions in pure Boolean queries can exclude relevant literature that must then be found (hopefully) by different means, such as following citations and asking experts in the field [8]. However, failure to find relevant documents that contain evidence for life-threatening conditions can lead to fatal outcomes for patients [9]. If a larger number of relevant documents can be found during the initial search process, the likelihood of such incidents will be reduced.

One of the reasons that key documents might be missed is that the queries have to be specific enough that they retrieve a set of documents that it is feasible to exhaustively review. But the size of a Boolean result set is unknown until after the query has been processed, and the quality of the result set is unknown until it has been sampled. Indeed, it can be a laborious process to generate queries as complex as those reported, with months of time being typical [7]. As well, the process requires query reformulation and repeated assessment of result quality via sampling, even as a prelude to the main evaluation stage.

Extended Boolean retrieval has been shown to provide advantages over pure Boolean retrieval in these two regards [10]. Table 1, reproduced from that previous work, compares the retrieval performance of complex structured Boolean queries with extended Boolean queries (using the p-norm approach that is described shortly), for which the queries have been transformed to more balanced query trees containing only the three basic Boolean operators.

As another reference point in the comparison, we also report the outcomes that are achieved when the obvious keyword queries are executed in a ranked sense using BM25, a state-of-the-art retrieval function (see Armstrong et al. [11] for an evaluation of relative retrieval effectiveness) and one possible alternative to the more complex EBR approach. To accomplish this part of the comparison, queries were generated by extracting the leaf terms of the structured queries, and using parts of the review as query; we only report results of the best queries (refer to Pohl et al. [10] for further details). Note however, that a similarity-based ranking, such as achieved by BM25, language models, or other mechanisms based on the cosine measure, can change in unpredictable ways as a collection is updated, for example—a characteristic that is unacceptable for repeatable retrieval.

The rankings produced by BM25 on keyword queries were less versatile and not able to outperform Boolean retrieval using high-quality structured queries, mirroring an effect that has also been noted in the legal domain [12] and for ad hoc retrieval using language models [13]. In contrast, the EBR models performed at the same level as did Boolean retrieval when the same number of results is produced, but have the advantages of ranking. Because scrutability, repeatability, and maximum effectiveness are indispensable, we do not provide a detailed evaluation of ranked keyword approaches in this paper.

The high level of scrutability attained by EBR also benefits from the finding that binary EBR term weights achieve better retrieval results than using *TF·IDF* term weights, which facilitates transparency by allowing the calculation of document scores with just the concise information provided in the review. The superiority of the binary approach might be due to the fact that retrieval is performed on document abstracts and metadata, so that most term frequencies are one, and reducing the importance of common but important metadata terms is likely to be unhelpful.

To complete this brief overview of the experimental design, it should be noted that systematic reviewing is a recall-oriented task performed on document result sets, not document rankings. It follows that a set-based effectiveness metric should be used, and to this end we employ relative recall, the number of retrieved relevant documents as a fraction of all known relevant documents. The same metric can also be evaluated on any given subset of an answer set or ranking.

2.1 Extended Boolean Retrieval

The various extended Boolean retrieval models all use Boolean queries as (formal) descriptions of information needs, but differ from each other in terms of the scoring functions and term weights used to generate a final similarity. Lee [2] gives an overview of the different models that have been proposed, and shows that only the p-norm model [3] has two key properties that, if not present, are detrimental to retrieval effectiveness. Approaches to EBR include fuzzy-set (soft) Boolean retrieval [14], Waller-Kraft [15], Paice [16], p-norm [3], Infinite-One [17], and inference networks [18]. The `wand` retrieval function described by Broder et al. [19] can also be viewed as an EBR approach, recursively applied to Boolean query specifications. Here, we build on the work of Lee [2], and focus on the p-norm model. Worth noting, however, is that the methods presented are applicable to a broad range of EBR approaches.

Boolean queries can be represented as trees, where the leaves are drawn from the set of terms $T = \{t_1, \ldots, t_n\}$, and the internal nodes are Boolean operators B, with $B.type \in \{\mathtt{AND}, \mathtt{OR}\}$. Extended Boolean retrieval models differ in the score functions used for each of the basic Boolean operators, that is, how individual scores s_c in the children of each internal node are aggregated, where each child clause $c \in \mathcal{C}$ is either a term or a query subtree. Scores are typically in the range $[0, 1]$, with 0 signifying complete absence, and 1 signifying absolute presence, of whatever concept that node or leaf represents.

For disjunctions (OR), the p-norm model defines

$$f^{\mathrm{OR}}(\mathcal{C}, p) = \left(\frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} s_c^p \right)^{1/p} , \tag{1}$$

and for conjunctions (AND) the corresponding score function is

$$f^{\mathrm{AND}}(\mathcal{C}, p) = 1 - \left(\frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} (1 - s_c)^p \right)^{1/p} . \tag{2}$$

The p-value controls the strictness of the operator, and can be set separately at each internal node; see, for example, the tree shown in Fig. 1. In the limit, $p = \infty$ reduces to strict Boolean evaluation when binary leaf weights are used, and $p = 1$ to inner product similarity scoring. Salton and Voorhees [20] discuss values for p.

The computed similarity scores can be used to rank documents relative to the query, allowing the user to consciously choose to inspect any desired number of documents. Note, however, that always retrieving (say) k documents might violate the important reproducibility requirement if new documents have been subsequently added to the document collection. Instead, the associated EBR document scores of the ranking can be analyzed and an appropriate score cut-off chosen that would then also be reported alongside the final query. Also useful in the initial stages of query formulation is that the effectiveness of any particular query can quickly be estimated by looking at the head of the ranking it generates.

A straightforward implementation of EBR would compute a score for every document that has at least one term in common with the query (the OR-set of the query terms), by recursively processing the whole query tree and propagating scores bottom-up from the leaves, drawing leaf scores as required from a set of inverted lists—a mode of operation that is usually referred to as being *document-at-a-time* processing. This is the method originally used in the SMART system [3].

Smith [17] proposed to recursively aggregate inverted lists, calculating and storing intermediate scores for every document that is encountered in any of the lists, in what is referred to as being the *term-at-a-time* approach. In effect, not all nodes in the query tree are visited for every document, but all of the inverted lists are fully inspected, and temporary memory proportional to the total size of the relevant inverted lists is required. Smith's *Infinity-One* method gives an approximation to the p-norm model, with the aim of reducing computational cost by reducing the volume of floating point operations. As is demonstrated below, the number of score calculations can be greatly reduced via an exact lossless pruning approach.

Inference networks are another alternative to the p-norm model, in that they also supersede strict Boolean retrieval when binary leaf weights are used [18]. However, to the best of our knowledge, they only have been shown to perform well using either ad hoc TF·IDF weights, or weights derived using language modeling approaches [13]; both of which involve document collection statistics which are not applicable in the domain of interest due to the scrutability requirement. Language models have the additional complexity of requiring parameter tuning.

2.2 Optimization Principles

Inverted lists are generally accepted to be the most appropriate data structure for information retrieval systems [21], with the two main query evaluation strategies being term-at-a-time and document-at-a-time. In the former, the inverted list of each query term is fully processed before the next is opened, and intermediate scores for all candidate answers are stored in a set of accumulators. This leverages fast sequential disk access, and when only one value has to be stored per candidate document—as is the case with ranked queries—is an attractive approach. In addition, the number of accumulators can be restricted without any great loss in effectiveness [22]. However, when complex Boolean queries are being processed, each accumulator is a complex structure corresponding to the complete state of a partially processed query. Nor can pruning be used to reduce the cost, because the set of answers to be produced is deterministic, rather than heuristic.

Document-at-a-time processing accesses all of the inverted lists at the same time, stepping through them concurrently and fully considering any document that appears in any of the lists before moving on to the next. Note that, because of compression considerations, inverted lists are typically ordered by document number, so document-at-a-time systems operate as a type of multiway merge, and do not need to backtrack through the lists. This simplifies the implementation of nested and complex operators, and there is no storage of intermediate results for any documents except the current one. The drawback is that the resultant data access pattern is multilocation sequential rather than single-location sequential, and explicit or implicit (by allowing the operating system to prefetch blocks of data) buffering must be used, so that the great majority of "get next document pointer" operations are still performed out of memory. That is, document-at-a-time evaluation strategies should be preferred for complex queries. Strohman et al. [23] also make this observation.

In order to facilitate pruning in ranked queries, as well as accelerate conjunctive Boolean queries, Moffat and Zobel [22] propose that *skip* pointers be inserted within inverted lists, so that groups of pointers known to be redundant to the computation can be bypassed. In the context of ranked querying they propose the *Continue* and *Quit* term-at-a-time methods that process inverted lists until the number of accumulators reaches some preset limit. Processing then stops in the *Quit* method, with lower effectiveness levels being achieved. The alternative is the *Continue* strategy, which inspects the remaining terms, but directly skips to the documents for which an accumulator has already been established, to calculate their exact final scores. Restricting the set of documents for which scores are computed is an effective technique because, if ranked results are to be produced, only the top-k documents, perhaps 10, or 100, or even 1,000, but not 100,000, are returned.

Finally, we note that other inverted list orderings and optimizations have been proposed [5], [24]. However, they do not apply if exact results are required, or when there is no difference in the term contributions between documents, which is the case if binary term weights are used.

2.3 Ranking Algorithms

Turtle and Flood [25] describe the `max-score` ranking mechanism, to accelerate keyword query evaluation when sum-score aggregation functions are used and only the top-k documents are required. Using document-at-a-time evaluation, the algorithm commences by fully scoring the first k documents in the `OR`-set of the query terms. Thereafter, the kth largest document score is tracked, as an *entry threshold* that candidate documents must exceed before they can enter the (partial) ranking. The `max-score` algorithm uses the information conveyed by the entry threshold to reduce two cost factors: 1) the number of candidate documents that are scored; and 2) the cost associated with scoring each candidate document. To achieve this, the terms in the ranked query are ordered by decreasing document frequency. Then, for each term t_i in the ordering, the highest achievable score is computed for a document containing all of the terms $t_1 \cdots t_i$. To compute the threshold score for t_{i+1}, the maximal term-contribution of t_{i+1} is determined, and then added to the score of t_i.

During query processing, the entry threshold is monotonically increasing, and at some point is likely to become sufficiently large that it can be concluded that a document containing only the commonest term t_1 (and none of the other terms) cannot make it into the top k. At that moment in time the set of candidate documents to be checked is reduced to the `OR`-set of $t_2, \ldots, t_n$, and processing continues until the threshold associated with t_2 is also less than the entry threshold. Independently, these score bounds also allow short-circuiting of the evaluation of each candidate document, so that not all terms are necessarily inspected. Note that the document frequency dictates the order in which terms are evaluated.

In contrast, Broder et al. [19] propose an algorithm that continuously maintains the terms sorted by the next document identifier in their inverted lists and skips one of the terms on smaller document identifiers to the next candidate. This possibly decreases the number of posting accesses by exploiting the document identifier distribution over the inverted lists, but is at the expense of additional sorting overhead.

An alternative to tracking the score of the kth document is to specify a minimal entry threshold prior to any documents being scored, making the retrieval of documents reproducible, but meaning that the result set for any given query will grow as the collection grows. Or, if term contributions differ between documents, then a higher initial threshold can be attained when top-scoring documents for each term are precomputed and stored as additional lists at indexing time, and then merged for each query before query evaluation starts. Strohman et al. [23] demonstrate that these methods do indeed reduce the number of documents that have to be scored, and that retrieval times were also improved. Note, however, that to date these methods have primarily been applied to ranking of flat keyword queries, possibly extended with proximity operators and phrases, and that they have not been applied to structured queries because the overall scoring functions do not decompose into a sum over term contributions.

3 Optimizations

Generally speaking, any form of ranked query evaluation, including both conventional keyword-based ranking and EBR, takes time that is linear in the product of the number of documents that match at least one term (the `OR`-set size) and of the query complexity. If a query contains very common terms, the first of these two factors can result in every document in the collection needing to be scored. This is common with complex Boolean queries. Worse, complex queries sometimes contain hundreds of terms, meaning that the second factor can also be high. We investigate ways to reduce both of these costs.

3.1 Scoring Method

Rather than being a simple sum, the overall scoring function in the p-norm model is a nested application of (1) and (2), determined by the query tree. Hence, it is not possible to aggregate the score for a document starting with any arbitrary term, including the one with the lowest document frequency. The recursive nature of EBR queries makes it necessary to calculate the scores on lower levels in the query tree first. One obvious possibility would be to try and add processing logic to each query node as it acts on its clauses. But optimizations such as `max-score` could only be employed at the query root node, as a threshold is only available for the overall query score. Instead, we follow a holistic approach and prefer to be able to calculate the document score given a set of query terms $S \subseteq T$ present in a document, no matter where they appear in the query tree.

Our approach, described in Algorithm 1, assumes that each query tree node N_i, for $i = 1, \ldots, n$, is assigned a smaller index identifier than any of its children, so that $N_i.P < i$, where $N_i.P$ is the index of the parent node of N_i, and where each node N is either a term drawn from T, or a Boolean operator drawn from B. It necessarily follows that N_1 denotes the query's root node; Fig. 1 gives an example.

Algorithm 1: *CalcScore*() – Query Tree Scoring

```
   Input: $T$, a set of numbered terminals, and $B$, a set of numbered
          internal nodes; collectively they form $N$, a set of tree
          nodes describing a Boolean expression
 1 $S \leftarrow \{T_i \in T \mid T_i.s > 0\}$
 2 while $S \neq \{N_1\}$ do
 3     Determine largest parent node index:
       $j = \arg\max_j \{S_i \in S \mid j = S_i.P\}$
 4     Determine active clauses of $B_j$ in $S$:
       $A = \{S_i \in S \mid S_i.P = j\}$
 5     Split $A$ into the two sets $A^{s=1}$ and $A^{0<s<1}$
 6     if $|A^{0<s<1}| = 0$ then
 7         Lookup pre-computed score when operands are
           all-binary:
           $B_j.s \leftarrow \mathrm{TableLookup}(B_j, |A^{s=1}|)$
 8     else if $B_j.type$ = OR then
 9         $B_j.s \leftarrow \left(\frac{1}{|B_j.\mathcal{C}|}\left(|A^{s=1}| + \sum_i (A_i^{0<s<1}.s)^{B_j.p}\right)\right)^{\frac{1}{B_j.p}}$
10     else if $B_j.type$ = AND then
11         $k^{s=0} \leftarrow |B_j.\mathcal{C}| - |A^{0<s<1}| - |A^{s=1}|$
12         $B_j.s \leftarrow 1 - \left(\frac{1}{|B_j.\mathcal{C}|}\left(k^{s=0} + \sum_i (1 - A_i^{0<s<1}.s)^{B_j.p}\right)\right)^{\frac{1}{B_j.p}}$
13     end
14     Remove the processed nodes from $S$, and add their parent:
       $S \leftarrow S - A + \{B_j\}$
15 end
16 return $N_1.s$
```

Within the tree, each operator node $B_i \in N \setminus T$ has clauses $B_i.\mathcal{C}$, a p-value $B_i.p$, a Boolean operator type $B_i.type \in \{\texttt{AND}, \texttt{OR}\}$, and (once it is computed) a score, $B_i.s$. Similarly, each leaf node $T_i \in N \backslash B$ has a parent node with index $T_i.P$, and an initial (binary) score, $T_i.s$. Technically, both $T_i.P$ for leaf nodes and $B_i.P$ for internal nodes are multisets, since terms and subexpressions can be shared; but for simplicity of description we assume single values. The actual implementation described below makes use of multisets, and alters step 14 accordingly.

The algorithm iterates from leaves to root, following the node numbering downward, computing scores for internal nodes based on their (already computed) children. When the query root node N_1 is reached, the overall score can thus be calculated. Only the nodes that have at least one term in the subtrees of any of their clauses are visited, pruning the query tree to its active parts, and making the algorithm suited for scoring small sets of terms.

For example, given the query depicted in Fig. 1, to score a document containing only the query terms $S = \{t_4, t_6\} = \{Humans/, Valium\}$, the algorithm would first calculate directly the intermediate score for N_2, considering N_5 to have a score of 0; and would then reach N_1 to calculate the overall score for the document. While a naive implementation might have to initialize all eight queries term nodes with values and calculate scores for all five operator nodes, the implementation is initialized with two query term nodes and involves scorings for two operator nodes only. Although the algorithm involves some additional (partial) sort overhead, we gain comparable reductions in the number of visited nodes compared to the algorithm of Smith [17], without having to store any intermediate lists. Even smaller numbers of documents are scored when the optimizations in the next sections are also applied.

Lee et al. [4] state that computing p-norms is much slower than other, simpler scoring functions. However, for operators with (binary) term clauses only, or in other cases where all clause scores are either 1 or 0, the score function simplifies to a simple calculation dependent only on the number of terms with a score of 1, denoted as $|A^{s=1}|$ in the algorithm. For each operator B_i, all possible scores can be precomputed and stored in a lookup table $[B_i, 0 \leq k^{s=1} \leq |B_i.\mathcal{C}|] \mapsto s$, used in step 7. This happens to be frequent for binary term weights and parent-of-leaf nodes, but not at other internal nodes. Hence, we also seek methods to reduce the number of calls to the *CalcScore*() function.

3.2 Scoring Fewer Documents

A critical observation is that the `max-score` optimization can be applied to score aggregation functions other than summation. The property required is that the scoring function be monotonic—that, given a set of terms S appearing in a document, no superset $S' \supset S$ can have a lower score. The p-norm model possesses this property, provided that there are no negations in the query. A proof is appended to this paper. In the medical domain that is the subject of this paper it is entirely plausible to assume that all negations appear at the leaves; and we explain shortly as to how negations can be handled in general queries.

To take advantage of the `max-score` optimization, the terms are sorted by decreasing document frequency. Then, instead of calculating cumulative term contributions as is the case for ranked keyword querying, we calculate afresh the overall EBR score L_i for every incremental term subset $t_1, \ldots, t_i$ with $1 \leq i \leq n$. This score is always 0 for the empty set, and, when the term weights are binary, is likely to be 1 when $i = n$. The monotonicity requirement means that these scores are increasing.

During document-at-a-time processing, an entry threshold is maintained, being the minimum score of the current top k documents. At any given moment the entry threshold will exceed some subset of the precomputed score bounds. When the entry threshold exceeds L_i, documents that *only* contain subsets of the first i terms $t_1, \ldots, t_i$ cannot obtain scores large enough to compete for membership of the top k, and it is enough to consider for scoring only the documents that appear in the inverted lists of terms $t_{i+1}, \ldots, t_n$, that is, documents that are in a reducing `OR`-set. The first term to be removed from the reducing `OR`-set is t_1, the most common one, with the longest inverted list. The inverted list for t_1 is still consulted for documents that appear in $t_2, \ldots, t_n$, but some of the entries in t_1's inverted list can be bypassed, with no scoring at all performed for the corresponding documents. Once L_2 has been exceeded, many of the documents in the inverted lists of both t_1 and t_2 can be completely bypassed, and so on.

Returning to the example in Fig. 1, the first term that would be excluded (*Humans/*) is present in almost 10 million documents. After enough documents are found with scores larger than for documents that only contain *Humans/* (0.184), the number of documents that have to be scored decreases from more than 9,912,282 to between 733,025 and 1,555,104. Computing the term bounds L_i is more expensive than for `max-score` in ranked keyword queries, in that each one involves a full-blown computation of an extended Boolean query. However, they can be generated in an on-demand manner, so that L_{i+1} is computed only if the entry threshold has exceeded L_i and L_{i+1} is actually required; and, when i is small, only a few nodes are evaluated using *CalcScore*().

One potential drawback of our proposal is the restriction on negations. There are two reasons why we do not consider this to be a limiting factor. First, the use of negations is generally discouraged in the domains of interest because intemperate use can lead to reductions in numbers of identified relevant documents through unintended exclusion [26], [8]. Second, De Morgan's laws extend to the p-norm model [3], and can be applied to propagate negations toward the leaves of the query tree, at which point the term values themselves can be inverted, and the desired outcome achieved.

3.3 Term-Independent Score Bounds (TIB)

Our adaptation of `max-score` to the EBR p-norm model allows for reductions in the number of candidate documents, but at face value does not allow short-circuiting of the evaluation of candidate documents, because the term contributions are nonadditive. That is, every candidate document is fully scored, no matter how many (and which) terms actually are present. To provide early termination of document scoring, we also propose the use of *term-independent score bounds* that represent the maximum attainable score for a given number of terms. A lookup table of score bounds M_r is created, indexed by r, that is consulted to check if it is possible for a candidate document containing r of the terms to achieve a score greater than the current entry threshold. That is, for each $r = 1, \ldots, n$, we seek to determine

$$M_r = \max\{CalcScore(S, B) \mid S \subseteq T \text{ and } |S| = r\} .$$

The number of possible term combinations that could occur in documents is $O(\binom{n}{r})$ for each r, which is $O(2^n)$ in total, and a demanding computation. However, the scoring functions only depend on the clause scores, that is, the overall score of each subtree, meaning that the problem can be broken down into subparts, solved for each subtree separately, and then aggregated. The simplest (sub-)tree consists of one term, for which the solution is trivial. For a particular operator node with clauses $\mathcal{C}$, let n_c denote the number of terms in the subtree of clause $c \in \mathcal{C}$. A table with $r = 0, \ldots, \sum_{c \in \mathcal{C}} n_c$ possible terms present is then computed; and to compute each M_r, all possibilities to decompose r into a sum over the clauses $r = \sum_{c \in \mathcal{C}} r_c$ have to be considered.

Instead of using the p-norm scoring functions directly, the computation can be performed in a different space by applying the inner part of the scoring functions (exponentiation) to the table scores for each clause. Then, the search translates into a maximization of score sums for disjunctions, and minimization of score sums for conjunctions, respectively. This reduces the cost of computing candidate solutions. Once the search has finished, the corresponding maximum score can be regenerated.

The cost of generating the whole table for a node is generally $O(\prod_{c \in \mathcal{C}} n_c)$. However, on average, it is much smaller in practice if some optimizations are applied. Similar to the adaptation of max-score, we only need to calculate M_{r+1} in the top-level node if the threshold surpasses M_r. However, M_r grows quite fast and we are most interested in bounds for small r to eliminate (the many) documents early from consideration that only have a few terms in common with the query. To generate M_r for a particular node, the tables of all clauses have to be filled only up to r. Hence, no node in the query tree has to calculate more than r table entries. Further, for nodes containing only terms with binary term weights, the table can be directly computed from the number of terms that appear; as well, TIB-score computations can be stopped when a predefined time limit is reached. Finally, it can be shown (proof similar to the one provided, but omitted for brevity) that the OR-scoring formula is convex and the AND-scoring formula concave, respectively. As long as a query subtree contains only one kind of operator, these properties can be used to further reduce computational complexity.

There are at least two possible applications for term-independent score bounds. First, they can be used instead of the adaptation of max-score. While max-score imposes an order on the terms in which they are excluded from the OR-set, term-independent bounds are able to dynamically exclude a number of arbitrary terms. When the current entry threshold exceeds M_r, all query terms can be (partially) sorted by their next candidate document identifier, and then the first r terms advanced (by a skipping process) until they are equal to or exceed the value of the $r+1$'th one. This is repeated until the first $r+1$ terms have the same value. Only then is it necessary for a document to be fully scored using *CalcScore*(), since to enter the answer set a document must (currently) contain more than r of the query terms. This approach is similar to the processing performed in the wand algorithm [19], but we do not calculate score bounds dynamically due to the demanding score calculations. Note that although this has the potential to exploit the document identifier distribution in the inverted lists, we are using less tight bounds than in adapted max-score, or the ones that wand uses in conjunction with sum-score functions.

Second, TIB can be combined with the adaptation of max-score. Even after the max-score approach has indicated that a document needs to be scored, the TIB filter might successfully eliminate that document before all inverted lists are consulted, based purely on how many of the query terms it *could* contain. For example, by inspection of inverted lists in order of document frequency, it might be known that a candidate document only contains one term plus possibly one other of the query terms that are yet to be consulted. If the current entry threshold score translates into a minimum of three query terms, this document cannot make it into the answer set and can thus be discarded. This saves us from scoring the document, and also increases the skip size on common terms, such as *Humans/* is in Fig. 1.

We explore both of these options in Section 4.

3.4 Conditional Bounds

Further extensions are also possible. The term-independent bounds M_r grow quickly, because they are based on a worst case assignment of terms, and particular terms might be responsible for large increases, but not occur in many documents. In that case it might be more advantageous to compute *term-dependent bounds* (TDB) that apply when a particular term is known to not appear in the document. In choosing the term to condition on, a tradeoff arises between a small document frequency of the term (which is desired, because it gives rise to a high rate of applicability) and the influence that term has on the bounds. The latter is correlated with the level in the query in which the term appears, but also the parent operators (type, p-value, number of clauses) and appearance of other terms have an influence.

Table 2 continues the previous example, and shows L_r, M_r, and the conditional M_r values corresponding to the term "Valium."

4 Experiments

We implemented document-at-a-time prototypes in Java, on top of Lucene.[4] In version 2.4, Lucene stores multilevel skip pointers in inverted lists that guarantee logarithmic access times to any posting and uses them in its document-at-a-time implementation of conjunctive operators. We retained the default settings (skip interval of 16, maximum skip levels of 10).

4. http://lucene.apache.org/java/.

TABLE 2
Maximally Scoring Term Sets and Their Scores L_r and M_r for Different Numbers of Terms r Present in a Document for `max-score`, Term-Independent Bounds, and Term-Dependent Bounds, Applied to the Example Query Given in Fig. 1

r	0	1	2	3	4	5	6	7	8
`max-score`									
Set	$\emptyset$	{4}	{4,12}	{4,12,10}	{4,12,10,13}	{4,12,10,13, 7}	{4,12,10,13, 7, 9}	{4,12,10,13, 7,9,11}	T
L_r	0	0.184	0.186	0.199	0.391	0.433	0.442	0.712	1
Term-Independent Bounds (`TIB`)									
Set	$\emptyset$	{4}	{4,7$\vee$9}	{4,6,7$\vee$9}	{4,6,7,9}	{4,6,7$\vee$9, 10,11}	{4,6,7,9, 10,11}	{4,6,7,9, 10,11,12$\vee$13}	T
M_r	0	0.184	0.374	0.623	0.693	0.756	0.895	0.895	1
Term-Dependent Bounds (`TDB`), Term 6 ("Valium") not present									
Set	$\emptyset$	{4}	{4,7$\vee$9}	{4,7,9}	{4,7$\vee$9,10, 11}	{4,7,9,10, 11}	{4,7,9,10, 11,12$\vee$13}	$T-\{6\}$	–
M_r^6	0	0.184	0.374	0.414	0.623	0.693	0.693	0.712	–

4.1 Data and Setup

Throughout the experiments, we use a late-2008 snapshot of MEDLINE, consisting of 17,104,854 citation entries with abstracts, collectively generating 6 GB of compressed inverted list data (excluding positional data and term information). Three query sets were used against this collection: 1) 50 simple, short PUBMED queries randomly sampled from the query log reported in Herskovic et al. [27], consisting of a Boolean conjunction only; 2) 50 structured queries sampled from the same query log, containing both conjunctive and disjunctive operators at least once in each query; and 3) 15 complex queries from AHRQ, as typically used in systematic reviews (available from Cohen et al. [28]). Properties of these queries are summarized in Table 3. To account for the fact that query structure plays an important role in EBR models but the queries were not intended to be used in this way, we flattened nested repetitions of the same Boolean operator (as has previously shown to provide good effectiveness [10]), and assigned fixed p-values to all operators. Where necessary, we also applied De Morgan's laws to propagate negations to the term level.

While we would have liked to have employed a large query set (perhaps thousands) to measure average throughput for complex EBR queries, it was not possible to do so because of the nontrivial nature of the queries involved, and the significant resources required to execute them. Also worth noting is that the test machine (Quad-Core Xeon 2.33 GHz, 64-bit and 16 GB memory) could accommodate the whole index in main memory, meaning that the cached timings are indeed free of any disk effects, and thus relatively stable. Further note that we do not report efficiency results for ranked keyword retrieval because the queries used are comparable in neither effectiveness nor size. In all but the pure Boolean queries, $k = 100$ documents were retrieved, as an estimate of the number of documents that might be inspected during each iteration while a review query is being developed.

We counted the number of scored documents with scores below the entry threshold (denoted redundant scorings in Table 4) and above the entry threshold (Table 3). The latter are the only scorings that a perfect or "clairvoyant" heuristic would permit. Also counted was the number of postings processed. Implementations of pure Boolean retrieval, and for several variants of EBR retrieval were measured, including the use of two different p-values. A p-value of 1 is of particular interest because it does not incur the computational costs of exponentiation, and for simple queries containing only one operator returns the same rankings. We also investigated the effect of retrieval of different numbers of documents (Fig. 2). Having measured operation counts, we then removed all of the instrumentation code, and executed the queries again, and report worst case (cold-start, caches of the operating system flushed, that is, disk-IO-bound) and best-case (maximal caching, that is, CPU-bound) timings. We ran each query set multiple times

TABLE 3
Properties of Query Sets

	PUBMED		AHRQ
	Simple	Structured	Complex
Queries	50	50	15
Terms (avg.)	2.3	7.1	75.1
Levels (avg.)	2.0	4.0	5.5
Nodes (avg.)	3.3	12.8	91.2
Queries with no results	19	6	0
Resultset size (min.)	0	0	587
(med.)	4.5	190	2,070
(max.)	61,104	65,090	17,330
Required scorings			
(Boolean)	1,412	3,588	3,255
(EBR, $p = 1$)	76,880	181,204	42,281
(EBR, $p = 10$)	76,880	179,966	23,678

Counts of terms and nodes are for unexpanded queries and are larger after MESH and wildcard expansions. The last three rows list the number of documents that entered the Boolean result set or the top-k heap (with two values of p), averaged over all queries in each set.

TABLE 4
Average Per-Query Counts of the Number of Redundant Document Scorings and of the Number of Postings Read from the Inverted Lists, for the Query Sets and for Two Choices of p; Plus Average Execution Times in Seconds

Query set and system		$p=1$				$p=10$			
		Redundant scorings	Postings processed	Time [s] Cold	Time [s] Cached	Redundant scorings	Postings processed	Time [s] Cold	Time [s] Cached
PUBMED Simple	Boolean	–	25,090†	0.07†	0.01†				
	EBR, Tree Iteration baseline	1,954,153†	2,107,220†	0.52†	0.46†				
	EBR, CalcScore() baseline	1,954,153†	2,107,220†	0.27†	0.20†				
	EBR, `max-score` only	319,919	462,621	0.17	0.05				
	EBR, `TIB` only	109,096	321,761	0.18†	0.06				
	EBR, `max-score + TIB`	11,598†	444,109†	0.26†	0.11†				
PUBMED Structured	Boolean	–	89,899†	0.47†	0.03†				
	EBR, Tree Iteration baseline	5,544,283†	15,391,506†	5.30†	4.50†	5,545,521†	15,391,506†	8.80†	7.78†
	EBR, CalcScore() baseline	5,544,283†	15,391,506†	2.77†	2.15†	5,545,521†	15,391,506†	5.50†	4.60†
	EBR, `max-score` only	1,078,782	4,660,586	1.40	0.62	1,102,909	4,707,938	2.12	1.27
	EBR, `TIB` only	803,041	4,147,582†	1.44†	0.68†	1,315,693	7,009,658†	2.67†	1.85†
	EBR, `max-score + TIB`	387,820†	3,708,108†	1.23	0.48	762,653†	4,455,036†	1.97	1.14
AHRQ Complex	Boolean	–	913,963†	10.38†	0.38†				
	EBR, Tree Iteration baseline	14,316,891†	105,541,788†	40.96†	30.03†	14,335,493†	105,541,788†	65.87†	53.00†
	EBR, CalcScore() baseline	14,316,891†	105,541,788†	33.61†	23.86†	14,335,493†	105,541,788†	48.98†	37.81†
	EBR, `max-score` only	2,958,833	65,316,498	22.25	10.20	2,589,252	61,355,097	25.12	12.18
	EBR, `TIB` only	2,253,448†	64,204,793	25.59†	12.78†	5,109,325†	89,370,092†	31.77†	17.67†
	EBR, `max-score + TIB`	2,057,016†	61,388,434†	20.19†	8.84†	2,487,645†	61,178,140†	22.10†	10.20†

Because the rankings produced for "PUBMED Simple" are unaffected by the choice of p, they were only executed with $p = 1$. All results are for $k = 100$ documents retrieved per query, with execution times averaged over five runs. Dagger superscripts indicate significance at the 0.01 level using a paired, two-sided Wilcoxon signed-rank test compared to the "`max-score` only" outcomes. Note that many of the pure Boolean queries returned no answers (Table 3), but that the EBR queries always returned $k = 100$ answers.

and got repeatable results with small standard deviation. All measurements are for the query evaluation process only, without access to the underlying documents.

Apart from localized rank-swaps due to rounding errors, all runs for the different systems returned the same rankings.

4.2 Results

A number of observations can be made from the results. First, although the *CalcScore*() scoring method requires partial sorting of tree node identifiers during document scoring, it has comparable execution times to a straightforward implementation that recursively iterates the query tree and computes scores for the next document in the OR-set of that subtree (Tree Iteration), often being faster. However, this advantage might be due to localized access patterns and optimization of the scoring method.

Second, the scoring method in the adaptation of `max-score` yields significant reductions in the number of candidate documents scored, postings processed, and execution times, for all query sets.

Third, nonunit p values lead to increased computational costs due to exponentiation. The more documents are to be retrieved (Fig. 2), the less effective the top-k optimizations are, but even so, execution times are significantly below the baselines.

Fourth, the `TIB` method for short-circuiting candidate document scoring is indeed able to reduce the number of score calculations, particularly on simpler queries and when combined with `max-score`. Indeed, for the "PUBMED Structured" queries the number of score calculations actually performed drops to within a factor of two of the minimal number that must inevitably be performed (documented in Table 3). On the other hand, the `TIB` approach does not translate into significant computational time savings over the simpler adapted `max-score`. Only if nonbinary term weights are being used—when the computational cost associated with scoring is likely to be larger—the `TIB` methods can be anticipated to have a clear advantage. The time to compute the necessary `TIB` bounds rarely reached our limit of 50 ms, validating the feasibility of the approach. In experiments not reported here because of space constraints, conditional term-dependent bounds

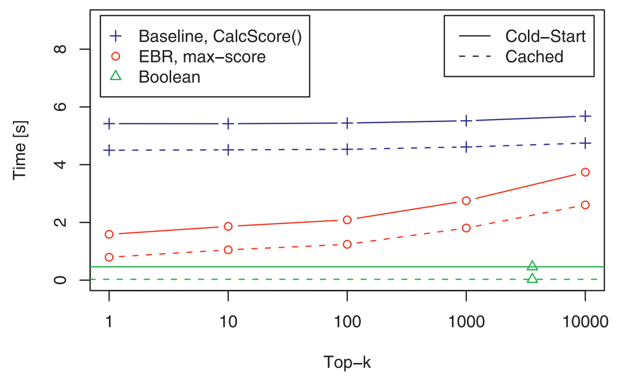


Fig. 2. Average query execution times on the "PUBMED Structured" query set ($p = 10$) for the adapted `max-score` variant as a function of k, compared with two baselines.

(TDB) turned out to be slightly better than TIB, but are very dependent on the choice of the term that is conditioned on.

Fifth, while the query execution times of strict Boolean execution appear to be magnitudes faster, it must be remembered that the result of a Boolean query is of indeterminate size, and that a nontrivial number of these queries in fact return no results at all, while others return many thousands of documents (Table 3). This uncertainty means that more Boolean queries are likely to be submitted by a user. Also, search service providers are free to decide which of the incoming queries they execute with the suggested EBR implementation, for instance, on basis of subscriptions by users that require the added benefits of the EBR model.

Finally, the gap between the disk-dominated and computational timings neatly correlates with the number of terms typically used in the query sets, and roughly reflects the number of term lookups necessary. It is also worth noting that "TIB only" sometimes has the best performance, but is not consistent in that regard.

5 Conclusion and Future Work

Having noted that ranked keyword querying is not applicable in complex legal and medical domains because of their need for structured queries including negation, and for repeatable and scrutable outputs, we have presented novel techniques for efficient query evaluation of the p-norm (and similar) extended Boolean retrieval model, and applied them to document-at-a-time evaluation. We showed that optimization techniques developed for ranked keyword retrieval can be modified for EBR, and that they lead to considerable speedups. Further, we proposed term-independent bounds as a means to further short-circuit score calculations, and demonstrated that they provide added benefit when complex scoring functions are used.

A number of future directions require investigation. Although presented in the context of document-at-a-time evaluation, it may also be possible to apply variants of our methods to term-at-a-time evaluation. Second, to reduce the number of disk seeks for queries with many terms, it seems desirable to store additional inverted lists for term prefixes (see, for example, Bast and Weber [29]), instead of expanding queries to hundreds of terms; and this is also an area worth exploration. We also need to determine whether or not term-dependent bounds can be chosen to consistently give rise to further gains. As another possibility, the proposed methods could further be combined and applied only to critical or complex parts of the query tree. Finally, there might be other ways to handle negations worthy of consideration.

We also plan to evaluate the same implementation approaches in the context of the inference network and wand evaluation models. For example, it may be that for the data we are working with relatively simple choices of term weights—in particular, strictly document-based ones that retain the scrutability property that is so important—can also offer good retrieval effectiveness in these important medical and legal applications.

6 Proof of Monotonicity

We consider term scores $s_i \in [0,1]$ and p-values in $[1,\infty)$. The special case of $p=\infty$ can be ignored since it is best evaluated with a pure Boolean implementation.

Theorem A.1. *The final score of an arbitrary query tree in the p-norm extended Boolean query model is monotonically increasing with term scores, or the presence of terms, respectively.*

The proof is by induction on query (sub-)trees:

Basis. The simplest query (sub-)tree consists of one term only. Then, the scoring function becomes $f(s)=s$. It follows trivially that $f'(s)=1>0$, hence, term clauses produce strictly increasing scores.

Induction. Now, we generalize this result to arbitrary query trees. Every query node contains a set of terms T distributed over its set of clauses $\mathcal{C}$ and aggregates the clauses' individual scores $f_{c\in\mathcal{C}}$. Each clause c is a subtree that contains a subset of terms $T_c \subset T$ with a particular assignment of scores s_j, where $j \in T_c$. The partial derivatives of the OR scoring function are all described by

$$\begin{aligned}
\frac{\partial f^{\mathrm{OR}}(f_{c\in\mathcal{C}}(s_j))}{\partial s_i} &= \frac{\partial\left[\frac{1}{|\mathcal{C}|}\sum_{c\in\mathcal{C}} f_c(s_j)^p\right]^{1/p}}{\partial s_i}\\
&= \underbrace{\frac{1}{p}}_{>0}\cdot\underbrace{\left[\frac{1}{|\mathcal{C}|}\sum_{c\in\mathcal{C}} f_c(s_i)^p\right]^{1/p-1}}_{\geq 0}\cdot\underbrace{\frac{p}{|\mathcal{C}|}}_{>0}\cdot\underbrace{f_c(s_i)^{p-1}}_{\geq 0}\cdot\frac{\partial f_c(s_i)}{\partial s_i}\\
&\geq 0 \iff \frac{\partial f_c(s_i)}{\partial s_i}\geq 0.
\end{aligned}$$

A similar argument holds for the AND scoring function. Then, because $f_c(s_i)$ is one of the two scoring functions that inherit monotonicity from its clauses, and a leaf in a query tree is always a term, the overall query tree score is monotonically increasing with term scores. Note that this proof does not hold if we allow for negations in the query tree. In such cases, negations can be propagated to the term level and removed by inversion of the leaf node scores.

Acknowledgments

National ICT Australia (NICTA) is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

[1] S. Karimi, J. Zobel, S. Pohl, and F. Scholer, "The Challenge of High Recall in Biomedical Systematic Search," *Proc. Third Int'l Workshop Data and Text Mining in Bioinformatics*, pp. 89-92, Nov. 2009.

[2] J.H. Lee, "Analyzing the Effectiveness of Extended Boolean Models in Information Retrieval," Technical Report TR95-1501, Cornell Univ., 1995.

[3] G. Salton, E.A. Fox, and H. Wu, "Extended Boolean Information Retrieval," *Comm. ACM,* vol. 26, no. 11, pp. 1022-1036, Nov. 1983.

[4] J.H. Lee, W.Y. Kin, M.H. Kim, and Y.J. Lee, "On the Evaluation of Boolean Operators in the Extended Boolean Retrieval Framework," *Proc. 16th Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval*, pp. 291-297, 1993.

[5] V.N. Anh and A. Moffat, "Pruned Query Evaluation Using Pre-Computed Impacts," *Proc. 29th Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval*, pp. 372-379, 2006.

[6] *Cochrane Handbook for Systematic Reviews of Interventions,* Version 5.0.2 [updated September 2009], J.P.T. Higgins and S. Green, eds., The Cochrane Collaboration, 2009, http://www.cochrane-handbook.org.

[7] L. Zhang, I. Ajiferuke, and M. Sampson, "Optimizing Search Strategies to Identify Randomized Controlled Trials in MEDLINE," *BMC Medical Research Methodology*, vol. 6, no. 1, p. 23, May 2006.

[8] M. Sampson, J. McGowan, C. Lefebvre, D. Moher, and J. Grimshaw, "PRESS: Peer Review of Electronic Search Strategies," Technical Report 477, Ottawa: Canadian Agency for Drugs and Technologies in Health, 2008.

[9] F. McLellan, "1966 and All that—When Is a Literature Search Done?," *The Lancet*, vol. 358, no. 9282, p. 646, Aug. 2001.

[10] S. Pohl, J. Zobel, and A. Moffat, "Extended Boolean Retrieval for Systematic Biomedical Reviews," *Proc. 33rd Australasian Computer Science Conf. (ACSC '10)*, vol. 102, Jan. 2010.

[11] T.G. Armstrong, A. Moffat, W. Webber, and J. Zobel, "Has Adhoc Retrieval Improved Since 1994?," *Proc. 32nd Int'l ACM SIGIR Conf. Research and Development in Information Retrieval*, pp. 692-693, 2009.

[12] D.W. Oard, B. Hedin, S. Tomlinson, and J.R. Baron, "Overview of the TREC 2008 Legal Track," *Proc. Seventh Text Retrieval Conf. (TREC)*, E.M. Voorhees and L.P. Buckland, eds., 2008.

[13] D. Metzler and W.B. Croft, "Combining the Language Model and Inference Network Approaches to Retrieval," *Information Processing and Management*, vol. 40, no. 5, pp. 735-750, 2004.

[14] T. Radecki, "Fuzzy Set Theoretical Approach to Document Retrieval," *Information Processing and Management*, vol. 15, no. 5, pp. 247-259, 1979.

[15] W.G. Waller and D.H. Kraft, "A Mathematical Model of a Weighted Boolean Retrieval System," *Information Processing and Management*, vol. 15, no. 5, pp. 235-245, 1979.

[16] C.D. Paice, "Soft Evaluation of Boolean Search Queries in Information Retrieval Systems," *Information Technology Research Development Applications*, vol. 3, no. 1, pp. 33-41, Jan. 1984.

[17] M.E. Smith, "Aspects of the p-Norm Model of Information Retrieval: Syntactic Query Generation, Efficiency, and Theoretical Properties," PhD dissertation, Cornell Univ., May 1990.

[18] H. Turtle and W.B. Croft, "Inference Networks for Document Retrieval," *Proc. 13th Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval*, pp. 1-24, 1990.

[19] A.Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J.Y. Zien, "Efficient Query Evaluation Using a Two-Level Retrieval Process," *Proc. 12th Int'l Conf. Information and Knowledge Management*, pp. 426-434, 2003.

[20] G. Salton and E. Voorhees, "Automatic Assignment of Soft Boolean Operators," *Proc. Eighth Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval*, pp. 54-69, 1985.

[21] J. Zobel and A. Moffat, "Inverted Files for Text Search Engines," *ACM Computing Surveys*, vol. 38, no. 2, p. 6, July 2006.

[22] A. Moffat and J. Zobel, "Self-Indexing Inverted Files for Fast Text Retrieval," *ACM Trans. Information Systems*, vol. 14, pp. 349-379, 1996.

[23] T. Strohman, H. Turtle, and W.B. Croft, "Optimization Strategies for Complex Queries," *Proc. 28th Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval*, pp. 219-225, 2005.

[24] M. Theobald, G. Weikum, and R. Schenkel, "Top-k Query Evaluation with Probabilistic Guarantees," *Proc. 30th Int'l Conf. Very Large Data Bases*, vol. 30, pp. 648-659, 2004.

[25] H. Turtle and J. Flood, "Query Evaluation: Strategies and Optimizations," *Information Processing and Management*, vol. 31, no. 6, pp. 831-850, 1995.

[26] M. Sampson, J. McGowan, E. Cogo, J. Grimshaw, D. Moher, and C. Lefebvre, "An Evidence-Based Practice Guideline for the Peer Review of Electronic Search Strategies," *J. Clinical Epidemiology*, vol. 62, no. 9, pp. 944-952, 2009.

[27] J.R. Herskovic, L.Y. Tanaka, W. Hersh, and E.V. Bernstam, "A Day in the Life of PubMed: Analysis of a Typical Day's Query Log," *J. Am. Medical Informatics Assoc.*, vol. 14, no. 2, pp. 212-220, 2007.

[28] A.M. Cohen, W.R. Hersh, K. Peterson, and P.-Y. Yen, "Reducing Workload in Systematic Review Preparation Using Automated Citation Classification," *J. Am. Medical Informatics Assoc.*, vol. 13, no. 2, pp. 206-219, 2006.

[29] H. Bast and I. Weber, "Type Less, Find More: Fast Autocompletion Search with a Succinct Index," *Proc. 29th Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval*, pp. 364-371, 2006.

Stefan Pohl received the master's of science degree from the University of Darmstadt, Germany, and was a visiting scholar at Cornell University, New York. He is a doctoral candidate in the Department of Computer Science and Software Engineering at the University of Melbourne, where he is currently completing a dissertation on effectiveness and efficiency of complex structured queries in the biomedical domain. His research interests include information retrieval, data mining, and machine learning.

Alistair Moffat received the PhD degree at the University of Canterbury, New Zealand, in 1986. Since then, he has been a member of the academic staff at the University of Melbourne, Australia, where he was appointed a full professor in 2002. His research interests include text and index compression, techniques for indexing and efficiently searching large document collections, and methodologies for evaluation of information retrieval systems. He is a coauthor of the books *Managing Gigabytes: Compressing and Indexing Documents and Images* (second edition with Morgan Kaufmann 1999); *Compression and Coding Algorithms* (Kluwer, 2002); and *Programming, Problem Solving, and Abstraction with C* (Pearson SprintPrint, 2003); and more than 150 refereed papers.

Justin Zobel received the PhD degree from the University of Melbourne and for many years was based at RMIT University, where he led the Search Engine group. He is a professor of computational bioinformatics at the University of Melbourne, and leads the Computing for Life Sciences activities within National ICT Australia's Victorian Research Laboratory. In the research community, he is best known for his role in the development of algorithms for efficient text retrieval, which underpin applications such as search engines deployed on the web. His research areas include search, bioinformatics, fundamental algorithms and data structures, plagiarism detection, compression, and research methods, and he is the author of a research methods textbook. He is an editor-in-chief of the *International Journal of Information Retrieval*, and an associate editor of *ACM Transactions on Information Systems*, *Information Processing & Management*, and *IEEE Transactions on Knowledge and Data Engineering*.