Available online at www.sciencedirect.com

SCIENCE DIRECT®

Information Processing and Management 42 (2006) 729–750

INFORMATION PROCESSING & MANAGEMENT

www.elsevier.com/locate/infoproman

Fast query evaluation through document identifier assignment for inverted file-based information retrieval systems

Cher-Sheng Cheng [*], Chung-Ping Chung, Jean Jyh-Jiun Shann

Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu 30050, Taiwan, ROC

Received 8 December 2004
Available online 28 June 2005

Abstract

Compressing an inverted file can greatly improve query performance of an *information retrieval system* (IRS) by reducing disk I/Os. We observe that a good *document identifier assignment* (DIA) can make the document identifiers in the posting lists more clustered, and result in better compression as well as shorter query processing time. In this paper, we tackle the NP-complete problem of finding an optimal DIA to minimize the average query processing time in an IRS when the probability distribution of query terms is given. We indicate that the *greedy nearest neighbor* (Greedy-NN) algorithm can provide excellent performance for this problem. However, the Greedy-NN algorithm is inappropriate if used in large-scale IRSs, due to its high complexity O($N^2 \times n$), where N denotes the number of documents and n denotes the number of distinct terms. In real-world IRSs, the distribution of query terms is skewed. Based on this fact, we propose a fast O($N \times n$) heuristic, called *partition-based document identifier assignment* (PBDIA) algorithm, which can efficiently assign consecutive document identifiers to those documents containing frequently used query terms, and improve compression efficiency of the posting lists for those terms. This can result in reduced query processing time. The experimental results show that the PBDIA algorithm can yield a competitive performance versus the Greedy-NN for the DIA problem, and that this optimization problem has significant advantages for both long queries and parallel *information retrieval* (IR).
© 2005 Elsevier Ltd. All rights reserved.

Keywords: Inverted index; Inverted file compression; Query evaluation; Document identifier assignment; *d*-Gap technique

[*] Corresponding author. Tel.: +886 3 573 1828; fax: +886 3 572 4176.
E-mail addresses: jerry@csie.nctu.edu.tw (C.-S. Cheng), cpchung@csie.nctu.edu.tw (C.-P. Chung), jjshann@csie.nctu.edu.tw (J.J.-J. Shann).

0306-4573/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.ipm.2005.05.003

1. Introduction

Information retrieval systems (IRSs) that are wildly used in many applications, such as search engines, digital libraries, genomic sequence analyses, etc. (Kobayashi & Takeda, 2000; Williams & Zobel, 2002), are overwhelmed by the explosion of data. To efficiently search vast amounts of data, an inverted file is used to evaluate queries for modern large-scale IRSs due to its quick response time, high compression efficiency, scalability, and support for various search techniques (Witten, Moffat, & Bell, 1999; Zobel, Moffat, & Ramamohanarao, 1998). An inverted file contains, for each distinct term in the collection, a list (called a posting list or synonymously an inverted list) of the identifiers of the documents containing that term. A query consists of keyword terms. To retrieve information, the query evaluation engine reads and decompresses the posting lists for the terms involved in the query, and then merges (intersection, union, or difference) corresponding posting lists to obtain a candidate set of relevant documents.

Compressing an inverted file can greatly increase query throughput (Williams & Zobel, 1999; Zobel & Moffat, 1995). This is because the total time of transferring a compressed posting list and subsequently decompressing it is potentially much less than that of transferring an uncompressed posting list. The document identifiers in a posting list are usually stored in ascending order. By using the popular *d*-gap compression approach (Moffat & Zobel, 1992; Witten et al., 1999), efficient compression of an inverted file can be achieved. In addition, we observe that the *d*-gap compression approach can result in good compression if the document identifiers in the posting lists are clustered.

The query processing time in a large-scale IRS is dominated by the time needed to read and decompress the posting lists for the terms involved in the query (Moffat & Zobel, 1996), and we observe that the query processing time grows with the total encoded size of the corresponding posting lists. This is because the disk transfer rate is near constant, and the decoding processes of most encoding methods used in the *d*-gap compression approach are on a bit-by-bit basis. If we can reduce the total encoded size of the corresponding posting lists without increasing decompression times, a shorter query processing time can be obtained.

A *document identifier assignment* (DIA) can make the document identifiers in the posting lists evenly distributed, or clustered. Clustered document identifiers generally can improve the compression efficiency of the *d*-gap compression approach without increasing the complexity of decoding process, hence reduce the query processing time. In this paper, we consider the problem of finding an optimal DIA to minimize the average query processing time in an IRS when the probability distribution of query terms is given. The DIA problem, that is known to be NP-complete via a reduction to the rectilinear *traveling salesman problem* (TSP), is a generalization of the problems solved by Olken and Rotem (1986), Shieh, Chen, Shann, and Chung (2003), and Gelbukh, Han, and Sidorov (2003). Their research results showed that this kind of optimization problem can be effectively solved by the well-known TSP heuristic algorithms. The *greedy nearest neighbor* (Greedy-NN) algorithm performs the best on average, but its high complexity discourages its use in modern large-scale IRSs.

In this paper, we propose a fast heuristic, called *partition-based document identifier assignment* (PBDIA) algorithm, to find a good DIA that can make the document identifiers in the posting lists for frequently used query terms more clustered. This can greatly improve the compression efficiency of the posting lists for frequently used query terms. Where the probability distribution of query terms is skewed, as is the typical case in a real-world IRS, the experimental results show that the PBDIA algorithm can yield a competitive performance versus the Greedy-NN for the DIA problem. The experimental results also show that the DIA problem has significant advantages for both long queries and parallel *information retrieval* (IR).

The remainder of this paper is organized as follows. Section 2 describes the inverted index and explains why a DIA can affect the storage space required and change query performance. Section 3 derives a cost model for the DIA problem, and presents how to use the well-known TSP heuristic algorithms to solve this optimization problem. In Section 4, we propose a fast PBDIA algorithm. We show the experimental results in Section 5. Finally, Section 6 presents our conclusion.

2. General framework

The data structures of an inverted index are depicted in Fig. 1. An inverted index consists of an index file and an inverted file. An index file is a set of records, each containing a keyword term t and a pointer to the posting list for term t. An inverted file contains, for each distinct term t in the collection, a posting list of the form

$$\mathrm{IL}_t = \langle \mathrm{id}_1, \mathrm{id}_2, \ldots, \mathrm{id}_{f_t} \rangle,$$

where id_i is the identifier of the document that contains t, and frequency f_t is the number of documents in which t appears. The document identifiers are within the range $1, \ldots, N$, where N is the number of documents in the indexed collection. In a large document collection, posting lists are usually compressed, and decompression of posting lists is hence required during query processing.

Zipf (1949) observed that the set of frequently used terms is small. According to Zipf's law, 95% of words in all documents fall in a vocabulary with no more than 8000 distinct terms. This suggests that it is advisable to store the index records of frequently used terms in RAM to greatly reduce index search time. Hence, the significant portion of query processing time is to read and decompress the compressed posting list for each query term. This paper restricts attention to inverted file side only and investigates the DIA problem to improve the efficiency of an inverted file and the overall IR performance.

The d-gap compression approach (Moffat & Zobel, 1992; Witten et al., 1999), the most popular approach for inverted file compression, consists of two steps. It first sorts the document identifiers of each posting list in increasing order, and then replaces each document identifier (except the first one) with the distance between itself and its predecessor. For example, the posting list $\langle 3, 8, 12, 15, 32 \rangle$ can be represented in d-gaps as $\langle 3, 5, 4, 3, 17 \rangle$. And the second step is to encode (compress) these d-gaps using an appropriate coding method. Many coding methods, such as γ coding (Elias, 1975), Golomb coding (Golomb, 1966; Witten et al., 1999), skewed Golomb coding (Teuhola, 1978), and batched LLRUN coding (Fraenkel & Klein, 1985), have been proposed to compress posting lists through the estimates of d-gap probability distributions. The more accurately the estimate, the greater the compression can be achieved.

One common characteristic of coding methods used in the d-gap compression approach is that small d-gap values can be coded more economically than large ones. If we can shrink the d-gap values, the compression ratio and query performance can be improved. Consider a document collection of six documents

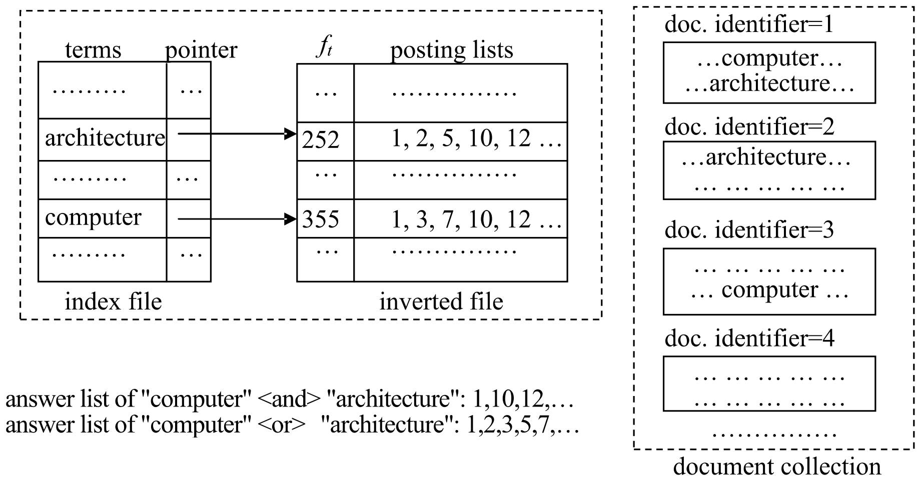
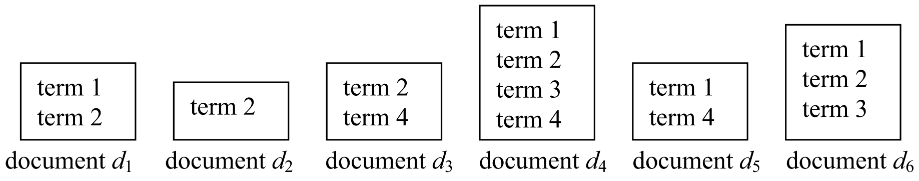


Fig. 1. Inverted index and document collection.



(a)

DIA I: { $d_1 \rightarrow 1$, $d_2 \rightarrow 2$, $d_3 \rightarrow 3$, $d_4 \rightarrow 4$, $d_5 \rightarrow 5$, $d_6 \rightarrow 6$}

Posting list of term 1: <1, 4, 5, 6>	*d*-gap list of term 1: <1, 3, 1, 1>
Posting list of term 2: <1, 2, 3, 4, 6>	*d*-gap list of term 2: <1, 1, 1, 1, 2>
Posting list of term 3: <4, 6>	*d*-gap list of term 3: <4, 2>
Posting list of term 4: <3, 4, 5>	*d*-gap list of term 4: <3, 1, 1>

Total bits required to encode *d*-gaps with γ code = 26 bits

(b)

DIA II: { $d_1 \rightarrow 3$, $d_2 \rightarrow 5$, $d_3 \rightarrow 4$, $d_4 \rightarrow 1$, $d_5 \rightarrow 6$, $d_6 \rightarrow 2$}

Posting list of term 1: <1, 2, 3, 6>	*d*-gap list of term 1: <1, 1, 1, 3>
Posting list of term 2: <1, 2, 3, 4, 5>	*d*-gap list of term 2: <1, 1, 1, 1, 1>
Posting list of term 3: <1, 2>	*d*-gap list of term 3: <1, 1>
Posting list of term 4: <1, 4, 6>	*d*-gap list of term 4: <1, 3, 2>

Total bits required to encode *d*-gaps with γ code = 20 bits

(c)

Fig. 2. An example to show different DIAs result in different compression results: (a) Example documents, (b) DIA I result, and (c) DIA II result.

shown in Fig. 2(a). Each document contains one or more terms. For example, the document d_1 contains terms 1 and 2, document d_2 contains term 2, etc. In Fig. 2(b) and (c), the notation $d_i \rightarrow j$ in DIAs I and II denotes that the document identifier j is assigned to the document d_i. According to the documents in Fig. 2(a) and the DIAs I and II, the obtained posting lists and *d*-gap lists are shown in Fig. 2(b) and (c). For DIA I, the *d*-gap values have nine 1s, two 2s, two 3s and one 4; whereas for DIA II, the *d*-gap values have eleven 1s, one 2 and two 3s. With γ coding in Table 1, the compressed inverted file requires 26 bits for DIA I, whereas it requires 20 bits for DIA II. If every term is queried with equal probability, the query processing costs for DIA II will be much lower than that of DIA I. This is because DIA II can result in better compression for the given coding method without increasing the complexity of decoding process, hence

Table 1
Some example codes for γ coding

d-gap value x	γ code
1	0
2	10 0
3	10 1
4	110 00

improve query throughput by reducing both the retrieval and decompression times of posting lists. This example shows that different DIAs can result in different compression results and different query throughputs for a given coding method. In next section, we will introduce a query cost function for the DIA problem, and then derive a method to find a good DIA to shorten average query processing time when the probability distribution of query terms is given.

3. Document identifier assignment problem and its algorithm

The DIA problem is the problem of assigning document identifiers to a set of documents in an inverted file-based IRS in order to minimize the average query processing time when the probability distribution of query terms is given. In this section, we first formalize the problem, and then show how to use the well-known *greedy nearest neighbor* (Greedy-NN) algorithm to solve this problem.

3.1. Problem mathematical formulation

Let $D = \{d_1, d_2, \ldots, d_N\}$ be a collection of N documents to be indexed, and $\pi: \{d_1, d_2, \ldots, d_N\} \rightarrow \{1, 2, \ldots, N\}$ be a DIA that assigns a unique identifier within the range $1, \ldots, N$ to each document in D. Let f_t be the total number of documents in which term t appears and $d_{t(1)}, d_{t(2)}, \ldots, d_{t(f_t)}$ be documents containing term t, then the posting list of the term t can be represented as $\mathrm{IL}_t = \langle \pi(d_{t(1)}), \pi(d_{t(2)}), \ldots, \pi(d_{t(f_t)}) \rangle$. Without loss of generality, we assume that $\pi(d_{t(1)}) < \pi(d_{t(2)}) < \cdots < \pi(d_{t(f_t)})$. Assume a coding method C which requires $C(x)$ bits to encode a d-gap x. The size of a posting list IL_t for term t can then be expressed as

$$\sum_{i=1}^{f_t} C\big(\pi(d_{t(i)}) - \pi(d_{t(i-1)})\big), \tag{1}$$

where we let $d_{t(0)} = 0$ and $\pi(d_{t(0)}) = 0$ to simplify the expression of Eq. (1). Assume that the probability of a term t appearing in a query is p_t. Let X_t be a random Boolean variable representing whether term t appears in a query: $X_t = 1$ if term t appears in a query and $X_t = 0$ otherwise. The query processing time $\mathrm{Time}_{\mathrm{QP}}$ of posting list processing includes (1) retrieval time $\mathrm{Time}_{\mathrm{R}}$ of posting list IL_t for each query term t, (2) decompression time $\mathrm{Time}_{\mathrm{D}}$ of posting list IL_t for each query term t, and (3) document identifier comparison time $\mathrm{Time}_{\mathrm{Comp}}$. Since the document identifier comparison time is relatively small (about 10% of query processing time) and does not change with different DIAs, the query processing time in this paper is defined only as

$$\mathrm{Time}_{\mathrm{QP}} = \sum_t X_t \times (\mathrm{Time}_{\mathrm{R}}(\mathrm{IL}_t) + \mathrm{Time}_{\mathrm{D}}(\mathrm{IL}_t)). \tag{2}$$

The average query processing time $\mathrm{AvgTime}_{\mathrm{QP}}$ is the expected value of $\mathrm{Time}_{\mathrm{QP}}$. That is,

$$\mathrm{AvgTime}_{\mathrm{QP}} = \sum_t p_t \times (\mathrm{Time}_{\mathrm{R}}(\mathrm{IL}_t) + \mathrm{Time}_{\mathrm{D}}(\mathrm{IL}_t)). \tag{3}$$

Since the disk transfer rate is near constant and the decoding processes of most coding methods used in d-gap compression approach are on a bit-by-bit basis, the retrieval and decompression times of a posting list IL_t for the term t appearing in a query grows with the size of the posting list IL_t. So

$$\mathrm{Time}_{\mathrm{R}}(\mathrm{IL}_t) + \mathrm{Time}_{\mathrm{D}}(\mathrm{IL}_t) = \mathrm{constant} \times \sum_{i=1}^{f_t} C\big(\pi(d_{t(i)}) - \pi(d_{t(i-1)})\big). \tag{4}$$

Substituting Eq. (4) into Eq. (3), we obtain

$$\mathrm{AvgTime_{QP}} = \mathrm{constant} \times \sum_t p_t \times \sum_{i=1}^{f_t} C\big(\pi(d_{t(i)}) - \pi(d_{t(i-1)})\big). \tag{5}$$

We thus define the objective function Cost(π) to reflect the average query processing time $\mathrm{AvgTime_{QP}}$:

$$\mathrm{Cost}(\pi) = \sum_t p_t \times \sum_{i=1}^{f_t} C\big(\pi(d_{t(i)}) - \pi(d_{t(i-1)})\big). \tag{6}$$

The objective of this research is to find a DIA $\pi : D \to \{1, 2, 3, \ldots, N\}$ such that Cost(π) is minimal. This optimization problem is called the DIA problem, and it is reduced to the *simple DIA* (SDIA) problem if the value of p_t for each term t is set to 1. The SDIA problem is the problem of finding a DIA to minimize the size of inverted file, and it is known to be NP-complete via a reduction to the rectilinear traveling salesman problem (Olken & Rotem, 1986). Since the DIA problem is a generalization of the SDIA problem, the DIA problem is also a NP-complete problem.

3.2. Solving DIA problem via the well-known Greedy-NN algorithm

The research works of Shieh et al. (2003) and Gelbukh et al. (2003) indicated that finding the near-optimal solution for the SDIA problem can be recast as the traveling salesman problem (TSP), and also showed that heuristic algorithms for the TSP can be applied to the SDIA problem to find a near-optimal DIA. Compared with those well-known TSP heuristic algorithms, such as insertion heuristic algorithm and spanning tree based algorithm, Shieh et al. (2003) showed that the Greedy-NN algorithm performs better for the SDIA problem on average. In Section 3.2.1, we show how to solve the SDIA problem using the Greedy-NN algorithm. Then, in Section 3.2.2, we show how to transform the DIA problem into the SDIA problem, and explain why the Greedy-NN algorithm can provide better performance than the other TSP heuristic algorithms for the DIA problem.

3.2.1. Solving SDIA problem via Greedy-NN algorithm

Shieh et al. (2003) showed that the SDIA problem can be solved by using TSP heuristic algorithms. Given a collection of N documents, a *document similarity graph* (DSG) can be constructed. In a DSG, each vertex represents a document, and the weight on an edge between two vertices represents the similarity of these two corresponding documents. The similarity Sim(d_i, d_j) between two documents d_i and d_j is defined as

$$\mathrm{Sim}(d_i, d_j) = \sum_{t \in (T(d_i) \cap T(d_j))} 1, \tag{7}$$

where $T(d_i)$ and $T(d_j)$ denote the set of terms appearing in d_i and d_j, respectively, and $\cap$ denotes the intersection operator. Hence, the similarity between two documents is the number of common terms appearing in both documents. The DSG for the example documents in Fig. 2(a) is shown in Fig. 3. A TSP heuristic algorithm can then be used to find a path of the DSG visiting each vertex exactly once with maximal sum of similarities. If we follow the visiting order of vertices on the path to assign document identifiers, the sum of d-gap values for an inverted file can be decreased, and the size of inverted file compressed via the d-gap compression approach can be reduced. Shieh et al. (2003) showed that the Greedy-NN algorithm (Fig. 4) can provide excellent performance for the SDIA problem.

We now show how to obtain a DIA for the DSG described in Fig. 3 using the Greedy-NN algorithm, where $V = \{d_1, d_2, d_3, d_4, d_5, d_6\}$. In Step 1, we pick d_4 as v_1 since the sum of similarity values associated with its adjacent edges is maximal (=10). In Step 2, we have $V' = \{d_1, d_2, d_3, d_5, d_6\}$. In Step 3, we pick d_6 as v_2 since d_6 is the vertex v in V' such that the edge (v, v_1) has the maximal similarity value. In Step 4, we have $V' = \{d_1, d_2, d_3, d_5\}$. Repeat Steps 3 and 4 as needed, we can then sequentially pick d_1 as v_3, d_3 as v_4, d_2 as v_5,

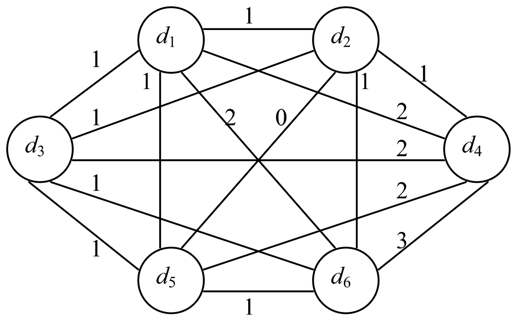


Fig. 3. DSG for the example documents in Fig. 2(a).

Algorithm Greedy_nearest_neighbor

```
Input:
  $D=\{d_1, d_2, \ldots, d_N\}$: a collection of $N$ documents to be indexed.
Output:
  A TSP path: the visiting order of vertices is $\{v_1, v_2, \ldots, v_n\}$
Method:
  0. Construct the DSG(V, E), where V is a set of vertices (in which each vertex represents a
     document) and E is a set of edges (in which each edge has a similarity value associated
     with it);
  1. Pick a vertex $v \in V$ as $v_1$ such that the sum of similarity values associated with the adjacent
     edges of v is maximal;
  2. $V' := V - \{v_1\}$;   $i := 1$;
  3. Find v in $V'$ such that the similarity value of the edge $(v, v_i)$ is maximal: if more than one
     such vertex exist, select one randomly;
  4. $i := i + 1$;   $v_i := v$;   $V' := V' - \{v_i\}$;
  5. If $i < N$ then goto 3;
  6. Output a TSP path with its visiting order of vertices being $\{v_1, v_2, \ldots, v_n\}$
```

Fig. 4. The Greedy-NN algorithm for the SDIA problem.

and d_5 as v_6. Hence, we have a TSP path: $\{d_4, d_6, d_1, d_3, d_2, d_5\}$, and a DIA $\pi = \{d_1 \to 3, d_2 \to 5, d_3 \to 4, d_4 \to 1, d_5 \to 6, d_6 \to 2\}$.

3.2.2. Transforming DIA problem into SDIA problem

We use a matrix A to represent the input document collection, in which a row corresponds to a term and a column corresponds to a document. The entry $A_{i,j}$ is a 1 if term i appears in document d_j, and 0 otherwise. The SDIA problem is to determine whether there exists a permutation of the columns of A that results in a matrix B such that

$$\sum_{i=1}^{n}\left(\sum_{j=2}^{f_i} C(z(i,j) - z(i,j-1)) + C(z(i,1))\right) \leqslant k, \tag{8}$$

where C is a coding method which requires $C(x)$ bits to encode a d-gap x, n is the number of terms, f_i is the total number of documents in which term i appears, $z(i,j)$ is a function that returns the column index of the jth nonzero entry at row i, and k is a given integer used to determine whether there exists a permutation of columns of A such that the total encoded size of an inverted file is less than k. The DIA problem is to determine whether there exists a permutation of the columns of A that results in a matrix B such that

$$\sum_{i=1}^{n} p_i \times \left(\sum_{j=2}^{f_i} C(z(i,j) - z(i,j-1)) + C(z(i,1)) \right) \leqslant k', \tag{9}$$

where p_i is the probability of a term i appearing in a query and k' is a given integer used to determine whether there exists a permutation of columns of A such that the mean encoded size needed to read and decompress a posting list during query processing is less than k'.

To show how to transform the DIA problem into the SDIA problem, we use the document collection in Fig. 2(a) as an example instance of the DIA problem, and assume that the probabilities of terms being queried are $p_1 = 0.2$, $p_2 = 0.3$, $p_3 = 0.1$, and $p_4 = 0.4$. Fig. 5(a) shows the matrix A of Fig. 2(a). Then we construct a new matrix A' for the SDIA problem by duplicating each row of matrix A in a certain number of times based on the given probabilities of terms appearing in a query, as shown in Fig. 5(b). In matrix A', the row of matrix A corresponding to term i is duplicated m_i times, where $m_i = \text{rows}(A') \times p_i$ and rows(A') denotes the number of rows of matrix A'. The rows(A') can be any positive integer such that $m_i = \text{rows}(A') \times p_i$ is an integer for every i. In this example, we let rows(A') be 10. One can easily show that the optimal solution of matrix A' for the SDIA problem is also the optimal solution of matrix A for the DIA problem when the probabilities $p_1 = 0.2$, $p_2 = 0.3$, $p_3 = 0.1$, and $p_4 = 0.4$ are given.

Matrix A:

probability		d_1	d_2	d_3	d_4	d_5	d_6
p_1=0.2	term 1	1	0	0	1	1	1
p_2=0.3	term 2	1	1	1	1	0	1
p_3=0.1	term 3	0	0	0	1	0	1
p_4=0.4	term 4	0	0	1	1	1	0

(a)

Matrix A':

	d_1	d_2	d_3	d_4	d_5	d_6
Row$_{\text{term1}}$ of matrix A is duplicated m_1=$rows(A')\times p_1$=2 times	1	0	0	1	1	1
	1	0	0	1	1	1
Row$_{\text{term2}}$ of matrix A is duplicated m_2=$rows(A')\times p_2$=3 times	1	1	1	1	0	1
	1	1	1	1	0	1
	1	1	1	1	0	1
Row$_{\text{term3}}$ the matrix A is duplicated m_3=$rows(A')\times p_3$=1 time	0	0	0	1	0	1
Row$_{\text{term4}}$ of matrix A is duplicated m_4=$rows(A')\times p_4$=4 times	0	0	1	1	1	0
	0	0	1	1	1	0
	0	0	1	1	1	0
	0	0	1	1	1	0

(b)

Fig. 5. An example to illustrate how to transform an instance of the DIA problem into an instance of the SDIA problem. (a) An example instance for the DIA problem: Matrix A corresponds to the document collection in Fig. 2(a), and the probabilities of terms appearing in a query are $p_1 = 0.2$, $p_2 = 0.3$, $p_3 = 0.1$, and $p_4 = 0.4$. (b) (b) Matrix A' is the corresponding instance of (a) for the SDIA problem. In matrix A', Row$_{\text{term}\,i}$ of matrix A is duplicated m_i times, where $m_i = \text{rows}(A') \times p_i$ and rows(A') denotes the number of rows of matrix A'.

Using the same approach, it is obvious that one can transform any instance A of the DIA problem into an instance A' of the SDIA problem such that the optimal solution of matrix A' for the SDIA problem is also the optimal solution of matrix A for the DIA problem when the probabilities p_i for $1 \leqslant i \leqslant n$ are given, where n denotes the number of distinct terms. Since the research work of Shieh et al. (2003) showed that the Greedy-NN algorithm performs the best for the SDIA problem on average, one can show that the Greedy-NN algorithm can provide better performance than the other TSP heuristic algorithms for the DIA problem. Therefore, the DIA problem can be solved using the Greedy-NN algorithm described in Fig. 4, if the similarity $\mathrm{Sim}(d_i, d_j)$ between two documents d_i and d_j in a DSG is redefined as

$$\mathrm{Sim}(d_i, d_j) = \sum_{t \in (T(d_i) \cap T(d_j))} p_t, \tag{10}$$

where the probability of a term t appearing in a query is known to be p_t.

Although the Greedy-NN algorithm is very simple to implement, it is not very applicable to large-scale IRSs due to its high complexity. Given a collection of N documents and n distinct terms, the number of comparisons for calculating $\mathrm{Sim}(d_i, d_j)$ given fixed i and j is $\mathrm{O}(n)$, hence the total number of comparisons to construct a DSG for the Greedy-NN algorithm is $\mathrm{O}(N^2 \times n)$. An algorithm with lower complexity yet still generates satisfactory results should be developed.

4. Partition-based document identifier assignment algorithm

Since the DIA problem is an NP-complete problem, the effort in search for an effective low-complexity method is needed. Although the Greedy-NN algorithm can be used to solve the DIA problem, its complexity is too high. In this section, we first present an optimal DIA algorithm for a single query term, and then propose an efficient *partition-based document identifier assignment* (PBDIA) algorithm for the DIA problem.

4.1. Generating an optimal DIA for a single query term

Consider a posting list IL_t for term t with f_t document identifiers in a collection of N documents. Using the d-gap technique, we can obtain f_t d-gap values: $d\text{-gap}_1, d\text{-gap}_2, \ldots, d\text{-gap}_{f_t}$. Assume a coding method C which requires $C(x)$ bits to encode a d-gap x. We want to know which d-gap probability distribution can minimize the size of posting list IL_t after compression using method C. That is, we want to know which d-gap probability distribution can minimize

$$\sum_{i=1}^{f_t} C(d\text{-gap}_i) \tag{11}$$

subject to

$$f_t \leqslant \sum_{i=1}^{f_t} d\text{-gap}_i \leqslant k \tag{12}$$

and

$$1 \leqslant d\text{-gap}_i \leqslant k \quad \text{for all } i, \;\; 1 \leqslant i \leqslant k, \tag{13}$$

where k is the largest document identifier in the posting list IL_t. It is known that $C(x)$ is approximately proportional to $\log_2(x)$ for many popular coding methods, such as γ coding, skewed Golomb coding, and batched LLRUN coding. For these coding methods, we can use dynamic programming technique (Bellman &

Dreyfus, 1962) and find that minimizing Eq. (11) should meet two requirements: (1) maximize the number of *d*-gap values of 1; and (2) minimize the largest document identifier, i.e., k, in the posting list IL_t. If a DIA for term t can satisfy the above two requirements, the best compression and the fastest query speed for the posting list IL_t can be achieved.

According to the above observation, we propose the *simple partition-based document identifier assignment* (SPBDIA) algorithm to generate optimal DIAs for a given query term t. The SPBDIA algorithm consists of a partitioning procedure, an ordering procedure, and a document identifier assignment procedure. The partitioning procedure divides the given documents into two partitions in terms of query term t: one partition $P(t)$ consists of documents containing query term t; the other partition $P(t')$ is made up of the documents without t. Then, the ordering procedure sets the order of partitions as $P(t)$ followed by $P(t')$. Finally, the document identifier assignment procedure generates an appropriate DIA for the ordered partitions according to query term t: the documents in partition $P(t)$ are assigned smaller consecutive document identifiers, while the documents in partition $P(t')$ assigned larger consecutive document identifiers. The SPBDIA algorithm is illustrated in the following example.

Example. There is a collection of 500 documents, among which 300 documents contain query term t. After partitioning, $P(t)$ has 300 documents and $P(t')$ has 200 documents. Then, the ordering procedure sets the order of partitions $P(t)$ followed by $P(t')$. Finally, the document identifier assignment procedure assigns the document identifiers 1–300 to the 300 documents in partition $P(t)$ and assigns the document identifiers 301–500 to the 200 documents in partition $P(t')$.

Documents in a partition can be arbitrarily assigned identifiers within the given range, hence the number of possible DIAs for the above Example is $300! \times 200!$. Each of the $300! \times 200!$ DIAs satisfies the two requirements for minimizing Eq. (11), and hence gives both the best posting list compression and fastest query speed for query term t. The SPBDIA algorithm is simple, and its complexity is $O(N)$.

4.2. *Efficient partition-based document identifier assignment algorithm for DIA problem*

In a real-world IRS, a few frequently used query terms constitute a large portion of all term occurrences in queries (Janson, Spink, Bateman, & Saracevic, 1998). This fact indicates that a DIA algorithm that allows those frequently used query terms to have better posting list compression can result in reduced average query processing time. Based on the SPBDIA algorithm, an efficient *partition-based document identifier assignment* (PBDIA) algorithm for the DIA problem can be developed.

Like the SPBDIA algorithm, the PBDIA algorithm also partitions the document set, orders these partitions, and then assigns document identifiers. The flowchart of the PBDIA algorithm is shown in Fig. 6. The partitioning and ordering procedures of the PBDIA algorithm iterate n times given that there are n query terms. Then, the document identifier assignment procedure is performed as the last step of the PBDIA algorithm. Terms that are queried more frequently should take higher priority in document partitioning and partition ordering. Let the most frequently queried term be assigned rank 1, the second most frequently queried term rank 2, and so on. We use $t_{\mathrm{rank}\,i}$ to represent the ith ranked query term. The partitioning and ordering procedures of the PBDIA algorithm should proceed by considering $t_{\mathrm{rank}\,1}$ first, then $t_{\mathrm{rank}\,2}$, and so on.

Both the PBDIA partitioning and ordering procedures are invoked once per iteration. The PBDIA partitioning procedure first divides each partition generated in the previous iteration into two partitions using the SPBDIA partitioning procedure. The PBDIA ordering procedure then assigns each newly generated partition a partition order. Each partition P in the PBDIA algorithm hence can be uniquely identified by an iteration number i and a partition order j, and we use the notation $P_{i,j}$ to represent the jth ordered partition of the ith iteration. For example, the notation $P_{2,3}$ represents the third ordered partition of the second iteration. Initially, we use the notation $P_{0,1}$ to represent the partition that contains all documents

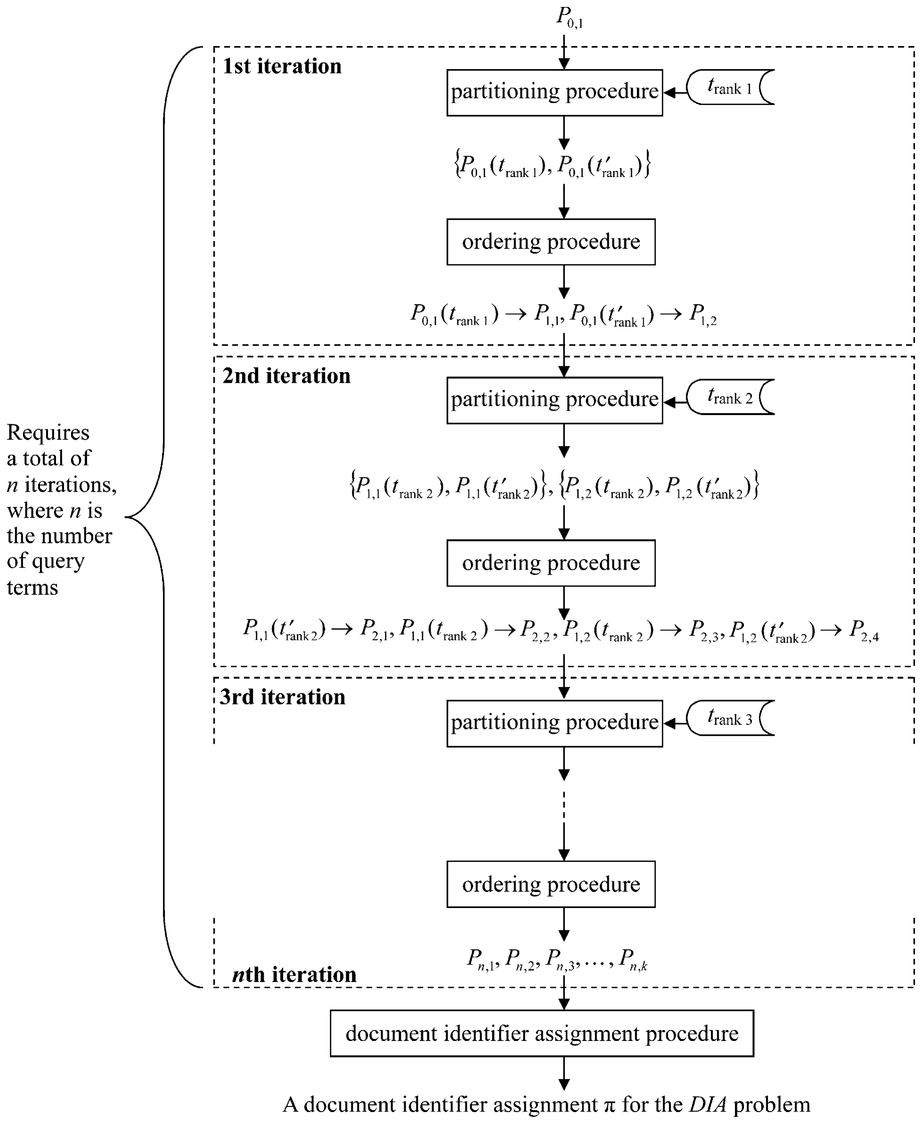

$P_{0,1}$
1st iteration
partitioning procedure
$t_{\text{rank }1}$
$\{P_{0,1}(t_{\text{rank }1}), P_{0,1}(t'_{\text{rank }1})\}$
ordering procedure
$P_{0,1}(t_{\text{rank }1}) \to P_{1,1}, P_{0,1}(t'_{\text{rank }1}) \to P_{1,2}$
2nd iteration
partitioning procedure
$t_{\text{rank }2}$
$\{P_{1,1}(t_{\text{rank }2}), P_{1,1}(t'_{\text{rank }2})\}, \{P_{1,2}(t_{\text{rank }2}), P_{1,2}(t'_{\text{rank }2})\}$
ordering procedure
$P_{1,1}(t'_{\text{rank }2}) \to P_{2,1}, P_{1,1}(t_{\text{rank }2}) \to P_{2,2}, P_{1,2}(t_{\text{rank }2}) \to P_{2,3}, P_{1,2}(t'_{\text{rank }2}) \to P_{2,4}$
3rd iteration
partitioning procedure
$t_{\text{rank }3}$
ordering procedure
$P_{n,1}, P_{n,2}, P_{n,3}, \ldots, P_{n,k}$
nth iteration
Requires a total of n iterations, where n is the number of query terms
document identifier assignment procedure
A document identifier assignment π for the DIA problem

Fig. 6. The flowchart for the PBDIA algorithm.

in an input document collection. In the following, we describe in detail the partitioning, ordering, and document identifier assignment procedures of the PBDIA algorithm.

4.2.1. PBDIA partition procedure

Let $P_{i-1,1}$, $P_{i-1,2}$, ..., and $P_{i-1,k}$ be nonempty partitions generated in iteration $i - 1$. The PBDIA partitioning procedure invoked in the ith iteration divides each partition $P_{i-1,j}$ into a partition pair $\{P_{i-1,j}(t_{\text{rank}\,i}), P_{i-1,j}(t'_{\text{rank}\,i})\}$ for $j = 1,2,\ldots,k$, where the partition $P_{i-1,j}(t_{\text{rank}\,i})$ consists of the documents in $P_{i-1,j}$ containing the query term $t_{\text{rank}\,i}$, and $P_{i-1,j}(t'_{\text{rank}\,i})$ consists of the documents in $P_{i-1,j}$ without the query term $t_{\text{rank}\,i}$. Since $P_{i-1,j}$ is nonempty, at least one of the two partitions $P_{i-1,j}(t_{\text{rank}\,i})$ and $P_{i-1,j}(t'_{\text{rank}\,i})$ is nonempty for $j = 1,2,\ldots,k$.

4.2.2. PBDIA ordering procedure

Let $\{P_{i-1,1}(t_{\text{rank}\,i}), P_{i-1,1}(t'_{\text{rank}\,i})\}$, $\{P_{i-1,2}(t_{\text{rank}\,i}), P_{i-1,2}(t'_{\text{rank}\,i})\}$, ..., and $\{P_{i-1,k}(t_{\text{rank}\,i}), P_{i-1,k}(t'_{\text{rank}\,i})\}$ be the partition pairs generated by PBDIA partitioning procedure in iteration i. Let $|P_i|$ denote the number of nonempty partitions of the above partitions. The PBDIA ordering procedure invoked in the ith iteration assigns a unique partition order, from $|P_i|$ to 1 and in descending order, to each nonempty partition, starting from $\{P_{i-1,k}(t_{\text{rank}\,i}), P_{i-1,k}(t'_{\text{rank}\,i})\}$, then $\{P_{i-1,k-1}(t_{\text{rank}\,i}), P_{i-1,k-1}(t'_{\text{rank}\,i})\}$, and so on.

Now let us consider the ordering of partition pair $\{P_{i-1,k}(t_{\text{rank}\,i}), P_{i-1,k}(t'_{\text{rank}\,i})\}$. Three cases exist.

Case 1: Both $P_{i-1,k}(t_{\text{rank}\,i})$ and $P_{i-1,k}(t'_{\text{rank}\,i})$ are nonempty
The ordering procedure assigns $|P_i|$ to $P_{i-1,k}(t'_{\text{rank}\,i})$, and $|P_i| - 1$ to $P_{i-1,k}(t_{\text{rank}\,i})$. $P_{i-1,k}(t'_{\text{rank}\,i})$ is hereafter denoted as $P_{i,|P_i|}$, and $P_{i-1,k}(t_{\text{rank}\,i})$ as $P_{i,|P_i|-1}$.

Case 2: $P_{i-1,k}(t_{\text{rank}\,i})$ is empty, and $P_{i-1,k}(t'_{\text{rank}\,i})$ is nonempty
The ordering procedure assigns $|P_i|$ to $P_{i-1,k}(t'_{\text{rank}\,i})$, and ignores $P_{i-1,k}(t_{\text{rank}\,i})$. $P_{i-1,k}(t'_{\text{rank}\,i})$ is hereafter denoted as $P_{i,|P_i|}$.

Case 3: $P_{i-1,k}(t_{\text{rank}\,i})$ is nonempty, and $P_{i-1,k}(t'_{\text{rank}\,i})$ is empty
The ordering procedure assigns $|P_i|$ to $P_{i-1,k}(t_{\text{rank}\,i})$, and ignores $P_{i-1,k}(t'_{\text{rank}\,i})$. $P_{i-1,k}(t_{\text{rank}\,i})$ is hereafter denoted as $P_{i,|P_i|}$.

Next we consider the ordering of partition pairs $\{P_{i-1,j}(t_{\text{rank}\,i}), P_{i-1,j}(t'_{\text{rank}\,i})\}$, where $j = 1,2,\ldots,k-1$. Let the next largest partition order to be assigned be s. Since PBDIA ordering procedure orders $\{P_{i-1,j+1}(t_{\text{rank}\,i}), P_{i-1,j+1}(t'_{\text{rank}\,i})\}$ before $\{P_{i-1,j}(t_{\text{rank}\,i}), P_{i-1,j}(t'_{\text{rank}\,i})\}$, $P_{i,s+1}$ is hence used to denote either $P_{i-1,j+1}(t_{\text{rank}\,i})$ or $P_{i-1,j+1}(t'_{\text{rank}\,i})$. Again, three cases exist for $\{P_{i-1,j}(t_{\text{rank}\,i}), P_{i-1,j}(t'_{\text{rank}\,i})\}$:

Case 1: Both $P_{i-1,j}(t_{\text{rank}\,i})$ and $P_{i-1,j}(t'_{\text{rank}\,i})$ are nonempty
There exist two subcases.
SubCase 1.a: $P_{i,s+1}$ is used to denote $P_{i-1,j+1}(t_{\text{rank}\,i})$
The ordering procedure assigns s to $P_{i-1,j}(t_{\text{rank}\,i})$, and $s-1$ to $P_{i-1,j}(t'_{\text{rank}\,i})$. $P_{i-1,j}\times(t_{\text{rank}\,i})$ is hereafter denoted as $P_{i,s}$, and $P_{i-1,j}(t'_{\text{rank}\,i})$ as $P_{i,s-1}$.
SubCase 1.b: $P_{i,s+1}$ is used to denote $P_{i-1,j+1}(t'_{\text{rank}\,i})$
The ordering procedure assigns s to $P_{i-1,j}(t'_{\text{rank}\,i})$, and $s-1$ to $P_{i-1,j}(t_{\text{rank}\,i})$. $P_{i-1,j}(t'_{\text{rank}\,i})$ is hereafter denoted as $P_{i,s}$, and $P_{i-1,j}(t_{\text{rank}\,i})$ as $P_{i,s-1}$.

Case 2: $P_{i-1,j}(t_{\text{rank}\,i})$ is empty, and $P_{i-1,j}(t'_{\text{rank}\,i})$ is nonempty
The ordering procedure assigns s to $P_{i-1,j}(t'_{\text{rank}\,i})$, and ignores $P_{i-1,j}(t_{\text{rank}\,i})$. $P_{i-1,j}(t'_{\text{rank}\,i})$ is hereafter denoted as $P_{i,s}$.

Case 3: $P_{i-1,j}(t_{\text{rank}\,i})$ is nonempty, and $P_{i-1,j}(t'_{\text{rank}\,i})$ is empty
The ordering procedure assigns s to $P_{i-1,j}(t_{\text{rank}\,i})$, and ignores $P_{i-1,j}(t'_{\text{rank}\,i})$. $P_{i-1,j}(t_{\text{rank}\,i})$ is hereafter denoted as $P_{i,s}$.

4.2.3. PBDIA document identifier assignment procedure

The document identifier assignment procedure, the last step of PBDIA algorithm, is straightforward. Let $P_{n,1}$, $P_{n,2}$, ..., and $P_{n,k}$ be the generated ordered partitions of the iteration n. This procedure assigns consecutive document identifiers to documents in the same partition, and consecutive identifier groups to consecutive ordered partitions. The first (smallest) document identifier is assigned to a document in the first ordered partition ($P_{n,1}$). And the ordering of documents in a partition is irrelevant and can be arbitrary.

To obtain a good DIA, the partitions must be properly ordered. We explain why the PBDIA ordering procedure is proper: Note that the PBDIA ordering procedure always assigns consecutive partition orders to two nonempty partitions of a partition pair. This makes documents in the same partition in iteration i remain in the same or neighboring partitions in iteration $i+1$. According to the PBDIA document

```
Algorithm Partition_based_document_identifier_assignment
Input:
  $D=\{d_1, d_2, \ldots, d_N\}$: a collection of $N$ documents to be indexed.
  $T=\{t_1, t_2, \ldots, t_n\}$: a set of $n$ distinct terms appearing in $D$.
  $Prob=\{p_1, p_2, \ldots, p_n\}$: $p_i$ denotes the probability of the term $t_i \in T$ appearing in a query.
Output:
  A document identifier assignment $\pi: \{d_1, d_2, \ldots, d_N\} \rightarrow \{1, 2, \ldots, N\}$ for the DIA.
Method:
 1. Create an empty doubly linked list PartList;   // to store partition
 2. Create an empty doubly linked list TempList;   //to store partition pairs
 3. Assign all documents in D to a new partition P, and add P to the PartList;
 4. Sort the terms in T in descending order according to their probabilities. Let $t_{\text{rank }1}$, $t_{\text{rank }2}, \ldots,$
    $t_{\text{rank }n}$ represent the sorted list.
 5. for i:=1 to n do
   5.1 while PartList is not empty do   /*partitioning procedure*/
     5.1.1 Get a partition P from the head of PartList, and then remove P from PartList;
     5.1.2  // At least one of the partitions $P(t_{\text{rank }i})$ and $P(t'_{\text{rank }i})$ should be nonempty
           Let $P(t_{\text{rank }i})$ be the partition containing the documents that are included in P and
           do contain the term $t_{\text{rank }i}$; let $P(t'_{\text{rank }i})$ be the partition containing the documents
           that are included in P and do not contain the term $t_{\text{rank }i}$;
     5.1.3 Add the partition pair $\{P(t_{\text{rank }i}),P(t'_{\text{rank }i})\}$ to the tail of TempList;
   5.2 while TempList is not empty do   /*ordering procedure*/
     5.2.1 Get a partition pair $\{P(t_{\text{rank }i}),P(t'_{\text{rank }i})\}$ from the tail of TempList, and then remove
           $\{P(t_{\text{rank }i}),P(t'_{\text{rank }i})\}$ from TempList;
     5.2.2 if $P(t_{\text{rank }i})$ is empty then add $P(t'_{\text{rank }i})$ to the front of PartList and go to step 5.2;
     5.2.3 if $P(t'_{\text{rank }i})$ is empty then add $P(t_{\text{rank }i})$ to the front of PartList and go to step 5.2;
     5.2.4 if PartList is empty then
             Add $P(t'_{\text{rank }i})$ to the PartList; add $P(t_{\text{rank }i})$ to the front of PartList;
           else   //PartList is not empty
             Get a partition P from the head of PartList, and get a document $d \in P$ ;
             if the document d contain the term $t_{\text{rank }i}$ then
               Add $P(t_{\text{rank }i})$ to the front of PartList; add $P(t'_{\text{rank }i})$ to the front of PartList;
             else // the document d does not contain the term $t_{\text{rank }i}$
               Add $P(t'_{\text{rank }i})$ to the front of PartList; add $P(t_{\text{rank }i})$ to the front of PartList;
 6. i:=1;
 7. while PartList is not empty do   /*document identifier assignment procedure*/
   7.1 Get a partition P from the head of PartList, and then remove P from PartList;
   7.2 while P is not empty do
     7.2.1 Get a document $d \in P$, and remove d from P;
     7.2.2 Assign document identifier i to the document d, and then i:=i+1;
```

Fig. 7. The PBDIA algorithm for the DIA problem.

identifier assignment procedure, documents in the same partition in iteration *i* will eventually be assigned consecutive or at least adjacent document identifiers. That is, once the order of partitions is generated at the end of iteration *i*, the compression performance for the posting list of $t_{\text{rank }i}$ is determined. Hence, the posting list of $t_{\text{rank }1}$ has the best compression, then that of $t_{\text{rank }2}$, and so on. This is because the PBDIA algorithm considers the $t_{\text{rank }1}$ first, then $t_{\text{rank }2}$, and so on, in its iterations.

The PBDIA algorithm is given in Fig. 7. A doubly linked list is used to store the partitions, and the two links of a partition maintain the ordering among these partitions. Given a collection of N documents and n distinct query terms, the number of comparisons for assigning documents to partitions in each iteration is $O(N)$. Since the PBDIA algorithm iterates for n times, the total number of comparisons for the PBDIA algorithm is $O(N \times n)$. Compared with the Greedy-NN algorithm, this complexity of PBDIA algorithm is distinctively low. This advantage brings the PBDIA algorithm a dark side, of course. Although the PBDIA algorithm targets on improving the compression efficiency for the frequently used query terms, it unavoidably decreases that for the other query terms. In reality, it is often the case that the popularities of the assorted query terms are very unbalanced. And this imbalance nature makes the PBDIA algorithm achieve very good query performance. In Section 5, we compare the search performance of the Greedy-NN and PBDIA algorithms for real-life document collections.

5. Experiments

This section describes our experiments for evaluating the different DIA algorithms. Experiments were conducted on real-life document collections, and the average query processing time and the storage requirement for each DIA algorithm were measured. We also investigated the DIA problem in parallel IR.

5.1. Document collections and queries

Three document collections were used in the experiments. Their statistics are listed in Table 2. In this table, N denotes the number of documents; n is the number of distinct terms; F is the total number of terms in the collection; and f indicates the number of document identifiers that appear in an inverted file. The collections FBIS (Foreign Broadcast Information Service) and LAT (LA Times) are disk 5 of the TREC-6 collection that is used internationally as a test bed for research in IR techniques (Voorhees & Harman, 1997). The collection TREC includes the FBIS and LAT.

We followed the method (Moffat & Zobel, 1996) to evaluate performance with random queries. For each document collection, 300 documents were randomly selected to generate a query set. A query was generated by selecting words from the word list of a specific document. To form the word list of a document, words in the document were folded to lower case, and stop words such as "the" and "this" were eliminated. The number of terms per query ranged from 1 to 65. For example, a query containing 5 terms may be "inverted file document collection built". For each query, there existed at least one document in the document collection that was relevant to the query. We also made the generated query set for each document collection have the following characteristics: (1) Query repetition frequencies followed a Zipf distribution $\Pr(q) \sim \frac{1}{\rho^{0.6}}$, where $\Pr(q)$ is the probability of query q appearing in generated query set, and ρ is the popularity rank of query q; (2) The terms per query distribution followed the shifted negative binomial distribution $f(x) = \binom{x-0.8}{x-2}(0.85)^{1.2}(0.15)^{x-1}$, where $f(x)$ is the probability of a query containing x words. This made the distribution of generated queries closely resemble the distribution of real queries (Wolfram, 1992; Xie & O'Hallaron, 2002).

5.2. Experimental results

In Section 5.2.1, we first present the actual times taken by the Greedy-NN and the PBDIA algorithms. In Section 5.2.2, we then present the query performance of different DIA algorithms. In Section 5.2.3, we present the compression performance of different DIA algorithms. Finally, we study the DIA problem in parallel IR in Section 5.2.4.

Table 2
Statistics of document collection

		Collection		
		FBIS	LAT	TREC
# of documents	N	130,471	131,896	262,367
# of terms	F	72,922,893	72,087,460	145,010,353
# of distinct terms	n	214,310	168,251	317,393
# of document identifier count	f	28,628,698	32,483,656	61,112,354
Total size (Mbytes)		470	475	945

The inverted files of the three test collections were constructed according to the DIAs generated by different DIA algorithms. We tested four different DIA algorithms: "Random", "Default", "Greedy-NN", and "PBDIA". The Random algorithm means that the document in a collection is randomly assigned document identifier. The Default algorithm means that the document in a collection is assigned document identifier in chronological order. The Greedy-NN and PBDIA algorithms were described in Sections 3.2 and 4.2, respectively. For each DIA algorithm, we also tested five coding methods: γ coding (Elias, 1975), Golomb coding (Golomb, 1966; Witten et al., 1999), skewed Golomb coding (Teuhola, 1978), batched LLRUN coding (Fraenkel & Klein, 1985), and unique-order interpolative coding method (Cheng, Shann, & Chung, 2004). For the following experiments, the parameter b for each posting list in Golomb coding was calculated using Witten's approximation (Witten et al., 1999), and the parameter g for unique-order interpolative coding was set to 4 (Cheng et al., 2004).

All experiments were run on an Intel P4 2.4 GHz PC with 512 MB DDR memory running Linux operating system 2.4.12. The hard disk was 40 GB, and the data transfer rate was 25 MB/s. Intervening processes and disk activities were minimized during experimentation.

5.2.1. Time taken by Greedy-NN and PBDIA algorithms

In Table 3, the performance in terms of completion time is shown. The times reported are the actual times taken by the algorithms to generate a DIA for the given document collection that has been inverted. Please note that the times presented in Table 3 consider neither the time spent in preliminary inversion of the document collection, nor the time needed to rebuild an inverted file with a new DIA.

Table 3 shows that the PBDIA algorithm is much faster than the Greedy-NN algorithm. This fact makes the PBDIA algorithm viable for use in large-scale IRSs. Such a fast DIA algorithm can be very useful for situations such as

1. Dynamically changing probability distribution of query terms, and
2. Dynamically changing document collection.

5.2.2. Query performance of different DIA algorithms

In Table 4, the average query processing time (AvgTime$_{QP}$) and the speedup relative to the Default algorithm (SP) were measured according to Eq. (3). In Table 5, the average number of bits required to retrieve

Table 3
Time consumed by the Greedy-NN and the PBDIA algorithms

DIA algorithm	Collection		
	FBIS	LAT	TREC
Greedy-NN	23 h 59 min	24 h 37 min	198 h 2 min
PBDIA	9 s	10 s	18 s

Table 4
Query performance of different DIA algorithms (AvgTime$_{QP}$ is the average query processing time, and SP is the speedup relative to the Default algorithm)

Collection	DIA algorithm	Coding methods									
		γ coding		Golomb coding		Skewed Golomb coding		Batched LLRUN coding		Unique-order interpolative coding	
		AvgTime$_{QP}$ (μs)	SP	AvgTime$_{QP}$ (μs)	SP	AvgTime$_{QP}$ (μs)	SP	AvgTime$_{QP}$ (μs)	SP	AvgTime$_{QP}$ (μs)	SP
(a) *Short queries*											
FBIS	Random	2989	0.93	2858	0.98	3894	0.96	3748	0.97	2746	0.95
	Default	2789	1.00	2802	1.00	3754	1.00	3636	1.00	2614	1.00
	Greedy-NN	2431	1.15	2790	1.00	3348	1.12	3275	1.11	2315	1.13
	PBDIA	2529	1.10	2808	1.00	3427	1.10	3320	1.10	2333	1.12
LAT	Random	2829	0.96	2704	0.99	3737	0.98	3654	0.97	2564	0.97
	Default	2724	1.00	2688	1.00	3645	1.00	3542	1.00	2476	1.00
	Greedy-NN	2268	1.20	2653	1.01	3137	1.16	3143	1.13	2085	1.19
	PBDIA	2379	1.15	2644	1.02	3234	1.13	3231	1.10	2150	1.15
TREC	Random	5822	0.90	5573	0.97	7556	0.93	7217	0.94	5448	0.91
	Default	5244	1.00	5380	1.00	7026	1.00	6781	1.00	4942	1.00
	Greedy-NN	4431	1.18	5353	1.01	6139	1.14	6032	1.12	4256	1.16
	PBDIA	4606	1.14	5292	1.02	6254	1.12	6171	1.10	4313	1.15
(b) *Medium-length queries*											
FBIS	Random	9388	0.93	8972	0.98	12,222	0.97	11,749	0.97	8613	0.95
	Default	8758	1.00	8795	1.00	11,795	1.00	11,402	1.00	8201	1.00
	Greedy-NN	7563	1.16	8746	1.01	10,426	1.13	10,225	1.12	7205	1.14
	PBDIA	7838	1.12	8798	1.00	10,650	1.11	10,387	1.10	7223	1.14
LAT	Random	8997	0.97	8605	1.00	11,842	0.98	11,562	0.97	8192	0.97
	Default	8684	1.00	8564	1.00	11,580	1.00	11,229	1.00	7932	1.00
	Greedy-NN	7126	1.22	8407	1.02	9851	1.18	9852	1.14	6607	1.20
	PBDIA	7434	1.17	8359	1.02	10,098	1.15	9982	1.12	6755	1.17
TREC	Random	18,475	0.92	17,689	0.97	23,936	0.94	22,724	0.95	17,273	0.93
	Default	16,935	1.00	17,153	1.00	22,594	1.00	21,666	1.00	16,004	1.00
	Greedy-NN	14,069	1.20	16,942	1.01	19,493	1.16	19,058	1.14	13,598	1.18
	PBDIA	14,611	1.16	16,713	1.03	19,809	1.14	19,280	1.12	13,722	1.17
(c) *Long queries*											
FBIS	Random	20,210	0.92	19,399	0.98	26,526	0.95	26,049	0.96	18,423	0.94
	Default	18,594	1.00	18,939	1.00	25,316	1.00	24,984	1.00	17,269	1.00
	Greedy-NN	15,882	1.17	18,971	1.00	22,131	1.14	21,957	1.14	14,979	1.15
	PBDIA	15,871	1.17	18,953	1.00	21,972	1.15	22,143	1.13	14,377	1.20
LAT	Random	18,029	0.96	17,116	1.00	23,591	0.98	22,646	0.97	16,477	0.97
	Default	17,392	1.00	17,035	1.00	23,011	1.00	22,033	1.00	15,964	1.00
	Greedy-NN	13,875	1.25	16,624	1.02	19,173	1.20	18,984	1.16	13,046	1.22
	PBDIA	13,996	1.24	16,298	1.05	19,023	1.21	19,212	1.15	12,817	1.25
TREC	Random	37,881	0.93	36,023	0.98	49,012	0.95	46,584	0.96	35,266	0.94
	Default	35,096	1.00	35,231	1.00	46,547	1.00	44,588	1.00	33,008	1.00
	Greedy-NN	28,372	1.24	34,469	1.02	39,489	1.18	38,592	1.16	27,523	1.20
	PBDIA	29,152	1.20	33,809	1.04	39,766	1.17	39,089	1.14	27,401	1.20

Table 5
AvgBPI$_{QP}$ of different DIA algorithms (AvgBPI$_{QP}$ is the average number of bits required to retrieve and decode an identifier during query processing, and Imp is the improvement over the Default algorithm)

Collection	DIA algorithm	Coding methods									
		γ coding		Golomb coding		Skewed Golomb coding		Batched LLRUN coding		Unique-order interpolative coding	
		AvgBPI$_{QP}$	Imp (%)	AvgBPI$_{QP}$	Imp (%)	AvgBPI$_{QP}$	Imp (%)	AvgBPI$_{QP}$	Imp (%)	AvgBPI$_{QP}$	Imp (%)
(a) *Short queries*											
FBIS	Random	3.56	−10.6	3.21	0.3	3.31	−7.1	3.25	−5.5	3.15	−7.9
	Default	3.22	–	3.22	–	3.09	–	3.08	–	2.92	–
	Greedy-NN	2.78	13.7	3.24	−0.6	2.73	11.7	2.69	12.7	2.63	9.9
	PBDIA	2.95	8.4	3.23	−0.3	2.84	8.1	2.76	10.4	2.69	7.9
LAT	Random	3.32	−6.8	2.98	0.0	3.05	−4.8	3.00	−3.8	2.87	−4.7
	Default	3.11	–	2.98	–	2.91	–	2.89	–	2.74	–
	Greedy-NN	2.56	17.7	3.00	−0.7	2.48	14.8	2.47	14.5	2.35	14.2
	PBDIA	2.73	12.2	2.97	0.3	2.59	11.0	2.59	10.4	2.42	11.7
TREC	Random	3.75	−13.3	3.38	0.3	3.46	−9.5	3.40	−8.2	3.34	−10.6
	Default	3.31	–	3.39	–	3.16	–	3.14	–	3.02	–
	Greedy-NN	2.78	16.0	3.41	−0.6	2.72	13.9	2.69	14.3	2.65	12.3
	PBDIA	2.94	11.2	3.37	0.6	2.81	11.1	2.81	10.5	2.70	10.6
(b) *Medium-length queries*											
FBIS	Random	3.57	−10.9	3.21	0.3	3.31	−6.8	3.25	−5.5	3.15	−7.9
	Default	3.22	–	3.22	–	3.10	–	3.08	–	2.92	–
	Greedy-NN	2.75	14.6	3.24	−0.6	2.70	12.9	2.66	13.6	2.61	10.6
	PBDIA	2.92	9.3	3.24	−0.6	2.81	9.4	2.75	10.7	2.66	8.9
LAT	Random	3.37	−6.3	3.03	0.3	3.11	−4.4	3.06	−3.7	2.94	−4.6
	Default	3.17	–	3.04	–	2.98	–	2.95	–	2.81	–
	Greedy-NN	2.58	18.6	3.06	−0.7	2.50	16.1	2.48	15.9	2.39	14.9
	PBDIA	2.73	13.9	3.02	0.7	2.59	13.1	2.60	11.9	2.44	13.1
TREC	Random	3.83	−12.0	3.42	0.3	3.53	−8.3	3.47	−7.1	3.40	−9.0
	Default	3.42	–	3.43	–	3.26	–	3.24	–	3.12	–
	Greedy-NN	2.82	17.5	3.45	−0.6	2.76	15.3	2.74	15.4	2.71	13.1
	PBDIA	2.99	12.6	3.41	0.6	2.85	12.6	2.86	11.7	2.75	11.9
(c) *Long queries*											
FBIS	Random	3.31	−12.2	3.02	0.3	3.09	−8.4	3.03	−6.7	2.90	−9.0
	Default	2.95	–	3.03	–	2.85	–	2.84	–	2.66	–
	Greedy-NN	2.50	15.3	3.06	−1.0	2.47	13.3	2.43	14.4	2.37	10.9
	PBDIA	2.57	12.9	3.05	−0.7	2.47	13.3	2.48	12.7	2.34	12.0
LAT	Random	3.58	−6.2	3.21	0.3	3.28	−4.1	3.23	−3.5	3.13	−4.3
	Default	3.37	–	3.22	–	3.15	–	3.12	–	3.00	–
	Greedy-NN	2.66	21.1	3.24	−0.6	2.58	18.1	2.55	18.2	2.50	16.7
	PBDIA	2.73	19.0	3.19	0.9	2.58	18.1	2.63	15.7	2.48	17.3
TREC	Random	3.85	−10.6	3.43	0.3	3.54	−7.3	3.47	−6.1	3.41	−7.9
	Default	3.48	–	3.44	–	3.30	–	3.27	–	3.16	–
	Greedy-NN	2.78	20.1	3.46	−0.6	2.73	17.3	2.70	17.4	2.69	14.9
	PBDIA	2.92	16.1	3.41	0.9	2.79	15.5	2.81	14.1	2.71	14.2

and decode an identifier during query processing ($\text{AvgBPI}_{\text{QP}}$) and the improvement over the Default algorithm (Imp) were measured according to Eq. (6). For each document collection, the generated query set was divided into three subsets: the short query set, the medium-length query set, and the long query set. The number of terms per query for the short, medium-length, and long query sets range from 1 to 8, 9 to 20, and 21 to 65, respectively.

All decoding mechanisms were optimized, including:

1. Replaced subroutines with macros.
2. Replaced calls to the log function with fast bit shifts.
3. Careful choice for compiler optimization flags.
4. Implementation used 32-bit integers, as that is the internal register size of the Intel P4 CPU.

Furthermore, the Huffman code of batched LLRUN coding was implemented with canonical prefix codes that can be decoded via a fast table look-up (Turpin, 1998). With these optimizations, decoding of a document identifier only required tens of nanoseconds.

The experimental results are shown in Tables 4 and 5. Key findings are:

1. Table 4 shows that the query performance of the Default algorithm can be 10% faster than the Random algorithm. This indicates that the Default algorithm already captures some clustering nature, thus can serve as a rigid baseline in comparison with other fine-tuned algorithms.
2. Comparing Tables 4 and 5, one should observe that $\text{AvgTime}_{\text{QP}}$ is proportional to $\text{AvgBPI}_{\text{QP}}$. This verifies Eq. (4) in Section 3.1, and explains why a good DIA can result in better compression and reduced query processing time.
3. From Table 5, one should observe that both the Greedy-NN and PBDIA algorithms can result in better compression of posting lists for all tested coding methods except Golomb coding. This indicates that the Greedy-NN and PBDIA algorithms can improve the cache efficiency if a posting list cache is implemented.
4. Table 4 shows that both the Greedy-NN and PBDIA algorithms can reduce average query processing time for all tested coding methods except Golomb coding. And the query speedup differences between the Greedy-NN and PBDIA algorithms were only 3% on average. Considering the algorithm complexity, the PBDIA algorithm is a good choice for the DIA problem.
5. From Table 4, one should observe that Golomb coding cannot benefit much from the Greedy-NN and PBDIA algorithms in terms of query performance. This is because Golomb coding assumes that the *d*-gap values in a posting list following a Bernoulli model (Witten et al., 1999), hence both the compression result and the query processing time of Golomb coding are independent of *d*-gap distribution.
6. From Table 4, one should observe that the query speedup obtained by the PBDIA algorithm becomes higher as the query length increases. This is because that, as the number of query terms increases, more frequently used query terms are likely to be included, resulting in more advantage due to the PBDIA algorithm.
7. Table 4 shows that both γ coding and unique-order interpolative coding are recommended for real-world IRSs due to their fast query throughputs. In addition, compared with the other tested coding methods, these two coding methods benefit more from the PBDIA algorithm. We conclude that the PBDIA algorithm is viable for use in real-world IRSs.
8. Table 4 shows that the PBDIA algorithm can reduce average query processing time by up to 20% for an inverted file in which the document identifiers in a posting list are sorted in ascending order. To allow extremely fast processing of conjunctive queries and ranked queries using the same index, most IRSs in use today adopt the skipped inverted files (Moffat & Zobel, 1996) or the blocked inverted files (Moffat, Zobel, & Klein, 1995). Both the skipped and blocked inverted files are identifier-ordered arrangement.

Table 6
Compression performance of different DIA algorithms (BPI is the average bits per identifier of the inverted file for the test collection, and Imp is the improvement over the Default algorithm)

Collection	DIA algorithm	Coding methods									
		γ coding		Golomb coding		Skewed Golomb coding		Batched LLRUN coding		Unique-order interpolative coding	
		BPI	Imp (%)	BPI	Imp (%)	BPI	Imp (%)	BPI	Imp (%)	BPI	Imp (%)
FBIS	Random	7.06	−19.7	5.28	0.0	5.75	−10.6	5.38	−8.5	5.36	−10.3
	Default	5.90	–	5.28	–	5.20	–	4.96	–	4.86	–
	Greedy-NN	5.86	0.7	5.28	0.0	5.33	−2.5	4.88	1.6	4.85	0.2
	PBDIA	6.17	−4.6	5.28	0.0	5.42	−4.2	5.06	−2.0	4.95	−1.9
LAT	Random	7.12	−6.6	5.33	0.0	5.73	−3.2	5.43	−2.8	5.42	−3.8
	Default	6.68	–	5.33	–	5.55	–	5.28	–	5.22	–
	Greedy-NN	6.06	9.3	5.32	0.2	5.26	5.2	5.00	5.3	4.91	5.9
	PBDIA	6.35	4.9	5.32	0.2	5.33	4.0	5.12	3.0	5.01	4.0
TREC	Random	7.39	−16.7	5.50	−0.4	5.92	−9.2	5.59	−7.5	5.59	−9.6
	Default	6.33	–	5.48	–	5.42	–	5.20	–	5.10	–
	Greedy-NN	6.08	3.95	5.49	−0.2	5.39	0.6	5.03	3.3	4.99	2.2
	PBDIA	6.36	−0.5	5.49	−0.2	5.45	−0.6	5.18	0.4	5.08	0.4

Therefore, the PBDIA algorithm can also be applied to those inverted files, and reduce the time needed to process a query against those inverted files. Since skipped inverted files and blocked inverted files are widely used in modern large-scale IRSs, we believe that the PBDIA algorithm can contribute in real-world IRSs.

5.2.3. *Compression performance of different DIA algorithms*

The compression results are shown in Table 6, and the metric used is the average number of *bits per identifier* (BPI), defined as follows:

$$\mathrm{BPI} = \frac{\text{The size of the compressed inverted file}}{\text{number of document identfiers } f}.$$

To reduce average query processing time, both the Greedy-NN and PBDIA algorithms target on improving the compression efficiency for the frequently used query terms. However, this is at the cost of sacrificing the compression efficiency for the less frequently used query terms. We need to know how much space overhead is needed to trade for this speed advantage. Results in Table 6 shows that the Greedy-NN and PBDIA algorithms can speed up query processing with very little or no storage overhead.

5.2.4. *DIA in parallel IR*

This subsection investigates the DIA problem in an IRS that runs on a cluster of workstations. Assuming k workstations, the inverted file is generally partitioned into k disjoint sub-files, each for one workstation. When processing a query, all workstations have to consult only their own sub-files in parallel, and the query processing time is shortened. Ma, Chen, and Chung (2002) indicated that near-ideal speedup on query processing can be obtained if an inverted file is partitioned using the interleaving partitioning scheme. For such a partitioning, DIA plays a crucial role in load balancing. The PDBIA algorithm can be applied to the inverted file to enhance the clustering property of posting lists for frequently used query terms, and can aid the interleaving partitioning scheme to yield better load balancing.

Table 7
Speedup of parallel query processing

Method	The number of workstations					
	1[a]	2	4	6	8	10
Default algorithm + Interleaving partitioning	1.00	1.90	3.75	5.61	7.44	9.35
PBDIA algorithm + Interleaving partitioning	1.17	2.23	4.41	6.57	8.70	10.93

[a] Without interleaving partitioning.

Table 7 shows the performance of parallel query processing using interleaving partitioning scheme with either the Default algorithm or the PBDIA algorithm. The metric is the speedup relative to sequential query processing with Default algorithm. Experiments were conducted on the TREC collection. The sub-file on each workstation was compressed using the unique-order interpolative coding method. The parallel query processing time was defined as $\max[T_1, T_2, \ldots, T_k]$, where T_i $(1 \leqslant i \leqslant k)$ was the time needed to retrieve and decompress the (partial) posting lists for the query terms on the ith workstation. Note that T_i did not include the document identifier comparison time (the reason being the same as described in Section 3.1). The experimental results show that the interleaving partitioning scheme can yield near-ideal speedups, as reported in Ma et al. (2002). In addition, using the PBDIA algorithm to enhance the clustering property of posting lists for frequently used query terms, the interleaving partitioning scheme yields super-linear speedups. Hence the DIA problem should deserve much attention in parallel IR.

6. Conclusion

In this paper, we study the DIA-based query optimization techniques for an IRS in which the inverted file is used to evaluate queries. We first define a cost model for query evaluation. Based on this model, we propose an efficient heuristic, called *partition-based document identifier assignment* (PBDIA) algorithm, for generating a good DIA to reduce average query processing time.

The PBDIA algorithm can efficiently assign consecutive document identifiers to the documents containing frequently used query terms. This makes the *d*-gaps of posting lists for frequently used query terms very small, and results in better compression for popular coding methods without increasing the complexity of decoding processes. This can result in reduced query processing time.

Experimental results show that the PBDIA algorithm can reduce the average query processing time by up to 20%. We also point out that the DIA problem has vital effects on the performance of long queries and parallel IR. Compared with the well-known Greedy-NN algorithm, the PBDIA algorithm is much faster and yields very competitive performance for the DIA problem. This fact should make the PBDIA algorithm viable for use in modern large-scale inverted file-based IRSs.

Acknowledgements

This work was supported by National Science Council, ROC: NSC93-2213-E-009-025.

References

Bellman, R. E., & Dreyfus, S. E. (1962). *Applied dynamic programming*. Princeton, NJ: Princeton University Press.

Cheng, C. S., Shann, J. J.-J., & Chung, C. P. (2004). A unique-order interpolative code for fast querying and space-efficient indexing in information retrieval systems. In P. K. Srimani et al. (Eds.). *Proceedings of ITCC 2004 international conference on information* *technology: coding and communications, Las Vegas, Nevada, April* (Vol. 2, pp. 229–235). Los Alamitos, CA: IEEE Computer Society Press.

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

Fraenkel, A. S., & Klein, S. T. (1985). Novel compression of sparse bit-string—preliminary report. In A. Apostolico & Z. Galil (Eds.), *Combinatorial algorithms on words. NATO ASI serials F* (Vol. 12, pp. 169–183). Berlin: Springer-Verlag.

Gelbukh, A., Han, S. Y., & Sidorov, G. (2003). Compression of Boolean inverted files by document ordering. In *Proceedings of 2003 IEEE international conference on natural language processing and knowledge engineering (IEEE NLPKE-2003), Beijing, China, October* (pp. 244–249). Los Alamitos, CA: IEEE Computer Society Press.

Golomb, S. W. (1966). Run length encoding. *IEEE Transactions on Information Theory, IT-12*(3), 399–401.

Janson, B. J., Spink, A., Bateman, J., & Saracevic, T. (1998). Real life information retrieval: a study of user queries on the web. *SIGIR Forum, 32*(1), 5–17.

Kobayashi, M., & Takeda, K. (2000). Information retrieval on the web. *ACM Computing Surveys, 32*(2), 144–173.

Ma, Y. C., Chen, T. F., & Chung, C. P. (2002). Posting file partitioning and parallel information retrieval. *Journal of Systems and Software, 63*(2), 113–127.

Moffat, A., & Zobel, J. (1992). Parameterised compression for sparse bitmaps. In N. Belkin, P. Ingwersen, & A. M. Pejtersen (Eds.), *Proceedings of 15th annual international ACM-SIGIR conference on research and development in information retrieval, Copenhagen, June* (pp. 274–285). New York: ACM Press.

Moffat, A., & Zobel, J. (1996). Self-indexing inverted files for fast text retrieval. *ACM Transactions on Information Systems, 14*(4), 349–379.

Moffat, A., Zobel, J., & Klein, S. T. (1995). Improved inverted file processing for large text databases. In R. Sacks-Davis, & J. Zobel (Eds.), *Proceedings of 6th Australasian database conference, Adelaide, Australia, January* (pp. 162–171).

Olken, F., & Rotem, D. (1986). Rearranging data to maximize the efficiency of compression. In *Proceedings of the fifth ACM SIGACT-SIGMOD symposium on principles of database systems, Cambridge, MA, United States, March* (pp. 78–90). New York: ACM Press.

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

Teuhola, J. (1978). A compression method for clustered bit-vectors. *Information Processing Letters, 7*(6), 308–311.

Turpin, A. (1998). *Efficient prefix coding*. Ph.D. thesis, University of Melbourne, Melbourne.

Voorhees, E., & Harman, D. (1997). Overview of the sixth text retrieval conference (TREC-6). In E. M. Voorhees & D. K. Harman (Eds.), *Proceedings of the sixth text retrieval conference (TREC-6)* (pp. 1–24). Gaithersburg, MD: NIST.

Williams, H. E., & Zobel, J. (1999). Compressing integers for fast file access. *The Computer Journal, 42*(3), 193–201.

Williams, H. E., & Zobel, J. (2002). Indexing and retrieval for genomic databases. *IEEE Transactions on Knowledge and Data Engineering, 14*(1), 63–78.

Witten, I. H., Moffat, A., & Bell, T. C. (1999). *Managing gigabytes: compressing and indexing on documents and images* (2nd ed.). San Francisco, CA: Morgan Kaufmann Publishers.

Wolfram, D. (1992). Applying informetric characteristics of databases to IR system file design, Part I: Informetric models. *Information Processing and Management, 28*(1), 121–133.

Xie, Y. & O'Hallaron, D. (2002). Locality in search engine queries and its implications for caching. In P. Kermani, F. Bauer, & P. Morreale (Eds.), *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'02), New York, June* (pp. 1238–1247).

Zipf, G. (1949). *Human behavior and the principle of least effort*. New York: Addison-Wesley.

Zobel, J., & Moffat, A. (1995). Adding compression to a full-text retrieval system. *Software Practice and Experience, 25*(8), 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.

Cher-Sheng Cheng received the B.S. and M.E. degrees in computer engineering from the Department of Computer Science and Information Engineering, National Chiao Tung University, Taiwan, ROC in 1994 and 1996, respectively. 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.

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 returned to his teaching position after this three-year assignment. His research interests include computer architecture, parallel processing, and parallelizing compiler.

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.