A Software Architecture for Effective Document Identifier Reassignment

Roi Blanco and Álvaro Barreiro

Computer Science Department, University of Corunna, Spain
{rblanco, barreiro}@udc.es

Abstract. This works presents a software solution for enhancing inverted file compression based on the reassignment of document identifiers. We introduce different techniques recently presented in the Information Retrieval forums to address this problem. We give further details on how it is possible to perform the reassignment efficiently by applying a dimensionality reduction to the original inverted file and on the evaluation results obtained with this technique. This paper is devoted to the software architecture and design practises taken into account for this particular task. Here, we show that making use of design patterns and reusing software components leads to better research applications for Information Retrieval.

1 Introduction

Indexing mechanisms are a critical part of Information Retrieval systems, as they provide fast access to term information needed for query evaluation [7]. Inverted files are by far the most used indexing structure in Information Retrieval (and even in large-scale database systems), specially when dealing with very large sets of data. Indexing structures store different information related to each term appearing in the collection, depending on the granularity specified. In this paper we will focus on document-level inverted files, this is, we only take into account the occurrences of terms in documents. Figure 1 is an illustrative example, where each line stands for a different document, and the structure stores term document frequency (number of different documents that contain the given term) and document identifiers.

This indexing structure is organised in posting lists, where each one holds the information for a different term. Therefore, an inverted file can be expressed as:

$$\{< t_i; f_{t_i}; d_1, d_2, \ldots, d_{ft_i} >, d_i < d_j \forall i < j\} \forall t_i \in T, \tag{1}$$

where T is the set of terms, f_{t_i} stands for the document frequency of the term t_i, and d_i is the document identifier. As the notation implies, the document identifiers are ordered.

Usually, posting lists are compressed with a suitable coding for the data involved. Compression methods need a suitable model of the data to perform

R. Moreno Díaz et al. (Eds.): EUROCAST 2005, LNCS 3643, pp. 254–262, 2005.

Doc.Id	Input Text:
1:	Rain on the green grass
2:	and rain on the tree
3:	And rain on the housetop
4:	but not on me
5:	Rain, rain, go away

term	Documents
rain	$\langle 4; 1, 2, 3, 5\rangle$
on	$\langle 4; 1, 2, 3, 4\rangle$
the	$\langle 3; 1, 2, 3\rangle$
green	$\langle 1; 1\rangle$
grass	$\langle 1; 1\rangle$
and	$\langle 2; 2, 3\rangle$
tree	$\langle 1; 2\rangle$
housetop	$\langle 1; 3\rangle$
but	$\langle 1; 4\rangle$
not	$\langle 1; 4\rangle$
me	$\langle 1; 4\rangle$
go	$\langle 1; 5\rangle$
away	$\langle 1; 5\rangle$

Fig. 1. Inverted File Example

the coding and decoding operations. Nevertheless there exists a family of methods, known as *static codes* [6] that work without having to store any information about the data. These coding methods are useful for posting lists, as they only contain integers and a model can become larger than the real compressed data.

Actually, not every integer corresponding to a document identifier is coded, but the difference between two consecutive ones, also known as *d-gaps* [7]. Static codes use the fact that small integers occurs often, and give shorter codes to them (measured in bits). So the shorter the differences between consecutive documents, the higher gain in compression, which translates into a smaller inverted file.

The reassignment of document identifiers is a very recent technique for enhancing compression with static codes [2]. The main idea is to map each original document identifier appearing in the collection into a new one, trying to reduce the distances between occurrences in the posting lists. Some papers proved that reordering can lead into gains in compression ratio for static inverted files of medium size collections [2,8,9,10]. These works approached the problem from different perspectives, using other well-known problems for approximating the original one. In this paper, we will show how to build a software architecture for the technique described in [10], which tries to solve the problem in an efficient way by reducing the dimensionality of the input data through matrix transformations. The set of programs to be used must be able to index, perform matrix operations with large sets of data and recompress inverted files using a previously calculated order, everything in an independent manner and using reusable and modular data structures. We focus on an object oriented programming paradigm and make extensive use of design patterns [12]. Section 2 describes the known approaches to the document identifier reassignment problem. Section 3 shows the main technique implemented in the software architecture we are describing. Sections 4 and 5 presents the architecture developed for the referred solution and implementation details. Finally, section 6 summarises the work and shows some future research lines.

2 Previous Work

Next, we describe the state of the art concerning the reassignment of document identifiers. Different approaches to the problem consider different data structures. Most works build a *weighted similarity graph* G, where the nodes v_i, v_j represent the document identifiers i, j and an edge (v_i, v_j) represents the similarity between documents i and j [2,8,10]. On the other hand, the work in [9] uses document clusters for reordering the identifiers.

Blandford and Blelloch (B&B) [2] developed a technique that improved the compression ratio about fourteen percent in TREC text collections. The technique employs a similarity graph as described before, and operates in three different phases. The first phase constructs the document-document similarity graph from the original inverted file. The second part of the algorithm calls to a graph partitioning package which implements the Metis [11] algorithm for splitting recursively the similarity graphs produced by the first part. Finally, the algorithm applies rotations to the clustered graph outputted by the second part for optimising the obtained order. The final assignment for the document identifiers is obtained by simply depth-first traversing the resulting graph. As is stated in [2], constructing a full similarity graph is $O(n^2)$, so the raw technique may not suitable for very large collections. Nevertheless, the efficiency of the algorithm can be controlled by two parameters: τ and ρ. The first parameter, τ, acts as a threshold for discarding high-frequency terms, i.e., if a term t_i has size $|t_i| > \tau$ it is discarded for the construction of the similarity graph. Actually the algorithm works with a sample of the full similarity graph. The parameter ρ stands for how aggressively the algorithm sub-samples the data: if the index size is n it extracts one element out of $\lfloor n^\rho \rfloor$. Tuning τ and ρ the technique may lead to a tradeoff between efficiency and time and memory usage.

Shie et al. [8] proposed a different graph-based approach, based on the well known Travelling Salesman Problem. The TSP is stated as follows: given a weighted graph $G = (V, E)$ where $e(v_i, v_j)$ is the weight for the edge from v_i to v_j, find a minimal path $P = \{v_1, v_2, \ldots, v_n\}$ containing all the vertexes in V, such as if $P' = \{v'_1, v'_2, \ldots, v'_n\}$ is another path in G, $\sum_{i=2}^{n} e(v_i, v_{i-1}) \leq \sum_{i=2}^{n} e(v'_i, v'_{i-1})$. Considering Sim a weighted adjacency matrix, it is possible to build a Document Similarity Graph (DSG). The idea is to assign close document identifiers to similar documents as this will likely reduce the d-gaps in common terms postings. This traversing problem can be transformed into a TSP just by considering the complement of the similarity as the weights in the edges of the DSG. The solution found by the TSP is the path that minimises the sum of the distances between documents, therefore the algorithm is an appropriate strategy to the document reassignment problem.

Silvestri et at [9] proposed a method which aimed at enhancing the clustering property of the index. Prior to reassigning, the technique computes a so called *transactional* representation for the documents, which consists in storing a 4-bytes truncated MD5 [13] digest for the terms appearing in them. Starting from that, the authors follow into two different assigning schemes, differing in the starting point of the algorithms: *top-down*, considering the whole collection

and recursively splitting it, and *bottom-up* assignment, starting from a flat set of documents and extracting disjoint sequences containing similar documents, grouping them.

The previously described techniques are only approximations for solving the real problem, and have some efficiency drawbacks, as stated in [10], like computing and storing the full similarity graph or in [9] the linear dependence of time and memory usage respect to the document size.

3 Document Identifier Reassignment Through Dimensionality Reduction

The architecture to be presented here, solves the document identifier reassignment problem as a TSP like in [8] but avoiding some efficiency problems. The main idea is to reduce the input similarity matrix data via Singular Value Decomposition. That leads to the development of a new software framework, in which reordering and recompressing techniques can operate after carrying out a dimensionality reduction. This way, the memory usage by such algorithms is controlled and as it is possible to determine the total amount of memory used in each step. Moreover, by assigning document identifiers through dimensionality reduction, results are consistent between different collections and are comparable with those obtained by working with the full dimension schema, as stated by [10].

Next, the main method for effectively reducing the dimensionality by SVD is described. Section 5 gives further details on the algorithm used.

3.1 Singular Value Decomposition

Singular Value Decomposition (SVD) is a well known mathematical technique used in a wide variety of fields. It is used to decompose an arbitrary rectangular matrix into three matrices containing singular vectors and singular values. This matrices show a breakdown of the original relationships into linearly independent factors. The SVD technique is used as the mathematical base of the Latent Semantic Indexing (LSI) IR model [14].

Analytically, we start with X, a $t \times d$ matrix of terms and documents. Then, applying the SVD X is decomposed into three matrices:

$$X = T_0 S_0 D_0' \tag{2}$$

T_0 and D_0 have orthonormal columns, and S_0 is diagonal and, by convention, $s_{ii} \geq 0$ and $s_{ii} \geq s_{jj} \forall i \geq j$. T_0 is a $t \times m$ matrix, S_0 is $m \times m$ and D_0', the transposed matrix of D_0, is $m \times d$ where m is the rank of X. However it is possible to obtain a k-ranked approximation of the X original matrix by keeping the k largest values in S_0 and setting the remaining ones to zero obtaining the matrix S with $k \times k$ dimensions. As S is a diagonal matrix with k non-zero values, the corresponding columns of T_0 and rows D_0 can be deleted to obtain T, sized $t \times k$, and D', sized $k \times d$, respectively.

This way we can obtain $\hat{X}$ which is a reduced rank k approximation of X:

$$X \approx \hat{X} = TSD' \tag{3}$$

$\hat{X}$ is the closest rank k approximation of X in terms of the Euclidean or Frobenious norms, i.e. the matrix which minimises $||X - \hat{X}||_N^2$ where $||\cdot||_N^2$ is the involved norm.

The i-th row of DS gives the representation of the document i in the reduced k-space and the similarity matrix $\Theta(X)$ is k-approximated by $\Theta(\hat{X})$:

$$\Theta(X) \approx \Theta(\hat{X}) = \hat{X}'\hat{X} = DS^2D', \tag{4}$$

where $\hat{X}'$ is the transposed matrix of $\hat{X}$ and D' is the transposed of D.

If $D_{d\times k} = \{z_{ij}\}$ and $\{s_i\}$ is the set of diagonal elements of S, it is easy to prove that

$$\Theta(\hat{X})_{ij} = \sum_{\gamma=0}^{k-1} z_{i\gamma} z_{j\gamma} s_\gamma^2 \tag{5}$$

Therefore it is possible to calculate $\Theta(\hat{X})_{ij}$ only storing the set of k elements $\{s_i\}$ and the $d \times k$ matrix D instead of computing and writing the full rank matrix $\Theta(X)_{d\times d}$.

The output of the SVD of X, $\hat{X}$ has been used in the computation of $\Theta(\hat{X})$ (equation 4). The same result could be obtained by calculating the SVD of $\Theta(X)$ due to the uniqueness property of SVD [1]. Since SVD computes the best rank k approximation, it is proved that the best rank k approximation of $\Theta(X)$ is obtained starting from X and without the need of computing $\Theta(X)$.

3.2 Results

Applying this technique and tackling the TSP with a Greedy Nearest Neighbour algorithm (Greedy-NN), we obtain good values in compression ratios, measured in bits per document gap used. For a detailed reference of this results see [10]. Tests were driven in the LATimes and FBIS collections, which form the TREC-5 disk, and with different values of the k parameter (which reflects the matrix dimension in the reduced space).

With k=200, for the LATimes collection (FIBS collection) we achieved a 13.65% (8.02%) gain in compression ratio respect to the original document identifier order with the gamma encoding, 13.2%(8.7%) for the delta encoding, and 11.32% (5.15%) for the interpolative coding. These values are 17.67% (21.92%), 17.8%(21.1%) and 13.66% (14.58%) repectively for both collections and the three encoding schemes, respect to a random reassignment. Computing the Greedy-NN TSP with the reduced space approximation $\Theta(\hat{X})$ gives worthy compression ratios in every case. The gains in the FBIS collection are worse than the ones in the LATimes, although starting from a randomized order the result is inverted. This is the expected behaviour if the FBIS collection exhibits a better original document order. One point to remark is that even in the case of interpolative coding, where the starting point is much better, the method is able to produce gains in bits per document gap.

4 Architecture

Figure 2 describes the system built for testing this approach. Nodes represent the data and components represent the different modules deployed. A solid arrows means direct dependency between data and a module (either input or output), and dashed arrows drive the flow of the program through the different components.

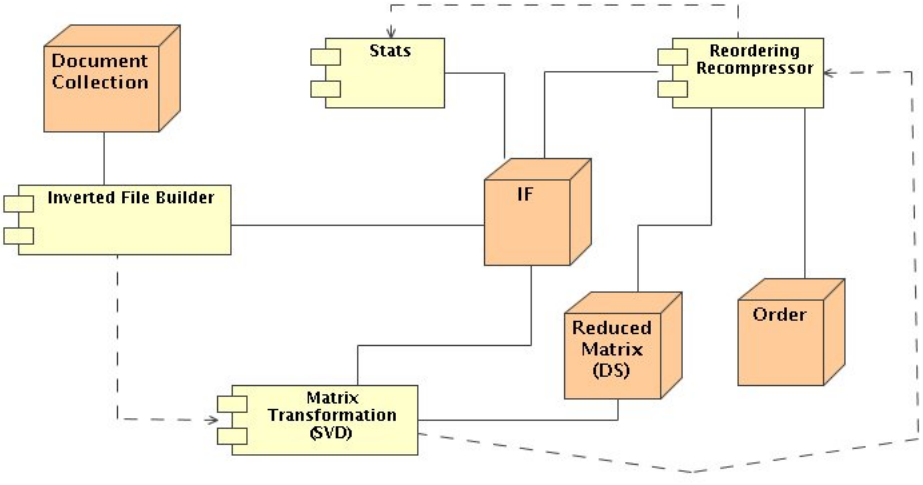

Fig. 2. Main component diagram for the software architecture

The full process is driven by a Mediator [12], which controls the interaction between the different components and reduces the dependency between the processing steps. The mediator serves as an intermediary and keeps modules from referring to each other explicitly, thereby reducing the number of interconnections. This has a purpose of generality for the architectural design, as it is a feature that facilitates the development and extension of the current software by allowing the construction of program pieces completely independent between each other. This is interesting for having different indexing, compression, dimensionality reduction, statistical and reordering modules. Extensions could be, for example, a component for building Direct Files, which are convenient for doing query expansion.

The process starts with the inversion of the text collection. The inverted file builder mechanism produces a complete version of the inverted file, considering the document identifiers in natural order, this is, as they appear originally in the collection. Also, this module outputs the X data matrix to a SVD module (see 3 for more details on this). This module produces the matrices $D_{d\times k}$ and $S_{k\times k}$ that allow the computation of $\Theta(\hat{X})$, therefore there is no longer needed to store the similarity matrix $\Theta(X)_{d\times d}$. The reassignment module uses the SVD output

matrix to compute the TSP approach fully described in [10], however the open design allows the interchangeability with the other techniques introduced in 2.

The output of the TSP reassignment module is used by an inverted file recoding program which exploits the new locality of the documents to enhance d-gap compression. Finally, some statical information is taken to make suitable comparisons between compression ratios achieved by the original encoding and those obtained after reassignment.

Respect to the complexity involved, as k is a constant factor, we can conclude that the space usage of the algorithm now is $O(d)$, i.e., linear in collection size and not dependant on document size. The main difference with respect to previous implementations for the TSP technique for the document identifiers reassignment problem, is that computing the similarity between two documents d_i and d_j involves k operations $(\sum_{\gamma=0}^{k-1}(DS)_{i\gamma}(DS)_{j\gamma})$ and storing k real pointers per document, making a total of $k \times d$ for the full matrix. This representation can fit smoothly into memory by adjusting the parameter k and uses considerably less space than the original $d \times d$ matrix. Even more, the space usage can be precalculated so suitable scalable algorithms can be easily developed. Considering 32 bits per float (real number), our implementation uses $4 \times k \times d$ bytes of main memory.

5 Implementation

In this section, it is explained how the architecture was developed. The modules implemented make extensive use of design patterns: descriptions of problems that appears repeatedly and solutions to them, in such way that the solution can be applied in different situations. Design patterns are an extended technique for developing Object Oriented software providing a good balance between space and time [12].

For indexing and compressing tasks we used the MG4J [4] from the University of Milan, a free Java implementation of the indexing and compression techniques described in [7] and originally implemented in the MG software [3]. The indexing is made in three different passes to the original text of the collection, using a technique called in-memory text-based partitioning inversion [7]. As an overview, the technique builds the index entirely in main memory and avoids random accesses to disk by controlling the amount of storage needed in each step of the algorithm and using compression techniques. This way it is possible to perform an efficient map from memory to disk, avoiding seeking and swapping times. The first pass collects statistics from the text, like term document frequency and in-document term frequency, and builds a dictionary file, also known as lexicon. The second pass reads the lexicon and calculates appropriate values for compressing efficiently the data structures used in the inversion process, and allocates disk space for guarantying an optimum usage of the resources. After that, the program builds a random access in-memory inverted file for chunks of the collection, and the inverted file in disk for the whole collection. Finally, a third an optional pass over the text is done for recompressing and producing a new final inverted file.

For the SVD module we used the SVDLIBC [5], a C library based on the SVDPACKC library. It should be pointed that we needed to modify the MG4J software to output data directly to the SVDLIBC module. This module computes the singular values associated to a matrix, which can be inputted in different formats, by the algorithm *las2*, standing for Single-Vector Lanczos Method [15].

The rest of the program code (reassignment, recoding and statistical software) is written in Java, taking into account that some routines and methods are shareable between different components. Some excerpts of the written code may include modifying coding routines and generating generic libraries for graph manipulation, matrix transformations and standard data formats for matrices. As a brief example, in the figure 3 we present an usage of the strategy pattern [12] for coding routines, which was adapted for including interpolative coding [6] for document pointers.

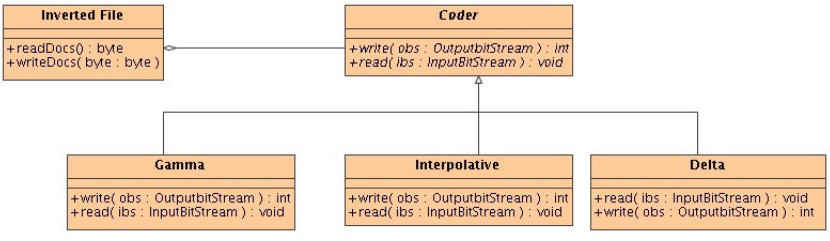

Fig. 3. Main component diagram for the software architecture

The inverted file acts as the context for the pattern, and the individual strategies implement an interface with the coding and decoding methods. This way the coding scheme is a behaviour of the different subclasses involved in the pattern, thus, adding new coding methods is just a matter of implementing the interface with different classes and behaviours.

6 Conclusions

Previous works established that reassigning the identifiers of the documents appearing in a collection improves compression ratios of inverted files, when coding the differences between consecutive identifiers with static codes. Moreover, the work in [10] shows how the reassignment can be done effectively by making a prior dimensionality reduction of the term-document matrix. In this paper we have focused on the software architecture and the concrete design and implementation issues. We underlined the importance of reusing software pieces and making use of good design practises. These methodologies must have an impact not only in a corporate environment but also for developing software tools with research purposes, where software lies above well-founded mathematical concepts. Therefore, working with a reusable framework for Information Retrieval

research avoids a lot of unnecessary work when reusing software pieces, and facilitates the deployment and development of more robust software kits that can be used by the IR community.

Acknowledgements

The work reported here was co-founded by the "Secretaría de Estado de Universidades e Investigación" and FEDER funds under research projects TIC2002-00947 and Xunta de Galicia under project PGIDT03PXIC10501PN.

References

1. B. T. Bartell, G. W. Cottrel and R. K. Belew. Latent Semantic Indexing is an optimal special case of Multidimensional Scaling. In *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 161-167, 1992.
2. D. Blandford and G.Blelloch. Index compression through document reordering. In *Proceedings of the IEEE Data Compression Conference (DCC'02)*, pp. 342-351, 2002.
3. `http://www.cs.mu.oz.au/mg/` Managing Gigabytes.
4. `http://mg4j.dsi.unimi.it/` MG4J (Managing Gigabytes for Java).
5. `http://tedlab.mit.edu/~dr/SVDLIBC/` SVDLIBC.
6. A. Moffat, A. Turpin. *Compression and Coding Algorithms*, Kluwer 2002.
7. I. H. Witten, A. Moffat and T. C. Bell. *Managing Gigabytes - Compressing and Indexing Documents and Images*, 2nd edition. Morgan Kaufmann Publishing, San Francisco, 1999.
8. W.-Y. Shieh, T.-F. Chen, J. J.-J. Shann and C.-P. Chung. Inverted file compression through document identifier reassignment. *Information Processing and Management*, 39(1):117-131, January 2003.
9. F. Silvestri, S. Orlando and R. Perego. Assigning identifiers to documents to enhance the clustering property of fulltext indexes. In *Proceeding of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 305-312, 2004.
10. R. Blanco, A. Barreiro. Document identifier reassignment through dimensionality reduction. In *Proceeding of the 27th European Conference on IR Research, ECIR 2005*, LNCS 3408, pp. 375-387, 2005.
11. G. Karypis and V. Kumar. A fast and high quality multilevel scheme for patitioning irregular graphs. Technical Report TR 95-035, 1995.
12. E. Gamma, R. Heml, R. Johnson, J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software* Addison Wesley 1995.
13. R. Rivest, RFC 1321: The md5 algorithm.
14. S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer and R. Harshman. Indexing by Latent Semantic Analysis. In *Journal of the American Society for Information Science*, 41(6):391-407, 1990.
15. M. Berry. Large Scale Singular Value Computations. In *International Journal of Supercomputer Applications*. 6:1, (1992), pp. 13-49