Document identifier reassignment through dimensionality reduction

Roi Blanco and Álvaro Barreiro

AILab. Computer Science Department, University of Corunna, Spain
roi@mail2.udc.es, barreiro@udc.es

Abstract. Most modern retrieval systems use compressed *Inverted Files* (IF) for indexing. Recent works demonstrated that it is possible to reduce IF sizes by reassigning the document identifiers of the original collection, as it lowers the average distance between documents related to a single term. Variable-bit encoding schemes can exploit the average gap reduction and decrease the total amount of bits per document pointer. However, approximations developed so far requires great amounts of time or use an uncontrolled memory size. This paper presents an efficient solution to the reassignment problem consisting in reducing the input data dimensionality using a SVD transformation. We tested this approximation with the Greedy-NN TSP algorithm and one more efficient variant based on dividing the original problem in sub-problems. We present experimental tests and performance results in two TREC collections, obtaining good compression ratios with low running times. We also show experimental results about the tradeoff between dimensionality reduction and compression, and time performance.

Keywords: Document identifier reassignment, SVD, indexing, compression.

1 Introduction

Large-scale Information Retrieval (IR) systems need an indexing mechanism for efficient retrieval. The most extended data structure used is the *inverted file* (IF). This IF is a traversed representation of the original document collection, organised in *posting lists*. Each entry in the inverted file contains information about a single term in the document collection. The format of the posting lists reflects the granularity of the inverted file, addressing in which documents and positions the term appears. In this work we suppose a document level granularity, therefore the posting list for term t_i is:

$$< t_i; f_{t_i}; d_1, d_2, \ldots, d_{ft_i} >, d_i < d_j \forall i < j \quad (1)$$

where f_{t_i} stands for the frequency of the term t_i (number of documents in which t_i appears), and d_i is the document identifier. As the notation implies, the document identifiers are ordered.

Since this structure requires a large amount of storage space, posting lists usually are compressed. Several works aimed at efficiently encoding the document identifiers contained in each entry. Posting lists are stored as a sequence of differences between consecutive documents identifiers (*d-gaps*). This method improves compression, as variable-length encoding schemes represent small integers with less bits than large ones. Small d-gaps are more frequent than large ones, so inverted files can be compressed efficiently. Recent works have tried to increase the number of small d-gaps by reordering the document identifiers, hoping that the average d-gap is lowered. This process is done after the collection is traversed and the inverted file is built. These works are presented in [2] and [9], in the following of this paper, the *B&B* (Blandford and Blelloch) and the *TSP approach* (Travelling Salesman Problem) respectively. Results show that the document identifier reassignment technique is effective in lowering the average d-gap, and therefore allowing gains in compression ratios. Both solutions 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.

The *B&B* algorithm recursively splits G into small subgraphs $G_{l,i} = (V_{l,i}, E_{l,i})$ representing smaller subsets of the collection until every subgraph becomes a singleton. After that, the technique performs a reordering of the document identifiers, by *depth-first* traversal. The TSP approaches the problem by considering it a *Travelling Salesman Problem* which can be solved by several ways pointed in graph literature. The objective is to find the traverse that minimizes the d-gaps by reassigning document identifiers according to the order in which they were visited.

Unfortunately these solutions lack of efficiency, and they turn nonviable for large collections in terms of space and time. In [10] the authors could only test the *B&B* algorithm in a collection of 60.000 documents, while the TSP implementation in [9] needs 2.17 GB of main memory and 23 hours to process a collection of 132.000 documents.

On the other hand, the work in [10] proposes a different approach by *assigning* the document identifiers *on the fly* during the inversion of the text collection. For this approach a *transactional* representation form of the documents is used, which stores for each document d_i a set of 4-byte integers representing the MD5 Message-Digest [8] of each term appearing in d_i. Using this representation, two families of algorithms were developed to compute an efficient document assignment: the *top-down assignment* and the *bottom-up assignment*. The top-down assignment schemes start with the whole collection and recursively split it into sub-collections, inserting similar documents into the same sub-collections. After this phase, the algorithm merges the sub-collections obtaining a single and ordered group of documents, which is used to compute the assigning order. Bottom-up schemes start from a set of documents, extracting disjoint sequences containing similar documents. Each sequence is ordered, and the final assignment is computed by considering an arbitrary order between sequences. Considering a document collection in transactional form $\widetilde{D} = \{\widetilde{d_1}, \widetilde{d_2}, \ldots, \widetilde{d_N}\}$, the space storage needed by the top-down methods in the asymptotic analysis is

$O(|\widetilde{D}|\log(|\widetilde{D}|))$ and for the bottom-up approaches is $O(|\widetilde{D}|)$ which gives superlineal and lineal order respectively. However, the total space is also dependant on a factor $|\bar{S}|$, that stands for the average size of documents. For the collection used in the experiments reported in [10], the Google Programming Contest collection, $|\bar{S}|$ has a value of 256 terms.

In this paper we propose a way to make the reassignment methods operational by arranging the input data into a lower dimensionality space, which reflects the major association patterns between documents and terms. Furthermore, the space storage needed can be parameterized so this technique has good behaviour also in collections where the average document size is high. To achieve this dimensionality reduction, we used the *Singular Value Decomposition* (SVD) technique. This way we built a representation of the document similarity matrix (the graph G). Instead of working with the original $d \times d$ similarity matrix (where d is the number of documents in the collection), we use a $d \times k$ matrix, where k is a chosen constant. Using the reduced dimension we can precalculate the amount of memory used by the reassignment algorithm. In addition it is possible to lower/upper the memory bounds, finding a compromise between space usage and performance. We tested our implementation with the *TSP* Greedy-NN algorithm and give evidence of the feasibility of the technique in some TREC collections. We also implemented and tested a more time efficient version, dividing the original problem in a number of similar sub-problems.

The rest of the paper is organised as follows. Section 2 describes the TSP algorithm. Section 3 shows the way of reducing the document similarity matrix dimensionality by computing its singular value decomposition and how we applied this result to the reassignment problem. Section 4 describes the experimental conditions and the tests results of our approach. In Section 5 we discuss the leading lines in further research derived from this work. Section 6 presents some conclusions from the experimental results.

2 The TSP approach to the document reassignment problem

Given that we will illustrate the use of the dimensionality reduction technique for document reassignment with the TSP algorithm, we briefly review the work in [9].

2.1 The document reassignment problem as a TSP

An inverted file can be seen as a posting list set. Each list contains the information for a single term appearing in the document collection, expressed as a sequence of encoded d-gaps $G_t = \{g_1, \ldots, g_{f_t}\}$. The document reassignment problem tries to find the bijective function f that

- maps each document identifier into a new identifier in the range $[1 \ldots d]$
- minimizes the average document gap.

Similarity between documents is defined as the number of common terms, and maintained in a similarity matrix Sim, where Sim_{ij} represents the similarity between the document i and the document j.

Shieh et al. [9] proposed a gap-reduction strategy based in the transformation of the problem into a *Travelling Salesman Problem* (TSP). 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 an weighted adjacency matrix, it is possible to build a Document Similarity Graph (DSG) expressing the similarities between documents. This graph can be traversed by a gap-reduction strategy based on the similarity factor between documents. The idea is assigning 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 weight in the TSP. The solution found by the TSP is the path that minimizes the sum of the distances between documents, therefore the algorithm is an appropriate strategy to the document reassignment problem.

2.2 Heuristic approximations

The TSP is an *NP*-complete problem, so some polynomial-time heuristic approximations were modified for the reassignment problem. These algorithms were classified as greedy algorithms and spanning tree algorithms. We tested our low-dimension approximation with the Greedy-NN algorithm.

Greedy-NN algorithm

Input:
    The Graph G
    The Vertex set V
    The weighted Edges set E
Output:
    A global path P maximizing the similarity between vertexes
Select the edge $e(v_i, v_j) \in E$ with the largest weight;
Add v_i and v_j to P;
$v_{last} \leftarrow v_j$;
while $(|P| \neq |V|)$ **do**
    Choose $v_k \in V$ and $v_k \notin P$ such that $e(v_{last}, v_k)$ is maximal;
    Add v_k to P;
    $v_{last} \leftarrow v_k$;
end while
return P**;**

The Greedy-NN (Nearest Neighbor) expands the path by adding the closest vertex to the tail of the current path. In each iteration the algorithm adds a new vertex (document) chosen that its similarity is the largest with the last vertex in the path. This approximation is high time consuming. Each vertex is inserted only once in the path P and at iteration i the algorithm does $d - i$ comparisons (the remaining documents) involving the term size t of both documents. Therefore the overall complexity is $O(d^2 t)$.

2.3 Implementation considerations

The TSP approximation for the identifier reassignment problem was evaluated in [9]. The solution demonstrated good improvements in the compression ratio, although it presented some design challenges and poor performance time and space results.

First, this approach requires a big amount of space. The similarity matrix is symmetric ($Sim_{ij} = Sim_{ji}$) and the elements in the diagonal are not relevant, so it is easy to prove that we need to store $\frac{d(d-1)}{2}$ *similarity pointers* ($O(|d^2|)$). Even with a suitable coding schema this amount can become unmanageable, so a matrix partitioning technique has to be developed. Second, building this matrix can be very expensive if it does not fit into memory, as each update has to access the disk twice, involving big delays.

Experimental results were presented for two medium sized collections (FIBS and LATimes in TREC disk 5), to prove the effectiveness of this mechanism. These tests are summarized in table 1.

Table 1. Statistics of the pure TSP approach on two TREC collections reported by Shieh et al. [9]

Collection	FIBS	LATimes
Size of the Collection	470 MB	475 MB
Number of distinct terms	209,782	167,805
Number of distinct documents	130,471	131,896
Temporal Cost	19.63 h	23.28h
Space Cost	2.10 GB	2.17 GB

It is important to remark that the work in [9] provides bar graphs that show an approximated gain of one bit per gap when reassigning with the Greedy-NN for delta and gamma coding. The temporal costs include the process of building the similarity matrix, greeding and recompressing the inverted file. However, the results show that this full TSP approach may be unacceptable for very large collections, as it takes 23 hours and 2.17 GB to process a 475 MB collection.

3 Document reassignment by dimensionality reduction

Approaches proposed so far aimed at reducing the d-gaps using different representations of the full inverted file. Shieh et al. [9] and Blandford and Blelloch [2] built a full document similarity graph and traversed it by different algorithms such as the TSP and recursive splitting. Silvestri et al. [10] used an *on the fly* assignment technique with temporal and spacial complexity linear or super-linear on the number of documents, but also dependant on the average document length.

We propose a new approach based on dimensionality reduction in which reordering algorithms can operate efficiently. We aim at:

- allowing a controlled and efficient memory usage for such algorithms
- giving consistent results through different document and collection sizes and heuristics
- not being outperformed by the original working framework.

We tested our approach with the TSP reassignment algorithm described in section 2 with good results (section 4). In this section, the application of SVD to the document identifier reassignment problem is presented.

3.1 Single 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 [3].

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' 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 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 minimizes $||X - \hat{X}||_N^2$ 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 $\Theta(\hat{X}) = \hat{X}' \cdot \hat{X}$. The same result could be obtained by calculating the SVD of $\Theta(X) = X' \cdot X$ 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 SVD in the document reassignment problem

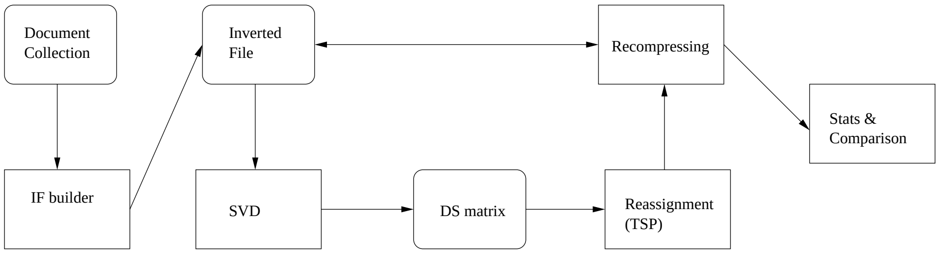

Fig. 1. Block diagram for the indexing and reassignment system

Figure 1 describes the system built for testing this approach. The inverted file builder mechanism outputs the X data matrix to a SVD module. 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 described in section 2.2. 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 output of the TSP reassignment module is used by an inverted file recoding program which exploits the new locality of the documents to enhance the d-gaps compression. Finally, some statical information is taken to make suitable comparisons between compression ratios achieved by the original encoding and those obtained after reassignment.

The main difference in this model 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.

One point to consider is the heuristic for choosing the starting node on the Greedy-NN algorithm which was also employed to solve the TSP. The algorithm (section 2.2) first chooses the edge (v_i, v_j) that has the maximum value. This involves the computation of the similarity for every document pair (d_i, d_j). In our approach selecting the first node this way takes more operations than the rest of the algorithm itself. Hence, we propose a less time-expensive heuristic, consisting in calculating, after dimensionality reduction, each (d_i, d_i) self-similarity and choosing the document (node) with the largest value.

4 Experiments and results

We performed several experiments for testing the low-dimension approach on the two TREC document collections described in table 1. These collections were not preprocessed, so indexing and reordering did include stop words and terms were not stemmed. The machine used was a 2.5 GHz AMD with 40 GB ATA disk and 1GB of main memory, running Linux OS. The original index file was built with MG4J [6] from the University of Milan, a free Java implementation of the indexing and compression techniques described in [12] and originally implemented in the MG software [5]. For the SVD module we used the SVDLIBC [11], a C library based on the SVDPACKC library. We wrote the reassignment, recoding and statistical software in Java. It should be pointed that we needed to modify the MG4J software to output data directly to the SVDLIBC module. Also some modifications were made that allowed us to encode document pointers with interpolative coding.

The first experiment assessed the performance of the system with the Greedy-NN algorithm, in terms of average bits per compressed document pointer (d-gap). The document collections were inverted, the IF was inputted to the SVD module and the program computed the Greedy-NN in the reduced dimension for the reassignment task. After reordering the collection, the inverted file was recompressed. The software measured the average bits per gap in the inverted file, before and after reordering and recompressing, which reflects the amount of compression gained by reordering the document collection. We ran several tests varying the following parameters:

– the parameter k which reflects the desired dimensionality reduction

– coding schemes for document pointers: delta coding, gamma coding or interpolative coding [7][12].

Best results are obtained considering X as a binary matrix in the reassignment process. The elements of X represent the presence or absence of a term in a given document. The recompressing module acts over the original index file which contains in-document term frequency and frequency of the term in the collection values. Results are given in bits per document gap because it is a measure independent of these indexing options. As stated in 3.2, the memory usage leads to $4 \times d \times k$ bytes, concretely $0.497707 \times k$ MB for the FBIS and $0.503143 \times k$ MB for the LATimes (for $k = 200$ less than 101 MB in both collections).

Table 2. LATimes bits per gap results

			k						
	Random	Original	200	100	50	20	10	5	1
Gamma	8.15	7.77	6.71	6.75	6.79	6.89	6.99	7.13	7.44
Delta	7.65	7.25	6.29	6.36	6.39	6.48	6.57	6.73	7.02
Interpolative	6.08	5.88	5.25	5.26	5.28	5.29	5.33	5.44	5.57

Table 3. FIBS bits per gap results

			k						
	Random	Original	200	100	50	20	10	5	1
Gamma	7.84	6.74	6.20	6.24	6.31	6.46	6.63	6.81	7.07
Delta	7.35	6.35	5.80	5.86	5.92	6.07	6.23	6.42	6.69
Interpolative	5.83	5.25	4.98	4.99	5.01	5.06	5.17	5.21	5.33

Tables 2 and 3 show the results for the different coding schemes. Columns refer to bits per document gap results for: random reassignment, original document identifiers and reassignment after reducing the dimensionality with different k values. Assigning values to k similar to those used in retrieval [4], the low-dimension algorithm operates with gains that give good benefits in bits per gap. As expected, the method behaves better as the k value increases. Also, the figures seem to have an asymptotic behaviour. 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 tha case of interpolative coding, where the starting point is much better, the method is able to produce gains in bit per document gap. Our tests did not include the computation of the full dimension solution as presented in [9], because it requires the development of matrix partition techniques and partial reading/writing, which is the task that we want to avoid. Shie et al [9] provided bar graph results for gamma and delta encoding in the LATimes and FBIS collections. However, exact compression values depends on the indexing software and particular indexing options. This information is not explicitly provided, thus it is not possible to make exact comparisons between their published full-dimension results and the k-dimension solutions.

Time measurement is divided in three parts: inverted file construction, SVD running time and reordering and recompressing time. As the system was built upon different modules, the different software pieces employ a lot of temporal I/O transfer time, which also is measured, so results are given in *elapsed time* in tables 4 and 5. Inversion takes 5m 20s for the FIBS collection and 6m 03s for the LATimes collection and it is not shown in the tables. Although the SVD software performs well for the collections and k values used, the TSP greedy algorithm running time still rises to high values. Anyway, we conclude that it is possible to achieve good compression ratios with reasonable time performance with our technique.

Table 4. FIBS running times

	k						
	200	100	50	20	10	5	1
SVD	34m 58s	15m 01s	5m 42s	3m 18s	2m 09s	1m 33s	58s
Reorder and recompress	8h 30m	4h 20m	1h 48m	58m 20s	31m 13s	17m 55s	7m 5s

Table 5. LATimes running times

	k						
	200	100	50	20	10	5	1
SVD	42m 31s	20m 37s	11m 25s	4m 5s	2m 33s	1m 55s	1m 09s
Reorder and recompress	8h 33m	4h 24m	2h 15m	58m 18s	32m 05s	18m 31s	7m 30s

Another advantage of the approach is that it is possible to propose more efficient reordering algorithms in time performance. We developed a simple new algorithm based on the division of the original problem in c subproblems, hereinafter c-GreedyNN. It operates as follows: first, it divides the DS matrix (which represents the document similarities in the k space) in c blocks of $[d/c]$ documents each. Then, each block is reordered by running the greedy algorithm.

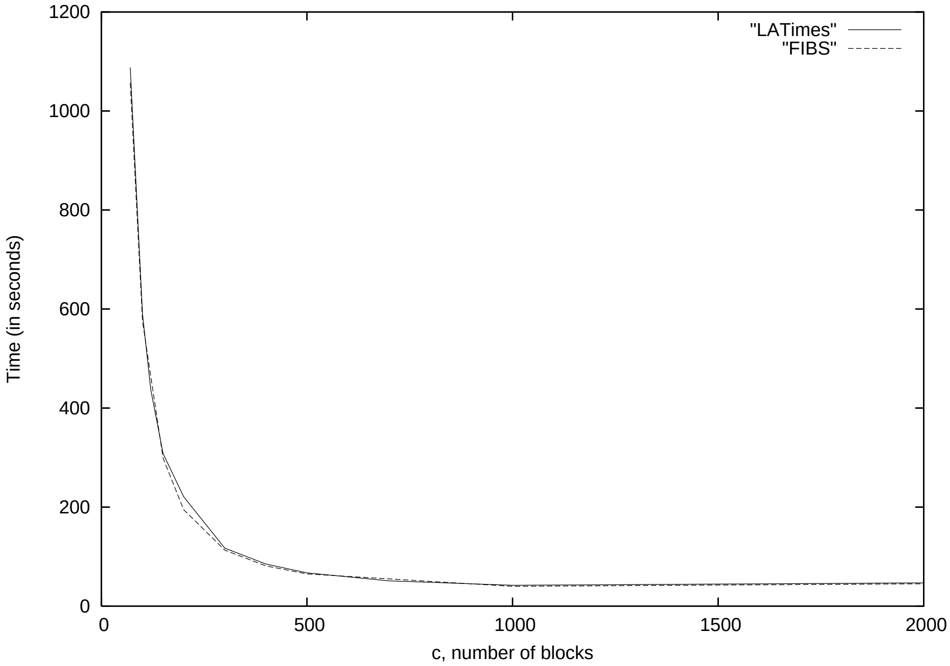

Fig. 2. Running times for the c-GreedyNN with the LATimes and FIBS collections under k=200 and delta coding

Finally, a block order is decided by running another greedy with c documents each one selected from different blocks. For a simpler explanation we consider d an exact multiple of c. Analytically, the Greedy-NN after dimensionality reduction does d comparisons to select the first document, and $\frac{d(d-1)}{2}$ for reordering, resulting in $\frac{d}{2}(d+1)$ comparisons involving k multiplications each. The new approach chooses c block-representatives and then performs c greedy runs with d/c documents, resulting in $d + c(\frac{\frac{d}{c}(\frac{d}{c}-1)}{2}) = \frac{d}{2}(\frac{d}{c}+1)$ comparisons, so the overall number of operations is reduced in a $1/c$ factor. Experimental results with different values of the number of blocks c are presented in figure 2 and tables 6 and 7. Results are provided for the LATimes and FIBS collections, $k = 200$ and delta coding. Tables 6 and 7 also show that the compression factor increases as the number of blocks decreases, with a goal value of 6.29 (5.80) for the LATimes (FIBS) collection, which is the value of considering the matrix as one whole block.

Running times are as expected from the analytical form and comparable as the ones presented in [10], and they give acceptable compression values. The method enhances the original compression ratio 7.25 (6.35) and the randomly ordered collection ratio 7.65 (7.35) with a minimum time usage, which can also be parametrized by selecting the c and k values.

Table 6. LATimes bits per gap and running time (k=200 and delta coding)

	c								
	70	100	150	200	300	400	500	1000	2000
Bits per gap	6.68	6.72	6.77	6.81	6.87	6.92	6.95	7.02	7.09
c-GreedyNN & recompress	18m8s	9m50s	5m8s	3m21s	1m57s	1m25s	1m7s	42s	47s

Table 7. FIBS bits per gap and running time (k=200 and delta coding)

	c								
	70	100	150	200	300	400	500	1000	2000
Bits per gap	5.98	6.00	6.02	6.05	6.09	6.12	6.14	6.22	6.3
c-GreedyNN & recompress	17m37s	9m35s	4m59s	3m15s	1m53s	1m21s	1m5s	40s	45s

5 Future work

In a first experimental line, immediate research involves three task:

- experimentation with different heuristics for selecting the first document in the Greedy-NN algorithm
- experimentation with web collections
- implementation of the solutions presented in [10] using the low-dimension scaling presented here and analysis of the behavior in collections with more than 600 terms per document in average, such as the LATimes or FIBS.

Another future working line is the following: as long as the TSP in the reduced dimension space performs well, we may pursue a formal characterization to the distance of the optimal solution reached, with this sort of heuristic solutions. On the other hand, dividing the TSP graph into c blocks allows effective reordering without compromising the index reduction. So, it is necessary to study different block-reordering strategies.

6 Conclusions

We presented a smart approximation for the document identifier reassignment problem by using a previous dimensionality reduction with SVD. Results presented provide time-efficient methods that yields good inverted file reduction gains. Concretely, we implemented the TSP Greedy-NN approach in the reduced dimension space and one variant, that applies this solution to sub-collections of the original data, reordering them next. It is possible to emphasize that the obtained data allows the exposition of future lines of work, as the design of algorithms and heuristics that could provide a better characterization of the result respect to the optimum compressed inverted file.

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 *Proceeding 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. 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.
4. S. T. Dumais. Latent Semantic Indexing (LSI): TREC-3 Report. In *NIST Special Publication 500-225: Proceedings of the Third Text REtrieval Conference (TREC-3)*, November 1994.
5. `http://www.cs.mu.oz.au/mg/` Managing Gigabytes.
6. `http://mg4j.dsi.unimi.it/` MG4J (Managing Gigabytes for Java).
7. A. Moffat, A. Turpin. *Compression and Coding Algorithms*, Kluwer 2002.
8. R. Rivest, RFC 1321: The md5 algorithm.
9. 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.
10. 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.
11. `http://tedlab.mit.edu/~dr/SVDLIBC/` SVDLIBC.
12. 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.