Vector Query with Signature Filtering

Pavel Moravec  
Department of Computer Science, VSB-TU Ostrava, Czech Republic  
pavel.moravec@vsb.cz

Jaroslav Pokorny  
Department of Software Engineering, Charles University, Prague, Czech Republic  
pokorny@ksi.ms.mff.cuni.cz

Vaclav Snasel  
Department of Computer Science, VSB-TU Ostrava, Czech Republic  
vaclav.snasel@vsb.cz

Abstract

In recent years, many approaches to filter documents before query evaluation were adopted for optimising query processing over a text data collection. We investigate a mixture of signatures and vector querying in our approach. We present a number of approaches of filtration before the vector query execution. We use the concept of weight-partitioned signature files and propose an efficient method for signature file traversal by S-trees. We verified experimentally that usage of S-trees leads to the decreased number of signature comparisons. The experiments have been done on real text collections.

1. Introduction

Large textual database are increasingly popular now not only in a digital library environment but also with the growth of the Internet and technologies, which are useful for information search and retrievals on the Web. The number of users, and thus queries, handled by an information retrieval system is potentially huge, and the system has to process queries in a reasonable time. The efficiency of the filtering process is thus an important issue.

In recent years, many approaches to filter documents before query evaluation were adopted to optimizing query processing. It means that the query evaluation is split into two phases. One category of these methods uses signature files [Faloutsos 1990] as an efficient filter to eliminate documents, which do not match a submitted Boolean query (especially the most frequent conjunctive one). In the vector model, a modification of signature filtering is needed. In this paper, we propose a two-step filter-and-refine query processing strategy based of a combination of signatures and the vector model.

In sections 2 and 3, we will discuss signatures and a modification of B-tree (so called S-tree [Deppisch 1986]) designed to store them. In section 4, we mention the vector model and calculation of term weights of processed documents. Section 5 discusses the possibility to combine both signatures and vector model, namely so called weight-partitioned signature files proposed by Lee and Ren [Lee 1996]. Then we offer a faster traversal method of weight-partitioned signature files by using S-trees instead of sequential access and experimental results to confirm improved speed on original test data. Finally, we summarize the approach and point out further research issues.

2. Signatures

Signatures can be used as a quick filter to eliminate some non-relevant documents before final query evaluation. A signature is a bit vector of \( F \) bits, where \( F \) is called the signature length. They are used to record presence of terms in a document. Each term can be encoded into a signature using hashing function to set \( m \) bits to 1. Number of bits set to 1 is called the signature weight.

The most common method for generating signatures is superimposed coding – signature of given document (or text block) is generated by ORing signatures of words contained in the block (see Figure 1).

When a query is evaluated, the query signature \( S_Q \) is compared with the documents’ signatures \( S_{Di} \). A match with document \( i \) is found, if and only if \( S_{Di} \land S_Q = S_Q \).

<table>
<thead>
<tr>
<th>Word</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>000 001 110 010</td>
</tr>
<tr>
<td>block</td>
<td>100 101 100 000</td>
</tr>
<tr>
<td>Block signature (v)</td>
<td>100 101 110 010</td>
</tr>
</tbody>
</table>

**Figure 1.** Superimposed signature

Because usual hash functions and the superimposing process are not uniquely invertible functions, some signatures can be matched even they are not relevant to the query. They are referred to as false drops. The...
probability of a false drop is an important factor for signature files, because it is quite expensive to eliminate false drop – selected documents must be searched by a different method (e.g. pattern matching) to do so. Roberts inferred an approximation for optimal signature length and term signature weight [Roberts 1979].

3. S-Tree

Uwe Deppisch introduced in [Deppisch 1986] a balanced tree structure for storing signatures. S-tree is a modification of $B^*$-tree and is able to store, insert and delete objects (or their references). Each object is stored with its corresponding signature in S-tree’s leaf nodes. The inner nodes contain signatures too. They are computed for every child node by superimposing its stored signatures.

Every S-tree (of the type $(K, k, h)$) must satisfy following conditions:
- All paths from root to leaf node have the same length $h$ (height),
- the root has at least 2 and at most $K$ sons, otherwise it must be a leaf node,
- every non-root node contains at least $k$ and at most $K$ sons.

While $k$ must be equal to $K/2$ in $B^*$-tree, in case of S-tree is valid any $k$ which satisfies condition $1 \leq k \leq K/2$. An example of S-tree is depicted in Figure 2.

In case of insertion, an appropriate leaf node is selected by descending into nodes where superimposing inserted object’s signature causes the least weight increase. After insertion, all signatures on insertion path are updated by superimposing object’s signatures. When the leaf node cannot accommodate another object, a node split occurs. Object is virtually inserted and some objects from the leaf node are moved to the newly created node, namely object $\beta$ with the highest weight increase against object $\alpha$ with the lowest weight in the original node; and all objects, which have the lower weight increase against $\beta$ than $\alpha$. Minimal number of sons $k$ must be preserved even it breaks last rule. If the parent node cannot accommodate child nodes, the split propagates upwards. If the root is to be split, a new root is created.

By deletion is chosen object removed from leaf node and corresponding signatures are recomputed. If the leaf node contains after deletion less than $k$ nodes, it is disposed and remaining nodes are reinserted at appropriate level.

When an object is searched for, its signature is compared with all signatures in node and the search recursively continues in nodes whose signatures satisfied condition $S_D \land S_Q = S_Q$. In the end, all objects matching given query are returned. Tousidou et al. introduced in [Tousidou 2002] improved page splitting techniques, which reduce query response time.

4. Vector Model

In the vector model, each document $D$ is represented as a vector in $n$-dimensional space,

$$D = (w_{D_1}, w_{D_2}, \ldots, w_{D_n})$$

where $n$ stands for the number of terms in all documents contained in document collection. A non-zero weight is assigned for every term in document. Accordingly we define query vector as

$$Q = (w_{Q_1}, w_{Q_2}, \ldots, w_{Q_n})$$

To determine how a document is “similar” to the given query, we define a similarity coefficient as

$$sim(Q, D) = \Sigma_{i=1..n} w_{Qi} \cdot w_{Di}, \text{ or}$$
sim(Q, D) = \frac{\sum_{i=1}^{n} w_{Qi} w_{Di}}{\sqrt{\sum_{i=1}^{n} w_{Qi}^2 \sum_{i=1}^{n} w_{Di}^2}}

First alternative is usually used when \( w_{Qi}, w_{Di} \in \{0, 1\} \). The second formula (called the cosine measure) is used more frequently, because the result is normalized. Document is considered to be relevant, if the computed similarity coefficient exceeds a given threshold.

Term weights \( w_{Qi} \) and \( w_{Di} \) can be computed in many different ways. According to Salton and Buckley [Salton 1988], every term weight has three basic components, whose multiplication forms term weight. They are

- **term frequency component** stands for the frequency of term in the current document,
- **collection document frequency component** depends on the number of documents which contain a specified term,
- **normalisation component** specifies, if the document vector is normalized and in which way.

Salton and Buckley present in their paper some possible values for each component, which are summarized in tables 1-3.

### Table 1. Term frequency component

<table>
<thead>
<tr>
<th>Component</th>
<th>Formula</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b )</td>
<td>1</td>
<td>term frequency is ignored</td>
</tr>
<tr>
<td>( t )</td>
<td>( tf )</td>
<td>raw term frequency – no. of times each term is present in document</td>
</tr>
<tr>
<td>( n )</td>
<td>( \frac{1}{2} \left( 1 + \frac{tf}{\max {tf}} \right) )</td>
<td>augmented normalized term frequency</td>
</tr>
</tbody>
</table>

### Table 2. Collection frequency component

<table>
<thead>
<tr>
<th>Component</th>
<th>Formula</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>1</td>
<td>collection frequency is ignored</td>
</tr>
<tr>
<td>( f )</td>
<td>( \log \frac{N}{n} )</td>
<td>where ( N ) stands here for total count of documents; ( n ) for number of documents term is assigned to.</td>
</tr>
<tr>
<td>( p )</td>
<td>( \log \frac{N-n}{n} )</td>
<td>probabilistic inverse document frequency</td>
</tr>
</tbody>
</table>

### Table 3. Normalisation component

<table>
<thead>
<tr>
<th>Component</th>
<th>Formula</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>1</td>
<td>normalisation is not applied</td>
</tr>
<tr>
<td>( f )</td>
<td>( 1/\sqrt{\sum_{vector} w_i^2} )</td>
<td>cosine normalisation</td>
</tr>
</tbody>
</table>

Since it is expensive to compute \( \sum_{k=1,n} (w_{Di}^2) \) it was suggested in [Lee 1994] to replace it with square root of the number of terms in document.

5. **Combining Vector Queries with Signatures**

Although the signatures had been studied intensively in last two decades, their use with vector queries was researched only marginally. The major problem concerning signatures in the vector model is that not all terms used in a query have to appear in relevant document. This means, that condition \( S_{Di} \land S_{Q} = S_{Q} \) is unusable, because relevant documents would be filtered out.

Other possibility is to expect that both query and document must have at least \( k\% \) of signature length or weight in common. However this does not ensure that relevant documents will not be filtered out and in case of both \( tf \) and \( idf \) components and signatures with uniform term weight (either unit or given) relevance drops quite quickly as the common portion increases while number of false drops remains rather high. Better results may be reached by involving weight from document vector into signature weight. An experiment was done to support this proposition and its results are shown in Figures 4 and 5.

To ensure 100% relevance ratio, a different method must be used. One possibility is searching the signature file for signatures of all terms in given query. While collection frequency component for every term can be stored and added to document’s score in corresponding document and normalisation can be simplified (as mentioned above), it is hard to store term frequency component for every term and document in the signature file.

Lee and Ren suggest in [Lee 1996] a concept of weight-partitioned signature file. For every term frequency (TF) group, which consists of terms with term frequency in a given interval, a separate signature file is created. The lengths of signatures must be precomputed to minimise false drop probability in signature files with higher term frequency. When evaluating a query, for every document a term-by-term search is executed, traversing TF groups either from TF group with highest term frequency (so called H-L method) or vice versa (L-H method). Once a term signature is found in one partition, its weight is computed and added to document score and next term can be searched for. However, if the searched document does not contain this term at all, all document’s signatures will be unsuccessfully searched. Because TF groups with lower frequency have higher probability of false drop, precision of H-L method is better than in case of L-H one.
5.1 Improvement of Weight-Partitioned Signature File

The original method used with weight-partitioned signature files has one major disadvantage – almost all document signatures must be compared with each term signature. In fact, if a queried document does not contain this term and there will be no false drop, then all document signatures in every partition will be used.

A simple test showed that using optimal signature files configuration and test file from [Lee 1996] the total reduction of the number of compared signatures is not more than 8%. An improved search technique should be used to archive higher reduction ratio. S-trees are an example of such technique, because many branches of S-tree are automatically skipped when executing a search. On the other hand, S-tree structure imposes higher overhead and more disk space is needed. We have decided to create an S-tree for every TF group, searching all documents for a given term at once. It may be possible to create an S-tree for every document and every TF group, and use it with the original algorithm, but an additional overhead for its parameters would be needed. One limitation should be taken in account. S-trees perform better, when the query signature weight is at least 10% of block signatures. So when we choose the maximal number of superimposed signatures per block as described in the original method, then values below 10 are desirable. In fact, this will be no problem, even authors suggest low numbers not exceeding round up of the average number of terms per document in TF group.

To eliminate inclusion of term weight more than once (because of a false drop), a blank bit vector with the number of bits corresponding to the number of documents can be created. Once a term is found in document, 1 is recorded on the corresponding position, so that the next time term will be skipped. When moving to a new term, all values will be reset to 0. Even for 1 million documents this vector would need cca 122 Kbytes of memory, so the overhead is acceptable. This allows us to get the same results as with the original search algorithm. An overall architecture of such modified retrieval system is in Figure 3.

5.2 Test Results

To ensure a uniform distribution among all bits of signatures, table quazigroups [Dvorsky 2002] were used. In the first test, we have compared usage of classical signatures together with a vector query. A collection of RFC 1-2040 (RFC stands here for “Request for Comment”; these documents published by IETF are de-facto Internet standards, describing for example internet protocols like TCP, UDP, HTTP, etc.) was indexed using both tf and idf factors and three sets of superimposed document signatures with the length of 2048 bits were generated.

In the first set, a unit weight was assigned to every term signature. In the second one, signatures with uniform weights \( m=4 \) were generated. In the third set, signatures with 0–5 bits were created according to the normalized term weight. (In case the computed term weight was too low, the term was skipped).

Relevance and precision was computed relatively to the result of the vector query. The cosine measure was used and the threshold for relevant vector query results was chosen to be 0.3. As one can see in Figures 4 and 5, the number of false drops in case of classical signatures stays quite high even if the relevance factor is low. Weight-generated signatures seem to gain better results: between 15-20% of signature size false drops ratio falls quickly, while the relative relevance still remains high. In the second test classical weight-partitioned signature files were compared with their S-tree modification under the same test conditions as described in the original article. Classical S-trees were implemented; better page-splitting techniques and parallelism were not taken in account.

A TREC subset of 10,000 Wall Street Journal articles with raw size of 27 MB was used. 30 TF groups for term frequencies from 1 to 30 were created. The optimal signature file configuration for document signatures that occupy ca. 25% of the original collection size was chosen and appropriate signature sizes and term weights computed according to Theorem 2 in Lee’s paper [Lee 1996].
Ten queries were executed and the number of signature comparisons was recorded and compared with the original Lee’s algorithm. Under this configuration, S-trees were always better than original H-L method, with average reduction factor between 24.6% in TF<sub>1</sub> group and 83% in TF<sub>30</sub> group. In Figure 6 average results are recorded together with the worst and best reduction values from every TF group.

6. Conclusions and future work

We present a number of approaches of filtration before the vector query execution. We have used the concept of weight-partitioned signature files and proposed a better method for TF group signature file traversal than it is known from literature. We verified experimentally that usage of S-trees leads to the decreased number of signature comparisons. Further improvement is, however,
possible by replacing page-split method in S-tree with a better variant or by increasing page size and involving parallel processing of S-tree.

A different technique might be however used when we are unable to identify original idf and tf components, like when we obtain existing term-document matrix or in case of decreased dimension of document vector, e.g. after a random projection [Bingham 2001] or computation of LSI [Berry 1999].

7. Acknowledgement

This research was supported in part by GACR grant 201/03/0912.

8. References


