Entry Pairing in Inverted File

Hoang Thanh Lam[1], Raffaele Perego[2],
Nguyen Thoi Minh Quan[3], and Fabrizio Silvestri[2]

[1] Dip. di Informatica, Università di Pisa, Italy
lam@di.unipi.it
[2] ISTI-CNR, Pisa, Italy
{r.perego,f.silvestri}@isti.cnr.it
[3] Lomonosov Moscow State University, Russia
ntmquan@yahoo.com

Abstract. This paper proposes to exploit content and usage information to rearrange an inverted index for a full-text IR system. The idea is to merge the entries of two frequently co-occurring terms, either in the collection or in the answered queries, to form a single, paired, entry. Since postings common to paired terms are not replicated, the resulting index is more compact. In addition, queries containing terms that have been paired are answered faster since we can exploit the pre-computed posting intersection. In order to choose which terms have to be paired, we formulate the term pairing problem as a Maximum-Weight Matching Graph problem, and we evaluate in our scenario efficiency and efficacy of both an exact and a heuristic solution. We apply our technique: (*i*) to compact a compressed inverted file built on an actual Web collection of documents, and (*ii*) to increase capacity of an in-memory posting list. Experiments showed that in the first case our approach can improve the compression ratio of up to 7.7%, while we measured a saving from 12% up to 18% in the size of the posting cache.

1 Introduction

Compression plays an important role in modern information retrieval systems, particularly in large-scale Web Search Engines (WSEs), which crawl and index tens of billions of pages, thus managing an extremely huge inverted index and document repository. The benefits of effective compression techniques are twofold. First, they allow cost savings for storage. Second, the memory hierarchy is better utilized thus resulting in a lower query processing time [1]. It is well known that fetching compressed data from disk and then decompressing them in memory, is generally faster than fetching from disk the same data stored uncompressed. Thus, in the case of a disk-resident inverted index, compressing posting lists decreases the time required to fetch them from secondary storage. On the other hand, posting list compression remains valuable even for in-memory index settings, since it remarkably increases the portion of the index that can fit the available memory at the cost of a negligible decompression overhead [2–4]. Previous work has focused on devising effective and efficient coding methods to compress the posting lists of inverted indexes [4–7], or to reduce their size by an ad-hoc assignment of document identifiers [8–12]. Differently from previous works on index compression, this paper proposes a new technique for the lossless compression of an inverted index by directly reducing the number of entries stored as well as the number of postings coded. Based on the observation that many terms are highly correlated, i.e. co-occur frequently in the same document, and/or in the same query, we propose an algorithm to code only once the postings that are shared by different, correlated terms. In principle, coding only once the postings of two frequently co-occurring terms results in a reduction of space occupancy proportional to the number of postings saved. In addition, the pairing can be exploited to speed-up query processing time, when these highly related terms occur in the same query. Indeed, choosing the terms to pair is a complex task since the number of terms in the lexicon is huge. In fact, typical candidate terms are likely to be globally frequent and highly correlated in the indexed collection and/or in the queries submitted to the information retrieval system. We formulate the pairing decision problem as an optimization problem in graph theory known as the *Maximum Weight Matching Problem* (MWMP) [13]. MWMP is a classical graph theory problem, which can be exactly solved in time proportional to the cube of the number of vertices. Alternative linear-time approximation algorithms are also known.

Two different series of experiments are conducted on an inverted index built on a large Web Collection. First, we apply our technique to compress the whole inverted file index, obtaining up to 7.7% better compression ratio. Second, we apply the same technique to store in an in-memory posting cache the most frequently accessed posting lists and their intersections [14]. With this sort of static cache, a 12%-18% reduction in size was measured on the given test set.

Note that multi-term indexing is not novel. In particular in [15], an inverted index with multi-term entries was proposed to boost query processing. Beside the traditional, single-term index, the authors built an additional inverted index with frequent multi-keyword entries. The main drawback of their solution is the extra amount of memory needed to store the multi-term entries. On the other hand, our solution actually reduces the memory needed thus leaving free space for other important in-memory structures such as the cache.

In summary, our main contributions are:

– a novel inverted index compression technique based on pairing posting lists of frequently co-occurring terms.
– two types of representations for paired posting lists. The two representations have different pros and cons, which are discussed and evaluated.
– the demonstration on real data of the advantages of using our technique on two different scenarios: (*i*) inverted file compression, and (*ii*), static index caching.

The rest of the paper is organized as follows. The related work is presented in the next Section. Preliminaries and background information about inverted index representation and compression techniques are introduced in Section 3.

Section 4 describes the basic idea of terms pairing and the two different representation methods proposed. The problem formulation and the proposed algorithms are discussed in Section 4.1, while the experimental results are reported in Section 5. Finally, Section 6 discusses future work and draws our conclusions.

2 Related Work

As above mentioned, several papers focused on devising effective and efficient methods to encode the document identifiers (DocIDs) contained in the posting lists of Inverted File (IF) indexes, which allows efficient retrieval of documents containing the set of terms specified in a query. Since posting lists are ordered sequences of integer DocID values, and are usually accessed by scanning them from the beginning, these lists are stored as sequences of d-gaps, i.e. differences between successive DocID values. d-gap lists are then compressed by using variable-length encodings, which represent small integers in less space than large ones. Basic d-gaps coding methods are Variable Byte, which does not give the best compression performance, but it has a very short decompression time, and Gamma, which works best for encoding very small values [4]. More complex encoding techniques are Golomb [5], Rice [6], Simple9, Simple16, and PForDelta [7]. By following a radically different approach, Blandford et. al. [8], and Shieh et. al. [9], proposed methods to improve the IF compression performance by globally re-ordering document identifiers. Their approach aimed to reduce the average values of d-gaps by cleverly re-assigning document identifiers. Smaller the d-gaps are, shorter their representation with any variable length encoding method. On the same direction, Silvestri et. al. [10, 11], Blanco et. al. [12] studied efficient approaches to reduce the complexity of re-ordering document identifiers while maintaining a similar compression performance. As previously said, multi-terms indexing is not novel. For example, it was proposed in [15] for supporting efficiently phrase searches. The most frequent phrases mined from query logs were indexed as unique index entries, thus allowing the fast resolution of queries asking for such subset of supported phrases. Such multi-term entries were chosen on the basis of actual frequency of occurrence extracted from query logs. As the distribution of query terms is highly skewed, their experiments showed that the additional multi-terms inverted index could remarkably speed up query processing at the cost of an increase in the size of the whole index. This work is highly related to ours, and their accurate experiments strongly reinforce the validity of our assertion regarding the performance improvement in the query processing process resulting also from our technique. The main difference with respect to the work in [15] is that our proposal does not require extra storage space for storing paired entries, but instead allows the size of the index to be reduced without incurring in any information loss.

3 Preliminaries

Let $\mathcal{D} = \{d_1, d_2, \ldots, d_N\}$ be a collection of N documents, each one identified by a distinct DocID, and $\mathcal{T} = \{t_1, t_2, \ldots, t_T\}$ the set of T distinct terms appearing in the documents of $\mathcal{D}$. Based on $\mathcal{D}$ and $\mathcal{T}$, an IF is constructed by listing, on a term basis, the sequences of documents of $\mathcal{D}$ containing occurrences of terms of $\mathcal{T}$. A DocID list associated with a term of $\mathcal{T}$, is called posting list. Generally, each posting is annotated with additional information regarding the frequency of the term in the document, and the position of each occurrence. This information is however generally stored in different streams of the IF posting list, and the following discussion does not affect such meta-information. Thus, without loss of generality, hereinafter, with the term posting list we will refer to a simple sequence containing the (encoding of) the integer identifiers of the documents in the associated IF entry.

As an example, let us consider two terms $t_1, t_2 \in \mathcal{T}$ such that t_1 and t_2 appear in documents of $\mathcal{D}$ having identifiers 10, 15, 80, 1070, 2000, 2008, and 6, 15, 1070, 1090, 2000, respectively. From the previous assumption, the above ordered sequences of DocIDs constitute the posting lists for terms t_1 and t_2. It is worth noting that each posting list is generally transformed into a sequence of d-gaps before its encoding, i.e., a sequence of differences between consecutive DocIDs appearing in the posting list. Obviously, resulting d-gap values are smaller than original DocIDs, and, they can be more compactly represented by means of variable length encodings. Decoding d-gap values requires extra computation to obtain valid DocIDs. However, this computational load is negligible, as it requires to perform only fast addition operations on data with high spatial locality. In the following, we will briefly recall two of the most commonly used variable length d-gap encoding techniques.

- **Variable Byte (VB) Coding.** According to this scheme, a d-gap x is encoded with a sequence of consecutive bytes, the number of which depends on the value of x. The first bit of each byte indicates wether the byte is the last of the sequence (it is set if that byte is the last one), while the other 7 bits are used to actually encode the d-gap. For example, the codes of the two integer values 2 and 129 are 00000010 and 0000000110000001, respectively.
- **Gamma Coding (GC).** The previous encoding scheme is not particularly efficient to code very small d-gaps, as may happen for very frequent terms of the collection. Gamma uses instead a variable number of bits. A gamma code has two parts: The first, unary part contains a sequence of 1's followed by one 0, where the number of 1's defines the length of the binary representation of the d-gap value that is stored in the second part. With gamma, the codes of the above two integer values 2 and 129 are 110.10 and 111111110.10000001, respectively.

Gamma coding uses single bits to encode integers, while variable byte coding is byte aligned. In the context of very small integral values, gamma code is more effective than variable byte, but decoding is more expensive. In other conditions, variable byte behaves better. As we will see in Section 5, the characteristics of each coding technique affect also the performance of our term-pairing approach.

4 Representing IF Paired Terms

We base our study on the reasonable assumption that: *terms appearing frequently together within the same documents share a lot of common postings*. For example, terms like *FIFA* and *Football*, *Windows* and *Microsoft*, *Linux* and *GNU*, etc, are very likely to co-occur. The union of two posting lists has a number of postings equal to the sum of the postings of the single posting lists, minus the number of common postings appearing in both the paired lists. When the merged posting lists are highly correlated, their union could be remarkably shorter than the sum of the single lists. Let us consider again the above toy example: t_1 and t_2 have the following posting lists associated with: 10, 15, 80, 1070, 2000, 2008, and 6, 15, 1070, 1090, 2000. Pairing $t_1 \cup t_2$ will result in a list made up of 8 postings instead of 11 (the sum of the cardinality of the two single lists). Pairing must preserve the ability of answering queries in which terms are not paired. Therefore, some extra bits are needed to code such information that is necessary for supporting a correct query answering.

We propose two different approaches to encode such paired posting lists. The first representation provides that two extra bits per posting are used to indicate the source of a given posting, e.g., 10 for the first term, 01 for the second, 11 for both. Apart from these extra information, the postings of the merged lists can be coded with any technique. We called this type of representation *Mixed Union* (MU). In the second type of representation the postings of the paired terms are split into 3 partitions. The first partition contains postings common to both terms, the second partition postings for the first term only, and, finally, the last partition belongs to the second term. The DocIDs of each partition can

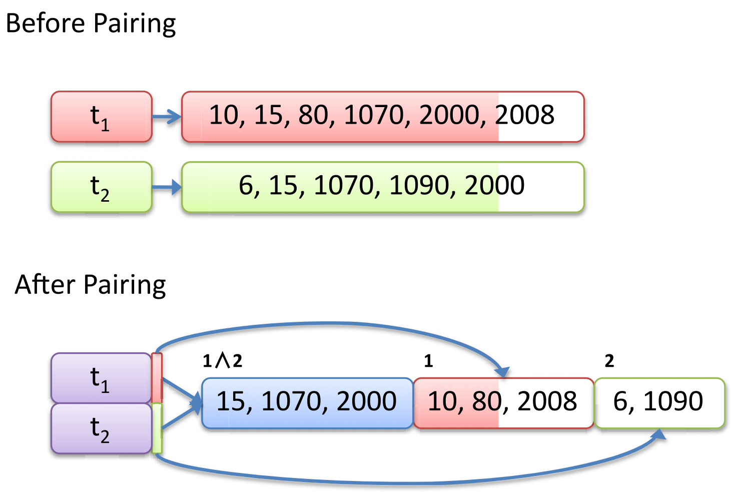


Fig. 1. A simple example of pairing two correlated posting lists: the paired posting list $t_1|t_2$ has 8 postings instead of 11

be ordered and represented with difference coding with the preferred technique. Some extra space is needed also in this case to store the offsets marking the starting points of the second and the third region. Normally, 8 bytes are sufficient for representing the two offsets in each paired list. We called this second kind of representation *Separated Union* (SU). The example in Figure 1 shows two entries paired and encoded with SU. SU has many advantages compared to MU. First, the number of extra space required is fixed (8 bytes) regardless posting list length, while in MU, the number of extra bits required is proportional to posting list length. In the case of a long posting list, MU is not likely to be a good choice. Second, decompression in SU involves each partition separately. In contrast, in MU we have to decompress the whole paired posting lists at a time to filter out the posting list of a single term or of the intersection. On the other hand, the disadvantage of SU is that the d-gap values must be computed locally for each partition, so that their average values are larger than in MU. Choosing which type of representation is the best may depend also on the compression technique adopted for coding the postings.

Paired entries speed-up processing of every kind of query. Conjunctive ones benefit from the intersection portion, to resolve disjunctive we just need to scan the paired posting list, to resolve set-difference (e.g. t_1 and not t_2) we just need to scan the appropriate portion of the paired posting list. Note that lexicon needs not only a slight change. Paired terms will point to two lists the intersection portion and the list of posting for the term not contained within the intersection. Suppose t_1 and t_2 have been paired up. When user requires a query containing both t_1 *and* t_2, the system will retrieve the relative list pointers. They have been paired up, thus, pointers are found to be the same, and the system will read only the intersection portion; otherwise from the lexicon we can obtain what is the relative portion to scan. For instance Figure 1 shows the case for SU. If we want to read the whole posting list for t_1 we have to scan the two lists pointed by the lexicon structure.

4.1 Problem Formulation

The compression benefits deriving from our technique clearly depend from the frequency and correlation degree of the paired terms. The choice of which terms to pair is thus crucial. We formulate the pairing decision problem as an optimization problem, and reduce it to a classical graph theory problem known as the *Maximum Weight Matching Problem* (MWMP) [13]. The MWMP can be solved with an exact algorithm in polynomial time, or with an approximation algorithm in linear time. In this section, we formalize our term-pairing problem.

Let $G(V, E)$ be a graph with a set of vertexes V, and a set of edges E. Suppose that each vertex $v_i \in V$ corresponds to a term t_i of $\mathcal{T}$, and that an edge e_{ij} exists between each couple of vertexes v_i and v_j. Moreover, let us assume that each edge $e_{ij} \in E$ is weighted with the value returned by the following function:

$$w(e_{ij}) = \begin{cases} 0 \; iff \; B_{tech}(t_i, t_j) \leq 0 \\ B_{tech}(t_i, t_j) \; otherwise. \end{cases}$$

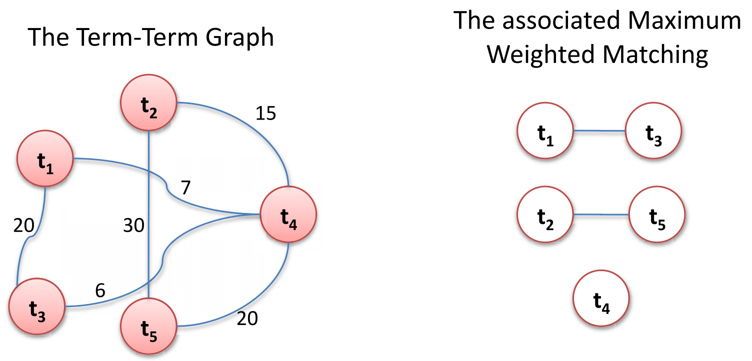


Fig. 2. An example of compression benefit graph. The Maximum Weight Matching for this graph is shown on the right hand side.

where $B_{tech}(t_i, t_j)$ measures (in number of bits) the benefit of pairing terms t_i and t_j according to the technique *tech* adopted, i.e. according to the type of representation for paired entries (MU or SU), and the encoding method used for representing d-gaps (e.g., VB, GC, etc.).

For the above graph G, a set $M \subseteq E$ of edges is a *matching* if no pair of edges of M has a common vertex. Given our benefit weight function $w(e_{ij})$, which assigns weights to the edges of G, a matching M is a *Maximum Weight Matching*, if M is a matching, and $\sum_{e \in M} w(e)$ is maximal. It is straightforward to derive from the Maximum Weight Matching of G the optimal pairing. It suffices to pair together the terms corresponding to nodes at both ends of edges in the matching. Figure 2 shows a simple example of graph with compression benefit weights. The rightmost graph in Figure 2 represents the associated Maximum Weight Matching.

Several exact algorithms for solving the MWMP in polynomial time were proposed such as the Blossom algorithm by J. Edmonds [16], or the one by Gabow [17]. Approximated methods can compute good quality solutions to the MWMP in linear time. In this paper, we use the Preis's linear time half-approximation algorithm [18]. We also experiment the exact algorithm by Gabow [17] with complexity $O(|E|^3)$.

5 Experimental Results

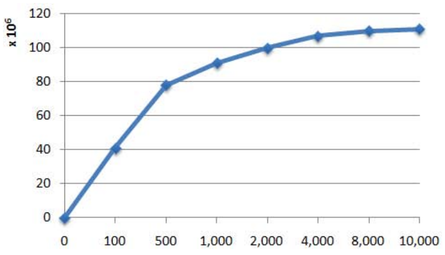
Our terms pairing technique was tested on an IF obtained from WBR99, a real Web collection consisting of 5,939,061 documents, occupying about 22 GB, and containing about 2,700,000 different terms [10, 11]. WBR99 is a snapshot of the Brazilian Web as crawled by TodoBr, a Brazilian search engine (`www.todobr.com.br`). The lexicon and the uncompressed postings of WBR99 occupy a space of about 3.6 GB. For experimenting the posting cache case we used also a query log of 51 millions queries collected by the same search engine. Two different series of experiments were conducted on the data above. In the first series of tests we evaluate the efficacy of our method in compressing the posting lists. We measure the compression ratio with respect to the size of a traditional, single-term IF adopting the same encoding technique (VB or GC), and containing exactly the same information. The second series of experiments evaluate the applicability of the same idea to the representation of postings within a static posting list cache.

In each experiment conducted, both the exact and approximate algorithms for the MWMP, as well as the Separated and Mixed Union representations for the paired posting lists, were tested. The results of the experiments are reported and discussed in the following subsections.

5.1 Term Pairing for IF Compression

During the analysis of the IF built on the WBR99 collection, we observed that the distribution of term occurrences is highly skewed. Indeed, the posting lists of the $10,000$ most frequent terms account for more than 75% the size of the whole IF! For this reason, in order to prevent our MWMP exact solver from performing exhaustive searching over a very large graph without significant improvement in compression performance, we restricted the size of the problem, and considered the graph of the $k = 10,000$ most frequent terms only. This simple pruning technique makes our problem instance tractable also with the Gabow's exact algorithm with cubic complexity.

To justify our pruning technique, the leftmost plots of Figure 3 reports the compression improvement (in number of bytes) resulting from applying our technique (with VB encoding and SU representation) on the basis of the results of Preis's approximate algorithm for different values of k. As we can see, the improvement increases rapidly when k grows up to 500, but tends to converge for higher values. The rightmost plot of Figure 3 shows instead the ratio between compression improvement and the number of paired terms. As expected, the benefit of our technique is very high when the first most frequent tterms are paired but tends to decrease as the length of posting lists of the paired terms decreases. In the following experiments we thus fixed $k = 10,000$, and considered the MWMP for our benefit graph with $10,000$ most frequent terms.



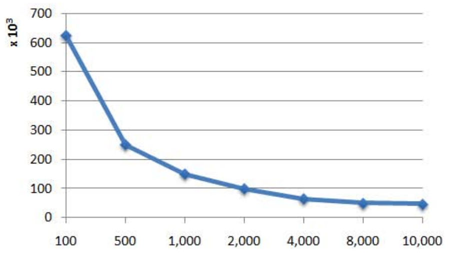


Fig. 3. Compression performance for values of the pruning factor k increasing up to $10,000$. Leftmost plot shows compression improvement, while the plot on the right hand shows the ratio between compression improvement and the number of paired terms.

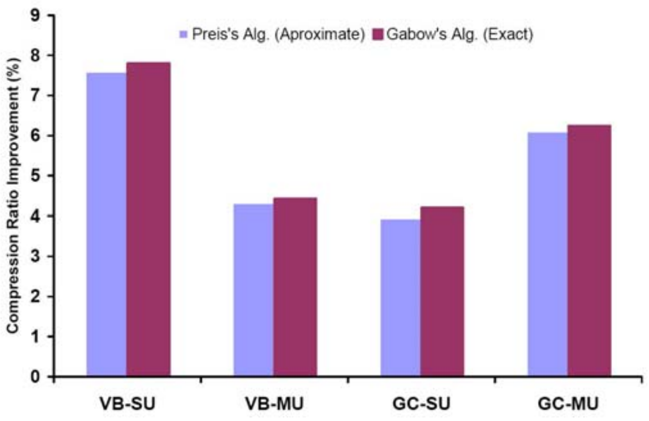


Fig. 4. Compression ratios achieved for VB-SU, VB-MU, GG-SU, and GC-MU, the four possible different combinations of Variable Byte and Gamma Coding, with Separated and Mixed Union representations. The two bars in each case refer to the compression ratios obtained with Gabow and Preis algorithms. The compression improvement is compared with the baseline VB code and Gamma code.

Figure 4 plots the results of the experiment conducted to evaluate the compression improvement resulting from different settings of our term pairing technique. In particular, Variable Byte (VB), and Gamma Code (GC) were combined with either Separated Union or Mixed Union, thus obtaining four different cases denoted with VB-SU, VB-MU, GC-SU, and GC-MU. Moreover, each case corresponds to two bars in the plot: one for the exact Gabow's algorithm, and one for the Preis' approximate one. We can see from the figure that the term pairing approach VB-SU is the most effective with a gain of 7.73% in the compression ratio. SU representation is remarkably better than mixed union with VB encoding. However, with Gamma encoding the opposite holds. This is very likely due to SU increasing the average values of d-gaps. We deserve to a future work to better investigate this fact. We can also observe that the exact algorithm is only slightly better than the approximate algorithm. For example, in the case of VB-SU, the Gabow's exact algorithm gives the a solution resulting in 7.73% better compression ratio, while the Preis's approximate solution reaches an improvement of 7.58%.

5.2 Pairing Cached Posting Lists

In this section, we consider another possible application of our term pairing approach: the representation of posting lists cached in main memory for faster query processing. Previous work [7, 14] has shown that posting list caching in memory is an effective technique to speed up query response time because it save up the cost of expensive disk accesses. Currently, commercial memory devices can be thousands time faster than hard disk, thus, by keeping the most frequently accessed posting lists inside memory, we can reduce the number of expensive

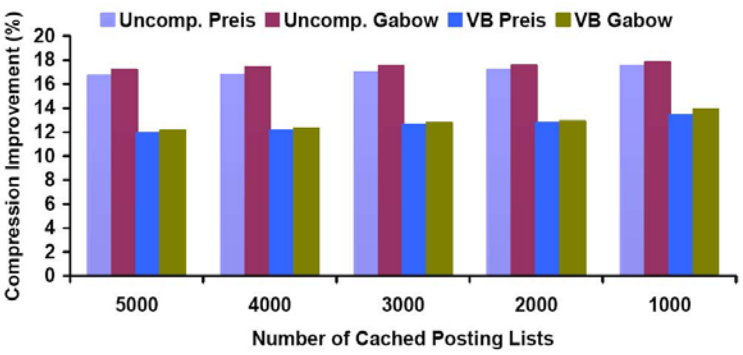


Fig. 5. Term pairing applied to the cache of posting lists: compression ratio improvement for different sizes of the cache (varying from 5,000 to 1,000), and different settings of the term pairing technique. The compression improvement is compared with the baseline VB code.

disk accesses. However, due to limit size of memory devices, cached posting lists must be compressed as much as possible. The more compacted the cached posting lists are the more posting lists can be resident inside memory, thus, reduce the number of disk accesses. In this section, we show empirically that our approach can reduce the cached posting lists size from $12-18\%$ compared to the other types of compression techniques. The cache into which the posting lists are kept can be managed with a static or dynamic policy. We will consider here only static posting list caching, since the choice of the best replacement policy to adopt in this case is out of the scope of this paper.

For testing the term pairing technique on cached posting lists we use a query log of 51 millions queries collected in 2003 by the Brazilian search engine TodoBr (the same source of the WBR99 collection). Frequency of terms appearing in the user queries were counted, and these terms then sorted by the ratio of frequency and posting list length [19]. Then, different instances of the associated MWMP were solved by considering the top 1,000, 2,000, 3,000, 4,000, 5,000 terms sorted by frequency and posting list length. Finally, we compute the space saving in the occupancy of a VB-encoded posting list cache of the same dimension by either applying or not the term pairing technique. We consider only SU since it better pairs up with the VB encoding. Moreover, we test also the behavior of our term pairing technique in the case uncompressed DocID-based postings. Recently, Baeza-Yates et. al. [19] showed that compression is advantageous also applied to cached posting lists. In fact, fixed the amount of available memory, compression allows a larger number of posting lists to be cached, and thus the cache hit ratio to be improved. However, our term pairing technique is itself a compression method, and it is interesting to evaluate its efficacy alone on uncompressed list of DocIDs.

Figure 5 reports the results of our tests. On the horizontal axis we report the size of cache in terms of number of cached posting lists, while on the vertical axis we plot the improvement in the actual size of the cache obtained applying our term pairing approach. We can see from the figure that term pairing is very effective in this case, allowing improvements ranging from 12% to 18%. We thus obtained better results from our technique when applied to posting list caching than to the whole IF. The reason for this behavior is that the cache is much smaller than the whole IF, and that it is filled with terms which frequently occur in queries. Even if the distribution of terms in queries and collections is not the same, these terms resulted to be frequent also in the collection, and as shown in the plot on the right of Figure 3, the larger the length of to-be-paired posting lists, the more the compression improvement obtained by pairing them.

6 Conclusions and Future Work

In this paper, we proposed a novel inverted index compression technique based on pairing the posting lists of highly correlated terms. We formulated term pairing as an optimization problem, and reduced it to the *Maximum Weight Matching Problem* that can be solved with exact polynomial algorithms, or approximated in linear time.

We demonstrated on real data the validity of our proposal by applying it to two different scenarios: the compression of the whole inverted file index, and of the posting lists cached in memory to fasten query processing. In both cases, the experiments conducted obtained very good results. In the case of IF compression we measured an improvement in terms of compression ratio up to 7.7 %. An impressive improvement (up to 18%) was measured for the case of static caching of index entries. Moreover, our technique is very efficient and can be applied even to huge IFs such as those managed by large-scale Web search engines. In fact, we showed that the size of the optimization problem can be strongly reduced to consider a few thousands of most frequent terms only, since the distribution of posting list length is highly skewed and our pairing technique works effectively for long, correlated posting lists. Moreover, results obtained by exploiting the (linear-time) approximated solution for the (reduced) Maximum Weight Matching Problem, were very similar to the one derived from the optimal solution obtained with a solver having cubic complexity. We plan to investigate in the future several important issue. First of all we will verify in which measure, our term pairing approach can speed up query processing. We based our assumptions on query processing speed-up on the results reported in [15]. We are confident our method will allow to obtain, at least, the same speed-up figures as their. Moreover, we will investigate the combination of pairing and index pruning techniques [20] for both cached posting lists and IF compression.

References

1. Roy, S., Kumar, R., Prvulovic, M.: Improving system performance with compressed memory. In: IPDPS 2001: Proceedings of the 15th International Parallel & Distributed Processing Symposium, p. 66. IEEE Computer Society, Washington (2001)
2. Turpin, A., Tsegay, Y., Hawking, D., Williams, H.E.: Fast generation of result snippets in web search. In: Kraaij, W., de Vries, A.P., Clarke, C.L.A., Fuhr, N., Kando, N. (eds.) SIGIR, pp. 127–134. ACM, New York (2007)

3. Zobel, J., Moffat, A.: Inverted files for text search engines. ACM Comput. Surv. 38(2), 6 (2006)
4. Witten, I.H., Moffat, A., Bell, T.C.: Managing Gigabytes – Compressing and Indexing Documents and Images, 2nd edn. Morgan Kaufmann Publishing, San Francisco (1999)
5. Golomb, S.: Run-length encodings. IEEE Transactions on Information Theory 12(3), 399–401 (1966)
6. Rice, R.F., Plaunt, J.R.: Adaptive variable-length coding for efficient compression of spacecraft television data. IEEE Trans. Commun. COM-19, 889–897 (1971)
7. Zhang, J., Long, X., Suel, T.: Performance of compressed inverted list caching in search engines. In: WWW 2008: Proceeding of the 17th international conference on World Wide Web, pp. 387–396. ACM, New York (2008)
8. Blandford, D., Blelloch, G.: Index compression through document reordering. In: DCC 2002: Proceedings of the Data Compression Conference (DCC 2002), p. 342. IEEE Computer Society, Washington (2002)
9. Shieh, W.Y., Chen, T.F., Shann, J.J.J., Chung, C.P.: Inverted file compression through document identifier reassignment. Inf. Process. Manage. 39(1), 117–131 (2003)
10. Silvestri, F., Orlando, S., Perego, R.: Assigning identifiers to documents to enhance the clustering property of fulltext indexes. In: SIGIR 2004: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval, pp. 305–312. ACM, New York (2004)
11. Silvestri, F.: Sorting out the document identifier assignment problem. In: Amati, G., Carpineto, C., Romano, G. (eds.) ECiR 2007. LNCS, vol. 4425, pp. 101–112. Springer, Heidelberg (2007)
12. Blanco, R., Barreiro, A.: Tsp and cluster-based solutions to the reassignment of document identifiers. Inf. Retr. 9(4), 499–517 (2006)
13. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York (1979)
14. Long, X., Suel, T.: Three-level caching for efficient query processing in large web search engines. In: WWW 2005: Proceedings of the 14th international conference on World Wide Web, pp. 257–266. ACM, New York (2005)
15. Chaudhuri, S., Church, K.W., Knig, A.C., Sui, L.: Heavy-tailed distributions and multi-keyword queries. In: Kraaij, W., de Vries, A.P., Clarke, C.L.A., Fuhr, N., Kando, N. (eds.) SIGIR, pp. 663–670. ACM, New York (2007)
16. Edmonds, J., Johnson, E.L., Lockhart, S.C.: Blossom i: a computer code for the matching problem. Unpublished report, IBM T. J. Watson Research Center (1969)
17. Gabow, H.N.: An efficient implementation of edmonds' algorithm for maximum matching on graphs. J. ACM 23(2), 221–234 (1976)
18. Preis, R.: Linear time 1/2-approximation algorithm for maximum weighted matching in general graphs. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 259–269. Springer, Heidelberg (1999)
19. Baeza-Yates, R., Gionis, A., Junqueira, F., Murdock, V., Plachouras, V., Silvestri, F.: The impact of caching on search engines. In: SIGIR 2007: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval, pp. 183–190. ACM, New York (2007)
20. Blanco, R., Barreiro, A.: Static pruning of terms in inverted files. In: Amati, G., Carpineto, C., Romano, G. (eds.) ECiR 2007. LNCS, vol. 4425, pp. 64–75. Springer, Heidelberg (2007)