Exploiting Progressions for Improving Inverted Index Compression

Christos Makris and Yannis Plegas
Department of Computer Engineering and Informatics, University of Patras, Patras, Greece

Keywords: Inverted Index, Compression, Arithmetic Progression, Interpolative Coding.

Abstract: In this paper, we present an algorithmic technique for compressing the document identifier lists of an inverted index which can be combined with state of the art compression techniques such as algorithms from the PForDelta family or Interpolative Coding and attain significant space compaction gains. The new technique initially converts the lists of document identifiers to a set of arithmetic progressions; the representation of an arithmetic progression uses at most three (!) numbers. This process produces an overhead which are the multiple identifiers that have to be assigned to the documents so we have to use a secondary inverted index and an extra compression step (PForDelta or Interpolative Coding) to represent it. Performed experiments in the ClueWeb09 dataset depict the superiority in space compaction of the proposed technique.

1 INTRODUCTION

An inverted index (also referred to as postings list or inverted file) is a data structure that is typically employed in Information Retrieval systems in order to index the terms (words) that are contained in a set of documents (Witten et al., 1999); (Baeza-Yates and Ribeiro-Neto, 2011). In the present paper we focus on inverted indices, where the documents are web pages. Traditionally an inverted index consists of all the different terms that exist in the processed pages and a list for each term that contains the identifiers of the pages where the term appears. A reference to an instance of a term in a web page in an inverted list is usually represented by a numerical identifier that uniquely identifies the page.

Since the size of an inverted index, if stored uncompressed, is of the same magnitude to the size of the whole collection, a great deal of work has been performed in order to compress the inverted index (Witten et al., 1999); (Zobel and Moffat, 2006); (Baeza-Yates and Ribeiro-Neto, 2011). The majority of the proposed techniques try to compress the inverted lists by storing the document identifiers in ascending order and compressing the gaps between successive values. In the present work we approach the problem of compressing inverted files from a different perspective, that we believe could shed new light on how space efficiently we can compress. We eliminate the long list of document identifiers that must be stored for each inverted list, by replacing it by a triple that represent an arithmetic progression; in order to achieve this we may be forced to assign extra identifiers to the documents and we handle these by using a *secondary* inverted index. An important advantage of our method is not only that we store a percentage of the initial identifiers but also that we may apply transparently previous inverted file compression methods in order to achieve better performance. A probable disadvantage of our technique is its decompression performance, however we feel that our technique is quite significant if using it for representing cached inverted lists, or the main memory component of a multi-tiered index (Zhang et al., 2008), (Ntoulas and Cho, 2007).

Before continuing we should note that an arithmetic progression (or numerical sequence) is a sequence of numbers such that the difference between any two successive members of the sequence is a constant. An arithmetic progression can be stored and retrieved using only three numbers the first element, the step between two consecutive elements and the number of elements in the sequence. The proposed method uses only sequences with step value one so it is not necessary to store the step of the sequence. Also the algorithm uses a base value which guarantees that the documents get different and uniquely decompressable identifiers. Consequently the algorithm needs only three

Makris C. and Plegas Y..
Exploiting Progressions for Improving Inverted Index Compression.
DOI: 10.5220/0004365402510256
In *Proceedings of the 9th International Conference on Web Information Systems and Technologies* (WEBIST-2013), pages 251-256
ISBN: 978-989-8565-54-9

numbers to store the sequence.

Our goal is to turn the lists of document identifiers in numerical sequences that represent arithmetic progressions and hopefully reduce the number of values that represent each initial inverted list to three (!) numbers per list. The cost we have to pay for this dramatic decrease of space is that we have to assign multiple extra identifiers to the documents. So our goal is to maximize the number of documents that do not have to take extra identifiers. The extra document identifiers are maintained by a secondary index that assigns to each original document identifier a list with the extra identifiers. Hence, we have to handle and compress these extra identifier values efficiently. Our technique is accompanied with a decompression algorithm that takes as input the compressed inverted index and returns as output the original inverted file.

In section 2 we describe the related work, in section 3 we present the main idea behind our algorithm, while in section 4 we present the decompression technique. Then the section 5 contains the experimental evaluation, and finally in section 6 we conclude with open problems and directions for future research.

2 RELATED WORK

There are many methods trying to compress inverted indexes. The most common methods use the gaps between document identifiers to represent values with fewer bits. In particular a set of well known methods, for compressing the inverted files are the Unary code, the γ-code, the δ-code and the Golomb code (more details for these methods appear in (Witten et al., 1999)). Concerning more recent methods, the most successful at this time are considered to be the *Interpolative Coding* (Moffat and Stuiver, 2000) and the *OptPFD* method in (Yan et al., 2009) that is an optimized version of the techniques appearing in the compression schemes described in (Heman, 2005); (Zukowski et al., 2006) and that are known under the umbrella of *PForDelta* (*PFD*) family.

The Interpolative Coding (IPC) compresses a sequence of ascending numbers by firstly encoding in binary the middle value of the ascending sequence and then by recursively encoding the left and the right interval by using the boundary values of the intervals to be coded, in order to reduce the number of used bits. The PFD firstly codifies using b bits, the values that are less than 2^b for a user defined threshold b, and then it codifies separately larger values that are called *exceptions*. The PFD works especially well if it is applied to blocks of bits of 32-bit values and patches the results. In (Yan et al., 2009) two extensions of the PFD methods were presented. One that employs more efficient coding procedure for the exceptions by storing more efficiently the involved offset values termed *NewPFD*, and another one that instead of selecting a given threshold for each exception, it judiciously selects the best value of each block by modeling it as an optimization problem, termed *OptPFD*. OptPFD is considered to be the more successful from the two variants.

Concerning enhancements of the compression techniques, there have appeared lately a plethora of methods that try to improve the compression performance by suitably rearranging the assignment of identifiers to each inverted list, see for example (Yan et al., 2009); (Ding et al., 2010). There are also methods making pruning-based performance optimization (Ntoulas and Cho, 2007); (Zhang et al., 2008).

In our constructions and following the findings of the various works mentioned in the bibliography we determined to choose as representative schemes with which to compare and combine in our technique IPC, and OptPFD that are considered to be amongst the best in the state of the art (Yan et al., 2009); (He et al., 2009).

3 CROPPING INVERTED LISTS

The inverted file represents each document with a document identifier called in our algorithm *original* identifier. As mentioned briefly above, our algorithm tries to condense the representation of each inverted list of a term, by a triple of numbers; we call the set of all these triples produced for all terms the *transformed index* representation. To achieve this we assign to the documents multiple extra document identifiers which called *extra* identifiers. The *extra* identifiers for each document constitute the list of extra identifiers for the document. The lists of extra identifiers for all the documents create the *secondary index* of extra identifiers. Below we analyze the process we follow to make the representation complete and reversible; the procedure is applied to each term separately moving from the first to the last in lexicographic order.

In subsections 3.1 and 3.2, we present the core idea of our algorithm while the subsection 3.3

contains a brief reference to simple heuristics for the original identifier assignment.

3.1 Core of Our Idea

For each inverted list, we use its size that is the number of documents contained in it, to define the range of the interval in the arithmetic progression that we seek. That is in the best of the scenario all the documents should have identifiers that are consecutive numbers in an interval with range equal to the size of the initial inverted list and this because in our case the difference between two successive values in a progression should be one.

Since the inverted list of a term can span multiple intervals of values of this range, we have to consider the range of values that are formed in each inverted list by the original identifiers of the documents and try to locate an interval of values that contains the more document identifiers and hence it is more suitable for being the arithmetic progression that we seek. The explanation of why we think the size of the inverted list is so important is that, as we move through the chosen interval in the identifiers of the inverted list, the original document identifiers that are outside this interval of values designate documents that should take extra identifiers; these extra identifiers should be added to fill the gaps between the values that exist in the size interval in order to form the desired arithmetic progression.

3.2 Selection of the Optimal Interval

Our goal in the conversion of inverted lists to numerical sequences is to use the smallest possible number of extra identifiers. The number of these extra identifiers used in our representation is equal to the number of original identifiers that are outside the selected interval. Hence selecting the interval containing the largest number of original identifiers minimizes the number of extra identifiers that are to be used.

For example, in Figure 1, we can see the process for selecting the optimal interval for an inverted list with ten documents. The algorithm selects the interval [1-10] hence it contains the maximum number of original identifiers. To verify what we said above, we see that in interval 1 [1-10], there are contained the documents 1,3,5,6,9,10 and outside this interval there are contained the original identifiers 12, 14, 15, 20. From interval 1, the missing values in order to form an arithmetic progression are 2, 4, 6 and 7. Therefore, the documents whose identifiers exist outside the interval (12, 14, 15 and 20) should get extra identifiers to represent the missing values (2, 4, 6 and 7) as we can see in Figure 2.

These extra identifiers should somehow be stored in a reversible way. Since we want to store them in an inverted like index where each original identifier will be associated to a list of extra identifiers, we need to assure that as we process each term from the smallest to the largest lexicographically, the extra identifiers that are assigned to the same original identifier are in ascending order.

We achieve this by introducing, when processing the inverted list of a term the so called *base number*; this guarantees that the range of values that will be assigned to each term, in the transformed index, are disjoint and in a total order according to the lexicographic ordering of the terms. Initially and for the first term in the inverted index the base number is equal to 0. The base number when processing the inverted list of a specific term will be equal to the last value of the previous inverted list, after this it is transformed into an arithmetic progression; that is, it will be equal to the upper bound of the progression *plus* the base value designated to the previous list.

Using this base value we define and store in the secondary inverted list as extra identifier the base value of the term in whose inverted list the document appears, *plus* its assigned missing value in the numerical sequence. Consequently, for each term, the lower limit of the interval which contains the extra identifiers values in the secondary index is the base number plus the initial value and the upper bound is the base number plus the initial value plus the size of the sequence. Hence the stored extra identifiers corresponding to the same original document identifier will be distinct and ascending as we process the terms, and thus form an eligible sequence of values for being stored and compressed in the secondary inverted index.

Figures 1 and 2 use small inverted lists in order to provide more comprehensible examples. Typically, the inverted lists are very large, as those that we will employ in our experiments. Finally Figure 3 initially contains the final representation of the compressed inverted index for the two terms. The compressed inverted index consists of the two term triplets and their secondary index.

3.3 Assignment of the Original Document Identifiers

The results of our method can be improved by intervening in the assignment of original identifiers to documents, in a way that will permit more

efficient compression using arithmetic progressions, in comparison to a random assignment. It could be possible to employ the plethora of the techniques that have been applied in the relevant literature for identifier assignment to documents, see for example (Yan et al., 2009); (Ding et al., 2010); however we have chosen to implement some simple heuristics that are fast and give quite satisfactory results. We will not expand further on this section to focus on the main idea of this work.

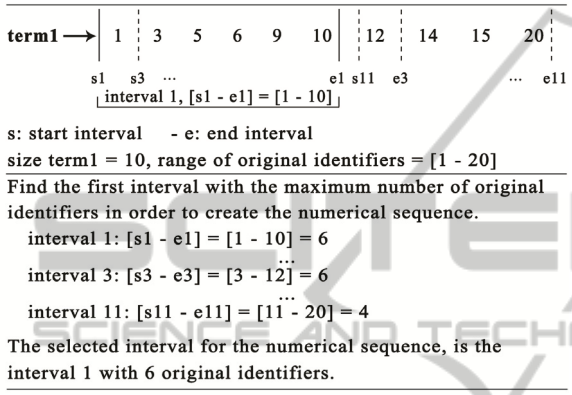

Figure 1: Optimal interval selection.

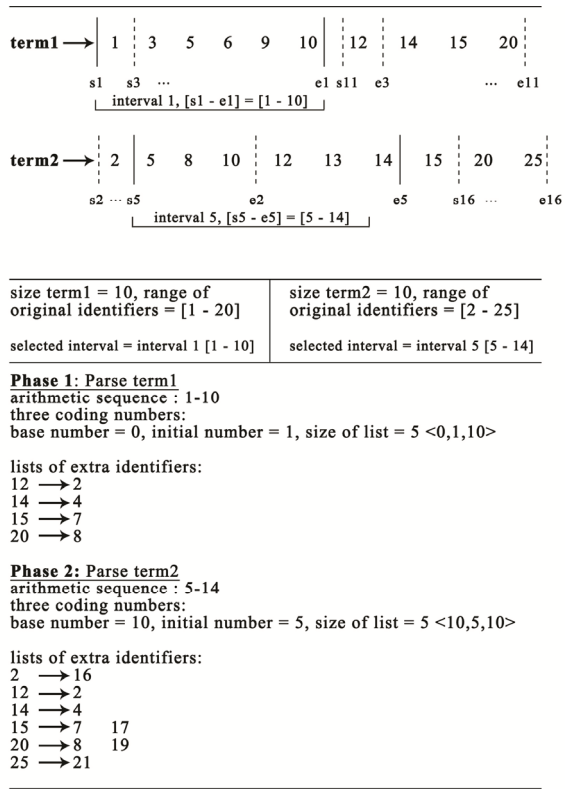

Figure 2: Conversion of the inverted lists into arithmetic progressions.

4 DECOMPRESSION

In order to complete the proposed algorithm we must describe the procedure for decompressing efficiently the compressed representation. This procedure ensures that we can take the initial lists self-same without any possibility of collision. The innovation of our idea for the decompression justifies the existence of the base value that is used in each term; this base value in combination with the extra identifiers of each document permit easy decompression. In particular each of the terms has a unique number as base value. The base value of each term is equal to the base value of the previously processed term plus the value of the last term of the sequence.

Every time that is necessary for an original document identifier to get an extra identifier value in our algorithms, this value is computed as (base value + the extra document identifier given in the numerical sequence). Then the extra document identifier is added to the inverted list of extra identifiers of the base identifier. Hence, all the extra identifiers, in a specific term, will lie in the interval from (base value + start value of numerical sequence progression) till (base value + last value of numerical sequence). The last value of the sequence is computed as (base value + start of sequence + length of sequence). In this way, we ensure that all extra identifiers that the processing of a term will cause to exist are in a unique interval that it is not covered by any one of the intervals of the rest of the term; this guarantees unique decompression. We have stored the numerical sequences and the extra identifiers lists in the disk, and we must decompress them to the initial representation.

To decompress the stored values in our compressed representation of the inverted index, we take the terms one-by-one and we follow the below procedure:

Step 1: We retrieve the list of the extra document identifiers that appear in the secondary index using the base value.

Step 2: We remove the base value from the secondary index identifiers in order to produce the extra document identifiers in the arithmetic progression of the term.

Step 3: We replace the extra document identifiers with the respective original document identifiers in the numerical sequence. The new list is the initial list of document identifiers for the specific term.

We employ the same procedure for the reconstruction of all the inverted lists.

In Figure 3 we decompress the compressed inverted lists from the Figure 2 and we produce the initial inverted lists. Initially Figure 3 contains the compressed inverted index and then are follow the steps of the decompression procedure.

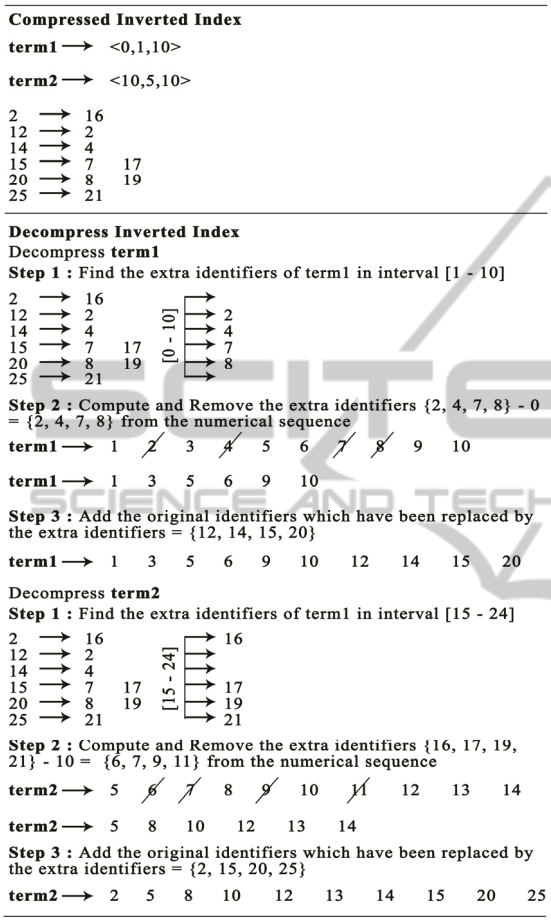

Figure 3: Decompression of the compressed index.

5 EXPERIMENTS

We performed the experiments by moving into two different directions. The first one was to leave the secondary index uncompressed, and measure the savings in compression that the unified scheme (transformed index plus secondary uncompressed index) attains. The second one was to apply to the secondary index, state of the art compression techniques and see the savings in performance that these achieve.

5.1 Dataset

In our experiments we employed the ClueWeb09 dataset (Callan, 2009). From this dataset we extracted two different parts of data, the Wikipedia and the Category B dataset. The Wikipedia dataset has almost 7 million pages and the Category B dataset has 50 million pages. Also, we used the Indri search machine in order to create their indices.

5.2 Experimental Process

We first run the proposed method of Section 3 with the secondary index uncompressed, and then we run the method with the secondary index compressed with IPC and then with the PFD compression technique (we used the OptPFD implementation). We compared the performance by compressing the inverted indices using only IPC and PFD. Following (Chierichetti et al., 2009) we use as measure of the compression performance the *compression ratio*, defined as the ratio of the compressed index size to the uncompressed index size.

The results are depicted in Tables 1 and 2. Table 1 show the compression ratio which was achieved in relation to the original size for the proposed techniques (when the secondary index is uncompressed). Table 2 depicts the compression ratio which was achieved by the compression technique when the secondary index has been compressed with the existing techniques. Table 1 depicts that our algorithm achieves 78-79% compression of the initial index which is not as good as the existing techniques; however this storage reduction justifies the employment of our technique as a preprocessing mechanism. Table 2 results depict a good increase in compression performance, outperforming IPC and PFD by almost 4-5%.

Table 1: The compression ratio achieved by the proposed algorithm, with the secondary index uncompressed.

	Algorithm	IPC	PFD
Wikipedia	78%	44%	42%
Category B	79%	45%	44%

Table 2: The compression ratio achieved by the proposed algorithm, with the secondary index compressed.

	Algorithm + IPC	Algorithm + PFD
Wikipedia	40%	39%
Category B	41%	41%

Hence the experiments show that using our technique as a preprocessing step, or more accurately as a transformation step, one can provide better compression results from the existing techniques. It is interesting to note that our construction can easily incorporate other

compression techniques by simply employing instead of IPC and PFD, other constructions.

6 CONCLUSIONS AND OPEN PROBLEMS

We have presented a novel technique for compressing efficiently inverted files, when storing the document identifiers. The proposed constructions can be harmonically combined with other techniques that have been proposed in the literature such as IPC and PFD and produce compression results that are competitive and in the majority of the cases even better than those of the previous works. As the careful reader should have noticed the handling of the secondary index with the extra identifiers constitutes the main burden of our technique. This burden can be relieved by using more arithmetic progressions when representing each initial inverted list, and here there exists a tradeoff that is worth the effort to be further explored, since it could lead to a whole set of parametric techniques. Moreover it could be interesting to investigate further techniques of handling the secondary index that could lead to faster decompression performance.

ACKNOWLEDGEMENTS

This research has been co-financed by the European Union (European Social Fund-ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF)-Research Funding Program: Heracleitus II. Investing in knowledge society through the European Social Fund.

This research has been co-financed by the European Union (European Social Fund-ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF)-Research Funding Program: Thales. Investing in knowledge society through the European Social Fund.

REFERENCES

Baeza-Yates, R., Ribeiro-Neto, B. 2011, Modern Information Retrieval: the concepts and technology behind search, second edition, Essex: Addison Wesley.

Callan, J. 2009, *The ClueWeb09 Dataset*. available at http://boston.lti.cs.cmu.edu/clueweb09 (accessed 1st August 2012).

Chierichetti, F., Kumar, R., Raghavan, P., 2009. Compressed web indexes. In: 18th Int. *World Wide Web Conference*, pp. 451–460.

Ding, S., Attenberg, J., Suel, T., 2010, Scalable Techniques for Document Identifier Assignment in Inverted Indexes, *Proceedings of the 19th International Conf. on World Wide Web*, pp. 311-320.

He, J., Yan, H., Suel, T., 2009. Compact full-text indexing of versioned document collections, *Proceedings of the 18th ACM Conference on Information and knowledge management*, November 02-06, Hong Kong, China

Heman, S. 2005. Super-scalar database compression between RAM and CPU-cache. MS Thesis, Centrum voor Wiskunde en Informatica, Amsterdam.

Moffat, A., Stuiver, L., 2000, Binary interpolative coding for effective index compression, *Information Retrieval*, 3, 25-47.

Navarro, G., Silva De Moura, E., Neubert, M., Ziviani, N., Baeza-Yates R., 2000, Adding Compression to Block Addressing Inverted Indexes, *Information Retrieval*, 3, 49-77.

Ntoulas A., Cho J., 2007. Pruning policies for two-tiered inverted index with correctness guarantee, *Proceedings of the 30th Annual International ACM SIGIR conference on Research and development in Information Retrieval,* July 23-27, Amsterdam, The Netherlands.

Scholer, F., Williams, H.E., Yiannis, J., Zobel, J. 2002. Compression of inverted indexes for fast query evaluation, *In 25th Annual ACM SIGIR Conference*, pp. 222-229.

Witten, I. H., Moffat, A., and Bell, T., 1999. *Managing Gigabytes: Compressing and Indexing Documents and Images.* Morgan Kaufmann Publishers, 2nd edition.

Yan H., Ding S., Suel T., 2009. Inverted index compression and query processing with optimized document ordering, *Proceedings of the 18th international conference on World Wide Web,* April 20-24, 2009, Madrid, Spain

Yan, H., Ding, S., Suel, T., 2009, Compressing term positions in Web indexes, pp. 147-154, *Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval.*

Zhang, J., Long, X., and Suel, T. 2008. Performance of compressed inverted list caching in search engines. *In the 17th International World Wide Web Conf. WWW.*

Zobel, J., Moffat, A., 2006. Inverted Files for Text Search Engines, *ACM Computing Surveys,* Vol. 38, No. 2, Article 6.

Zukowski, M., Heman, S., Nes, N., and Boncz, P. 2006. Super-scalar RAM-CPU cache compression. In the *22nd International Conf. on Data Engineering* (ICDE) 2006.