# A New Approach for Boolean Query Processing in Text Information Retrieval

**Leemon Baird and Donald H. Kraft**
**Department of Computer Science**
**U.S. Air Force Academy**
**USAFA, CO 80840**
**USA**
**leemon.baird@usafa.af.mil donald.kraft@usafa.af.mil**

**Special Session 3SS: The Application of Fuzzy Logic and Soft Computing in Flexible Querying**
**IFSA 2007 World Congress**
**Cancun, Mexico**
**June 2007**

## Abstract

The main objective of an information retrieval system is to be effective in providing a user with relevant information in response to a query. However, especially given the information explosion which has created an enormous volume of information, efficiency issues cannot be ignored. Thus, to be able to quickly process lists of documents that have the keywords stated in a given query assigned/indexed to them by merging via the Boolean logic of the query is essential in a Boolean query system. A new algorithm, based loosely on concurrent codes, is developed and discussed.

## Introduction

There is a need for good information retrieval, especially in this, the information age. Such systems need to be effective in providing relevant information without burdening the user with too much data. Thus, performance measures such as recall and precision help one determine how effective an information retrieval system is.

Information retrieval systems also need to be efficient, especially in terms of time. Efficiency considerations are vital in the information explosion era where huge volumes of information make it imperative to find the right information in a timely manner. Boolean query mechanisms are quite common, even today, in modern retrieval systems. Search engines such as Google and digital libraries such as the one at Stanford University [Stanford2006] provide such mechanisms. Users tend to use simple queries, but some users can and do incorporate Boolean logic connectives (e.g., and, or, and not) into their queries [Spink2001]. Thus, it behooves us to find the most efficient mechanism possible to search large collections of text documents to find those documents that satisfy Boolean queries.

In addition, some text retrieval systems have incorporated weights to indicate the importance of terms in describing document contents (indexing) and in terms of describing queries [Meadow2000]. These weights have been interpreted as coordinates for documents and queries in a vector space and as probabilities, but can also be interpreted as fuzzy set membership functions. This, too, will have an impact on efficiency, so that an efficient mechanism to process weighted Boolean queries is needed.

In addition, data mining is a field of increasing importance lately, including security issues such as identifying terrorists and keeping information secure. Such data

This work was sponsored in part by the Air Force Information Operations Center (AFIOC), Lackland AFB, TX, and was performed at the Academy Center for Information Security (ACIS) at the United States Air Force Academy.

mining often involves complex searches of extremely large databases that combine both structured data (e.g., relational databases) and unstructured data (e.g., text documents and emails). In unstructured document retrieval, the problem is often to find documents that contain certain keyword combinations, as described by a complex Boolean expression. In the structured case, the problem is generally to find records that contain certain attributes, as described by a complex Boolean expression. In both cases, the need for efficiency, in terms of both time and space (i.e., computer memory) is obvious.

**Background**

There has been but a little work of late in trying to improve the efficiency of query processing of Boolean queries using AND, OR, and NOT operators on indexed textual information retrieval systems. Consider the problem of retrieving documents in response to a query q = t1 AND t2, where t1 and t2 are given terms (e.g., t1 = "NLP" and t2 = "IR") that is posed to a collection of N documents. Suppose that an index has been set up as an inverted file [Meadow2000]. Suppose further that t1 has n1 documents with the tem t1 associated with them, while t2 has n2 documents with the term t2 associated with them, i.e., the lengths of the lists for terms t1 and t2 are n1 and n2, respectively. Suppose also that n3 the number of documents that have both terms t1 and t2 associated with them.

Zobel notes that to process query q, if the index is totally resident on the disk, a reasonable assumption, that one must fetch both lists which has O(n1+n2) time, while a straightforward merge would then cost an additional O(min(n1,n2)) [Zobel2006] . Zobel also notes that one could get O($\sqrt{n1}+\sqrt{n2}$) if one has random access to the lists and if the sum of the square roots of the list lengths is greater than n3 by employing skiplists.

Baeza-Yates [Baeza-Yates2004] offers a fast set intersection algorithm that assumes that the two lists of documents for each term are sorted. His algorithm involves a hybrid of binary search, as his original problem involves matching a query that is a multiset of terms against a document collection that is a larger multiset of terms. He arrives at a good average case by not inspecting all elements. Still, he is dealing with O(min(n1,n2) log (max(n1,n2))).

Work has been done on using comparison-based insertion and interpolation search for intersecting large, ordered sets [Barbay2006, Demaine2003, Demaine2001, Demaine2000]. More specifically, the authors look at Boolean query operators (union and difference, as well as intersection) for large sets of text records, using adaptive algorithms based on binary search.

The bottom line is that most results lead to algorithms with algorithmic complexity O(min(n1,n2)), while some hierarchical methods can yield O($\sqrt{n1}+\sqrt{n2}$), but with high overhead. We desire an algorithm that is linear, say O(n3).

**The Algorithm**

We employ a novel data structure and algorithm for speeding Boolean searches, optimizing for the case where the number of documents satisfying the query is small compared to the number of documents containing terms used in the query. The algorithm works for arbitrary Boolean queries, including arbitrarily-nested parentheses. The data structures are based on BBC codes, a recent development in concurrent code theory, a

new branch of coding theory that has been developed recently for a very different application (wireless jam-resistant communication) [Baird2007].

Assume every document is assigned a random, *n*-bit binary number as its identifier. For each possible search term, there will be a list of documents IDs containing that term. These are stored in a BBC structure, which is something like a Bloom filter. However, in a Bloom filter it is difficult to read out all the items stored in it. In a BBC structure, it is easy to do so. This difference allows the structure to optimize queries where the final set of satisfying documents is very small compared to the number of documents containing each term.

There is a separate BBC structure for each search term. The structure is a 1D array of bits. Initially, all bits are set to zero. Then each document ID is added to the structure. An ID is added to the structure by setting a bit to 1 in a location determined by the hash of each prefix of the document ID. For example, if the document ID is 11010 then the prefixes are {1, 11, 110, 1101, 11010). Hashing them gives the set {H(1), H(11), H(1110), H(1101), H(11010)}. Any hash function can be used, as long as it returns values in the range of $0\ldots2^m$-1, where *m* is the number of bits in the BBC structure. A simple approach is to use the first *m* bits of a SHA-1 hash or MD5 hash. In this example, 5 bits in the vector are set to 1, those in positions H(1) through H(11010). Contrast this with a Bloom filter, where only the bit in position H(11010) would be set. Figure 1 shows the structure that would be built for term A, assuming there are only two documents containing A, and their IDs are 11010 and 01100. The location of each 1 bit is chosen by a hash function. The locations corresponding to 11010 are written above the rectangle, and those for 10100 are written below it.

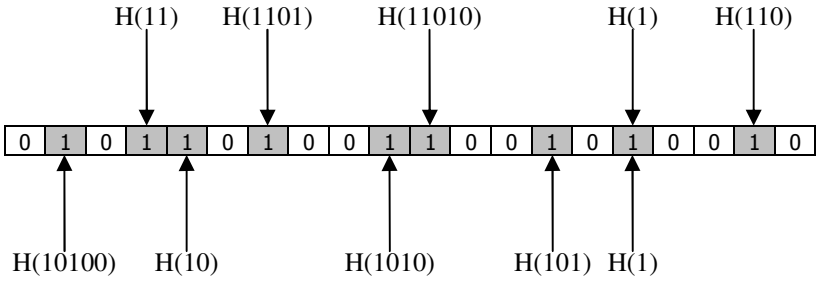


*Figure 1. A BBC structure for a given term, encoding the document IDs of all documents containing that term (in this example, 11010 and 10100).*

Once all the document IDs have been inserted into the BBC structures for all the search terms, it is possible to perform queries efficiently. For example, consider the query "(A OR B) AND NOT C". We will first determine whether any of the document IDs satisfying this query begin with a 0. This is done by performing an OR of the bit in position H(0) in the BBC structures that were built for terms A and B. The result is then ANDed with the NOT of the bit in position H(0) in the BBC structure for term C. If the result of this calculation is a 1, then we conclude that there is at least one satisfying document whose ID starts with 0. A similar check can be done for H(1). After performing these two checks, the *working set* of possible prefixes of document IDs

satisfying the query is either {}, or {0}, or {1}, or {0,1}. The process can then be repeated to find the second bit of each satisfying document ID. If the first step yielded a set with 0 in it, then the second step will check positions H(00) and H(01). If the first step yielded a set with 1 in it, then the second step will check positions H(10) and H(11). This can continue until all the bits of all the document IDs have been found. In other words, a search is performed on a tree such as that in figure 2.

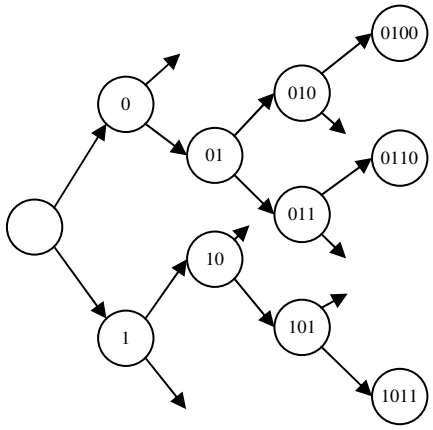


*Figure 2. A BBC decoding search that yields all documents that satisfy the query (in this case, documents 0110, 0100, and 1011).Upward branches add a 0, and downward add 1. Branches are pruned when the hash of a prefix doesn't satisfy the query.*

The leaves that occur in the far right layer are the IDs of exactly those documents that satisfy the query. Initially, the set of potential prefixes is empty {}. After one iteration, searching the second layer from the left, the set is {0, 1}, because the bits at position H(0) for each term satisfied the Boolean expression of the query, and so did the bits at position H(1). At the next iteration, for layer 3, the set is only {01, 10} because the bits at positions H(01) and H(10) for each term satisfied the query, but the bits at H(00) and H(11) did not. In the third layer, the set grows to three elements, because the prefix 01 could be expanded to both 010 and 011, but the prefix 10 could only be expanded to 101. It couldn't be expanded to 100 because the bits in position H(100) for each term didn't satisfy the Boolean expression of the query.

If the final solution has very few satisfying documents, then the set of possibilities will shrink very quickly, and very little time will be expended on the entire search. Note that the search pruning has an interesting holographic property. In a traditional search for a query such as A AND B AND C AND D AND E, the search might proceed from left to right, processing each term in order to successively shrink the set of possible documents to return. In the approach using the BBC structure, the search reveals progressively more bits of the document ID. But at every step, all of the variables in the entire query are taken into account. This means that if the final answer is going to be a small set, then there will be substantial pruning at every stage, not just at the end. The entire Boolean expression will help prune at every step.

An example of how this would work is an optimization of the query (Attack OR Bomb) AND Car. A BBC packet (signature) could be created that contains the document IDs of every document containing the word "Attack". A similar packet would be created for Bomb, and one for Car. These packets can then be combined bitwise. So the Attack

and Bomb packets would be combined with a bitwise OR, then the result would be combined with the Car packet with a bitwise AND. The resulting packet encodes all of the documents that satisfy the query. Previous methods would allow such a packet to be constructed and queried for whether it contains a given document. But now with BBC coding, we can also extract all of the documents contained in the final packet. In other words, this is an extension of traditional signatures, which actually allows all of the documents satisfying the query to be found efficiently. In addition, by using a depth-first search of the BBC decoding tree, it is very efficient to extract just a single document (or just *n* documents) from the set of satisfying documents, with an amount of computation that is independent of the number of documents in the set.

**Extending the Algorithm to Fuzzy Retrieval**

In classical information retrieval models involving Boolean queries, one assumes that text records have been indexed with terms such that term $t_j$ has or has not been assigned to document $d_i$. Thus, the "weight" of $t_j$ on document $d_i$ is either 1 (has been assigned) or 0 (has not been assigned). One can easily imagine an extension where these weights, rather than being in {0,1} can take on values in the interval [0,1]. One interpretation of these extended weights is to see them as fuzzy set membership functions, i.e., $d_i$ is in the subset of documents "about" the concept(s) represented by $t_j$ with a membership value if $w_{ij}$, the weight of $t_j$ on $d_i$. These weights might be generated according to any of many forms of relevance feedback [Kraft1999].

The BBC-based approach can be adapted for fuzzy queries as well. If each (document, term) pair has a fuzzy weight, then some system can be chosen for combining fuzzy values to obtain an overall degree that any given document satisfies the given query. For example, using traditional fuzzy operators, the weights for each term can be combined with a maximum, minimum, or 1-x operator (for OR, AND, and NOT, respectively). The goal then is to find the best document (or top *n* documents) according to this function.

There are several ways to modify the algorithm to work in this case. The simplest method is a two-stage process. First treat every variable as crisp, noting only whether a document contains a term, not how many times it occurs. Use the algorithm from the previous section to find the set of documents that satisfy the crisp query. The fuzzy value for each document in the set can then be calculated, and the best document (or best *n* documents) can be returned. In the case where the solution set is small, this calculation will be very fast.

If the result of the crisp query is a large set, then this simple approach may be unacceptably slow. In that case, there is a more sophisticated algorithm that can be much more efficient. It will only generate one document (or *n* documents) rather than generating the entire set resulting from the crisp query.

For this fuzzy search algorithm, it is necessary to store an additional data structure. For each term T and each possible prefix P of a document ID, it will store the $C_{min}(T,P)$ and $C_{max}(T,P)$, which are the minimum and maximum weight for that term in all documents whose IDs start with that prefix. So, $C_{max}(X, 011)$ for term X and the prefix 011 will store the count of how many times X appears in whichever document contains the most occurrences of X, from among all documents whose IDs start with the string 011. Similarly, $C_{min}(X, 011)$ will store the minimum.

Given the $C_{max}$ and $C_{min}$ tables, it is possible to optimize the search process through the BBC decoding tree, by performing a uniform-cost, best-first search [Russell1995]. At any given point in the search, there will be a set of prefixes of IDs that might satisfy the query. The one with the most promising fuzzy value should be expanded next. For example, if the query is "(A OR B) AND NOT C", and a given prefix is 011, then the maximum value that any document could have starting with 011 would be min(max($C_{max}$(A,011), $C_{max}$(B,011)), 1- $C_{min}$(C,011)). This was found by replacing OR with max, AND with min, and NOT with a subtraction from 1. The $C_{max}$ structure was used for both the non-NOTed variables, and the $C_{min}$ for the NOTed variable. This can be calculated for each prefix in the set, and the prefix with the highest value should be expanded next. For the crisp algorithm in the previous section, at any given time the working set always contains prefix strings of the same length. With this new approach, the working set will contain prefix strings of varying lengths, but which all have roughly the same maximum possible fuzzy weight.

Using this approach, the first time an entire ID is decoded, it's guaranteed to have the highest upper bound of all the IDs considered. As a heuristic, this document could simply be returned, and the search could end. But if it is desired to find the document with the highest possible fuzzy value, then additional work should be performed. First, the true fuzzy value for this document should be calculated. Then, any prefix in the working set that has an upper bound lower than this document should be deleted. Finally, the search should continue using all the remaining prefixes (if any). During the remainder of the search, a prefix can always be pruned if its maximum value is less than the best value already achieved. When no prefixes remain, the best answer discovered so far is guaranteed to be the best document possible for that query.

**Future Work - Testing**

In information retrieval research, one must go beyond algorithmic complexity and purely theoretical analysis by testing new algorithms against standard data. Such standard data must include textual documents, queries, and answer sets. The National Institute of Standards and Technology (NIST) has available such standard test data, known as the TREC or Tipster data collection, which would be employed to verify the improvements possible in the new algorithm. TREC collections vary in data type, query types, and data volumes. For example, some sample data sets include genomic, legal, and government oriented collections. Regardless of the data sets, they all provide document relevance judgments for each query listed. To demonstrate scalability, we plan in the future to employ the "Terabyte" collection. Although not actually a terabyte in terms of size (only 436 GB), efficiently and accurately processing this collection should demonstrate the potential of our approach.

**References**

[Baird2007] Baird, L. C., Bahn, W. L., and Collins, M. D., "Jam-resistant communication without shared secrets through the use of concurrent codes," U.S. Air Force Academy Technical Report USAFA-TR-2007-01, 14 Feb 2007.

[Baeza-Yates2004] Baeza-Yates, R., "A fast set intersection algorithm for sorted sequences," CPM 2004, Istanbul, Turkey, Springer Lecture Notes in Computer Science

[Baeza-Yates1999] Baeza-Yates, R. and Ribeiro-Neto, B. *Modern Information Retrieval*, England: ACM Press/Addison-Wesley

[Barbay2006] Barbay, J., Lopez-Ortiz, A., and Lu, T., "Faster adaptive set intersections for text searching," WEA 2006

[Demaine2003] Demaine, E.D. and Lopez-Ortiz, A. "A Lower Bound on Index Size for Text Retrieval," Journal of Algorithms, 48(1) pp. 2-15

[Demaine2001] Demaine, E.D., Lopez-Ortiz, A., and Munro, J.I., "Experiments on Adaptive Set Intersections for Text Retrieval Systems, *Proceedings of the 3rd Workshop on Algorithm Engineering and Experiments (ALENEX)*, Lecture Notes in Computer Science, Washington, DC, 2001.

[Demaine2000] Demaine, E.D. and Lopez-Ortiz, A. "Adaptive Set Intersections, Unions, and Differences," *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*

[Kraft1992] Kraft, D.H. and Buell, D.A., "Fuzzy Sets and Generalized Boolean Retrieval Systems," In Dubois, D., Prade, H, and Yager, R. (eds.), *Readings in Fuzzy Sets for Intelligent Systems*, San Mateo, CA: Morgan Kaufmann Publishers

[Kraft1999] Kraft, D.H., Bordogna, G., and Pasi, G., "Fuzzy Set Techniques in Information Retrieval," in Bezdek, J.C., Didier, D. and Prade, H. (eds.), Fuzzy Sets in Approximate Reasoning and Information Systems, vol. 3, The Handbook of Fuzzy Sets Series, Norwell, MA: Kluwer Academic Publishers, 1999

[Meadow2000] Meadow, C.T., Boyce, B.R., and Kraft, D.H., *Text Information Retrieval Systems*, second edition, San Diego, CA: Academic Press

[Grossman2004] Grossman, D. and Frieder, O., *Information Retrieval: Algorithms and Heuristics,* Second Edition, Springer Publishers

[Russell1995] Russell, S., Norvig, P., *Artificial Intelligence: A Modern Approach*, Prentice Hall, 1995.

[Sanchez2002] Sanchez, S.N., Triantaphyllou, E., and Kraft, D.H., "A Feature mining Based Approach for the Classification of Text Documents into Disjoint Classes," *Information Processing and Management*, 38(4), pp. 583-604

[Spink2001] Spink, A., Wolfram, D., Jansen, B.J., and Saracevic, T., "Searching the Web: The Public and Their Queries," *Journal of the American Society for Information Science,* 53(2), pp. 226-234

[Stanford2006] http://www-db.stanford.edu/~kevin/queryLanguage.html, October 16

[Zobel2006] Zobel, J. Personal communication. June