Sorting Out the Document Identifier Assignment Problem

Fabrizio Silvestri

Institute for Information Science and Technologies
ISTI - CNR, via Moruzzi, 1, 56126 Pisa, Italy
Italy
fabrizio.silvestri@isti.cnr.it

Abstract. The compression of Inverted File indexes in Web Search Engines has received a lot of attention in these last years. Compressing the index not only reduces space occupancy but also improves the overall retrieval performance since it allows a better exploitation of the memory hierarchy. In this paper we are going to empirically show that in the case of collections of Web Documents we can enhance the performance of compression algorithms by simply assigning identifiers to documents according to the lexicographical ordering of the URLs. We will validate this assumption by comparing several assignment techniques and several compression algorithms on a quite large document collection composed by about six million documents. The results are very encouraging since we can improve the compression ratio up to 40% using an algorithm that takes about ninety seconds to finish using only 100 MB of main memory.

1 Introduction

Indexes in Web Search Engines (WSEs) are usually represented using the popular *Inverted File* (IF) data structure [15]. Given a set of documents, an IF is composed by two distinct sets: the *Lexicon* and the *Posting Lists*. The *Lexicon* represents the set of terms that can be found within the whole document set. To each term of the lexicon a *Posting List* is associated containing information (the so-called *posting*) on all the documents containing that term. For example, the index entry $< t_1; 5; 3, 4, 10, 20, 23 >$ states that term t_1 (stored within the Lexicon) appears in five documents, namely 3, 4, 10, 20, and 23. The set containing all these lists is stored within the Posting Lists section.

One of the main reasons why IFs (or one of their variations) are usually adopted in real world WSEs, is that they can be easily compressed to reduce memory occupancy. Compressing indexes in WSEs has been also proved to enhance efficiency of the retrieval process [2, 11, 14]. A reduction in space occupancy, in fact, usually corresponds to a better utilization of the memory hierarchy.

The majority of the techniques adopted for compressing IFs are based on their d-gapped representation [15]. Posting lists are usually scanned sequentially. For this reason, it is possible to represent those lists by taking differences

among successive identifiers (with the obvious exception of the first one). This way, the previous list would be represented as $< t_1; 5; 3, 1, 6, 10, 3 >$. Encoding each integer with a technique requiring few bits for smaller values will result in a reduction of the utilized space. Variable-length encoding schemata allow IF indexes to be represented concisely since small d-gaps are much more frequent than large ones. This feature of posting lists, usually called *Clustering property* [7] is passively exploited by compression algorithms. However, by mapping DocIDs in a way that increases the frequency of small d-gaps, it is very likely that we can enhance the effectiveness of any variable-length encoding scheme [13]. It has been shown that an effective way to compute such a mapping is clustering the collection of documents and computing the mapping by considering the way documents are grouped within the clusters.

Indeed, clustering is an expensive operation especially in a highly dimensional domain like textual documents. Even if a number of scalable clustering techniques are known, a cheaper approach would be desirable.

As often happens when dealing with large scale WSEs, the simplest solution usually results to be the most effective one. What we are going to empirically validate in this paper is, in fact, that a very effective mapping can be obtained by considering the sorted list of URLs referencing the web documents of the collection. Furthermore, it is very likely that this lexicographical ordering of the URLs is already used by link databases; we will show, then, that it would introduce benefits also for assigning numerical identifiers to documents in Web Search Engines indexes.

As already pointed out in another paper [10], numbering the URLs following their lexicographical order improves the performance of web graph compression algorithms. In the following we are going to show that using this kind of numbering scheme we can also obtain better compression ratios by spending just a few seconds even on a very large number of documents. In fact, sorting a list of 5.9 million URLs would take about 90 seconds against about 45 minutes needed by our previous clustering solutions.

While the motivations presented in [10] clearly explain why sorting URLs can improve compression effectiveness for Web graphs, it is not so easy how to argument why sorting URLs can improve index compression too.

Luhn's hypothesis [9] can help in finding a reasonable motivation to this phenomenon. Luhn's hypothesis states that the significance of a word can be effectively measured by its frequency of occurrence. In particular, Luhn stated that words occurring less (more) than a given cut-off value are not significant as they are too rare (too common).

It is reasonable to say that lists referring to common terms are highly compressible since they contain relatively small d-gaps. On the other hand, rare terms occur only once or twice and do not impact on compression performance.

The lists we should care about are instead those whose length falls between the two Luhn's cut-offs. These lists, in fact, are likely to be referring to terms that are either highly correlated (e.g. "new" and "york"), or highly uncorrelated (e.g. "loudspeakers", and "octopussy").

At least in principle, by placing documents containing correlated terms closer and by separating documents containing uncorrelated terms, we should observe a gain in compressibility of postings. Unfortunately this problem has been shown to be NP-complete [3], and several heuristics have been proposed in the past [12, 5, 13, 4].

Our hypothesis is that *documents sharing correlated and discriminant terms are very likely to be hosted by the same site thus will also share a large prefix of their URLs.*

To partially evaluate the validity of our hypothesis we performed a simple experiment. We sorted the URLs identifying the documents of our test collection, and we took the first 35,000 documents. Then, between all the possible pairs of documents we measured the similarities using the Jaccard measure, and the similarity among their relative URLs by counting the number of tokens in common (i.e. the similarity among http://www.aaa.bbb.cc/people/n1.surname1 and http://www.aaa.bb.cc/people/n2.surname2 is 2 since they share the server part and the first subdirectory). Among those pairs of URLs with similarity 0, the large majority (about 89.7%) have also Jaccard similarity equal to 0. The document similarity increased by considering URLs with similarity equal to 1 and 2. In the former case the 18.1% of documents are at distance 0.1, in the latter the large majority of document pairs (52.1%) have similarity equal to 0.9 meaning that they share a large number of terms. This simple experiment empirically shown that the two measures might be linked in some manner.

In our opinion, the most important strength point of this paper is the simplicity of the algorithm employed. Instead of thinking of sophisticated metric, or algorithms, to measure the distance between documents, in the case of Web collections we simply need to sort the URL list lexicographically, and assign identifiers to documents accordingly.

The paper is organized as follows. Section 2 reviews the state-of-the-art on the assignment problem and analyzes pros and cons when compared against our solution. Section 3 briefly recalls the assignment problem definition and the main definitions that will be used throughout this paper. Section 4 shows the results of our empirical evaluation of the URL sorting assignment heuristics. Finally Section 5 concludes the paper, and present some issues that we are going to face in the future.

2 Related Work

In recent years several works have discussed approaches to the encoding of lists of integer values. These techniques have usually been applied to the compression of full-text indexes represented by means of d-gapped inverted lists [15]. Only recently, though, some works have been done in order to enhance the compressibility of indexes through a clever assignment of identifiers to documents [12, 5, 13, 4].

Shieh *et al.* [12] proposed a DocID reassignment algorithm adopting a Travelling Salesman Problem (TSP) heuristic. A *similarity* graph is built by consider-

ing each document of the collection as a vertex and by inserting an edge between any pair of vertices whose associated documents share at least one term. Moreover, edges are weighted by the number of terms shared by the two documents. The TSP heuristic algorithm is then used to find a cycle in the *similarity* graph having maximal weight and traversing each vertex exactly once. The suboptimal cycle found is finally broken at some point and the DocIDs are reassigned to the documents according to the ordering established. The rationale is that since the cycle preferably traverses edges connecting documents sharing a lot of terms, if we assign close DocIDs to these documents, we should expect a reduction in the average value of d-gaps and thus in the size of the compressed IF index. The experiments conducted demonstrated a good improvement in the compression ratio achieved. Unfortunately, this technique requires to store the whole graph in the main memory, and is too expensive to be used for real Web collections: the authors reported that reordering a collection of approximately $132,000$ documents required about 23 hours and 2.17 GBytes of main memory.

Blelloch and Blandford [5] also proposed an algorithm (hereinafter called *B&B*) that permutes the document identifiers in order to enhance the clustering property of posting lists. Starting from a previously built IF index, a similarity graph G is considered where the vertices correspond to documents, and the edges are weighted with the *cosine similarity* [8] measure between each pair of documents. The *B&B* algorithm recursively splits G into smaller subgraphs $G_{l,i} = (V_{l,i}, E_{l,i})$ (where l is the level, and i is the position of the subgraph within the level), representing smaller subsets of the collection. Recursive splitting proceeds until all subgraphs become singleton. The DocIDs are then reassigned according to a *depth-first* visit of the resulting tree. The main drawback of this approach is its high cost both in time and space: similarly to [12] it requires to store the whole graph G in the main memory. Moreover, the graph splitting operation is expensive, although the authors proposed some effective sampling heuristics aimed to reduce its cost. In [5] the results of experiments conducted with the TREC-8 ad hoc track collection are reported. The enhancement of the compression ratio obtained is significant, but execution times reported refer to tests conducted on a sub-collection of only $32,000$ documents. The paper addresses relevant issues, but due to its cost, the *B&B* algorithm also seems unfeasible for real Web collections. Several transactional-model-based solution have been compared against *B&B* in [13] and they showed that there is actually room for a lot of improvements to their method. In particular, not from an effectiveness point of view, but more from a scalability perspective.

In [4], Blanco and Barreiro studied the effect of dimensionality reduction on reassignment algorithms based on the Greedy-NN TSP algorithm. Basically, they first reduce dimensionality of the input matrix through a Singular Value Decomposition (SVD) transformation and then they apply the Greedy-NN TSP algorithm. They also tested the effect of *blocking* (i.e., the division of the dataset in subsets) on the effectiveness of the algorithm. Results are very good and they were able to obtain good compression ratios with low running times. Even thought the results are good, using the SVD transformation on the matrix might

spoil the scalability of the method. Indeed, the block partitioning approach proposed seems to reduce this effect but costs quite a lot in terms of effectiveness degradation.

In our opinion, another drawback of all the previous approaches is that they focus on *reassigning* DocIDs appearing in a previously built IF index. The strength point of this work, however, is that DocIDs are assigned on the fly, before (and not either during, or after) the inversion of the document collection. In order to compute efficiently and effectively a good assignment, a new model to represent the collection of documents is needed. In a previous work [13], there have been presented some results relative to four assignment algorithms based on clustering. The clustering approach resulted to be scalable and space-effective. This means that it can be used to assign DocIDs even before the spidered collection will be processed by the Indexer. Thus, when the index will be actually committed on disk, the new DocID assignment will be already computed. Conversely, the other methods proposed so far require that the IF index has already been computed in advance. They also proposed a new model that allowed the assignment algorithm to be placed into the typical spidering-indexing life-cycle of a WSE. The *Transactional Model* was based on the popular *bag-of-words* model. This work remain valid for generic textual document collections. Though, the main concern about this kind of techniques within Web collections is that usually document identifiers are used also for other purposes, like for instance addressing snippets, retrieve ranking information, and so on. Renumbering the collection, thus, may not be feasible in practice for large scale WSEs.

The solution of considering the sorted list of URLs has already been used in [10] to enhance the compressibility of Web Graphs. Web graphs may be represented by using adjacency lists. They are basically lists that contain, for each vertex v of the graph, the list of vertices directly reachable from v. It has been observed that almost 80% of all links are local, that is, point to pages of the same site. Starting from this observation, it is obvious that assigning closer identifiers to URLs referring to the same site will result in adjacency lists that will contain the around 80% of ids very close among them. Representing these lists using a d-gapped representation will thus lead to d-gapped adjacency lists having long runs of 1's. Starting from this assumption, in [10] and [6], authors show that exploiting the lexicographical ordering of URLs leads to an enhancement in performance of Web graphs encoding algorithms.

The main matter of this paper is an *evaluation of the compression algorithms efficiency, when an inverted index is built over a collection whose numerical identifiers are assigned according to lexicographically sorted URLs.*

3 Assignment of DocIds

Basically, the previously proposed clustering-based (re)assignment algorithms were trying to reduce the average gap value by clustering together documents having a number of terms in common. The distance measure used in clustering

was thus based on this concept of number of shared terms and the complexity of clustering algorithms depended on the number of distance computations[1]. The complexity is generally linear ($O(|D|)$) in the case of k-means based clustering, or superlinear ($O(|D| log |D|)$) in the case of hierarchical clustering methods (like *B&B*). Indeed, this complexity results consider distance computations as a constant complexity ($O(1)$) operation. Computing the distance betweverticesen two documents, in fact, means finding the intersection among two sets of term, and this is clearly not a cheap operation.

The algorithm we are going to present instead is indeed trivial, since it just consists of sorting the list of URLs, yet very efficient, since it does not require any set intersection operations. The computational complexity of this approach is $O(|D| log |D|)$. Differently from the clustering methods, the complexity is expressed as the number of string comparisons instead of number of set intersections. Furthermore, sorting the list of URLs is very effective from the compression ratio point of view. Another non-trivial advantage of this solution with respect to the previously proposed ones, is that a lot of scalable external memory string sorting algorithms exists while, currently, no assignment (or even re-assignment) algorithms have been proposed using external memory based techniques. One could obviously use one of the many external memory clustering solutions that exists in literature, but these will still require a lot of time to complete their operations.

Throughout the rest of this paper we will compare our assignment strategy based on URLs sorting against the previously proposed k-scan algorithm [13].

4 Experiments

We experimented our solution on an index built upon the WBR99 collection. WBR99 consists of 5,939,061 documents, about 22 GB uncompressed, representing a snapshot of the Brazilian Web (domains .br) as spidered by www.todobr.com.br. As the hardware platform, we used a Pentium IV 3.2GHz, with 1GB of RAM, local disk, and Linux as the operating system.

The tests we perform are aimed at showing the superiority of our approach with respect to the clustering approach. We will show the improvements in both compression ratio, reordering time, and space consumed of various encoding algorithms.

We considered several encoding schemata in our experiments:

- Elias' Gamma (GAMMA);
- Elias' Delta (DELTA);
- Golomb Code (GOLOMB);
- Variable Byte (VB);
- Interpolative Coding (INTERP);

[1] The distance measure used was the *Jaccard* distance that depends on the cardinality of the intersection between the set of terms contained within two documents A, and B, and on the cardinality of their union. $d(A,B) = 1 - \frac{|A \cap B|}{|A \cup B|}$.

– Simple9 (S9).

Simple9, hereinafter S9, is the encoding scheme described in [1]. Gamma, Delta, Golomb, Interpolative (INTERP) and Variable Byte (VB) are five popular encoding schemata described in [15].

In order to assess the validity of our method we performed our experiments by comparing not only different encoding methods but also different orderings for assigning document IDs. The orderings compared are the following:

– A random ordering (*Random*). For each document an ID is assigned u.a.r. by considering the set of previously unassigned IDs.
– The original numbering of documents given to documents in the index (*Original*).
– Block, and transactional-model based k-scan clustering (*Clustering*). In this case k, the number of clusters, has been set equal to 900, and the block size has been set equal to $900,000$ documents. Since this kind of clustering is sensitive to initial ordering [13], we fed the algorithm with the ordering found in the original index.
– URL-based sorting (*Url sorting*).
– k-scan clustering of the document collection again using $k = 900$, and $900,000$ documents for each block (*Clustering + Url Sorting*). The sorted list of URLs has been taken as the initial ordering of the documents.

4.1 Results

Regarding the enhancements in compression ratios, Table 1 and Figure 1 show comparisons between six different compression ratios and five different orderings.

Table 1. Results of various assignment algorithms using different encoding schemata. In bold are represented the best results obtained for each encoding schemata.

	VB	GAMMA	DELTA	S9	INTERP	GOLOMB
Random	11.4	12.72	12.71	15.41	11.13	11.31
Original	11.25	12.34	12.32	15.2	10.94	11.12
Url sorting	**9.72**	**7.72**	**7.69**	14.34	7.48	**8.23**
Clustering	9.81	8.82	8.8	**14.03**	**7.26**	8.63
Clustering + Url sorting	10.03	8.96	8.95	14.15	7.31	8.9

In Table 1, the best compression ratio achieved by each methods and for each ordering schema has been represented in bold. As we expected the compression ratio in the case of identifiers assigned by using our URL sorting method performs better than clustering in almost all the cases except for S9, and INTERP. Anyway, the enhancements in terms of compression ratio of the URL sorting method against the Clustering one is visible only in the case of Gamma and

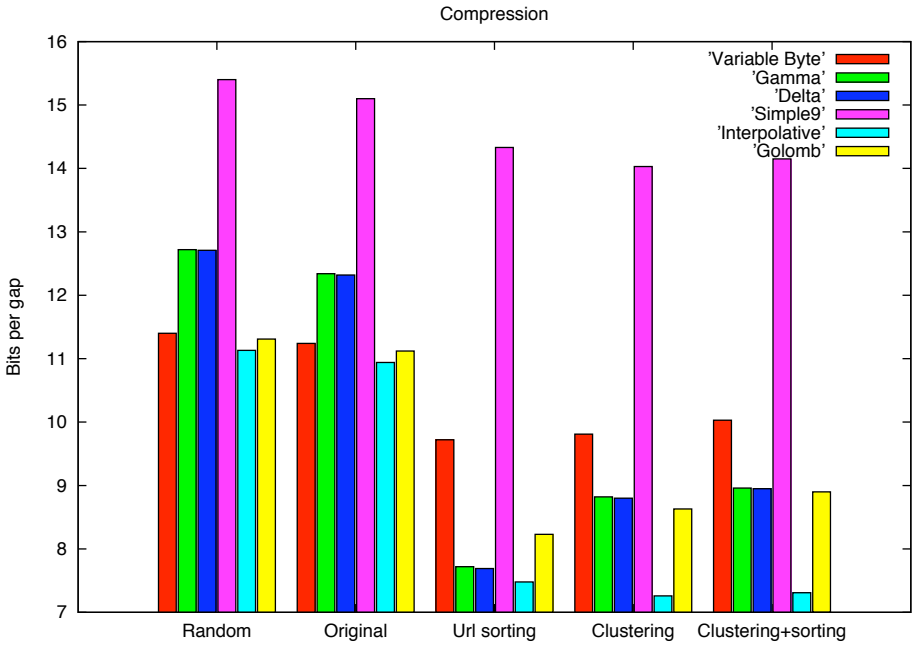

Fig. 1. Compression ratios of six different encoding techniques when applied to five different orderings.

Delta. Here the URL sorting has produced a posting list section that is 13% smaller than the Clustering ordering. In the remaining cases Clustering and URL-ordering are comparable.

Since Clustering is sensitive to the initial ordering, we also tested a hybrid solution consisting in performing Clustering on the list of documents ordered according to their URLs. Contrarily to what we expected this method did not perform better neither than URL sorting, nor than Clustering (except for S9 and INTERP). We have not found a good explanation of the reason why this happens, we reserve to better investigate this issue in the near future. Another positive result is that differently from what was observed in [13] VB can be improved.

To further confirm our results we also measured the distribution of d-gaps within three different ordering: original, clustering and URL sorting. Figure 2 reports the three distributions.

As the histogram shows, the number of very small d-gaps dramatically increases in clustering and URL sorting based ordering with respect to the original ordering. In particular the number of d-gaps equal to 1 and 2 increases in the cases of URL sorting from around $75,000,000$ to $325,000,000$. The URL sorting successfully increases the number of small d-gaps up to the d-gap equal to seven

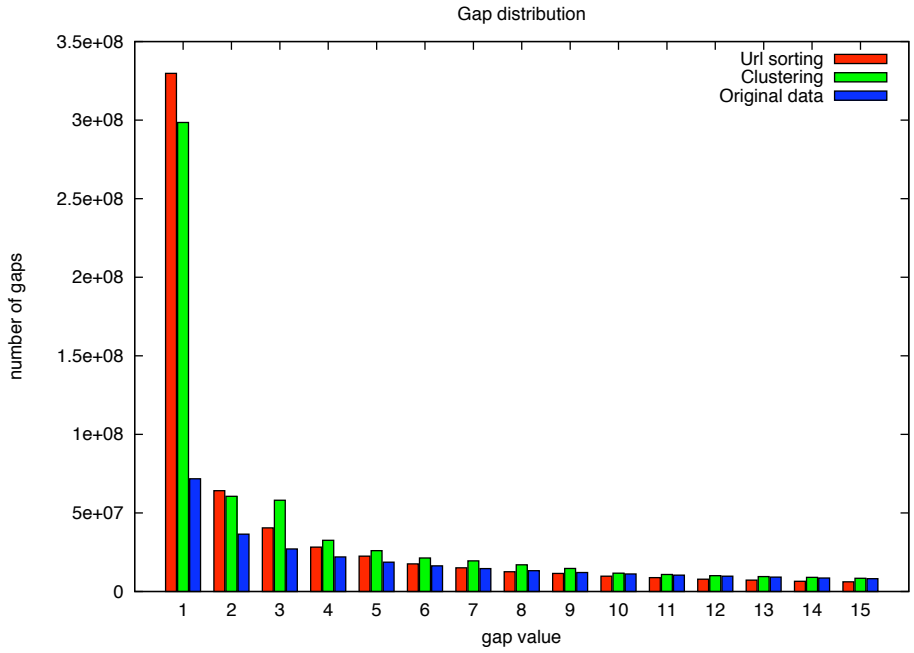

Fig. 2. Distribution of the d-gaps within the index organized according to the three different orderings: original, clustering and URL sorting.

and decreases the other. The clustering schema, instead, successfully increases the small d-gaps up to the d-gap equal to fifteen. This is the main reason why URL sorting is slightly better than clustering. This observation also confirms our hypothesis made in Section 3. *When we reduce the average gap, the resulting IF results smaller almost independently from the encoding method actually adopted.*

As we have seen, compression efficiency gains are comparable in most cases to those obtained by clustering. The main improvement, though, is in the resources consumed by our novel assignment algorithm compared to those needed by solutions based on clustering. As already said sorting a list of about six million URLs took just ninety seconds on the testing platform to complete and occupied just 95MB of main memory. On the other hand, clustering the same collection of documents required the partitioning of the collection into seven blocks of $900,000$ documents each (except for the last block composed by around $600,000$ documents), each block took about 352 seconds, for a total of about 45 minutes of CPU-time. The memory occupancy in the case of clustering has been around 1.2GB. It is thus clear that, while the ordering algorithm can scale to billions of documents without any particular problems, Clustering solutions cannot afford to achieve the same performance figures. In fact, as Table 2 shows, reducing the block size to $100,000$ documents will result in a slightly reduction of total completion time around 100 seconds less than the case with blocks of $900,000$

documents. In all the experiments the number of total clusters has been kept equal to $1,000$.

Table 2. Time (in seconds) spent for clustering each block of documents.

	Block Size				
	100,000	300,000	500,000	700,000	900,000
Time (s)	39	119	197	274	352

On the other hand a reduction of block size will impact in compression performance. Table 3 shows the relation among block size and compression ratio.

Table 3. Bits per gap of various encoding schemata when the collection is reordered using Clustering, and by varying block size.

	Block Size				
	100,000	300,000	500,000	700,000	900,000
VB	11.24	10.9	10.56	9.98	9.81
GAMMA	12.32	12.18	10.83	9.03	8.82
DELTA	12.32	12.17	10.81	9.02	8.8
S9	15.01	14.91	14.46	14.11	14.03
INTERP	10.89	9.23	8.01	7.62	7.26
GOLOMB	11.07	10.79	9.98	9.11	8.63

As it is possible to observe in the table, by reducing the number of documents for each block, the compression ratio dramatically decreases. By halving the size of the blocks, in fact, almost all of the methods loose performance between one and two bits per each posting. For example, in the case of GAMMA, passing from $900,000$ documents per block to $500,000$ documents per block, the number of bits per posting increases from 8.82 to 10.83 that is about 23% worse.

To be more precise, we should have performed our tests comparing the sorting technique against a clustering technique not requiring the prior partitioning of the collection into blocks. To date, there have not been proposed any external memory document assignment algorithm. Actually one could think about using one of the many out-of-core clustering methods that exist in the literature. Anyway, these clustering methods will be even slower than the blocked solution and is not really clear whether the compression ratios will increase further.

5 Conclusions and Future Works

We have shown that a simple sorting of the list of URLs associated to a collection of Web documents is very effective and results to be very fast. In all the experiments performed the compression ratio has increased by numbering the documents according to the sorted list of URLs.

This fully confirms our initial claim. *Ordering of the URLs used by link databases introduces benefits also if the same ordering would be used for assigning numerical identifiers to documents in Web Search Engines indexes.*

The benefits of sorting, when compared to clustering, are multiple. Compression ratios are about 5% better. The time needed to compute the assignment is two orders of magnitude smaller than the time needed by clustering. The space occupied by the clustering algorithms is dramatically bigger than the space needed to sort a list of URLs. In our tests, clustering used around one KB per each document assigned, requiring the prior subdivision of the collection in blocks. Furthermore, the URL-ordering method is more scalable to collections of even billion of URLs.

Nevertheless, due to the very limited time consumed by the sorting algorithm, one may think about placing the assignment module before the indexing phase will actually took place.

In conclusion, the URL sorting technique is the most efficient technique for assigning docIDs in the case of Web Search Engines when considering the classic time-space trade-off. In the other cases, for instance desktop search, enterprise search, email search systems, where URL information are not available, we might use folder's names or e-mail threads. Anyway, if none of this information is available clustering is still viable.

Some points remain to be investigated further. An important issue, when dealing with encoding methods is the time spent in decoding lists, and thus in resolving queries. So far, methods based on the Variable Byte schema (i.e. byte-aligned methods) have been shown to be the most effective, offering the best trade-off between decoding speed, and space occupancy. From what it can be seen in Figure 1, Gamma, and Delta scheme, now, has compression ratios far better than those of Variable-Byte. This could mean that the Delta schema may become that of reference for compressing posting lists in Web Search Engines. In the near future, we are going to evaluate the decoding speed of the various encoding scheme in light of this new ordering schema on a real IR system. It should be pointed out, in fact, that in real IR systems IF indices contain information about term frequencies and term positions. This data obviously affects the performance (in terms of retrieval time) of a retrieval system and can be only measured experimentally.

References

1. Vo Ngoc Anh and Alistair Moffat. Inverted index compression using word-aligned binary codes. *Inf. Retr.*, 8(1):151–166, 2005.

2. Vo Ngoc Anh and Alistair Moffat. Simplified similarity scoring using term ranks. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on research and development in information retrieval*, pages 226–233, New York, NY, USA, 2005. ACM Press.
3. Roi Blanco and Alvaro Barreiro. Characterization of a simple case of the reassignment of document identifiers as a pattern sequencing problem. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on research and development in information retrieval*, pages 587–588, New York, NY, USA, 2005. ACM Press.
4. Roi Blanco and Alvaro Barreiro. Document Identifier Reassignment Through Dimensionality Reduction. In *Advances in Information Retrieval: 27th European Conference on IR research, ECIR 2005, Santiago de Compostela, Spain, March 21-23, 2005. Proceedings*, pages 375 – 387, 2005.
5. Dan Blandford and Guy Blelloch. Index compression through document reordering. In *Proceedings of the Data Compression Conference (DCC'02)*, pages 342–351, Washington, DC, USA, 2002. IEEE Computer Society.
6. P. Boldi and S. Vigna. The webgraph framework i: compression techniques. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 595–602, New York, NY, USA, 2004. ACM Press.
7. A. Bookstein, S. T. Klein, and T. Raita. Modeling word occurrences for the compression of concordances. *ACM Trans. Inf. Syst.*, 15(3):254–290, 1997.
8. Chris Buckley. Implementation of the smart information retrieval system. Technical Report TR85–686, Cornell University, Computer Science Department, May 1985.
9. H. P. Luhn. The Automatic Creation of Literature Abstracts. *IBM Journal of Research Development*, 2(2):159–165, 1958.
10. Keith H. Randall, Raymie Stata, Janet L. Wiener, and Rajiv G. Wickremesinghe. The link database: Fast access to graphs of the web. In *DCC '02: Proceedings of the Data Compression Conference (DCC '02)*, page 122, Washington, DC, USA, 2002. IEEE Computer Society.
11. Falk Scholer, Hugh E. Williams, John Yiannis, and Justin Zobel. Compression of inverted indexes for fast query evaluation. In *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on research and development in information retrieval*, pages 222–229, New York, NY, USA, 2002. ACM Press.
12. Wann-Yun Shieh, Tien-Fu Chen, Jean Jyh-Jiun Shann, and Chung-Ping Chung. Inverted file compression through document identifier reassignment. *Information Processing and Management*, 39(1):117–131, January 2003.
13. Fabrizio Silvestri, Salvatore Orlando, and Raffaele Perego. Assigning identifiers to documents to enhance the clustering property of fulltext indexes. In *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on research and development in information retrieval*, pages 305–312, New York, NY, USA, 2004. ACM Press.
14. A. Trotman. Compressing inverted files. *Information Retrieval*, 6(1):5–19, January 2003.
15. Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes – Compressing and Indexing Documents and Images*. Morgan Kaufmann Publishing, San Francisco, second edition edition, 1999.