PERGAMON

Information Processing and Management 39 (2003) 117–131

INFORMATION PROCESSING & MANAGEMENT

www.elsevier.com/locate/infoproman

Inverted file compression through document identifier reassignment

Wann-Yun Shieh [a], Tien-Fu Chen [b], Jean Jyh-Jiun Shann [a], Chung-Ping Chung [a,*]

[a] *Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu 300, Taiwan, ROC*

[b] *Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi 621, Taiwan, ROC*

Received 21 August 2001; accepted 4 January 2002

Abstract

The inverted file is the most popular indexing mechanism for document search in an information retrieval system. Compressing an inverted file can greatly improve document search rate. Traditionally, the *d*-gap technique is used in the inverted file compression by replacing document identifiers with usually much smaller gap values. However, fluctuating gap values cannot be efficiently compressed by some well-known prefix-free codes. To smoothen and reduce the gap values, we propose a document-identifier reassignment algorithm. This reassignment is based on a similarity factor between documents. We generate a reassignment order for all documents according to the similarity to reassign closer identifiers to the documents having closer relationships. Simulation results show that the average gap values of sample inverted files can be reduced by 30%, and the compression rate of *d*-gapped inverted file with prefix-free codes can be improved by 15%.

Keywords: Information retrieval; Inverted file; *d*-gap; Document identifier reassignment; Traveling salesman problem

1. Introduction

To help users find useful information from a large-scale information cyberspace, information retrieval systems (IRSs) are widely developed recently. The large-scale information space requires a specialized indexing mechanism for efficient retrieval. Most indexing mechanisms can

* Corresponding author.
E-mail address: cpchung@csie.nctu.edu.tw (C.-P. Chung).

0306-4573/03/$ - see front matter
PII: S0306-4573(02)00020-1

be categorized into two types: inverted files and signature files (Faloutsos & Oard, 1995). But, the inverted file has been the most popular indexing mechanism used along the years. Rillof and Hollaar (1996) pointed out that most IRSs use inverted files as indexing mechanisms. Zobel, Moffat, and Ramamohanarao (1998) further showed that in terms of querying time, used space and functionality, inverted files perform better than signature files. However, the inverted files themselves require much storage space. We believe this space requirement can be alleviated to some extent, and this paper presents our study in this aspect.

1.1. Current methods and problems

In an inverted file-based IRS, a user sends a request containing some query terms to the system. The system searches these query terms in the inverted file to see which documents satisfy the request, and returns these documents' identifiers to the user. Thus, for each distinct term t, there is a corresponding list (called the inverted list) of the form

$$\langle t; f_t; D_1, D_2, D_3, \ldots, D_{f_t} \rangle$$

in the inverted file, where identifier D_i indicates the document that contains t, and frequency f_t indicates the total number of documents in which t appears.

When the number of collected documents in an IRS increases, the number of terms and the length of each inverted list in the inverted file increase accordingly. The increase in the term number slows down the search speed while the increases in the inverted list lengths slow down for Boolean query operations. Additionally, a large inverted file cannot fit into the main memory and usually has to be stored in the disk device whose access time is in the order of milliseconds. Compression is a common practice to reduce the size of an inverted file. A popular compression technique is to sort the document identifiers of each inverted list in increasing order, and then replace each document identifier with the difference between it and its predecessor to form a list of d-gaps (Witten, Moffat, & Bell, 1999; Zobel & Doffat, 1995; Moffat & Zobel, 1996). For example, the inverted list ⟨term; 7; 15, 43, 90, 8, 51, 130, 61⟩ can be sorted to ⟨term; 7; 8, 15, 43, 51, 61, 90, 130⟩ and transformed into d-gap representation as ⟨term; 7; 8, 7, 28, 8, 10, 29, 40⟩. This sequence of smaller numbers can be further effectively encoded by some prefix-free codes such as the gamma code, the delta code, and the Golomb code (Witten et al., 1999). The common nature of these codes is their variable-length representations in which smaller numbers are encoded more economically than larger ones.

Unfortunately, the gap values in an inverted file usually fluctuate a lot. A large gap value may potentially be as large as a document identifier, and the saving in space by the encoding techniques cannot be achieved easily (Witten et al., 1999). Most studies have focused on the improvement in encoding technique for d-gaps. To the best of our knowledge, none have emphasized on the re-assignment of document identifiers to reduce the gap values.

1.2. Research goal

We observed that if two documents deal with similar topics or belong to a specific domain, their contents usually share a large number of common terms. In this case, their document-identifiers will both appear in many inverted lists corresponding to their common terms. If we reassign new

closer identifiers to these documents, their gap values appearing in the inverted lists will likely be reduced.

In this paper, we propose a document identifier reassignment method to reduce the average gap values in an inverted file. This reassignment is based on a similarity factor between documents. We define the similarity between two documents as the number of common terms in their contents. All documents hence form a similarity-graph in which a vertex represents a document and an edge between two vertices represents their similarity. The reassignment order can be generated by traversing all vertices in this graph with a gap-reduction strategy. We transform this problem to the traveling salesman problem (TSP) and modify some heuristic algorithms for the TSP to generate such reassignment order.

This paper is organized as follows. In Section 2, we present the document similarity in detail. In Section 3, we present the document identifier reassignment method. Then we show the simulation results in Section 4. Finally, Section 5 presents our conclusions.

2. How document similarity can be used

In this section we define the document similarity which operates in our document identifier reassignment method.

2.1. Definition of document similarity

We define the similarity between two documents as the number of common terms both appearing in the documents' contents. If two documents have large similarities, their identifiers will both appear in many inverted lists. Therefore, it is worth reassigning new closer identifiers to these documents. We use this similarity factor to measure how closer identifiers should be reassigned to the documents.

Table 1 shows an example inverted-file with four inverted lists. Document similarities can be extracted from these inverted lists. For example, the first inverted list includes three documents having the term "cold". Thus each of three document pairs (1, 3), (1, 5), and (3, 5) has one similarity due to the common term "cold". If we traverse the inverted file term by term in one pass, the similarities among documents can be scored.

2.2. Maintaining document similarity with a graph

Given that a vertex represents a document identifier and the weight on the edge between two vertices represents the number of common terms in two corresponding documents, the similarities

Table 1
An example inverted file with four inverted lists

Number	Term	f_t	Document identifiers
1	Cold	3	1, 3, 5
2	Collect	4	2, 3, 4, 5
3	Company	4	1, 2, 3, 5
4	Computer	3	1, 4, 5

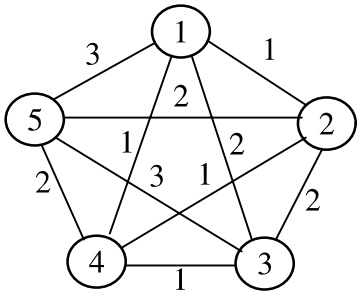


Fig. 1. Document similarity graph for Table 1.

2	1			
3	2	2		
4	1	1	1	
5	3	2	3	2
	1	**2**	**3**	**4**

Fig. 2. Similarity matrix derived from Fig. 1.

among all documents can be expressed as a graph, called the document similarity graph (DSG). The DSG for the inverted file of Table 1 is shown in Fig. 1.

A DSG can be represented by a similarity matrix in which an element $S_{i,j}$ represents the similarity between two documents i and j. For example, Fig. 2 shows the similarity matrix derived from Fig. 1. We use this data structure in our document identifier reassignment method. However, there are some design challenges for constructing such a matrix. First, a typical similarity matrix may be too large to be stored in the main memory. Second, while generating such a matrix, each update to it may need to access the disk twice—read from the disk, increment, and write back to the disk, and, thus, incur large delays. Finally, the number of such disk accesses increases as a square of the document count. We describe a matrix partitioning technique to solve these problems later.

3. Document identifier reassignment method

We present a gap-reduction strategy in our reassignment method and transform this problem to the TSP. Then we use some heuristic algorithms to generate the identifier reassignment order for all documents.

3.1. Gap reduction

Because all documents and the similarities among them can be presented by a DSG, an identifier reassignment order for all documents can be thought of as a new permutation for all vertices in the DSG. We traverse the vertices in the DSG one by one to generate a permutation order by a gap-reduction strategy. This strategy is based on the similarity factor between documents. That is, when we traverse to a node, say v_i, and look for a next node v_j to be the successor of v_i, we hope the similarity between v_i and v_j is maximal.

This problem can be transformed to the TSP. Given a graph containing a set of vertices and distances among the vertices, the TSP is to find a global path such that the total distance a salesman travels is minimum (Lawler, Lenstra, Rinnooy, & Shmoys, 1985; Manber, 1989; Cormen, Leiserson, & Rivest, 1990). If the "complement" of the similarity in the DSG is represented by the distance in the TSP, then the global path found by the TSP algorithm is the path of the *maximal* total similarity in the DSG. (Here the total similarity denotes the sum of similarities on the path.) Hence, when we follow this path to reassign document identifiers, the total gap value of these reassigned document identifiers in the inverted file can be minimized. Formal descriptions and proof of theory are ignored in this paper for clarity.

However, the TSP is an *NP*-complete problem and it seems unlikely there exists an optimal algorithm to find the minimum-distance path in a polynomial time when the number of vertices is large. Therefore, we modify some heuristic algorithms for the TSP to the algorithms for the document identifier reassignment problem.

3.2. Heuristic algorithms for generating identifier reassignment order

The algorithms can be classified into two categories: the greedy algorithm and the spanning tree (ST)-based algorithm (Lawler et al., 1985).

3.2.1. Greedy algorithms

A greedy algorithm for the document identifier reassignment problem is to expand the path by adding a vertex which is not on but is closest to the current path, i.e., with the largest similarity, one at a time. Each vertex-selection during the algorithm is in the sense of "maximizing the similarity" to get the new permutation. There are two alternatives for such a selection. First, we can select the vertex that is closest to the tail of the current path, i.e., expand the path from the tail. Second, the path expansion is not limited to the tail but open to all vertices in the current path. We call the first case Greedy-NN (Nearest Neighbor) algorithm and the second case Greedy-NA (Nearest Addition) algorithm.

Suppose that $G = (V, E)$ is a DSG with n vertices, and V and E are the sets of vertices and weighted edges in G, respectively. The Greedy-NN and Greedy-NA algorithms are listed in Figs. 3 and 4.

The Greedy-NN and the Greedy-NA algorithms are very simple to implement, but they involve higher time complexity. In the Greedy-NN algorithm, each vertex is inserted into path P exactly once, and expanding the path from the tail involves selecting a vertex from remain ones (Step 3 in Fig. 3). The overall complexity of the Greedy-NN algorithm is therefore $O(n^2)$. In the Greedy-NA algorithm, we need to check, for every pair of $v_k \notin P$ and $v_l \in P$, whether the similarity of (v_k, v_l) is

Input: *G, V, E*
Output: global path *P* that the total similarity is maximal
Begin
Step 1: Choose the edge $(v_i, v_j) \in E$ whose similarity is the largest in *G*.
Step 2: Start with a partial path consisting of a single vertex v_i, i.e., $P \leftarrow v_i$.
Step 3: Suppose that the current partial path *P* is $v_1, ..., v_l$, where $l < n$.
Find vertices $v_k \notin P$ such that the similarity of (v_k, v_l) is maximal.
Connect (v_k, v_l) and put v_k into the current path *P* just after v_l.
Step 4: Repeat Step 3 and halt when the current path *P* contains all vertices in *V*.
End.

Fig. 3. Greedy-NN algorithm.

Input: *G, V, E*
Output: global path *P* that the total similarity is maximal
Begin
Step 1: Choose the edge $(v_i, v_j) \in E$ whose similarity is the largest in *G*.
Step 2: Start with a partial path consisting of a single vertex v_i, i.e., $P \leftarrow v_i$.
Step 3: Find vertices $v_k \notin P$ and $v_l \in P$ such that the similarity of (v_k, v_l) is maximal.
Connect (v_k, v_l) and put v_k into the current path *P* just after v_l.
Step 4: Repeat Step 3 and halt when the current path *P* contains all vertices in *V*.
End.

Fig. 4. Greedy-NA algorithm.

maximal (Step 3 in Fig. 4). This step leads to a worst-case running time of $O(n^2)$ for the path expansion. Thus, the overall complexity of the Greedy-NA algorithm is $O(n^3)$. The other algorithms with lower complexity are described in the next subsection.

3.2.2. ST-based algorithms

A ST-based algorithm for the TSP first finds the minimum spanning tree (MinST) of a graph and then traverses the MinST to generate a global path. Manber (1989) showed that all edges in the MinST would be a near-optimal path for the TSP. This is so because a TSP path is a cycle containing all vertices; therefore, removing any edge from a TSP path makes it a spanning tree, whose total distance is thus at least that of the MinST.

Similarly, for the document identifier reassignment problem, because we want to find the path of the maximal total similarity among all vertices, we can transform a ST-based algorithm to first find a DSG's *maximum* spanning tree (MaxST), whose total similarity of edges is maximal in the graph, and then traverse the MaxST by breadth first search (BFS) or depth first search (DFS) to generate a global path. Without ambiguity, we call the former MaxST-BFS algorithm and the

Input: *G, V, E*
Output: global path *P* that the total similarity is maximal
Begin
Step 1: Find the maximum spanning tree *T* of *G*.
Step 2: Choose the edge $(v_i, v_j) \in T$ whose similarity is the largest in *T*.
Step 3: Start with v_i, i.e., $P \leftarrow v_i$.
Step 4: Perform BFS (or DFS) of *T* from vertex v_i and put the vertices during traversal into *P*.
Step 5: Halt when the current path *P* contains all vertices.
End.

Fig. 5. MaxST-BFS (DFS) algorithm for document identifier reassignment problem.

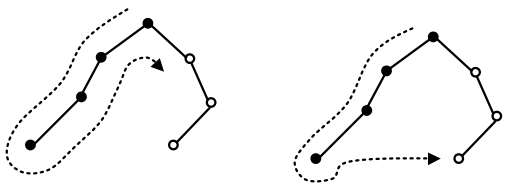
latter MaxST-DFS algorithm; both are shown in Fig. 5. Consequently, all edges in the MaxST will be a near-optimal path that forms a new reassignment order of document identifiers.

An obvious fact about performing DFS on a spanning tree is that when we traverse to a leaf vertex without adjacencies, we have to backtrack to an already-visited vertex, as shown in Fig. 6(a). This will cause the path to be discontinuous. In order to avoid such a discontinuous path, Lawler et al. (1985) introduced the idea of "shortcut" that does not increase the complexity of DFS too much. The shortcut occurs when we traverse to a leaf vertex of a spanning tree, instead of going back to an already-visited vertex, we go directly to an as-yet-unvisited vertex which is closest to the leaf vertex and then continually traverse the remaining vertices, as shown in Fig. 6(b). This step will greatly reduce the extra cost of backtracking. Without lost of generality, we implement the shortcut idea into the MaxST-DFS algorithm as described in Fig. 7.

All ST-based algorithms described above involve the spanning tree construction and tree traversal. The running time of the spanning tree construction is $O((|V| + |E|) \log |\mathrm{V}|)$, and that of the tree traversal is $O(|V| + |E|)$. Thus the overall running time of each ST-based algorithm is $O((|V| + |E|) \log |\mathrm{V}|)$.

3.3. Matrix partitioning

As described in Section 2.2, constructing a similarity matrix from an inverted file to represent a DSG needs many disk accesses. It is desirable to transform as many disk read/write accesses to



(a) DFS without shortcut (b) DFS with shortcut

Fig. 6. The concept of the shortcut. The dashed-line illustrates the traversal sequence and the solid vertex illustrates the vertices already visited.

Input: *G, V, E*
Output: global path *P* that the total similarity is maximal
Begin
Step 1: Find the maximum spanning tree *T* of *G*.
Step 2: Choose the edge $(v_i, v_j) \in T$ whose similarity is the largest in *T*.
Step 3: Start with v_i, i.e., $P \leftarrow v_i$.
Step 4: Perform DFS of *T* from vertex v_i and put the vertices during traversal into *P*.
If a leaf vertex v_j is visited, go directly to the closest unvisited vertex v_k of v_j, where $v_k \in T$ (i.e., the similarity between v_k and v_j is maximal).
Repeat Step 4 from vertex v_k.
Step 5: Halt when the current path *P* contains all the vertices.
End.

Fig. 7. MaxST-DFS with shortcut algorithm for the document identifier reassignment problem.

memory read/write accesses as possible. A matrix partitioning technique that stores a smaller block of the matrix in the memory for construction is adopted.

The key idea of the matrix partitioning technique is to divide the matrix into several independent and contiguous blocks of the maximal memory size. These blocks can then be moved to the memory to be processed, one at a time. In this case, when the inverted file is traversed, if a document pair falls in the range of the current block, its similarity is updated as necessary. When a matrix block finishes updating, it is written back to the disk (in one disk write), and the next block is loaded into the memory. Although each block update involves traversing the whole inverted file once, such construction time is much less than the time for updating the matrix globally in the disk.

4. Performance evaluation

In this section, we present the simulation model and compression results for the document identifier reassignment method.

4.1. Simulation environment

To allow experimental comparisons of various algorithms, simulations have been performed on three benchmark collections: paper collection (PC), Foreign Broadcast Information Service (FBIS), and Los Angeles Times (LATimes). The "PC", which represents a small-scale collection, is a set of papers collected from various proceedings and journals of computer science domain. The "FBIS" and "LATimes", which represent two large-scale collections, are contained in the fifth disk of Text REtrieval Coference (TREC). The TREC is a very large document collection distributed worldwide for comparative information retrieval experiments (Witten et al., 1999). Both of "FBIS" and "LATimes" are composed by full text of various newspapers and newswire articles plus government proceedings, and each has about 130,000 documents. The short

Table 2
Statistics of three uncompressed inverted files used in our experiments

Collection	PC7	FBIS	LATimes
Number of documents	7794	130,471	131,896
Number of distinct terms	286,738	209,782	167,805
Size of inverted file (MB)	43	263	297

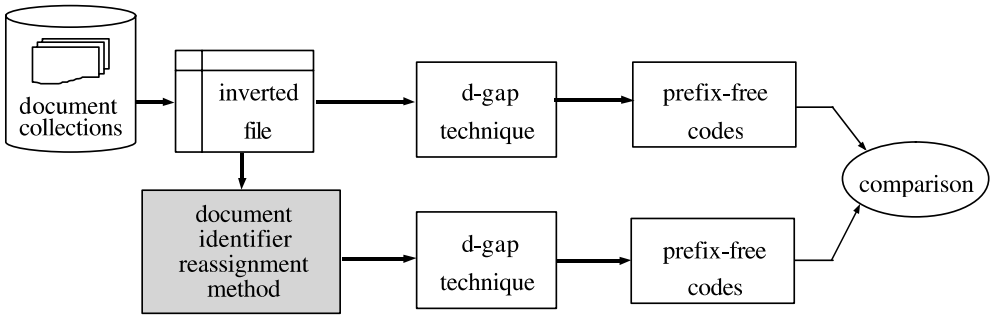


Fig. 8. Simulation model.

descriptions of these inverted files without *d*-gap encoding and compression are given in Table 2. Because we focus on reducing gap values in the inverted file, we store only document identifiers and frequency for each term; other augmented information such as word location in text or morphological data is not included.

Fig. 8 shows the simulation model with the proposed document identifier reassignment method. The inverted files are constructed from the document collections first. Then, the document identifier reassignment method and the *d*-gap technique are applied to the inverted files. Finally, the *d*-gapped inverted files are encoding through the gamma code, delta code, and Golomb code for the compression performance comparison. All experiments are performed on a PC with a 700 MHz CPU, a 128 MB SDRAM, and a 20 GB disk.

4.2. Simulation results

The simulation results include two parts. First, we show the document identifier reassignment effect on the change of gap-value distribution. Second, we show that this effect results in inverted file compression.

4.2.1. Reassignment effect on gap-value distribution

Fig. 9 shows the gap-value distribution changes after the document identifier reassignment. Note that the amounts of gap values decrease exponentially as the gap value increases. Thus we show only gap values 1–10. Here the Greedy-NN algorithm is used to generate the identifier reassignment path in the DSG. For every tested inverted file, we found that the number of gap value 1's after reassignment is about 1.5 times more than that before reassignment, and the numbers of other gap values all decrease. The average gap value can be reduced by about 30%.

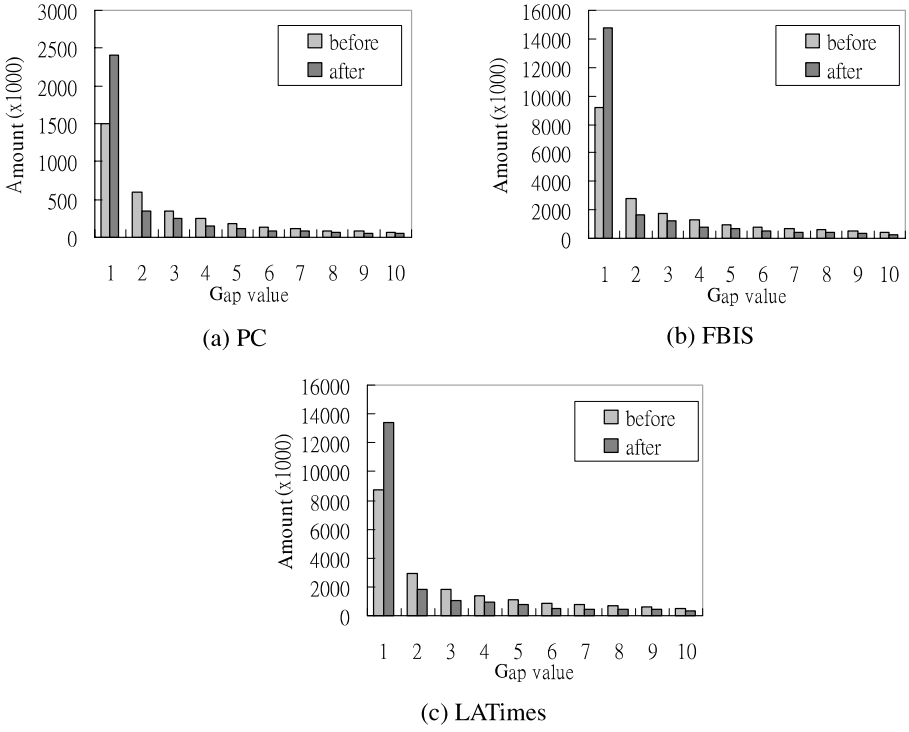


Fig. 9. Numbers of gap values 1–10 before and after document identifier reassignment.

This shows that documents with larger similarities are reassigned closer identifiers, and the gap values in the inverted file can be effectively reduced. These gap-value distributions after reassignment are more suitable for the prefix-free-code encoding.

4.2.2. Effect of the reassignment on the "PC" inverted file compression

Fig. 10 shows the compression performance of the heuristic algorithms to generate the identifier reassignment order for the "PC" inverted file under different prefix-free codes. Here, compression is shown in bits per gap value, which is the average number of bits used to represent each gap value in the inverted file. The Greedy-NN algorithm performs best on average while the Greedy-NA algorithm does not perform well. It is because that, in the Greedy-NA algorithm, the extension of the global reassignment path is from any vertex in the existing path; therefore, when a new vertex v_j is inserted into the path after vertex v_i, the gaps between vertex v_i and its original successors will be enlarged by v_j. This may cause a negative impact from identifier reassignment.

On the other hand, the Greedy-NN algorithm gives better compression than the ST-based algorithms. The reason is that performing BFS or DFS in the ST-based algorithms may also cause the similar negative impact from identifier reassignment. Consider the case of BFS shown in Fig. 11(a), where vertex v_j is visited after v_i but before v_k. The gap between v_i and v_k, which both belong to the same subtree, will be enlarged because the global path extension of BFS is from any already-visited node. Whereas for the case of DFS shown in Fig. 11(b), the gap between v_j and v_k, which belong to different subtrees, will be enlarged because the global path extension of DFS is

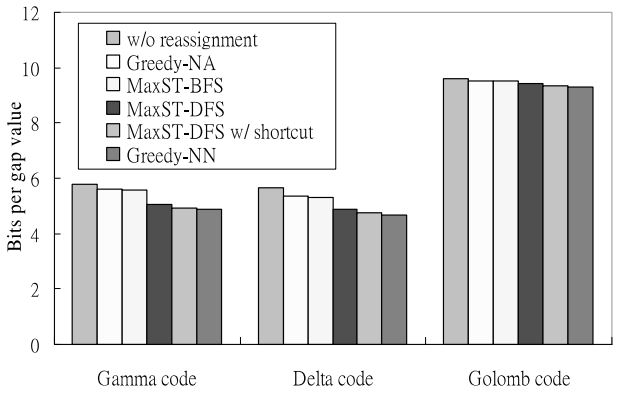


Fig. 10. Compression performance of different heuristic algorithms to generate the identifier reassignment order for the "PC" inverted file, measured in bits per gap value.

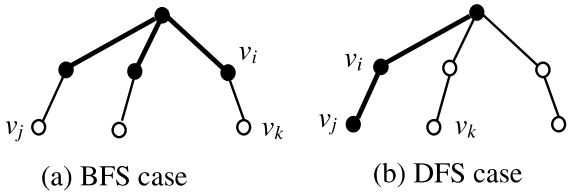


Fig. 11. Two cases of negative impacts from identifier reassignment.

from the most recently visited node. However, the "shortcut" will make up these drawbacks. That is why the MaxST-DFS with shortcut algorithm performs better than the MaxST-BFS and MaxST-DFS algorithms, and almost as well as the Greedy-NN algorithm.

In Fig. 10, the compression performance of the gamma code or the delta code is much better than the Golomb code. In fact, only when many terms appear with very high frequency does the optimality of the Golomb code lead to a significant improvement. These results are consistent with those derived in (Witten et al., 1999). Thus, we adopt the gamma code and the delta code to encode the inverted file in the following experiments.

4.2.3. Effect of the reassignment on the "FBIS" and the "LATimes" inverted file compression

Figs. 12 and 13 show the similar results for the "FBIS" and the "LATimes" inverted files, respectively. Although the document number of the "FBIS" or the "LATimes" inverted file is in the order of magnitude larger than that of the "PC" inverted file, the average number of bits to encode each gap value in the "FBIS" or the "LATimes" inverted file is only slightly larger than that in the "PC" inverted file. The reason we supposed is that the distributions of gap values in these inverted files are similar (as shown in Fig. 9) though the contents of these collections are different. Thus we conclude that the Greedy-NN algorithm (or the MaxST-DFS with shortcut algorithm) and the delta code are the best choices for the document identifier reassignment and *d*-gapped inverted file encoding, respectively, since they yield much better compression performance and lower complexity.

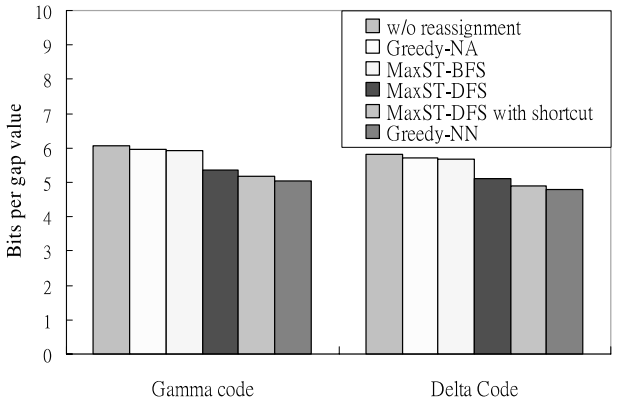


Fig. 12. Compression performance of different heuristic algorithms to generate the identifier reassignment order for the "FBIS" inverted file, measured in bits per gap value.

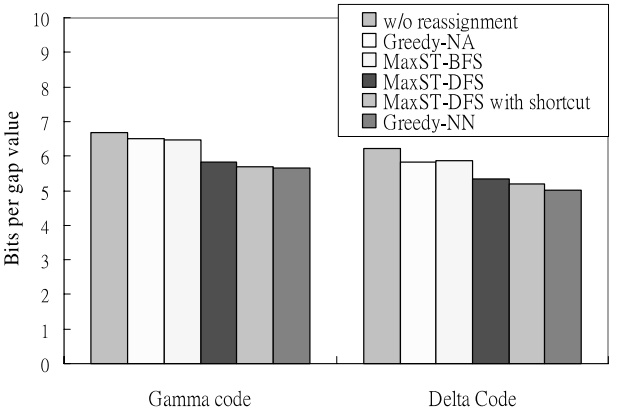


Fig. 13. Compression performance of different heuristic algorithms to generate the identifier reassignment order for the "LATimes" inverted file, measured in bits per gap value.

4.2.4. Time and space costs

We present the time and space costs involved in running the compression procedures for the PC, FBIS, and LATimes inverted files. The compression procedures include constructing the similarity matrix, reassigning identifiers by the Greedy-NN algorithm, and encoding *d*-gaps by the delta code. The space cost is principally the size of the similarity matrix in a compression form.

Table 3 shows that for each inverted file, the time of constructing the similarity matrix occupies a large part in the whole time. This is because that each inverted file is stored in the disk and constructing the matrix involves the most disk accesses to the inverted file in the compression procedures. Table 3 also shows that as the inverted file size increases from 43 MB (PC) to 297 MB (LATimes), the time cost goes up from 1.31 to 23.28 h and the space cost goes up from 9.2 MB to 2.17 GB. Such increase may be unacceptable for large-size database. However, improvements in processing and storage capacity and the introduction of efficient parallelism algorithms for

Table 3
Time and space costs of the compression procedures

	Time cost (h)				Space cost
	Similarity matrix construction	Greedy-NN reassignment	Delta code encoding	Total	
PC	1.12	0.11	0.08	1.31	9.2 MB
FBIS	15.24	4.14	0.25	19.63	2.10 GB
LATimes	17.73	5.29	0.26	23.28	2.17 GB

analyzing document similarities (Salton & Bergmark, 1981; Witten & Rasmussen, 1990) have made it feasible to deal with an increasingly large data set.

5. Conclusions

The goal of this paper is to reduce gap values in a *d*-gapped inverted file by the document identifier reassignment method. This method uses the similarity between documents to measure how closer identifier should be reassigned. We transform this problem to the TSP and modify some heuristic algorithms to generate the identifier reassignment order. Simulation results show that this reassignment method can reduce the average gap values by 30%, and a *d*-gapped inverted file with prefix-free code encoding can be compressed to 85% of its original size.

The contribution of the algorithms proposed in this paper is not limited to the inverted file compression. In fact, using the number of common terms as the similarity between two documents has been considered a useful and effective measurement in limiting the amount of computation required to calculate inter-document relevance (Frakes & Baeza-Yates, 1992). If we reassign identifiers to documents according to their similarities, the range in retrieving relevant documents can be limited to a small data set (i.e., neighbors) instead of the whole database. This would be helpful for those algorithms which need enhanced relevance assessments for information retrieval.

One issue requires further investigation. The DSG construction time in our reassignment algorithm depends greatly on the number of documents in the database collection. The processing time and the storage cost will be intolerable for collections commonly seen in modern search engines. Partitioning large database into smaller pieces can reduce processing time in searching local document similarities. However, without a global view, the similarities between documents may not be fully exploited. How to partition the database without this negative effect is under our investigation.

Acknowledgements

This work was supported by National Science Council, ROC: NSC-89-2213-E-009-062.

References

Cormen, T. H., Leiserson, C. E., & Rivest, R. L. (1990). *Introduction to Algorithms*. Cambridge, MA: MIT Press.
Frakes, W. B., & Baeza-Yates, R. (1992). *Information Retrieval: Data Structures and Algorithms*. Englewood Cliffs, NJ: Prentice Hall.

Faloutsos, C., & Oard, D. W. (1995). A Survey of Information Retrieval and Filtering Methods. Technical Report CS-TR-3514, Department of Computer Science, University of Maryland.

Lawler, E. L., Lenstra, J. K., Rinnooy, A. H. G., & Shmoys, D. B. (1985). *The Traveling Salesman Problem—A Guided Tour of Combinatorial Optimization*. New York: Wiley-Interscience Publication.

Manber, U. (1989). *Introduction to Algorithms a Creative Approach*. Addision-Wesley.

Moffat, A., & Zobel, J. (1996). Self-indexing inverted files for fast text retrieval. *ACM Trans. Information Syst., 14*(4), 249–279.

Rillof, E., & Hollaar, L. (1996). Text database and information retrieval. *ACM Comput. Surveys., 28*(1), 133–135.

Salton, G., & Bergmark, D. (1981). Parallel computations in information retrieval. *Lecture Notes in Computer Science, 111*, 328–342.

Willett, P., & Rasmussen, E. M. (1990). *Parallel Database Processing*. London: Pitman (Research Monographs in Parallel and Distributed Computing).

Witten, I. H., Moffat, A., & Bell, T. C. (1999). *Managing Gigabytes—Compressing and Indexing Documents and Images* (second ed.). Morgan Kaufmann Publishers, Inc.: Los Altos, CA.

Zobel, J., & Moffat, A. (1995). Adding compression to a full-text retrieval system. *Software Practice and Experience, 25*, 891–903.

Zobel, J., Moffat, A., & Ramamohanarao, K. (1998). Inverted files versus signature files for text indexing. *ACM Transactions on Database Systems, 23*(4), 453–490.

Wann-Yun Shieh received the B.S. degree in Computer Science and Information Engineering from the National Chiao-Tung University, Hsinchu, Taiwan, Republic of China in 1996. Currently he is pursuing the Ph.D. degree in computer science and information engineering at the National Chiao-Tung University, Hsinchu, Taiwan, Republic of China. His research interests include computer architecture, parallel and distributed systems, and information retrieval.

Tien-Fu Chen received the B.S. degree in Computer Science from National Taiwan University in 1983. After completing his military services, he joined Wang Computer Ltd., Taiwan as a software engineer for three years. From 1988 to 1993 he attended the University of Washington, receiving the M.S. degree and Ph.D. degrees in Computer Science and Engineering in 1991 and 1993 respectively. He is currently an Associate Professor in the Department of Computer Science and Information Engineering at the National Chung Cheng University, Chiayi, Taiwan. In recent years, he has published several widely-cited papers on dynamic hardware prefetching algorithms and designs. His current research interests are computer architectures, distributed operating systems, parallel and distributed systems, and performance evaluation.

Jean Jyh-Jiun Shann received the B.S. degree in Electronic Engineering from Feng-Chia University, Taichung, Taiwan, Republic of China in 1981. She attended the University of Texas at Austin from 1982 to 1985, where she received the M.S.E. degree in Electrical and Computer Engineering in 1984. She was a lecturer in the Department of Computer Science and Information Engineering, National Chiao-Tung University, Hsinchu, Taiwan, ROC, while working towards the Ph.D. degree. She received the degree in 1994 and is currently an Associate Professor in the department. Her current research interests include computer architecture, parallel processing, and information retrieval.

Chung-Ping Chung received the B.E. degree from the National Cheng-Kung University, Tainan, Taiwan, Republic of China in 1976, and the M.E. and Ph.D. degrees from the Texas A & M University in 1981 and 1986, respectively, all in Electrical Engineering. He was a lecturer in electrical engineering at the Texas A & M University while working towards the Ph.D. degree. Since 1986 he has been with the Department of Computer Science and Information Engineering at the

National Chiao-Tung University, Hsinchu, Taiwan, Republic of China, where he is a professor. From 1991 to 1992, he was a visiting associate professor of computer science at the Michigan State University. From 1998, he joined the Computer and Communications Laboratories, Industrial Technology Research Institute, ROC as the Director of the Advanced Technology Center, and then the Consultant to the General Director. He is expected to return to his teaching position after this three-year assignment. His research interests include computer architecture, parallel processing, and parallelizing compiler.