Experiments on Compressing Boolean Inverted Files by Document Ordering

Alexander Gelbukh [1], SangYong Han [2], Grigori Sidorov [1]

[1] Computing Research Center, National Polytechnic Institute,
Av. Juan Dios Bátiz, Zacatenco, 07738, DF, Mexico
{gelbukh, sidorov}@cic.ipn.mx; www.gelbukh.com

[2] Department of Computer Science and Engineering, Chung-Ang University,
221 Huksuk-Dong, DongJak-Ku, Seoul, 156-756, Korea
hansy@cau.ac.kr

ABSTRACT

Boolean queries are used to search a document collection for the documents that contain specific terms, independently of the frequency of a term in the document. To perform such queries, a search engine maintains an inverted file, which lists for each keyword the documents containing it. The size of such a file is comparable with that of the document collection, which is a considerable storage overhead. The inverted file can be compressed by reordering the documents in the collection in a specific way. We investigate the scalability of the efficiency of such compression with respect to the size of the collection and stemming settings.

Keywords: Information Retrieval, Boolean Search, Inverted File Size

1. INTRODUCTION

Information retrieval can be defined as the task of finding in a large natural language document collection those that satisfy some information need expressed by the user. The most common way of expressing such information need is a query consisting of a set of keywords. E.g., for the query *flu treatment*, the documents about flu treatment are selected from the collection at hand and presented to the user.

There are two major technical approaches to the task: the vector space model and the Boolean model [1]. In the vector space model, quantitative characteristics of the documents (such as the number of occurrences of the given term) and the keywords (such as the number of documents containing the given keyword) are taken into account to calculate the relevance of a retrieved document to the query, so that the user is first presented the most relevant documents. The greater the number of occurrences of the given keywords in a document the higher its relevance. This is especially useful when there are very many documents that contain the search term, which is usually the case with very large document collections such as Internet. However, the need to calculate and store this quantitative information produces a significant overhead on the retrieval system.

In the case of Boolean model, all documents that contain the given keywords are retrieved. The relevance of individual documents cannot be judged directly, since no frequency information is maintained.

Such a search strategy is adequate for relatively small specialized document collections, when the number of documents retrieved for a query is not very high. It is especially suitable for collections of small documents such as news headers, medical records, or abstracts. Such documents usually do not contain many occurrences of any word, so that the vector space model does not perform well on them and, in fact, is in effect reduced to the Boolean model (since most documents have at most one occurrence of any given keyword).

The advantages of the Boolean approach include:

- Simple implementation,
- Fast processing,
- Reduced storage overhead,
- Applicability of compression techniques (as the one described in this paper),
- Clear semantics,
- Logical expressions in the query.

The latter point refers to the possibility to specify queries with Boolean expressions, such as "*flu* AND (*treatment* OR *medicine*) AND NOT *aspirin*". Again, though this is not very relevant for general-purpose collections such as Internet, such queries are very useful for professionals working with specialized collections such as medical records.

The main disadvantage of the Boolean model is the lack of support for ordering the retrieved documents by relevance. However, under our assumptions (few documents retrieved for a query, lack of

J. Díaz de León, G. González, J. Figueroa (Eds.): Avances en Ciencias de la Computación, pp. 131-135, 2003.

statistical information in individual documents) the relevance can be judged by an independent module: first all documents satisfying the query are retrieved, and then their full texts are analyzed on the fly using statistical, linguistic, or other methods.

Hence Boolean model remains a popular search strategy suitable for many document collections.

Since such collections are likely to be distributed on CD-ROM or another media with restricted capacity, considerable effort has been devoted to reducing the storage overhead. One of popular methods applicable to the Boolean model (but not to the vector model) is blocking [1].

In this paper we present a method of compression of the internal structure used in Boolean search engines. Our method gives better results when the documents in the collection are unordered, so that we can choose a specific order of the documents. However, the method can be applied without this restriction. In the experiments we obtained about 7% of reduction in size as compared to the baseline.

The paper is organized as follows. Section 2 describes our improvement to the data structure representing the inverted file. Section 3 formalizes the problem of its compression. Section 4 explains the algorithm used for compression. Finally, Section 5 presents the experimental results and Section 6 draws the conclusions.

2. THE DATA STRUCTURE

In this paper we assume that all documents of the collection are available beforehand. Thus, our structures are optimized for storing and retrieval, but not for dynamically adding new documents.

To facilitate Boolean queries, the system uses an inverted file. It represents a $T \times D$ Boolean matrix $A = |a_{ij}|$, where T is the number of all different occurring in the documents of the collection (possibly except stop-words), D is the number of documents in the collection, and

$$a_{ij} = \begin{cases} 1 & \text{if } j\text{-th document contains } i\text{-th term,} \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

This very sparse matrix is stored in a format optimized for fast retrieval of the set of all documents containing a given term t.

Usually the matrix is stored as an array of the document numbers ordered by the terms, e.g.,

```
12, 17, 18, 19, 23, 24, 35
03, 07, 08, 12
01, 17, 78                    (2)
...
```

which means that, say, second term occurs in the documents number 3, 7, 8, and 12.

Note that the numbers of the documents are ordered in each line, which facilitates Boolean logic operations in case of a query formulated as a logical expression. To calculate an expression, say t_1 OR t_2, the sequences corresponding to t_1 and t_2 are simply merged. The complexity of this operation is linear in the total length of these two sequences.

However, such a data structure has the size comparable to that of the whole document collection, especially in the case of small documents where few terms are repeated in one document. Indeed, in this case the inverted file contains an element per nearly each occurrence of any word in any document.

In this paper we consider a possible way to reduce the size of such an inverted file.

Our idea is to represent contiguous intervals through the initial number and the number of additional documents in the interval. With this the data (2) will be represented as

```
12, 17+2, 23+1, 35
03, 07+1, 12
01, 17, 78
...
```

which means that, say, the second term occurs in documents 3, 7, one more (i.e., 8), and 12.

At the first glance, this representation is guaranteed to be more compact than (2). However, additional structure has its price. Indeed, in (2), the numbers can be allocated in fixed-length cells. Say, there are slightly less than 1000 documents, each number can occupy 10 bits. Note that under the sumption that the sizes of the documents are comparable, there is no point in variable-length cells store the numbers of the documents.

In case of representation (3), additional cost paid for maintaining such a more complex structure. Specifically, fixed-length storage is no longer possible (not to store zeroes for the majority of items do not represent non-trivial intervals); only nonzero interval lengths are stored explicitly.

3. THE PROBLEM

As we have seen, though joining contiguous runs document numbers into one interval results in items, each item now requires more bits to Thus, whether the representation (3) results compact than (2) depends on the number of tiguous runs in the rows of the matrix.

Reordering of the columns of the matrix change the total number of contiguous runs. means that reordering of the documents in the lection can result in more compact representation (3).

This does not imply any additional cost logical order of documents in the collection does matter, as is the case with many collections. ever, if the order really matters, an additional should be maintained to translate the document

dering used for (1) into the ordering in the actual collection.

In [2] we have shown that, though without re-ordering of the matrix the representation (3) proves to be worse than (2), it is possible to find such an order with which (3) results to be more compact, even taking into account the possible additional space needed for the translation array.

In this paper, we investigate the effect of the size of the document collection and stemming on the effectiveness of such compression.

4. THE ALGORITHM

In [2], two algorithms were suggested to find the near-optimal ordering.

Traveling Salesman Problem algorithm. Let us first consider a fixed-length storage scheme, so that both non-trivial intervals and individual numbers (we will call them trivial intervals) occupy the same number of bits (say, 15 bits), i.e.,

```
12+0, 17+2, 23+1, 35+0
03+0, 07+1, 12+0
01+0, 17+0, 78+0                    (4)
...
```

and find the optimal ordering of the documents in this case. Our experiments show that this order results to be near-optimal for the real scheme (3).

Assume that no term occurs in all documents. Consider also a minor improvement to our representation. Namely, consider the documents to be arranged in a circle instead of a sequence. Then, if a row of the matrix contains both last and first elements, they can be joined into an interval. This can be easily represented in the format (4): say, if the last interval in a row extends beyond the last column of the matrix, it is interpreted as including the first columns. E.g., if $D = 78$, the third row of (4) can now be written as

```
...
17+0, 78+1                          (5)
...
```

Though in practice there is no point to implement this detail (since it does not provide any significant improvement), we need it to simplify our reasoning.

The problem of minimizing the total number of items in the structure (4) with the improvement (5) can be recast, similarly to [3], as a Hamming-distance traveling salesman problem (TSP) [3], [5] over the complete graph K_D of documents thought of as sets of words. This problem can be formulated as follows. Consider the $D \times D$ matrix $L = |l_{ij}|$ of distances,

$$l_{ij} = \left|(d_i \cup d_j) \setminus (d_i \cap d_j)\right| \qquad (6)$$

i.e., the number of words in which the documents i and j differ. The problem is to find a path, i.e., an ordering $\sigma(i)$: $\{1, ..., D\} \leftrightarrow \{1, ..., D\}$, such that the total length between its adjacent nodes be minimal:

$$\sum_{i=1}^{D} l_{\sigma(i)\sigma(i+1)} \rightarrow \min. \qquad (7)$$

where $\sigma(D+1) \equiv \sigma(1)$.

Indeed, consider an ordering of the columns of the matrix. A word absent in a document but present in the next one corresponds to the beginning of a (trivial or non-trivial) interval in the corresponding row of the matrix; a word present in a document but absent in the next one corresponds to the end of such interval. Since the total number of beginnings and ends of the intervals is the same, the formula (7) expresses the double total number of items in (4).

TSP is a well investigated problem [3]; in particular, Hamming-distance TSP has been applied, for example, in molecular biology [5]. TSP is an *NP*-hard problem (i.e., *NP*-complete or harder), which means that no affordable exact algorithm to find its exact solution is known. However, many heuristics algorithms have been developed that give near-optimal solutions. In our experiments we tried, among others, the cheapest insertion algorithm:

1. Choose an arbitrary element d_0.
2. Choose an element d_1 closest to d_0 and form a round-trip path consisting of these two elements.
3. Repeat until all elements are inserted in the path:
4. Choose an element d_c not in the already constructed path and two adjacent elements d_a and d_b on the path such that the cost C of insertion of d_c between d_a and d_b be minimal.
5. Insert d_c between d_a and d_b in the path.

The cost of insertion here is

$$C = l_{d_a d_c} + l_{d_c d_b} - l_{d_a d_b}. \qquad (8)$$

A trivial direct implementation of this algorithm has the complexity D^3. However, it can be implemented with complexity $D^2 \log D$; we will not deep into details here. Since the calculation of the Hamming distance matrix is, to the best of our knowledge, D^2S, where S is the average number of words in a document, we conclude, given $S > \log D$, that the total complexity of our algorithm is D^2S.

Another possible class of algorithms for solving this problem is genetic algorithms.

Modified algorithm. In the previous section we assumed that the storage space does not depend on the interval length; in particular, that the space occupied by trivial and non-trivial intervals is the same, as in (4). Now let us return to the variable-length representation (3).

	Case		Configuration	Cost
1.	Adding a new single number		0 **1** 0	N
2.	Converting a single number into an interval		0 1 1 0 0 **1** 1 0	$M-N$
3.	Extending an interval		1 1 1 0 0 **1** 1 1 1 1 1	0
4.	Breaking an interval into:	two intervals	1 1 0 1 1	M
5.		an interval and a single number	0 1 0 1 1 1 1 0 1 0	N
6.		two single numbers	0 1 0 1 0	$2N-M$

Table 1. Cost of insertion of a document.

In this case the total cost of an ordering σ in question cannot be described by pair-wise distances l_{ij}, as in (8). Fortunately, most techniques used to solve TSP-like problems—among those the algorithm described above, as well as genetic algorithms—do not heavily rely on the exact form of the expression (8). It is enough for C to depend on a limited number of elements around the insertion point; the complexity of the algorithm is not affected. Though the theorems on worst-case behavior of the algorithm applied to TSP may not hold in this case, we expect that the quality of the results is comparable to that of the standard TSP case.

Let us construct a more precise expression for C. Assume that trivial intervals occupy N bits and non-trivial ones M bits (though our considerations can be generalized to a variable-length representation of non-trivial intervals). When a document is inserted into an existing path, for each row of the structure (3) we should distinguish the cases described in the left-hand column of Table 1 and add the appropriate cost specified in the right-hand column of the table.

For example, let us insert a document containing only the 2nd and 3rd words into the collection (3) between 17th and 18th documents (and renumber the rest of the documents so that the 18th becomes 19th, etc.). The new collection is represented as follows:

```
12, 17, 19+1, 24+1, 36
03, 07+1, 18, 12
01, 17+1, 79                    (9)
...
```

Indeed, the new document broke an interval in the first line (case 5 in Table 1), added a single number in the second line (case 1) and converted a single number into an interval in the third line (case 2). Thus, the total insertion cost is $C=N+N+(M-N)=2N+M$.

For each word—a row of (3)—the cases can be distinguished by the configurations shown in Table 1. Here, 1 stands for presence of the word in the document and 0 for absence. The figure in bold stands for the document being inserted; the others stand for the left and right neighbors of the insertion point, accordingly. As we see, it is enough to consider at most two documents to the left and two to the right the right of the insertion point. Note that the cases of only 1 and 2 elements in the already constructed path (where there are less than two neighbors of the insertion point) are handled separately (we omit here the details).

5. EXPERIMENTAL RESULTS

For our experiments we used the abstracts extracted from the Cystic Fibrosis standard document collection as available from [6]. The collection contains $D = 785$ abstracts, of the total size of 700 KB, ranging from 0.1 to 3.3 KB, with average size of 0.9 KB. As we have mentioned in Section 1, Boolean search model is adequate for such short documents.

We indexed all alphanumeric sequences in the documents, which amounted to a total of 680 KB, and converted all letters to uppercase. After removing the following most frequent stop-words: *of, the, in, and, with, to, a, was, were, for, is, from, that, by, be, this, or, as, these, an, not, on, are, than, have, no, at, had, it, may, I, been, which, between, one, but, two, 2, has, both, more, all, other, there, 5, 3, their, also, 0, those, only, after, 4, when, three, who,* the collection reduced to 530 KB. No stemming was applied.

This gave $T = 7856$ unique words for the whole collection. The resulting inverted file (2) contained 47314 elements. Then we compressed the inverted file to the representation (3), reordering the documents using different algorithms.

We have experimented with sub-collections of different sizes to estimate the scalability of the method.

Also, we experimented with original files as well as with stemmed words, i.e., the words like *government, governor, governmental*, etc. collated to the same stem *govern-* and thus occupying only one line in the inverted file matrix. We used the standard Porter stemmer.

Since our experiments are still in progress, the actual experimental results will be given in detail in the final version of this paper, though we do not give them in this Extended Abstract. What we can say already for sure is that the experiments show that the method is

robust both with respect to the size of the collection and with respect to stemming.

6. CONCLUSIONS

In [2] we have reported an algorithm for compressing the Boolean matrix representing the inverted file of a Boolean search engine for a static collection. The matrix can be compressed up to 8% by the combination of two factors: (1) a storage format that represents the contiguous runs of 1s in the matrix as one unit and (2) a specific ordering of the columns (i.e., of the documents in the collection), if needed without loss of information about the original order.

In this paper we have experimentally shown that the method is robust with respect to both the size of the collection and stemming.

References

[1] R. Baeza-Yates, B. Ribeiro-Neto: Modern Information Retrieval. ACM Press/Addison-Wesley 1999.

[2] A. Gelbukh, S.Y. Han, G. Sidorov. Compression of Boolean Inverted Files by Document Ordering. IEEE NLPKE-2003, to appear.

[3] G. Cohen, S. Litsyn and G. Zémor, On the traveling salesman problem in binary Hamming spaces, IEEE Trans. Info. Theory, vol. 42, 1996, pp. 1274–1276, citeseer.nj.nec.com/cohen96traveling.html.

[4] D.J. Rosenkrantz, R.E. Stearns, and P.M. Lewis II. An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing*, 6(3):563-581, 1977.

[5] Alizadeh, F., Karp, K., Newberg, L., and Weisser,D. (1993). Physical mapping of chromosomes: A combinatorial problem in molecular biology. In Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA), 371-381. ACM Press.

[6] Cystic Fibrosis text collection, cse.hanyang.ac.kr/~jmchoi/class-old/2001-1/ir/homeworks.html.