Using d-gap Patterns for Index Compression

Jinlin Chen
Computer Science Dept.
Queens College, CUNY
Flushing, NY, 11367, USA
Tel: 001-718-997-3497
jchen@cs.qc.edu

Terry Cook
Computer Science Dept.
Graduate Center, CUNY
New York, NY, 10016, USA
Terrycookd1@aol.com

ABSTRACT
Sequential patterns of d-gaps exist pervasively in inverted lists of Web document collection indices due to the cluster property. In this paper the information of d-gap sequential patterns is used as a new dimension for improving inverted index compression. We first detect d-gap sequential patterns using a novel data structure, UpDown Tree. Based on the detected patterns, we further substitute each pattern with its pattern Id in the inverted lists that contain it. The resulted inverted lists are then coded with an existing coding scheme. Experiments show that this approach can effectively improve the compression ratio of existing codes.

Categories and Subject Descriptors
E.4 [Data]: Coding and Information Theory – Data Compaction and Compression; H.3.1 [Information Storage and Retrieval]: Content Analysis and Indexing – Indexing methods;

General Terms: Algorithms, Performance, Experimentation, Theory.

Keywords: Inverted file, d-gap, Index compression, sequential pattern

1. INTRODUCTION
Efficient indexing of Web pages is crucial for the performance of search engines. Inverted file is the most popular indexing technique for today's Web search engines due to its relative small size and high efficiency for keyword–based queries. An inverted file index maps each term to an inverted list of all the documents containing the term. For a term t, the inverted list has the structure $<d_1, d_2, d_3, \ldots, d_{ft}>$, where f_t is the number of documents containing t, d_i is a document Id that identifies the document associated with the i^{th} occurrence of t, and $d_i < d_{i+1}$. The list can be stored as an initial position followed by a list of d-gaps by taking consecutive differences, $d_{i+1} - d_i$, to save memory. In this paper we assume that all inverted lists are in d-gap format.

Many codes have been proposed for compressing inverted lists. The performance of a code is mainly decided by whether the implicit d-gap distribution model of the code conforms to that of the document collection. A phenomenal feature that influences the d-gap distributions of many Web document collections is the cluster property [1], 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. A major impact of cluster property is that many frequent patterns of d-gap sequences may exist in the inverted lists. The motivation of this paper is to make use of d-gap sequential patterns to improve index compression.

For example, given the following inverted lists, (a) <1 3 4 5 5 2 6 1 1 7 4 5 3 2 9 5 5 2 6>; (b) <7 4 5 3 2 9 3 4 1 5 5 2 6 1>; (c) <1 7 7 4 5 3 2 9 3 6>, we can find that d-gap sequences <5 5 2 6> and <7 4 5 3 2 9> both occur 3 times in the lists. If we can represent them as a pattern (using a pattern Id) in the inverted lists that contain them, we may store the lists with fewer bits.

Here one issue is how to detect d-gap sequential patterns (DSP). In this paper we apply a top-down approach to detect DSPs using a special data structure, UpDown Tree. Each detected DSP is assigned a pattern Id. To improve index compression, we substitute each DSP with its pattern Id in the inverted lists that contain it. Experiment results show that this approach can effectively improve the compression ratio of existing codes.

2. DSP BASED INDEX COMPRESSION
2.1 DSP Detection
Taking each inverted list as a sequence and each d-gap as an item, detecting d-gap sequential patterns becomes a sequential pattern (SP) mining problem under four constraints. First, each element in a sequence consists of only one d-gap. Second, d-gaps appearing in the sequences that contain a pattern must be adjacent with respect to the underlying order as defined in the pattern. Third, if a pattern appears k times in a sequence, its support is added k instead of 1. The reason is that if a pattern occurs multiple times in the same inverted list, each of the occurrences can be compressed by using a pattern Id separately. Fourth, if two patterns are overlapped in the same sequence, only one pattern's support is counted (By default we count the one examined first). The reason is that during compression we can only make use of one of the overlapped patterns. For example, <5 2 6> occurs three times in the lists. However, each of its occurrences is overlapped with <5 5 2 6> and thus not counted. We call SP mining under such constraints d-gap sequential pattern Mining. Many approaches have been proposed for SP mining under constraints. However, these approaches are not suitable for DSP detection due to the large searching space and inefficient data structures.

Instead of using traditional approaches, we use a special data structure, UpDown Tree, to detect DSPs. Below we give a brief introduction to UpDown Tree. For a detailed description of UpDown Tree please refer to our other paper in this proceeding [2]. An UpDown Tree is a compact data structure to efficiently store all the sequences that contain a given item k. For a sequence

WWW 2007, May 8–12, 2007, Banff, Alberta, Canada.
ACM 978-1-59593-654-7/07/0005.

S that contains k, if k the m^{th} item, we define the full suffix of k in S as the subsequence of S from k to the last item of S, and the full prefix of k in S as the subsequence of S from the first item to the m^{th} item (k). We use a Down Tree to represent all the full suffixes of k, and an Up Tree to represent all the full prefixes of k. We further derive an UpDown Tree by merging the root nodes of Up and Down Tree. Such a combination ensures that any DSP pattern containing k corresponds to a path from the Up Tree to the Down Tree. Using a depth first order to check each node j in the Up Tree for all the DSPs that contain k and starting from j, we implement a top down approach for DSP mining which is more efficient than traditional bottom up approaches because it eliminates unnecessary candidate checking. Each detected DSP is assigned a pattern Id for index compression as described in Section 2.2.

2.2 Index Compression

DSP information can be combined with existing codes to improve inverted index compression. A popular code for inverted index is Gamma code [3], which represents an integer x by $1 + \lfloor \log x \rfloor$ stored as a unary code, followed by $\lfloor \log x \rfloor$ bits binary code of x without its most significant bit. Below we combine DSP information with Gamma code to demonstrate the usage of DSP for index compression.

Definition 1 For an inverted list $<d_1, d_2, d_3, \ldots, d_{ft}>$, based on the DSPs detected in Section 2.1, we can re-write it as $<g_1, g_2, \ldots, g_k>$, where g_i is either a d-gap or the pattern Id of a DSP, and k is the total number of d-gaps and DSPs. We call this new list the **Clustered Representation** of the original list.

To encode an inverted list in its clustered representation, we need clearly identify whether g_i is a d-gap or a DSP. The algorithm below realizes this by encoding DSP information (total numbers, positions) at the beginning of the compressed inverted list using Gamma code, and encoding each DSP pattern Id in the inverted list using Huffman code which conforms to the distribution model of pattern Ids. Here we do not give the details of Huffman code as it is a mature coding scheme.

Algorithm 1 (**DSP based Gamma Code**).

Input: an inverted list in its clustered representation;

Output: DSP based Gamma code for the list;

Method: 1) Encode the total number of DSPs in the list using 0-origin Gamma code;

2) Encode the position of each DSP in the clustered representation of the list sequentially using Gamma code. Here the position of a DSP is the total number of d-gaps in between the DSP and the preceding DSP (1 if the DSP is at the beginning of the list);

3) Encode each item in the list sequentially. If the item is a d-gap, it is encoded as Gamma code. If the item is a DSP, its pattern Id is encoded in Huffman code.

Example 1 For the example list (a) in section 1, the effective DSPs are <5, 5, 2, 6> and <7, 4, 5, 3, 2, 9>. The supports of both DSPs are 3, which are coded as 0 and 1 in Huffman code, respectively. The DSP based Gamma code of list (a) is, 11000 (total number of DSPs) 11000 (position of the 1st DSP) 101 (position of the 2nd DSP) 0 (position of the 3rd DSP) 0 (1) 101 (3) 11000 (4) 0 (DSP) 0 (1) 0 (1) 1 (DSP) 0 (DSP), which has 28 bits. Comparing to Gamma code which needs 75 bits, we save 47 bits. However, this comes at the cost of saving the Huffman code information of all the DSPs, which can be leveraged if the supports of DSPs are high.

3. EXPERIMENT RESULTS

The testing data set is TREC-8 web track data WT2g [4], which contains 28 sub-tracks (WT01-28), 250,000 documents. To study the performance of our codes over document collections with different size, we perform our experiments over WT01-05, WT01-10, WT01-15, WT01-20, WT01-25, and WT01-28, respectively. Based on practical experience, we use 10 as minimum support and minimum length to decide whether a DSP should be used for compression. Studying of the optimal values of these parameters will be performed in the future.

Fig. 1 compares the performances of DSP based Gamma code and Gamma code in terms of compression ratios (*bits per pointer*) over different collection sizes. Fig. 1 shows that DSP based Gamma code performs similar to Gamma code on small collections and better on large collections. The larger the collection is, the better it performs than Gamma code. The reason is that as the collection becomes larger, the support of many DSPs increase accordingly, which leverage the cost of saving the Huffman code information for the DSPs. Since the number of today's Web pages is continuously increasing, this indicates a promising future for DSP based Gamma code.

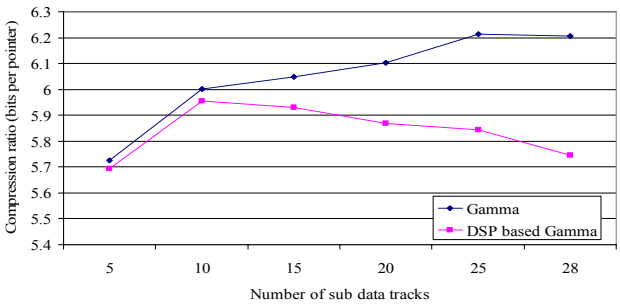

Figure 1. DSP based Gamma vs. Gamma codes.

4. CONCLUSIONS

In this paper we use the information of frequent patterns of d-gap sequences as a new dimension for improving inverted index compression. First we use a special data structure, UpDown Tree, to implement an efficient top-down approach for DSP mining. We then substitute each DSP with its pattern Id in the inverted lists that contain it. Experiment results show that this approach can effectively improve the compression ratio of existing codes.

5. REFERENCES

[1] Anh V. *N*. and Moffat A. Index compression using fixed binary codewords. in Proc. 15th Australasian Database Conference (2004). 61-67.

[2] Chen J., and Cook T. Mining Contiguous Sequential Patterns from Web Logs. in Proc. of the WWW2007. (Canada, 2007).

[3] Elias P. Universal codeword sets and representation of the integers. IEEE Trans. on Info. Theory, *21*, 2 (1975), 194–203.

[4] Hawking D., Voorhees E., Craswell N., and Bailey P. Overview of the TREC-8 Web Track. in Proc. of the 8th Text REtrieval Conf. (2000). 131-150.