Inf Retrieval (2006) 9:499–517
DOI 10.1007/s10791-006-6614-y

TSP and cluster-based solutions to the reassignment of document identifiers

Roi Blanco · Álvaro Barreiro

Received: 1 July 2005 / Revised: 9 December 2005 / Accepted: 9 December 2005

Abstract Recent studies demonstrated that it is possible to reduce *Inverted Files* (IF) sizes by reassigning the document identifiers of the original collection, as this lowers the distance between the positions of 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. This paper presents an efficient solution to the reassignment problem, which consists in reducing the input data dimensionality using a SVD transformation, as well as considering it a *Travelling Salesman Problem* (TSP). We also present some efficient solutions based on clustering. Finally, we combine both the TSP and the clustering strategies for reordering the document identifiers. We present experimental tests and performance results in two text TREC collections, obtaining good compression ratios with low running times, and advance the possibility of obtaining scalable solutions for web collections based on the techniques presented here.

Keywords Document identifier reassignment · SVD · Indexing · Compression · TSP · Clustering

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, organized 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 assume a document level

R. Blanco (✉) · A. Barreiro
IRLab. Computer Science Department, University of Corunna, Spain
e-mail: rblanco@udc.es

A. Barreiro
e-mail: barreiro@udc.es

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 studies have addressed the efficient encoding of document identifiers contained in each entry (Witten et al., 1999). Posting lists are stored as a sequence of differences between consecutive document 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 work has tried to increase the number of small d-gaps by reordering the document identifiers, hoping that the average length of the coded d-gaps would be lowered.

The reassignment of document identifiers has been addressed by two different strategies: splitting the collection with graph-partitioning or clustering algorithms and obtaining a new order which exploits locality (Blandford and Blelloch, 2002; Silvestri et al., 2004), and considering the task as a graph-layout problem (Shieh et al., 2003). In the work presented in Blanco and Barreiro (2005), the heuristic that considers the problem as a *Travelling Salesman Problem* (TSP) introduced in Shieh et al. (2003) allowed an efficient solution of the reassignment after reducing the input data dimensionality using a *Singular Value Decomposition* (SVD) transformation. The TSP-based strategy considers the reassignment of document identifiers as a graph-traversal problem and finds a path according to a global minimum. Although the strategy is not explicitly based on locality, it has obtained very appealing results in practice. As it is shown later in Section 2, this path is the one that maximizes the bitwise intersection of the document vectors. In Section 6 we give an explanation for the good behaviour of the TSP heuristic for this problem.

The main reason why a SVD reduction is useful in this context, is that the TSP technique needs to compute similarity values between documents for choosing the edges that will connect the nodes in the final path. Thus, SVD computation deals with a necessary efficiency matter given the dimensions of the space of terms by documents, while also being the best choice for matrix transformation, as it obtains the best similarity measure in a reduced-dimensionality space.

This paper extends the work presented in Blanco and Barreiro (2005) by reviewing and reimplementing two clustering algorithms previously adapted to the reassignment problem (Silvestri et al., 2004), in order to evaluate the two different approaches within the same conditions. In the proposal presented in Silvestri et al. (2004) the assignment of document identifiers is done *on the fly*, i.e. without the existence of a prior built inverted file. In fact, this is the main reason why the baseline for the evaluation of the approach is a random order and not the original collection ordering. In our work we measure the improvements after the reassignment of document identifiers with respect to the random order and the identity (original) collection. Then, we present a new technique that combines the two strategies in such a way that the best tradeoff between compression and efficiency is achieved.

The rest of the paper is organized as follows. Section 2 describes the TSP-based solutions in depth. Section 3 shows the way of reducing the dimension of the document similarity matrix by computing its Singular Value Decomposition and how we applied this result to the reassignment problem. Section 4 applies the SVD technique again, but after dividing the original problem into subproblems. We postpone the introduction of the clustering solutions and the technique that combines the TSP and clustering until Section 5, because the implementations, experiments and results of each technique are included in their corresponding section. The paper continues with the discussion and future work in Section 6, and ends with a conclusions section.

2. The TSP approach to the document identifiers reassignment problem

Given that we will illustrate the use of the dimensionality reduction technique for document reassignment with the TSP-based solutions, we briefly review the work in Shieh et al. (2003).

2.1. The document identifiers 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 cost in bits of coding the posting lists

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. (2003) 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 of 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) 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 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 weight in the TSP. The solution of 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.

When describing the heuristic approximation, we consider the TSP as obtaining the path that maximizes the similarity sum between consecutive documents. 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), by choosing the most similar vertex to

Table 1 Statistics of the pure TSP approach on two TREC collections reported by Shieh et al. (2003)

Collection	FBIS	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 h37′	23 h16′
Space cost	2.10 GB	2.17 GB

the last one 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

```
    Greedy-NN algorithm
1:  Input:
        The Graph $G$
        The Vertex set $V$
        The weighted Edges set $E$
2:  Output:
        A global path $P$ maximizing the similarity between vertexes
3:  Select the edge $e(v_i, v_j) \in E$ with the largest weight;
4:  Add $v_i$ and $v_j$ to $P$;
5:  $v_{last} \leftarrow v_j$;
6:  while $(|P| \neq |V|)$ do
7:      Choose $v_k \in V$ and $v_k \notin P$ such that $e(v_{last}, v_k)$ is maximal;
8:      Add $v_k$ to $P$;
9:      $v_{last} \leftarrow v_k$;
10: end while
11: return $P$;
```

(the remaining documents) involving the term size t of both documents. Therefore the overall complexity is $O(d^2t)$.

2.3. Implementation considerations

The TSP approximation for the identifier reassignment problem was evaluated in Shieh et al. (2003). The solution demonstrated significant 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 (FBIS and LATimes in TREC disk 5), to prove the effectiveness of this mechanism. These tests are summarized in Table 1.

It is important to remark that the work in Shieh et al. (2003) 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, running the greedy algorithm 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 identifiers reassignment by dimensionality reduction and the TSP strategy

The TSP strategy achieves very good compression ratios. For this reason, we propose a new approach based on dimensionality reduction in which the TSP algorithm can operate efficiently. This technique is based on the application of a SVD transformation to the input data, in order to obtain a new representation of the data structures that allows the reordering of the document identifiers in a dimensionality-reduced space. Once the new ordering is obtained, the posting lists in the inverted file are recompressed, using the document identifiers given by the assignment function. It is worth noting the different nature of this SVD application in Information Retrieval with respect to other typical SVD IR usages, like the LSI retrieval model.

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. These 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 (Deerwester et al., 1990).

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

$$X = T_0 S_0 D_0' \quad (2)$$

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, thus 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:

$$X \approx \hat{X} = TSD' \quad (3)$$

$\hat{X}$ 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 $\Theta(\hat{X})$:

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

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

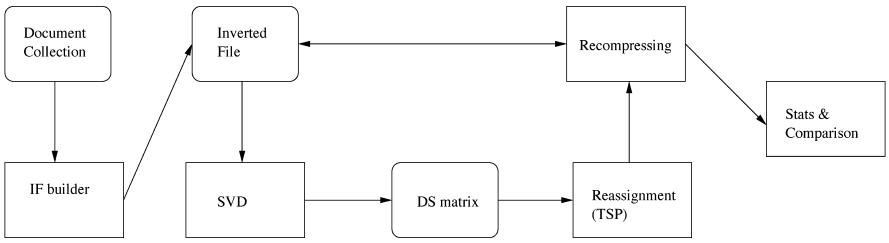

Fig. 1 Block diagram for the indexing and reassignment system

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

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

Therefore it is possible to calculate $\Theta(\hat{X})_{ij}$ solely by storing the k elements of the $\{s_i\}$ set 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 (Bartell et al., 1992). 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

Figure 1describes 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 it 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 is $O(d)$ now, i.e. linear in collection size and not dependent 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 into account to make suitable comparisons between compression ratios achieved by the original encoding and those obtained after reassignment.

The summarized steps are detailed next, where SVDGreedyNN is a modified version of the Greedy-NN algorithm that uses the $\Theta(\hat{X})$ matrix for computing the similarity measure between vertexes v_i and v_j.

SVD-based reassignment of document identifiers

```
1: Input:
       The Document Collection $C$
       The $k$ SVD parameter
2: Output:
       A reordered Inverted File
3: $originalIndex \leftarrow$ createIndex(C);
4: $reducedMatrix \leftarrow$ SVD($originalIndex$, $k$);
5: $documentOrder \leftarrow$ SVDGreedyNN($reducedMatrix$);
6: $newIndex \leftarrow$ recompress(C, $documentOrder$);
7: return($newIndex$);
```

The main difference in this model is that computing the similarity between two documents d_i and d_j requires 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. Furthermore, 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 involves 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.

3.3. 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 1 GB of main memory, running Linux OS. The original index file was built with MG4J from the University of Milan, a free Java implementation of the indexing and compression techniques described in Witten et al. (1999) and originally implemented in the MG software (http://www.cs.mu.oz.au/mg/). For the SVD module we used SVDLIBC (http://tedlab.mit.edu/∼ dr/SVDLIBC/), a C library based on the SVDPACKC library. We wrote the reassignment, recoding and statistical software in Java. It should be pointed out 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, golomb coding or interpolative coding (Moffat and Turpin, 2002).

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 within-document term frequency and frequency of the term in the collection. Results are given in bits per document gap because it is a measure independent of these indexing options. As stated in Section 3, 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).

We have experimented with different coding algorithms that had been successfully used for inverted file compression. These algorithms cover some of the different categories for static coding methods (Moffat and Turpin, 2002): two global non-parameterized methods (gamma γ and delta δ (Elias, 1975)), one global parameterized (Golomb code (Golomb, 1966)), and one local context-sensitive (interpolative coding (Moffat and Stuiver, 2000)).

Elias delta and gamma codes (Elias, 1975) do not take into account any data from the document collection, as they drive the same representation for an integer x regardless its frequency. For the gamma code the implied probability for a gap of length x is

$$Pr(x) = \frac{1}{2x^2}; \qquad (6)$$

for the case of delta code the probability is

$$Pr(x) = \frac{1}{2x(\log x)^2}. \qquad (7)$$

The associated code lengths for x are $1 + 2\lfloor \log x \rfloor$ and $1 + 2\lfloor \log\log x \rfloor + \lfloor \log x \rfloor$ respectively. On the other hand, global Golomb coding (Golomb, 1966) makes use of the density of pointers in the inverted file and is dependent on a parameter b. Golomb coding is optimal if the implied probability follows a geometric distribution:

$$Pr(x) = (1 - p)^{(x-1)} p. \qquad (8)$$

The code length for an integer x is $\lfloor (x - 1)/b \rfloor + \lfloor logb \rfloor$ bits. The b parameter can be chosen so the average number of bits for coding the inverted file is minimized (Gallager and van Voorhis 1975) and it depends on the geometric distribution. Finally, the interpolative coding (Moffat and Stuiver, 2000) is a context-sensitive model that takes into account the clustering properties of the collection. The behaviour of the interpolative code is nearly as efficient as the Golomb code in the worst case, although it can lead to greater gains of compression when coping with contiguous sets of integers.

Tables 2 and 3 show the results for the different coding schemes. In the rows labeled with each of the coding methods, columns refer to bits per document gap results for: random reassignment, original document identifiers and reassignment after reducing the dimensionality with different k values. Actually, the default starting point for any rearrangement is the pre-existing order, which already includes an element of clustering because of the

Table 2 LATimes bits per gap results and execution times

			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
Golomb	5.82	5.81	5.79	5.79	5.79	5.79	5.79	5.79	5.79
SVD	–	–	42′31″	20′37″	11′25″	4′05″	2′33″	1′55″	1′09″
Reorder	–	–	8h33′	4h24′	2h15′	58′18″	32′05″	18′31″	7′30″

Table 3 FBIS bits per gap results and execution times

			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
Golomb	5.61	5.60	5.59	5.59	5.59	5.59	5.59	5.59	5.59
SVD	–	–	34′58″	15′01″	5′42″	3′18″	2′09″	1′33″	58″
Reorder	–	–	8h30′	4h20′	1h48′	58′20″	31′13″	17′55″	7′05″

chronology of the documents. However, in previous studies (Blandford and Blelloch, 2002; Silvestri et al., 2004), the authors used the random reassignment as a baseline, and for comparison issues this column is included here as well. By assigning values to k similar to those used in retrieval (Dumais, 1994), the low-dimension algorithm operates with gains that produce improvements in bits per gap. As expected, the method behaves better as the k value increases. Also, the tables seem to show a asymptotic behaviour. With $k = 200$, for the LATimes collection (FBIS collection) we achieved a 13.65% (8.02%) gain in compression ratio with respect to the original document identifier order with the gamma encoding, 13.24%(8.66%) for the delta encoding, and 11.32% (5.15%) for the interpolative coding. These values are 17.67% (21.92%), 17.80% (21.10%) and 13.66% (14.58%) respectively for both collections and the three encoding schemes, with 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 upon is that even in the case of interpolative coding, where the starting point is much better, the method is able to produce gains in bit per document gap. Regarding the compression ratios with Golomb coding, it performed the best for the random assignment, and it is clear that it does not benefit from the locality of the documents. The reason for this behaviour is that the Golomb code is not affected by the skewness of the document distribution, as it renders a geometric distribution (Moffat and Turpin, 2002). However, interpolative coding with reordering is the option that achieves best compression results.

Our tests did not include the computation of the full dimension solution as presented in Shieh et al. (2003), because it requires the development of matrix partition techniques and partial reading/writing, which are the tasks we want to avoid. Shieh et al. (2003) provided bar graph results for gamma and delta encoding in the LATimes and FBIS collections. However, exact compression values depend 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.

The last two rows of Tables 2 and 3 are devoted to running times. Time measurement is divided in three parts: inverted file construction, SVD running time and reordering and recompressing time. Times are given for delta coding only, as the tables illustrate the time variation within the k parameter. The time variance due to k, is only dependent on SVD and reordering times, so times with delta coding only are suitable for this purposes (recompressing time is about 16 seconds for both collections). As the system was built upon different modules, the different software pieces employ a lot of temporal I/O transfer time, which is also measured, so results are given in *elapsed time*. Inversion takes $5'\ 20''$ for the FBIS collection and $6'\ 03''$ 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. Hence, we conclude that it is possible to achieve good compression ratios with reasonable time performance with our technique. In the rest of the paper, we will show results for one compression method, as the experiments are aimed at clarifying if the different reordering variants give better bits per gap values, along with their respective temporal cost, without focusing on coding methods.

4. The c-blocks algorithm

In order to further exploit the advantages of the TSP strategy and the dimensionality reduction, we developed a simple new algorithm based on the division of the original problem in c subproblems, hereinafter c-GreedyNN. Later on, we will be able to compare the performance between this algorithm and others based on collection partitioning.

The c-GreedyNN algorithm 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. 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 Tables 4 and 5.

Results are provided for the LATimes and FBIS collections, $k = 200$ and delta coding. Tables 4 and 5 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 (FBIS) collection, which is the value of considering the matrix as one single block.

The algorithm is efficient in running time, as expected from the analytical form, and it gives acceptable compression values. We ran all the experiments with $k = 200$, as it proved to drive the best performance results as well as the slowest reordering times, so lowering those times while keeping good compression ratios is a challenge for the c-blocks algorithm.

Table 4 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	18′08″	9′50″	5′08″	3′21″	1′57″	1′25″	1′07″	42′	47″

Table 5 FBIS 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	6.30
c-GreedyNN & recompress	17′37″	9′35″	4′59″	3′15″	1′53″	1′21″	1′05″	40″	45″

The method enhances the original compression ratio 7.25 (6.35) for the LATimes (FBIS) collections and delta coding. These improvements are higher when comparing to the randomly ordered collection compression ratio 7.65 (7.35). The tradeoff between the final compression achieved and the time usage, can be parameterized by selecting the c and k values.

5. Clustering strategies

5.1. Clustering solutions to the document identifiers reassignment problem

Recent work addressed the document identifiers reassignment problem with clustering techniques. The main idea is to enhance the locality in the collection by assigning similar documents to the same cluster, and hoping that the d-gaps are lowered if documents on the same cluster are assigned close identifiers. For this particular case, 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.

The first top-down algorithm was introduced by Blandford and Blelloch (2002), who developed a technique that improved the compression ratio in about 14% in TREC text collections. The technique employs a similarity graph as described in Section 2.1, 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 (Karypis and Kumar, 1995) algorithm for splitting recursively the similarity graph produced by the first part. Finally, the algorithm applies rotations to the clustered graph outputted by the second part in order to optimize the obtained order. The final assignment of the document identifiers is obtained by simply depth-first traversing the resulting clustering tree. As it is stated in Blandford and Blelloch (2002), constructing a full similarity graph is $O(n^2)$, so the raw technique may not be 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 a number of document occurrences $|t_i| > \tau$ 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 $\lfloor n^{\rho} \rfloor$. By tuning τ and ρ the technique may lead to a tradeoff between efficiency and time and memory usage.

The work in Silvestri et al. (2004) proposes a different approach by *assigning* the document identifiers *on the fly* during the inversion of the text collection. This is done by calculating a *transactional* representation form of the documents, which stores for each document d_i a set of 4-byte integers representing the MD5 Message-Digest (Rivest, 1992) of each term appearing in d_i. Using this representation, a new top-down algorithm called *bisecting* is presented and evaluated, as well as two bottom-up algorithms: the *k-means* and *k-scan*. These techniques were tested in the Google Programming Contest collection. As the whole collection is ordered while indexing, the baseline chosen for measuring the performance of these algorithms was a randomized collection. Our previous results with the FBIS and LATimes presented in Section 3.3, showed that randomly ordering the collection always yields worse compression ratios than those obtained with the *natural order*, i.e. leaving the collection with its original numbering of documents. This motivated us to reimplement these algorithms, but in a scenario of reassignment of document identifiers, with the existence of an inverted file, rather than performing the assignment on the fly.

In our implementation of the clustering algorithms the construction of the inverted file makes unnecessary the transactional hashing of the terms. In order to improve the computational times of the document-to-document similarities, we worked not only with the inverted file, but also with the direct file. This structure can be seen as an inverted file where the index keys are the documents, and the posting lists contain term identifiers. It is worth noting that this direct file is also compressed with the techniques used to compress the inverted file.

5.2. Bisecting and k-scan

In this section, we adapt the bisecting and k-scan algorithms from Silvestri et al. (2004) to the reassignment scenario. The bisecting algorithm belongs to the top-down family, and operates as follows: first, it randomly selects two documents as the centres of mass of a two-partition of the collection. Next, it assigns every document in the input data to one of the two partitions, according to the similarity between the document and the centre of mass. The technique keeps these two partitions equally sized, so many documents at the end of the input may fall into an undesired cluster. This partitioning scheme is called recursively until the input partition only contains one element. The merging phase is done by comparing the borders of the two partitions. If the similarity between the document at the right border of cluster D' and document at the left border of cluster D'' is less or equal than the similarity between the document at the right border of cluster D'' and document at the left border of cluster D', in the final order cluster D'' must precede D'. Otherwise D' must precede D''. In the pseudo-code, $\oplus$ represents the concatenation of two sets containing document identifiers.

The k-scan is a simplified version of the popular k-means clustering algorithm. The algorithm sorts the elements of the collection by descending document length, and chooses the largest unselected document as the cluster representative. Then it assigns the $|D|/k - 1$ unselected documents which are most similar to the cluster representative. There is an internal order in each cluster, given by this similarity measure. The selection of the cluster representative and the assignment of the documents to that cluster is repeated k times.

```
    Algorithm Bisecting($D$)
1:  Input:
        The document set $D$
2:  Output:
        A list with the final order
3:  if $|D| > 1$ then
4:      $c_1 \leftarrow$ randomDocument($D$);
5:      $c_2 \leftarrow$ randomDocument($D \setminus c_1$);
6:      $D' \leftarrow \{c_1\}$;
7:      $D'' \leftarrow \{c_2\}$;
8:      for all $d \in D \setminus \{c_1, c_2\}$
9:          if $(|D'| \geq \frac{|D|}{2}) \vee (|D''| \geq \frac{|D|}{2})$ then
10:             Assign $d$ to the smallest partition;
11:         else
12:             if similarity($c_1, d$) $\geq$similarity($c_2, d$) then
13:                 $D' \leftarrow D' \cup d$;
14:             else
15:                 $D'' \leftarrow D'' \cup d$
16:             endif
17:         endif
18:     endfor
19:     $D'_{\text{ord}} \leftarrow Bisecting(D')$
20:     $D''_{\text{ord}} \leftarrow Bisecting(D'')$
21:     if sim(right($D'_{\text{ord}}$),left($D''_{\text{ord}}$))
                  $\leq$ sim(right($D''_{\text{ord}}$),left($D'_{\text{ord}}$)) then
22:         $D_{\text{ord}} \leftarrow D''_{\text{ord}} \oplus D'_{\text{ord}}$
23:     else
24:         $D_{\text{ord}} \leftarrow D'_{\text{ord}} \oplus D''_{\text{ord}}$
25:     endif
26:     return $D_{\text{ord}}$
27: else
28:     return $D$;
```

The work in Silvestri et al. (2004) shows that the space storage and the computational cost of the bisecting algorithm are both superlineal $O(|D|\log(|D|))$ in the asymptotic analysis. For k-scan, the space occupied is lineal $O(|S||D|)$ and the complexity is $O(|D|k)$, where $|S|$ is the average document length.

5.3. Experiments and results

The experiments were designed to assess the performance of the clustering techniques and compare them to the TSP-based approaches, measuring the improvements in compression ratios with respect to both the original and randomized collections. The similarities between documents were measured with the Jaccard distance, repeating the conditions described in Silvestri et al. (2004). In order to exploit the SVD transformation previously computed, we repeated the experiments measuring the similarities between documents with the inner product in the reduced dimensionality space. In these cases, the k parameter in the SVD transformation was fixed to 200, as this value proved to be worthy (Sections 3.3 and 4). Hereinafter, the k parameter appearing in the tables refers to the number of scans of D in the k-scan algorithm. In this set of experiments the compression of the posting lists was done with delta coding.

```
    k-scan algorithm
1:  Input:
        The set of documents $D$
        The number of sequences $k$

2:  Output:
        $k$ ordered sequences representing the final order
3:  $result \leftarrow \oslash$
4:  Sort $D$ by descending lengths;
5:  for $i = 1 \ldots k$ do
6:      $center \leftarrow$ first($D$);
7:      for all $d_j \in D$ do
8:          sim[$j$] $\leftarrow$ similarity($center, d_j$);
9:      end for
9:      $M \leftarrow \frac{D}{k} - 1$ documents with higher $sim$ value;
10:     $result \leftarrow result \oplus \{center\} \oplus M$
11:     $D \leftarrow D \setminus (M \cup \{center\})$
12: end for
13: return $result$;
```

The bisecting algorithm achieves different gains in bits per gap in each run, due to the random centre selection for the different executions, although they are stable around a certain value. In 10 runs of the algorithm, for the LATimes collection the values obtained fall in the range [6.82, 6.87] for the Jaccard distance and [7.24, 7.32] for the inner product. Comparing the worse values in these ranges to the original (random) collection order 7.25 (7.65), the bisecting method obtains a gain in compression ratio of 5.24%(10.19%) for the Jaccard Distance and − 0.97% (4.30%) for the inner product.

The main point to consider is that the gains with respect to the natural order are very small, or they may not exist. This observation was ratified by the results of the bisecting algorithm in the FBIS collection, which we skip here. Comments on the difference between the results obtained with the two different measures of similarity for every experiment can be found at the end of this section.

Table 6(7) shows the final compression results measured in bits per gap for the LATimes (FBIS) collection for the k-scan algorithm. The first row presents the data obtained measuring the similarity with inner product and 200 dimensions in the reduced space, which are exactly the same conditions as those of the experiments with the c-blocks (Section 4) and the bisecting algorithms. The second row shows the results with the Jaccard distance. The k parameter stands for the number of scans of the algorithm. Overall, the results give good compression values for a large enough k value (1000). However, considering the variation of k, there is a point where the bits per document gap stop decreasing, and slightly increase until reaching the limit value where $k = |D|$, which is just ordering the collection by document length.

Table 6 k-scan LATimes bits per gap

	k										
	100	200	300	400	500	1000	2000	3000	30000	$\cdots$	131896
Inner product	6.79	6.72	6.70	6.67	6.66	6.62	6.60	6.60	6.70	$\cdots$	7.65
Jaccard	6.82	6.74	6.70	6.68	6.67	6.64	6.63	6.62	6.71	$\cdots$	7.65

Table 7 k-scan FBIS bits per gap

	k										
	100	200	300	400	500	1000	2000	3000	30000	$\cdots$	130471
Inner product	6.46	6.39	6.36	6.34	6.33	6.29	6.23	6.21	6.26	$\cdots$	7.35
Jaccard	6.54	6.47	6.44	6.41	6.39	6.29	6.23	6.22	6.27	$\cdots$	7.35

Table 8 k-scan + TSP LATimes bits per gap (delta coding)

	k				
	100	200	300	400	500
Inner product under 200 dimensions	6.34	6.35	6.36	6.37	6.37
Jaccard	6.25	6.28	6.29	6.30	6.30

Table 9 k-scan + TSP FBIS bits per gap (delta coding)

	k				
	100	200	300	400	500
Inner product under 200 dimensions	5.93	5.94	5.95	5.96	5.97
Jaccard	5.84	5.87	5.87	5.89	5.89

It is remarkable that the best values obtained with the k-scan algorithm in the LATimes collection (6.60) and the values given by the c-blocks algorithm are similar (6.68). This is not true for the FBIS collection, where the c-blocks (5.98) algorithm obtains a better value than the k-scan (6.23). In terms of gain with respect to the original compression ratio (6.35) this would be 1.9% vs 5.9%. These comparisons are fair, as the metric used by the two algorithms in this measure is exactly the same (inner product in the $k = 200$ space).

5.4. TSP and clustering combination

Finally, we present a technique that combines the best of the previous approaches: the real performance in bits per gap of the TSP and the efficiency in time consumption of clustering. The basic idea is the same as in the c-blocks algorithm (see Section 4), namely split the collection into several parts and run the TSP in each part, so the overall running time is lowered from several hours to just minutes. The difference lies in the way the blocks are chosen; instead of driving a *raw* partition of equally-sized blocks, we select each sub-collection with the k-scan algorithm. This way, it is expected that each partition holds similar documents, so the TSP would improve an already good result.

Tables 8 and 9 shows the results obtained for the LATimes and FBIS collection for this technique, respectively. The performance of the algorithm is approximately the same as running a Greedy-NN (TSP for the whole collection) with a 200 dimension reduced space (the best results obtained so far). One point to consider is that this technique obtained better results than the c-blocks in both collections. This proves that the TSP-based techniques perform well when the collection brings a *natural order* with a good compression ratio, which is harder to improve with any algorithm, but even more with the clustering approaches tested before. Also, the reordering times for the values of k shown are in the order of the c-blocks, because the partitions are size balanced. Of course, this also applies to the algorithms described in the previous section, although this particular technique performs a reorder of each sub-collection in addition to compute the clusters.

It is possible to notice that the Jaccard measure performs better than the inner product in the bisecting and k-scan + TSP algorithms, although the inner product behaves slightly better for the k-scan, especially with low values of k. The problem with the Jaccard distance is that it is an undefined measure in the SVD reduced space, because it is related to the intersection of the original terms. However, the results with the inner product are useful for comparing fairly the clustering and c-blocks techniques.

6. Discussion and future work

The work presented here shows empirically that the TSP heuristic is a good reordering strategy. This is supported by the implementations, tests and comparisons with clustering algorithms and in particular by the combination of the two main techniques. As a summary of the results presented throughout the paper, Figs. 2 and 3 show some of the results obtained by the algorithms described here. The plots give some details on the effectiveness and efficiency tradeoff that can be achieved with the TSP and cluster-based combinations. The x-axis shows reordering and recompressing time in a logarithmic scale, and the y-axis represents the bits

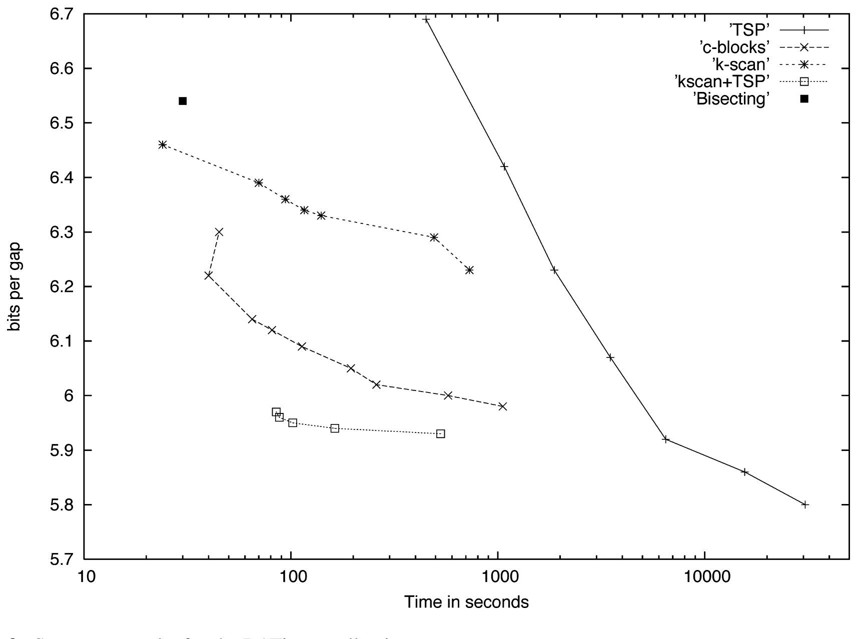

Fig. 2 Summary results for the LATimes collection

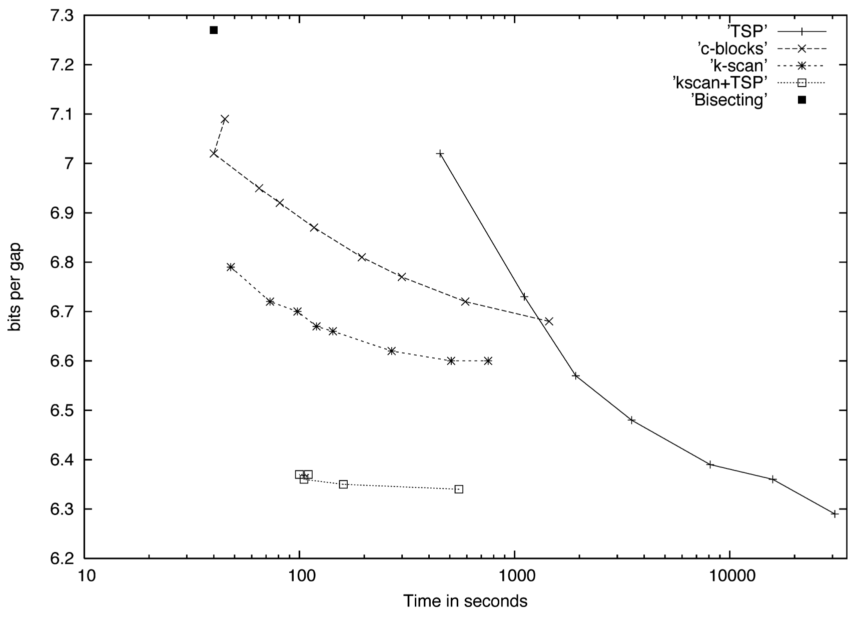

Fig. 3 Summary results for the FBIS collection

per gap achieved for delta coding. Each line in the plot stands for a different reordering technique, namely TSP (Section 2), c-blocks (Section 4), k-scan (Section 5.2), bisecting (Section 5.2) and the TSP and clustering combination using k-scan (Section 5.4).

For example, it is possible to achieve reduction in the compression ratios of 13.24% (6.29 bits per gap) (LATimes, delta coding) with a time cost of more than 8 hours driving a TSP algorithm only, while the TSP+k-scan combination gives a ratio of 12.55% (6.34 bits per gap) with much less computational effort (in the order of 10 min).

Despite of the fact that the TSP-based technique performs better than other approaches and that it can be adapted to other situations to fit time consumption and memory usage (SVD reduction, collection partitioning, and combination with clustering) it does not necessarily give the optimal solution to the problem of reassignment of document identifiers.

In order to find better solutions than those provided by the TSP strategy, it will be necessary to discover heuristics that take into account the exact cost function to be minimized. This minimization must allow for the coding scheme, as the aim is to minimize the function

$$\mathrm{cost}(\pi) = \sum_{t=1}^{T} \left(\mathrm{length}(\pi(d_{t,1})) + \sum_{i=1}^{f_t} \mathrm{length}(\pi(d_{t,i+1}) - \pi(d_{t,i})) \right) \quad (9)$$

where π is the reassignment bijective function that maps each document identifier in the range $\{1 \ldots d\}$ to a new one in the same interval, and length(x) gives the total number of bits needed for coding an integer x. Most static codes represent an integer x with $O(\log(x))$ bits. In order to obtain good compression ratios, we should minimize the d-gap products. This way, the log of the products and the sum of the logs would be minimal, with practical consequences in the final coded posting lists. In this context, considering the problem a TSP, although producing good results, is only a strategy to address the document identifier reassignment problem. However, it is possible an explanation of why it performs well in reducing the d-gap log.

Consider the distance matrix $L_{d \times d}$ where each l_{ij} measures the distance between documents i and j. Considering the documents as sets of terms, the distance can be measured as $|(d_i \cup d_j) \setminus (d_i \cap d_j)|$. The traversal found by the exact solution of the TSP minimizes the function $sumdist(\pi) = \sum_{i=1}^{D-1} l_{\pi(d_i)\pi(d_{i+1})}$. Imagine the inverted file as a $t \times d$ binary matrix B, where a one the position b_{ij} represents an occurrence of the term i in document j. For each row of this binary matrix, each time a 0–1 or 1–0 switch appears, $sumdist(\pi)$ increases by one, so the order that minimizes $sumdist(\pi)$ is the one which minimizes this number of switches. This would maximize the bitwise intersection of document vectors, but it does not imply minimizing the average d-gap logarithm sum. However, this ensures that the final configuration of the binary matrix has a big number of consecutive 1s in each row, so it is reasonable to expect a low average logarithm d-gap sum (as the logarithm does not penalize bigger d-gaps as much as it benefits from smaller ones).

It is possible to trace a new line of active research, coming up with new heuristics based on the optimal cost function and/or the characterization of the distance that a particular solution achieves with respect to the optimum. This also holds for TSP solutions: as long as this algorithm 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.

The combination of clustering and TSP would also hold the same scalability problem as the TSP-only approaches described in Shieh et al. (2003) in very large collections. The combination of TSP, SVD and clustering may be suitable for these cases, and it would be interesting to measure the gains that these techniques are able to achieve, as the clustering gives good results but they become higher by applying the TSP in each cluster. Moreover, this is especially indicated for collection partitioning techniques that may produce unbalanced clusters, such as the *k-means* algorithm (Jain et al., 1999). The negative effect of large clusters would probably be softened with the SVD, so the TSP technique may run with acceptable times.

Another scalability issue is the efficiency of the SVD step. As we have seen, the dimensionality reduction allows efficient implementations of the TSP-based approaches, but when dealing with web scalability, when several Gigabytes or even Terabytes are involved, it becomes necessary to come up with more efficient SVD implementations, see for example Kokiopoulou and Saad (2004), or other different matrix transformations, with similar properties as the SVD, but more efficient in computing time.

7. Conclusions

We presented a smart approximation for the document identifier reassignment problem by using a previous dimensionality reduction with SVD. The results presented allow to report time-efficient methods that yield 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. We also presented an implementation of two clustering techniques to the problem of reassignment and evaluated and compared them with TSP-based reordering. Finally, in the discussion section we showed a comparison of the different techniques and gave insights into both theoretical aspects and scalability issues of the proposed work, as well as future research lines.

Acknowledgments The work reported here was co-funded by the “Ministerio de Educación y Ciencia” and FEDER funds under research projects TIC2002-00947 and TIN2005-08521-C02-02 and “Xunta de Galicia” under project PGIDT03PXIC10501PN.

References

Bartell, B. T., Cottrel, G. W., & Belew, R. K. (1992). 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).

Blanco, R., & Barreiro, A. (2005). Document identifier reassignment through dimensionality reduction. In *Proceedings of the 27th European Conference on IR Research, ECIR 2005*, LNCS 3408 (pp. 375–387).

Blandford, D., & Blelloch, G. (2002). Index compression through document reordering. In: *Proceedings of the IEEE Data Compression Conference (DCC'02)*, pp. 342–351.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, *41*(6), 391–407.

Dumais, S. T. (1994). Latent Semantic Indexing (LSI): TREC-3 Report. In *NIST Special Publication 500-225: Proceedings of the Third Text REtrieval Conference (TREC-3)*, November 1994.

Elias, P. (1975). Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*, *21*(2), 194–203.

Gallager, R., & van Voorhis, D. (1975). Optimal source codes for geometrically distributed integer alphabets. *IEEE Transactions on Information Theory*, *21,* 228–230.

Golomb, S. (1966). Run-length encodings. *IEEE Transactions on Information Theory*, *12,* 399–401.

`http://mg4j.dsi.unimi.it/` `MG4J` (Managing Gigabytes for Java).

`http://tedlab.mit.edu/~dr/SVDLIBC/` SVDLIBC.

`http://www.cs.mu.oz.au.mg/` Managing Gigabytes.

Karypis, G., & Kumar, V. (1995). *A Fast and High Quality Multilevel Scheme for Patitioning Irregular Graphs* (Tech. Rep. No. TR 95-035). Department of Computer Science, University of Minnesota,

Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). Data clustering: A review. *ACM Computing Surveys*, *31*(3), 264–323.

Kokiopoulou, E., & Saad, Y. (2004). Polynomial filtering in latent semantic indexing for information retrieval. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 104–111.

Moffat, A., & Turpin, A. (2002). *Compression and coding algorithms*, Kluwer.

Moffat, A., & Stuiver, L. (2000). Binary interpolative coding for effective index compression. *Information Retrieval*, *3*(1), 25–47.

Rivest, R. (1992). RFC 1321: The md5 algorithm.

Shieh, W.-Y., Chen, T.-F., Shann, J. J.-J., & Chung, C.-P. (2003). Inverted file compression through document identifier reassignment. In *Information Processing and Management*, *39*(1), 117–131.

Silvestri, F., Orlando, S., & Perego, R. (2004). Assigning identifiers to documents to enhance the clustering property of fulltext indexes. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 305–312.

Witten, I. H., Moffat, A., & Bell, T. C. (1999). *Managing gigabytes—Compressing and indexing documents and images* (2nd edn.). San Francisco: Morgan Kaufmann Publishing.