Cluster based Mixed Coding Schemes for Inverted File Index Compression

Jinlin Chen[1], Ping Zhong[2], Terry Cook[3]
[1]Computer Science Department
Queen College, City University of New York
USA
jchen@cs.qc.edu

[2]Computer Science Department
Graduate Center, City University of New York
USA
pzhong@gc.cuny.edu

[3]Computer Science Department
Graduate Center, City University of New York
USA
terrycookd1@aol.com

Journal of Digital Information Management

ABSTRACT: *The cluster property of document collections in today's search engines provides valuable information for index compression. By clustering d-gaps of an inverted list based on a threshold, and then encoding clustered and non-clustered d-gaps using different methods, we can tailor to the specific properties of different d-gaps and achieve better compression ratio. Based on this idea, in this paper we propose a cluster based approach and presents two new codes for inverted file index compression: mixed gamma/flat binary code and mixed delta/flat binary code. Experiment results show that the two new codes achieve better or equal performance in terms of compression ratio comparing to interpolative code which is considered as the most efficient bitwise code at present. Besides, the two new codes have much lower complexity comparing to interpolative code and therefore enable faster encoding and decoding. By adjusting the parameters for the mixed codes, even better results may be achieved. Experiments show promising results with our approaches.*

Categories and Subject Descriptors
H.3.2 [Information Storage] File Organization; H.3.1 [Content analysis and indexing]; I.7.3 [Index generation]

General Terms
Document processing, Index generation

Keywords: Inverted file, *d*-gap, Index compression, inverted list
Received 28 Aug. 2006; Revised and accepted 29 August 2007

1. Introduction

Today Web search engines play an important role for people to access Web information. The large amount of information available on the Web requires an efficient indexing mechanism for search engines. Among the many indexing techniques, inverted file has been the most popular one due to its relative small size and high efficiency for keyword-based queries [10][16][17]. An inverted file index on a document collection maps each unique term to an inverted list of all the documents containing the term. For a term *t*, the inverted list has the structure $<f_t; d_1, d_2, d_3, \ldots, d_{ft}>$, where f_t is the number of documents containing *t*, d_i is a DocID that identifies the document associated with the i^{th} occurrence of *t*, and $d_i < d_{i+1}$. Since the inverted list is in ascending order of DocIDs, and all processing is sequential from the beginning of the list, the list can be stored as an initial position followed by a list of *d*-gaps by taking consecutive differences, $d_{i+1} - d_i$. In this way it is possible to code inverted lists using fewer bits per pointer on average.

Many codes have been proposed for compressing inverted lists. These codes use different codewords for different *d*-gaps. The performance of a code is decided by whether the implicit *d*-gap distribution model conforms to that of the document collection.

One way to improve inverted file compression is to use the cluster property [1] of document collection, which states that term occurrences are not uniformly distributed. Some terms are more frequently used in some parts of the collection than in others. The corresponding part of the inverted list will consequently be small *d*-gap values clustered. Interpolative code [9] exploits the cluster property of term occurrences and achieves very good performance. Other codes that favor small *d*-gaps also perform well on document collections with cluster property.

A major feature of most previous approaches is that they use the same code within a given inverted list, without considering the difference in between clustered and non-clustered *d*-gaps. Actually the knowledge of cluster and non-cluster property of *d*-gaps provides valuable information for improving index compression. By clustering *d*-gaps of an inverted list strictly based on a threshold, and then encoding clustered and non-clustered *d*-gaps using different methods, we can tailor to the specific properties of different *d*-gaps and achieve better compression ratio. Based on this idea, in this paper we propose a cluster based approach and present two new mixed codes for inverted file index compression: mixed *k*-base gamma/*k*-flat binary code and mixed *k*-base delta/*k*-flat binary code. Experiment results show that the two new codes achieve better or equal performance in terms of compression ratio comparing to interpolative code which is considered as the most efficient bitwise code at present. Besides, the two new codes have much lower complexity comparing to interpolative code and therefore enable faster encoding and decoding. By adjusting the parameters for the mixed codes, even better result may be achieved.

The rest of this paper is organized as follows. Section 2 describes related work on inverted file indexing. Section 3 presents the motivation of this paper. Section 4 discusses the concept of cluster based mixed code and presents two new mixed codes. Section 5 presents experiment results for performance evaluation. Section 6 concludes the paper.

2. Related work

Approaches for compressing inverted lists can be grouped into global methods which use the same model for all inverted lists and local methods which use different models for different inverted lists [16].

The simplest global code is flat binary code which requires $\lceil \log N \rceil$ bits for each pointer in a collection with N documents (*Note*: we will use base 2 logarithm throughout this paper). The implicit probability model is that each *d*-gap size is uniformly random in 1 … N. However, in reality frequent words are likely to have small *d*-gaps while infrequent words tend to have larger *d*-gaps. Thus variable length code should be considered in which small values are considered more likely and coded more economically than larger ones. One such code is the unary code, which codes a *d*-gap x with $x - 1$ '1' bits followed by a '0' bit. The corresponding probability model is binary exponential decay, which favors small *d*-gaps extremely.

There are many codes whose implicit probability distributions lie somewhere between the uniform distribution of flat binary code and the binary exponential decay distribution of unary code. Elias gamma code [5] represents an integer x by $1 + \lfloor \log x \rfloor$ stored as a unary code, followed by $\lfloor \log x \rfloor$ bits binary representation of x without its most significant bit ($x - 2^{\lfloor \log x \rfloor}$). Gamma code is efficient for small integers but inefficient for larger integers. Elias delta code [5], instead, is suitable for large integers but inefficient for small ones. For an integer x, a delta code stores the gamma code representation of $1 + \lfloor \log x \rfloor$ and then the binary representation of x without its most significant bit.

Gamma and delta codes use the same model for every inverted list. Parameterized local methods, on the contrary, take advantage of the different average *d*-gaps in different lists. Golomb code [6] is such an example. Coding of an integer x using Golomb code with respect to a parameter b is as follows. First, a zero-origin unary code of a quotient q is emitted, where $q = \lfloor (x - 1) / b \rfloor$; second, a binary code is emitted for the remainder r, $r = (x - 1) - q \times b$. This binary code requires either $\lfloor \log b \rfloor$ (if $r < p$), or $\lceil \log b \rceil$ (if $r \geq p$) bits to be represented, where p is the pivot point, and $p = 2^{\lfloor \log b \rfloor + 1} - b$.

Golomb code is effective when the probability distribution of *d*-gap is geometric, however, it does not adapt well to the cluster property which is quite normal in many of today's document collections [1]. The cluster property indicates that term occurrences are not uniformly distributed. Some terms are more frequently used in some parts of a collection than in others. The corresponding part of the inverted list will mainly be small *d*-gaps clustered. A mechanism that is sensitive to clustering may outperform Golomb code on non-uniform inverted lists. Interpolative code [9] is such an example. Interpolative code encodes a monotonic increasing sequence of integers using knowledge of its neighbors. If in a sequence of integers X, for a given integer x_i, the preceding integer x_{i-1} and following integer x_{i+1} are known, then because x_i must be in the interval $[x_{i-1} + 1, x_{i+1} - 1]$, the maximum number of bits required for x_i is $\lceil \log (x_{i+1} - x_{i-1} - 2 + 1) \rceil$. For typical document collections, the interpolative code achieves better compression than Golomb code. The major problem is its high encoding and decoding complexity [16].

All the above mentioned codes are bitwise schemes, in which each posting is stored using a bit-aligned code. Generally speaking, bitwise codes achieve very good compression ratios. The problem is the relatively high decoding time. Anh and Moffat [1] proposed fixed binary codes to enhance the decoding performance. Fixed binary code splits an inverted list into multiple segments, each of which includes *d*-gaps at similar scale. Thus each segment can be encoded with a fixed number of bits which best accommodate the segment. This idea of dividing an inverted list into segments is quite interesting and related to the work presented in this paper. However, the major interest of fixed binary code is to improve decoding performance without sacrificing too much encoding performance. And because of this, fixed binary code focuses on building efficient data structures for future decoding at encoding stage. Generally speaking it yields larger index due to the additional cost of storing selector and length information for each segment. Our research, instead, focus on how to build efficient encoding schemes to improve index compression by making better use of cluster property.

Another type of codes is bytewise codes [2][11][14], in which each posting is stored employing a byte-aligned code. In bytewise encoding, each integer is represented using a fixed and integral number of bytes. Thus retrieving a codeword doesn't involve expensive shift-mask operations. Bytewise codes generally gain low decoding time with the cost of lower compression ratio comparing to bitwise codes.

3. Motivation

Clustering property of *d*-gaps has been exploited to improve compression ratio [4][8][9]. However, previous approaches use the same code within a given inverted list without considering the difference of clustered and non-clustered *d*-gaps.

Actually the knowledge of clustered and non-clustered *d*-gaps provides valuable information for index compression. For example, given an inverted list <12; 38, 17, 13, 34, 6, 4, 1, 3, 1, 2, 3, 1>, if we define a cluster as continuous *d*-gaps that are less than 4, we can break the list into two segments <12; (38, 17, 13, 34, 6, 4), (1, 3, 1, 2, 3, 1)>. For the first segment, since every *d*-gap is larger than or equal to 4, we can represent a *d*-gap x in two parts: a gamma code (or any other suitable code) of $x/4$, followed by two bits binary code for the remainder of $x/4$. Gamma code of x has $1 + 2\lfloor \log x \rfloor$ bits, while in our new approach the total number of bits is: $1 + 2\lfloor \log (x/4) \rfloor + 2 = 1 + 2\lfloor \log x \rfloor - 2$, which is two bits less than the original gamma code. For the second segment, since every *d*-gap is smaller than 4, we can exploit codes that favor small *d*-gaps to minimize the total number of bits. For example, we can use two bits flat binary code to represent the possible *d*-gaps in a cluster, i.e., 00 (1), 01 (2), and 10 (3).

In this way mixed codes are used for the inverted list (gamma for non-cluster *d*-gaps and flat binary for *d*-gaps in a cluster). Here the major issue is how to indicate the start and end of a cluster, so that during decoding we know which decoding scheme to use. For this example, since gamma code for any integer larger than 1 always starts with bit '1', this gives us the idea of using '0' as the starting bit of a cluster. Inside a cluster the two bits binary code '11' is never used, which can be used to indicate the end of a cluster. Comparing to gamma coding, which use 2.33 bits in average to represent integer 1 (coded as 0), 2 (coded as 100), and 3 (coded as 101) if they occur uniformly, our two bits flat binary code only uses two bits in average. However, this comes at the cost of three additional bits (one starting and two ending bits). Fortunately, we can offset the cost of the two ending bits based on the observation that the *d*-gap x following the cluster is surely larger than 3. Based on this observation, we can encode x using gamma for $x/4$ plus two bits binary code for the remainder, which needs two less bits

than normal gamma code, so that the two ending bits of the cluster are offset. In case the cluster is at the end of an inverted list, there is no need to use the two ending bits, and the additional cost vanishes. Based on this analysis the only additional cost of representing a cluster is the one bit starting indicator, which is not a big issue because in average each *d*-gap inside a cluster uses 0.33 less bit, and each non-cluster *d*-gap uses 2 less bits comparing to normal gamma code.

A remaining issue is, if a non-clustered *d*-gap *x* is smaller than 8 (i.e., 4, 5, 6, or 7.), the quotient of *x*/4 will then be 1, whose gamma code is 0, which is the same as the proposed starting bit for a cluster and causes conflict. To solve this problem, we observe that the binary code '011' is never used in the first three bits of any cluster (otherwise it means a cluster with no *d*-gap inside, which is impossible). Therefore we can use '011' to indicate the starting bits of a *d*-gap 4, 5, 6, or 7. And because only these four values need special consideration (a non-clustered *d*-gap is always larger than or equal to 4 as defined. For a *d*-gap *x* larger than 7 the corresponding gamma code of *x*/4 is led by bit '1', which is distinguishable from the leading bit '0' of a cluster), we need only two additional bits after '011' to represent these four values ('00', '01', '10', and '11' for 4, 5, 6, and 7, respectively). In this case each of these four integers needs five bits, which is the same as its corresponding gamma code.

The above analysis can be further generalized in two aspects. First, instead of using 3, we can use any appropriate number as the maximal distance inside a cluster, preferably 2^k - l, where *k* is a positive integer. The larger *k* is, the more bits we can save for a non-clustered *d*-gap, at the cost of more bits for a clustered *d*-gap. Second, instead of using gamma code, we can also use other codes such as delta code for non-clustered *d*-gaps.

Since our approach makes use of clustering property of document collections, it will also benefit from those DocID reassignment algorithms [3][12][13] which have the purpose of increasing clustering property.

4. Cluster based mixed codes

In this section we first give formal definition of cluster based mixed code, we then present two mixed codes – mixed *k*-base gamma/*k*-flat binary code and mixed *k*-base delta/*k*-flat binary code.

Definition 1 For a segment of *d*-gaps $< d_i, d_{i+1}, \ldots, d_j > (i \le j)$ in an inverted list, if for $\forall k, i \le k \le j, d_k \le T$, (*T* is a positive integer), the segment is called a **Cluster** with **Maximal Cluster Distance** (MCD) *T*. A *d*-gap larger than MCD is called a **Non-Clustered** *d*-gap. (***Note***: unless otherwise specified, in this paper we represent an inverted list in its *d*-gap format.)

Definition 2 For an inverted list $<f_t; d_1, d_2, d_3, \ldots, d_{ft}>$, if we cluster all the possible clusters based on a given MCD, and re-write the inverted list as $<f_t; g_1, g_2, g_3, \ldots, g_{ft}>$, where g_i is either a non-clustered *d*-gap or a cluster, and f_t is the total number of clusters and non-clustered *d*-gaps, we call this new list the **Clustered Representation** of the original list.

Definition 3 For the clustered representation of an inverted list, if we encode non-clustered *d*-gaps and clustered *d*-gaps with different codes, we call this encoding approach a **Cluster based Mixed Code**.

Definition 4 A cluster with MCD *T*, $T = 2^k - 1$, $k \ge 1$, is called a ***k*-base Cluster**. A *k*-base cluster can have at most $2^k - 1$ different *d*-gap values inside the cluster, *i.*e., 1, 2, …, 2^k - 1.

Definition 5 For a *k*-base cluster, if each *d*-gap inside the cluster is coded using *k* bits one-origin binary code, we call this code ***k*-flat Binary Code** for the cluster, in which *d*-gap value 1 is coded as 0, 2 is coded as 1, … and 2^k - 1 is coded as $2^k - 2$ (all with *k* bits). The binary code $2^k - 1$ is never used and can be used for other purpose.

4.1 Mixed Gamma/Flat Binary Code

Definition 6 For a *d*-gap *x*, $x \ge 2^k$, if *x* is coded as gamma code of $x/2^k$, followed by *k* bits binary representation of the remainder of $x/2^k$, we call this coding scheme ***k*-base Gamma Code**.

Algorithm 1 For an inverted list, a **Mixed *k*-base Gamma/*k*-flat Binary Code** (for simplicity, we will call it *mixed gamma code*) is defined as follows,

Input: an inverted list; **Output:** mixed *k*-base gamma/*k*-flat binary code;

Algorithm:

1) Rewriting the inverted list in its *k*-base clustered representation;
2) Starting from the beginning of the clustered representation, each item is encoded sequentially as follows,
   2.1) If the current item is a cluster, it is encoded with a starting bit '0', and k '1' bits as ending bits if there is a non-clustered d-gap following the cluster. Each d-gap inside the cluster is encoded using k-flat binary code sequentially following the starting bit;
   2.2) If the current item is a non-clustered d-gap that follows a cluster, it is encoded using k-base gamma code;
   2.3) If the current item is a non-clustered d-gap x that does not follow a cluster, and if $x \ge 2 \times 2^k$, it is encoded using k-base gamma code; if $x < 2 \times 2^k$, it is encoded as follows: a leading bit '0', followed by k '1' bits, followed by k bits binary code of $x - 2^k$. Here the first bit '0' together with the following k '1' bits are called **Special Leading Sequence** of x.

Algorithm 2 For an inverted list coded with mixed *k*-base gamma/*k*-flat binary code, a **Mixed *k*-base Gamma/*k*-flat Binary Decoding** algorithm is defined as follows,
Input: mixed gamma code of an inverted list; **Output:** the original inverted list;

Algorithm: Starting from the beginning of the code,

1) If a leading bit is '0', and the following *k* bits are not all '1's, then it means the bits following the starting bit '0' are for a cluster, and we decode the cluster based on coding scheme defined by *k*-flat binary code, *i.*e., following the starting bit '0', each time we take *k* bits and the corresponding *d*-gap is 1+ binary value of the *k* bits. This process is continued until *k* all '1' bits are read, or the end of the inverted list is met;
   1.1) If *k* bits of all '1's at the end of a cluster are read, it means that the following *d*-gap *x* is a non-clustered *d*-gap. We then decode the bits following the ending bits of the cluster using gamma decoding (assume the decoded value is *y*), and further decode the following *k* bits after the gamma code using binary decoding (assume the decoded value is *z*). Then we have $x = 2^k \times y + z$;
2) If a leading bit is '0', and the following *k* bits are all '1's, then it means the *k* bits following all '1's are for a non-clustered *d*-gap *x*, $x < 2 \times 2^k$, and $x \ge 2^k$. We decode *x* as $x = 2^k$ + binary value of the *k* bits following the *k* '1' bits.

3) If a leading bit is '1', it means the following bits together with the leading bit '1' are for a non-clustered *d*-gap x, $x \geq 2 \times 2^k$. We first decode the bits starting from the leading bit '1' using gamma decoding (assume the decoded value is y), and further decode the following k bits after the gamma code using binary decoding (assume the decoded value is z). And we have $x = 2^k \times y + z$.

Theorem 1 There is no ambiguity to decode a mixed *k*-base gamma/*k*-flat binary code using mixed *k*-base gamma/*k*-flat binary decoding algorithm, and the decoded result is the same as the original inverted list.

Proof: The value of a *d*-gap x falls into one of the following three intervals:

(1) $x < 2^k$, (2) $2^k \leq x < 2 \times 2^k$, (3) $2 \times 2^k \leq x$.

Based on the location of x relative to clusters, the following three cases exist:

(a) x is inside a cluster; (b) x is outside a cluster and a successor of a cluster; (c) x is outside a cluster and not a successor of a cluster.

Based on the above partitions and the definition of mixed *k*-base gamma/*k*-flat binary code, only the following combinations can exist: 1a, 2b, 2c, 3b, 3c. During encoding, for case 1a, the corresponding cluster is led with bit '0', and the following k bits are guaranteed to be not all '1's; for case 2c, the leading bit is '0', followed by k bits all '1's; for case 3c, since $2 \times 2^k \leq x$, $x/2^k \geq 2$, the corresponding *k*-base gamma code of x is guaranteed to have the leading bit '1' (all gamma codes for $x > 1$ has a leading bit '1'). So there is no ambiguity for these three cases during decoding. For case 2b and 3b, since x is a direct successor of a cluster, the implicit leading bits are the ending bits of the precedent cluster. Therefore there is no ambiguity in between these two cases and the other three cases for decoding. Besides, since $2^k \leq x$, k base gamma coding ensures no ambiguity for corresponding decoding in between these two cases.

For each of the above cases, the decoding procedure defined in Algorithm 2 is a direct reverse of the corresponding encoding procedure defined in Algorithm 1, so the decoded list is the same as the original one.

Example 1 Using the same inverted list as used for fixed binary code in [1], <12; 38, 17, 13, 34, 6, 4, 1, 3, 1, 2, 3, 1>

1) Gamma, delta, Golomb (b=3), interpolative, and fixed binary codes need 60, 62, 64, 55, and 57 bits, respectively; *Note*: to test interpolative code, we need know the size of the document collection, which is not given in [1]. Here we take a reasonable assumption that the document collection size is the sum of all *d*-gaps in this inverted list plus the average *d*-gap of the list. In this way we use 134 as the collection size.
2) If we use mixed 2-base gamma/2-flat binary code, MCD = $2^2 - 1 = 3$. There exists one cluster <1, 3, 1, 2, 3, 1> inside the list, and the mixed gamma code is 1110001 10 (38) 11000 01 (17) 101 01 (13) 1110000 10 (34) 011 10 (6) 011 00 (4) 0 00 (1) 10 (3) 00 (1) 01 (2) 10 (3) 00 (1), which has 53 bits. Comparing to gamma code, we save 7 bits. Comparing to interpolative code which performs best among all codes in 1), we save 2 bits here.
3) If we use mixed 3-base gamma/3-flat binary code, MCD = $2^3 - 1 = 7$. There exists one cluster <6, 4, 1, 3, 1, 2, 3, 1> inside the list, and the mixed gamma code is 11000 110 (38) 100 001 (17) 0111 101 (13) 11000 010 (34) 0 101 (6) 011 (4) 000 (1) 010 (3) 000 (1) 001 (2) 010 (3) 000 (1), which has 54 bits. Comparing to gamma code, we save 6 bits. Comparing to interpolative code, we save 1 bit here.

This example indicates that different bases (k) have different effects for the coding results. Below are two theorems describing the number of bits consumed by mixed gamma code.

Theorem 2 For a *d*-gap x encoded with mixed *k*-base gamma/*k*-flat binary code, the number of bits (b) consumed is,

1) If $x < 2^k$, $b = k$;
2) If x is a direct successor of a cluster, $b = 1 + 2\lfloor \log x \rfloor - k$;
3) If x is not a direct successor of a cluster, and $x \geq 2 \times 2^k$, $b = 1 + 2\lfloor \log x \rfloor - k$;
4) If x is not a direct successor of a cluster, $x \geq 2^k$, and $x < 2 \times 2^k$, $b = 1 + 2\lfloor \log x \rfloor$.

Proof: 1) If $x < 2^k$, it is inside a cluster, and encoded with *k*-flat binary code. Therefore $b = k$;

2) If x is a direct successor of a cluster, it is encoded with *k*-base gamma code, which includes two parts. The first part is gamma code of $x/2^k$, which needs $1 + 2\lfloor \log (x/2^k) \rfloor = 1 + 2\lfloor \log x \rfloor - 2k$ bits. And the second part is k bits binary code of the remainder of $x/2^k$. Therefore $b = 1 + 2\lfloor \log x \rfloor - 2k + k = 1 + 2\lfloor \log x \rfloor - k$;
3) If x is not a direct successor of a cluster, and $x \geq 2 \times 2^k$, x is encoded with *k*-base gamma coding. Similar to 2), we have $b = 1 + 2\lfloor \log x \rfloor - k$;
4) if x is not a direct successor of a cluster, $x \geq 2^k$, and $x < 2 \times 2^k$, its code is led with a special leading sequence, which needs $k + 1$ bits, and followed by k bits binary representation of $x - 2^k$. Therefore $b = 1 + 2k$. Since $x \geq 2^k$, and $x < 2 \times 2^k$, we have $k = \lfloor \log x \rfloor$. Therefore $b = 1 + 2\lfloor \log x \rfloor$.

Theorem 3 The additional bits (b) other than coding *d*-gaps for a cluster encoded with mixed *k*-base gamma/*k*-flat binary code is, 1) If the cluster has no successor, $b = 1$; 2) If the cluster has a successor, $b = 1 + k$.

Proof: 1) If the cluster has no successor, it is encoded with a starting bit '0', and no ending bits, therefore $b = 1$.
2) If the cluster has a successor, it is encoded with a starting bit '0', and k '1' ending bits, thus $b = 1 + k$.

Using gamma code, an integer x requires $1 + 2\lfloor \log x \rfloor$ bits. Based on the above theorems, comparing to gamma code, the bits that mixed gamma code saves (bs) for each *d*-gap x is:

1) If $x < 2^k$, $bs = 1 + 2\lfloor \log x \rfloor - k$;
2) If x is a direct successor of a cluster, $bs = 1 + 2\lfloor \log x \rfloor - (1 + 2\lfloor \log x \rfloor - k) = k$;
3) If x is not a direct successor of a cluster, and $x \geq 2 \times 2^k$, $bs = 1 + 2\lfloor \log x \rfloor - (1 + 2\lfloor \log x \rfloor - k) = k$;
4) If x is not a direct successor of a cluster, $x \geq 2^k$, and $x < 2 \times 2^k$, $bs = 1 + 2\lfloor \log x \rfloor - (1 + 2\lfloor \log x \rfloor) = 0$.

This analysis does not count the additional cost of cluster starting and ending bits, which may be 1 or $1 + k$ as illustrated in Theorem 3. Based on this analysis, for a mixed gamma code, the larger the average *d*-gap is, the more possible we save more bits per pointer in average comparing to gamma code.

4.2 Mixed Delta/Flat Binary Code

Definition 7 For a *d*-gap x, $x \geq 2^k$, if x is coded as delta code of $x/2^k$, followed by k bits binary representation of the remainder of $x/2^k$, we call this coding scheme ***k*-base delta Code**.

Algorithm 3 For an inverted list, a **Mixed *k*-base delta/*k*-flat Binary Code** (for simplicity, we will call it *mixed delta code*) is defined as follows,

Input: an inverted list; **Output:** mixed *k*-base delta/*k*-flat binary code;
Algorithm: 1) Rewriting the inverted list in its *k*-base clustered representation;
2) Starting from the beginning of the clustered representation, each item is encoded sequentially as follows,
   2.1) If the current item is a cluster, it is encoded with a starting bit '0', and *k* '1' bits as ending bits if there is a non-cluster *d*-gap following the cluster. Each *d*-gap inside the cluster is encoded using *k*-flat binary code sequentially following the starting bit;
   2.2) If the current item is a non-cluster *d*-gap that follows a cluster, it is encoded using *k*-base delta coding;
   2.3) If the current item is a non-cluster *d*-gap *x* that does not follow a cluster, and if $x \geq 2 \times 2^k$, it is encoded using *k*-base delta coding; if $x < 2 \times 2^k$, it is encoded as follows: a leading bit '0', followed by *k* '1' bits, followed by *k* bits binary code of $x - 2^k$.

Algorithm 4 For an inverted list coded with mixed *k*-base delta/*k*-flat binary code, a **Mixed *k*-base** delta/***k*-flat Binary Decoding** Algorithm is defined as follows,

Input: mixed *k*-base delta/*k*-flat binary code of an inverted list;
Output: the original inverted list;

Algorithm: Starting from the beginning of the code,
1) If a leading bit is '0', and the following *k* bits are not all '1's, then it means the bits following the starting bit '0' are for a cluster, and we decode the cluster based on coding scheme defined by *k*-flat binary code, *i*.e., following the starting bit '0', each time we take *k* bits and the corresponding *d*-gap is 1+ binary value of the *k* bits. This process is continued until *k* all '1' bits are read, or the end of the inverted list is met;
   1.1) If *k* bits of all '1's at the end of a cluster are read, it means that the following *d*-gap *x* is a non-cluster *d*-gap. We then decode the bits following the ending bits of the cluster using delta decoding (assume the decoded value is *y*), and further decode the following *k* bits after the delta code using binary decoding (assume the decoded value is *z*). Then we have $x = 2^k \times y + z$;
2) If a leading bit is '0', and the following *k* bits are all '1's, then it means the *k* bits following all '1's are for a non-cluster *d*-gap *x*, $x < 2 \times 2^k$, and $x \geq 2^k$. We decode *x* as $x = 2^k$+ binary value of the *k* bits following the *k* '1' bits.
3) If a leading is bit '1', it means the following bits together with the leading bit '1' are for a non-cluster *d*-gap *x*, $x \geq 2 \times 2^k$.

We first decode the bits starting from the leading bit '1' using delta decoding (assume the decoded value is *y*), and further decode the following *k* bits after the delta code using binary decoding (assume the decoded value is *z*). And we have $x = 2^k \times y + z$.

Theorem 4 There is no ambiguity to decode a mixed *k*-base delta/*k*-flat binary code using mixed *k*-base delta/*k*-flat binary decoding algorithm, and the decoded result is the same as the original inverted list.

We skip the proof of Theorem 4 as it is similar to that of Theorem 1.

Example 2 For the inverted list shown in Example 1, <12; 38, 17, 13, 34, 6, 4, 1, 3, 1, 2, 3, 1>,
1) The mixed 2-base delta/2-flat binary code is 11000 001 10 (38) 101 00 01 (17) 100 1 01 (13) 11000 000 10 (34) 011 10 (6) 011 00 (4) 0 00 (1) 10 (3) 00 (1) 01 (2) 10 (3) 00 (1), which has 56 bits. Comparing to delta code (62 bits), we save 6 bits. Comparing to interpolative code (55 bits), we use one bit more.
2) The mixed 3-base delta/3-flat binary code is 10100 110 (38) 1000 001 (17) 0111 101 (13) 10100 010 (34) 0 101 (6) 011 (4) 000 (1) 010 (3) 000 (1) 001 (2) 010 (3) 000 (1), which has 55 bits. Comparing to delta code, we save 7 bits. Comparing to interpolative code, we have the same performance.

Similar to Example 1, we can find that different bases (*k*) have different effects for mixed delta code. Table 1 summarizes the performance of our mixed codes as well as other codes for the inverted list from [1].

Below are two theorems describing the number of bits consumed by mixed delta code.

Theorem 5 For a *d*-gap *x* encoded with mixed *k*-base delta/*k*-flat binary code, the number of bits (*b*) consumed is,
1) If $x < 2^k$, $b = k$;
2) If *x* is a direct successor of a cluster, $b = 1 + 2\lfloor \log (\log (2x) - k) \rfloor + \lfloor \log x \rfloor$;
3) If *x* is not a direct successor of a cluster, and $x \geq 2 \times t$; 2^k, $b = 1 + 2\lfloor \log (\log (2x) - k) \rfloor + \lfloor \log x \rfloor$;
4) If *x* is not a direct successor of a cluster, $x \geq 2^k$, and $x < 2 \times t$; 2^k, $b = 1 + 2\lfloor \log x \rfloor$;

Proof: 1) If $x < 2^k$, it is inside a cluster, and encoded with *k*-flat binary code. Therefore $b = k$;
2) If *x* is a direct successor of a cluster, it is encoded with *k*-base delta code, which includes two parts. The first part is delta code of $x/2^k$, which needs $1 + 2\lfloor \log \log (2x/2^k) \rfloor + \lfloor \log (x/2^k) \rfloor = 1 + 2\lfloor \log (\log (2x) - k) \rfloor + \lfloor \log x \rfloor - k$ bits. And the second part is *k* bits binary representation of the remainder of $x/2^k$. Therefore $b = 1 + 2\lfloor \log (\log (2x) - k) \rfloor + \lfloor \log x \rfloor - k + k = 1 + 2\lfloor \log (\log (2x) - k) \rfloor + \lfloor \log x \rfloor$;
3) If *x* is not a direct successor of a cluster, and $x \geq 2 \times t$; 2^k, *x* is encoded with *k*-base delta code. Similar to the proof of 2), we have $b = 1 + 2\lfloor \log (\log (2x) - k) \rfloor + \lfloor \log x \rfloor$;
4) if *x* is not a direct successor of a cluster, $x \geq 2^k$, and $x < 2 \times t$; 2^k, its code is led with a special leading sequence, which needs $k + 1$ bits, and followed by *k* bits binary representation of $x - 2^k$. Therefore $b = 1 + 2k$. Since $x \geq 2^k$, and $x < 2 \times t$; 2^k, we have $k = \lfloor \log x \rfloor$. Therefore $b = 1 + 2\lfloor \log x \rfloor$.

Theorem 6 The additional cost (*b*) other than coding *d*-gaps for a cluster encoded with mixed *k*-base delta/*k*-flat binary code is, 1) If the cluster has no successor, $b = 1$; 2) If the cluster has successor, $b = 1 + k$.

Method	Gamma	Delta	Golomb (b=3)	Interpolative	Fixed binary	Mixed Gamma (*k*=2)	Mixed Gamma (*k*=3)	Mixed Delta (*k*=2)	Mixed Delta (*k*=3)
Code length (bits)	60	62	64	55	57	53	54	56	55

Table 1. Comparison of different codes for the example inverted list

We skip the proof of Theorem 6 as it is similar to that of Theorem 3.

Using delta coding, an integer x requires $1 + 2\lfloor \log \log 2x \rfloor + \lfloor \log x \rfloor$ bits. Based on the above theorems, comparing to normal delta code, the bits that mixed delta code saves (bs) for each d-gap is:

1) If $x < 2^k$, $bs = 1 + 2\lfloor \log \log 2x \rfloor + \lfloor \log x \rfloor - k$ (this may be a net loss since $1+2\lfloor \log \log 2x \rfloor + \lfloor \log x \rfloor$ may be less than k);
2) If x is a direct successor of a cluster, $bs = 1 + 2\lfloor \log \log 2x \rfloor + \lfloor \log x \rfloor - (1 + 2\lfloor \log (\log (2x) - k) \rfloor + \lfloor \log x \rfloor) = 2\lfloor \log \log 2x \rfloor - 2\lfloor \log (\log (2x) - k) \rfloor$;
3) If x is not a direct successor of a cluster, and $x \geq 2 \times t$; 2^k, similar to 2), $bs = 2\lfloor \log \log 2x \rfloor - 2\lfloor \log (\log (2x) - k) \rfloor$;
4) If x is not a direct successor of a cluster, $x \geq 2^k$, and $x < 2 \times t$; 2^k, $bs = 1 + 2\lfloor \log \log 2x \rfloor + \lfloor \log x \rfloor - (1 + 2\lfloor \log x \rfloor) = 2\lfloor \log \log 2x \rfloor - \lfloor \log x \rfloor$.

The above analysis does not count the additional cost of cluster starting and ending bits, which may be 1 or $1 + k$ as illustrated in Theorem 6. Comparing to the gain of mixed gamma code from gamma code, generally speaking, the gain of mixed delta code from delta code is not that large when the average d-gap is large. This is due to the fact that delta code itself compresses more than gamma code when d-gap is large. Our experiments in next section confirm this analysis.

5. Experiments

TREC-8 web track data WT2g [7] is used as testing data. The collection contains 250,000 documents, 2 Gigabytes data, 1,002,779 distinct terms, and 76,147,063 pointers. We ran our tests on a Dell Inspiron 8600 laptop with 1.80 GHz Pentium M CPU, 1GB memory, and Ultra-ATA 80GB hard disk. The operating system is Windows XP.

We first built inverted file index using gamma code, and then built new indices using mixed codes based on existing index. In our system the time to build the inverted file index using gamma code for WT2g data set is around 40 minutes, and the additional time to build new indices using mixed gamma or mixed delta code is around 100 seconds, which is only about 4% additional cost. Generally speaking, the time complexity of compressing inverted file index using gamma code or delta code is much lower comparing to that of interpolative code [16]. Based on our experiment results, mixed gamma (delta) code has similar time complexity as gamma (delta) code, therefore it also has much lower time complexity comparing to interpolative code.

5.1 Compression Ratio Comparison of Different Codes

Table 2 shows the compression ratios in terms of *bits per pointer* (which is used for all the rest experiments) of our mixed gamma and mixed delta codes as well as other codes.

The first four methods are various base-line methods for comparison purpose. Generally speaking, Interpolative code is the most efficient one of these four traditional bitwise approaches, and delta code is more efficient than gamma code for large data collections. These are verified in Table 2. One interesting exception is Golomb code. Williams and Zobel [15] suggests that Golomb code is more space-efficient than gamma and delta codes. However, in our experiment Golomb code has the worst performance. The reason is that Golomb code does not adapt well to the apparent cluster property of WT2g data set, in which many related documents are closely located.

Compared to Interpolative code, both mixed gamma and mixed delta codes have better or equal compression ratio, and a much lower encoding and decoding complexity. Compared to their counterpart (mixed gamma vs. gamma, and mixed delta vs. delta), both mixed gamma and mixed delta have promising improvements, and the additional costs for encoding and decoding are almost neglectable (4%) at run time.

5.2 Comparison of Mixed Gamma and Gamma Codes

Based on the analysis in Section 4, for an inverted list with small average d-gap, mixed gamma code may not always outperform gamma code. However, when the average d-gap is large enough, it will almost always outperform gamma code. The larger the average d-gap is, the better mixed gamma code performs. This trend is verified in Table 3 when average d-gap increases from 10 to more than 50,000. Based on the analysis in section 4, the limit of improvement of mixed gamma code to gamma code should be k bits per pointer. This is also verified in Table 3.

Ave. d-gap	Gamma	Mixed gamma	Improvement of mixed gamma
10-11	4.33	4.25	0.08
100-110	8.05	7.29	0.76
1000-1100	11.7	10.69	1.01
10000-11000	15.97	14.77	1.2
50000-130000	23.11	21.4	1.71

Table 3. Mixed gamma (k=2) and gamma code for inverted lists with different average d-gaps

5.3 The impact of k to Mixed Gamma Code

Table 4 shows testing results of four settings of different k values for mixed gamma code. For setting 1, all inverted lists are encoded with mixed 2-base gamma/2-flat binary code. For setting 2, inverted lists with average d-gap ≤ 128 are encoded with 2-base mixed gamma/flat binary code, $129 \leq d$-gap ≤ 256 with 3-base, $257 \leq d$-gap ≤ 512 with 4-base, and d-gap > 512 with 5-base. Similarly we have setting 3 and 4. Table 4 shows that further improvement can be achieved if larger ks are used for inverted lists with larger average d-gaps. This conforms to our analysis in Section 4.

5.4 Comparison of Mixed Delta and Delta Codes

Similar to mixed gamma code, mixed delta code may not always outperform delta code. However, when the average d-gap is large enough, it will almost always outperform delta code. Generally speaking, the larger the average d-gap is, the better mixed delta code performs. This trend is verified in

Method	Gamma	Delta	Golomb	Interpolative	Mixed Gamma (k=2)	Mixed Delta (k=2)
Compression ratio	6.21	5.91	6.24	5.83	5.83	5.70

Table 2. Compression ratio (bits per pointer) of various codes

k	Ave. *d*-gap S1	Ave.*d*-gap S2	Ave. *d*-gap S3	Ave. *d*-gap S4
2	All	≤128	≤128	≤128
3		129-256	129-256	129-256
4		257-512	257-512	257-512
5		>512	513-1024	513-1024
6			>1024	1025-2048
7				>2048
Comp. ratio	**5.83**	**5.664**	**5.662**	**5.661**

Table 4. The impact of *k* to mixed gamma code

Table 5 when average *d*-gap increases from 10 to more than 50,000. From Table 5, we can further find that mixed delta code does not save that much from delta code as mixed gamma code does from gamma code. This conforms to our analysis in section 4.

Ave. d-gap	Delta	Mixed Delta	Improvement of mixed Delta
10-11	4.57	4.4	0.17
100-110	7.63	7.24	0.39
1000-1100	10.15	9.89	0.26
10000-11000	12.95	12.64	0.31
50000-130000	18.05	17.35	0.7

Table 5. Mixed delta (*k*=2) and delta code for inverted lists with different average d-gaps

5.5 The impact of *k* to Mixed Delta Code

Table 6 shows testing results of four settings of different *k*s for mixed delta code. In this table the settings of *k* for different *d*-gaps are similar to those in Table 4. Table 6 shows that for mixed delta code the best compression ratio is achieved when *k* is set to 2 for all inverted lists (setting 1).

6. Conclusions

By clustering *d*-gaps of an inverted list strictly based on a threshold, and then encoding clustered and non-clustered *d*-gaps using different methods, we can tailor to the specific properties of different *d*-gaps and achieve better compression ratio. In this paper we have verified this idea and presented two new codes for index compression: mixed gamma code and mixed delta code. Experiment results show that the two new codes achieve better or equal performance in terms of compression ratio comparing to interpolative code which is considered as the most efficient bitwise code at present. Besides, the two new codes have much lower complexity comparing to interpolative code and therefore enable faster encoding and decoding. By adjusting the parameters for the mixed codes, even better results may be achieved.

References

[1] Anh V. N.,Moffat, A (2004). Index compression using fixed binary codewords. *In*: *Proc. 15th Australasian Database Conference* (Dunedin, New Zealand, Jan. 2004. 61-67.

[2] Anh V. N., Moffat A. Inverted Index Compression Using Word-Aligned Binary Codes, *Information Retrieval, 8*, 1 (Jan. 2005), 151-166.

[3] Blandford, D., Blelloch, G (2002). Index compression

k	Ave. *d*-gap S1	Ave. *d*-gap S2	Ave. *d*-gap S3	Ave. *d*-gap S4
2	All	≤128	≤128	≤128
3		129-256	129-256	129-256
4		257-512	257-512	257-512
5		>512	513-1024	513-1024
6			>1024	1025-2048
7				>2048
Comp. ratio	**5.70**	**5.75**	**5.77**	**5.78**

Table 6. The impact of *k* to mixed delta code

through document reordering. *In*: *Proceedings of the Data Compression Conference* (Snowbird, UT, USA, 342-351.

[4] Bookstein A., Klein S., Raita, T (1995). Modeling word occurrences for the compression of concordances. *In*: *Proc. IEEE Data Compression Conference.* Snowbird, UT, USA, March 29-30, 462-462

[5] Elias, P (1975). Universal codeword sets and representation of the integers. *IEEE Transactions on Information Theory, 21*, 2 (Feb. 1975), 194-203.

[6] Golomb, S (1966). Run-length encodings, *IEEE Trans. on Information Theory, 12*, 399-401.

[7] Hawking, D., Voorhees E., Craswell N., Bailey, P (2000). Overview of the TREC-8 Web Track. *In*: *Proc. of the Eighth Text REtrieval Conference,* NIST Special Publication 500-246, 2000. 131-150.

[8] Moffat, A., Stuiver. L (1996). Exploiting clustering in inverted file compression. *In*: *Proceedings of the 1996 IEEE Data Compression Conference,* Snowbird, Utah, March 31 - April 3. 82-91.

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

[10] Rillof, E., Hollaar, L (1966). Text database and information retrieval. *ACM Comput. Surveys. 28*, 1 (1996), 133-135.

[11] Scholer F., Williams H. E., Yiannis H., Zobel. J (2002). Compression of inverted index for fast query evaluation. *In: ACM SIGIR02* (Tampere, Finland, Aug. 2002). 222-229.

[12] 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 (Jan. 2003), 117–131.

[13] Silvestri, F., Perego R., Orlando, S (2004). Assigning document identifiers to enhance compressibility of web search engine indexes. In *Proceedings of the 19th Annual ACM Symposium on Applied Computing - Data Mining Track,* Nicosia, Cyprus. 600-605.

[14] Trotman, A (2003). Compressing inverted files. *Information Retrieval, 6* (2003), 5-19.

[15] Williams H. E., Zobel, J (1999). Compressing integers for fast file access. The Computer Journal, 42, 3, 193-201.

[16] Witten I. H., Moffat A., Bell T. C (1999). Managing Gigabytes – Compressing and Indexing Documents and Images. Morgan Kaufmann Publishing, San Francisco, second edition, 1999.

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