INTERNATIONAL JOURNAL OF COMPUTERS COMMUNICATIONS & CONTROL
ISSN 1841-9836, 11(6):776-788, December 2016.

Lossless Compression of Data Tables in Mobile Devices using Co-clustering

B. Han, B. Li

Bo Han*, Bolang Li
International School of Software, Wuhan University
37 Luoyu Road, Wuhan, Hubei, China, 430079
*Corresponding author: bhan@whu.edu.cn

Abstract: Data tables have been widely used for storage of a collection of related records in a structured format in many mobile applications. The lossless compression of data tables not only brings benefits for storage, but also reduces network transmission latencies and energy costs in batteries. In this paper, we propose a novel lossless compression approach by combining co-clustering and information coding theory. It reorders table columns and rows simultaneously for shaping homogeneous blocks and further optimizes alignment within a block to expose redundancy, such that standard lossless encoders can significantly improve compression ratios. We tested the approach on a synthetic dataset and ten UCI real-life datasets by using a standard compressor 7Z. The extensive experimental results suggest that compared with the direct table compression without co-clustering and within-block alignment, our approach can boost compression rates at least 21% and up to 68%. The results also show that the compression time cost of the co-clustering approach is linearly proportional to a data table size. In addition, since the inverse transform of co-clustering is just exchange of rows and columns according to recorded indexes, the decompression procedure runs very fast and the decompression time cost is similar to the counterpart without using co-clustering. Thereby, our approach is suitable for lossless compression of data tables in mobile devices with constrained resources.
Keywords: data tables, lossless compression, co-clustering, redundancy.

1 Introduction

Internet of things and mobile Internet churn out huge volume of real-time data in mobile computing devices, such as smartphones, tablets and e-readers. By using ubiquitous mobile sensors, these digital devices constantly collect environment information or persons' behavior information. The information is generally recorded in a structured data table. For example, in a table of a healthcare application, rows represent the records in a sequential sampling time series and columns show the values of multiple measured parameters, such as heart rates. By cumulating records with the time, multi-columns mobile data tables can grow sharply in size, varying from few hundred kilobytes to few hundred megabytes dependent upon different applications. In the Internet context and big data era, these sensor data will generally be sent to cloud servers for further processing in a time frequency.

For a mobile device, a larger size of a data table not only takes more storage space, but its transportation to cloud servers takes more network transmission latencies and more energy costs in batteries. Therefore, lossless compression of data tables is critical to optimize the usage of system resources and thus play an important role in optimizing system performance.

Traditional lossless compression methods view a dataset as a large sequence of strings and employ the occurrence frequency of duplicated symbols to compress them in variable length codes in a dictionary, such as Huffman, Lempel-Ziv, run length encoding and other compression encoders [1]. These methods do not account for dependency patterns in a table and therefore they cannot compress a table with the size less than the limit given by the Shannon entropy [2–4].

Recently, some researchers studied data dependency and exploited the resulted redundancy for compression. Mielikainen et al. applied spatial dependency to propose an adaptive prediction length in a clustered differential pulse code modulation method for lossless compression of hyperspectral data [5]. Venugopal et al. applied Hadamard transformation to eliminate the correlation inside local blocks in medical data [6]. Patauner et al. combined vector quantization, delta calculation and a Huffman coding algorithm together to reduce the correlation among a sequence of data records acquired from pulse digitizing electronics and then apply lossless compression [7]. Kolo et al. proposed an adaptive lossless data compression algorithm for wireless sensor networks [8]. The network data sequence is partitioned into consecutive blocks, and the optimal compression scheme is applied for each block by dynamically analyzing data dependency in a block. They further improved their approach and proposed a fast lossless adaptive compression scheme with low memory requirements for wireless sensor networks [9]. The approach can generate its coding tables on the fly and compress data blocks very fast. Buchsbaum et al. proposed a novel dynamic programming algorithm to discover column dependencies in a table by a one-time, offline learning procedure from a small number of training examples [10]. By exploiting the dependencies, they contiguously partitioned table columns into disjoint groups and found that compressing each group of columns separately can significantly improve compression rates. In their further research, Buchsbaum et al. applied a TSP (traveling salesman problem) tour method to reorder table columns prior to partitioning for further improving compression rates [11]. It provided a unified theory of entropy-like functions to explain both contiguous partitioning and column rearrangement. However, the time cost of the algorithm is expensive and is not linearly proportional to a data table size. Yang et al. studied a transform compression approach for a Boolean matrix [12]. It firstly located the largest columnwise-constant submatrix, and next rearranged columns such that the columnwise-constant is moved to the left-upper corner of the matrix. Following the procedure, the approach recursively applied transformation on the rest of the matrix until the partition resulted in a matrix smaller than a user-defined threshold. However, its running-time cost is also very expensive and the choice of the user-defined threshold is a tricky problem and some improper choices can negatively affect the compressibility.

In a word, the above methods exploited data dependencies and correlations among columns in a table (or called a matrix in mathematics). However, rows in a table also can show dependencies and redundancies. A distinct characteristic of tables is that reordering of columns and rows will not lead to information loss. Therefore, we aim to apply an approach to group dependent or similar columns and rows together, such that the redundancy in grouped homogeneous blocks can be exposed and then we apply compression on these blocks.

Co-clustering is an approach simultaneously clustering of similar rows and columns for revealing hidden structures of a data matrix [13, 14]. It provides great potential for compression. Firstly, co-clustering reorders and groups rows and columns into similar or homogeneous rectangular regions, such that redundant information gets exposed and can be removed by a statistical encoder (such as a Huffman compressor). Next, the inverse transform of co-clustering is very simple and fast. This property will be desirable for uncompressing data. In addition, the time complexity of the algorithm is $O((m+n)kl)$ [15], where m is the number of rows, n is the number of columns, k is the number of row-clusters and l is the number of column-clusters. Since k and l are much smaller than m and n, the algorithm can run in linear time with the size of a data table. In another word, the algorithm is scalable to big size of data tables. Co-clustering has been widely studied for information clustering, pattern structure discovery, et al [16–19].

In this paper, we propose a novel lossless compression approach for data tables. Unlike traditional compression methods by exploiting data dependencies in a single view of either columns or rows, the proposed approach can shape homogeneous rectangular blocks by reordering columns and rows simultaneously via co-clustering. To the best of our knowledge, the proposed approach is the first algorithm that integrates co-clustering and information coding theory for the purpose of lossless compression of data tables. It not only constructs homogeneous blocks by co-clustering, but also optimizes columns/rows alignment in a block to further expose block redundancy, such that the downstream compression with a standard lossless encoder becomes more efficient. We compare empirically the performance of a standard compressor 7Z before and after the application of co-clustering and within-block alignment on a synthetic dataset and several public datasets with properties similar to tables in mobile systems, such as information collected from wearable sensors, clinical care, customer reviews, et al. The extensive experimental results suggest that the proposed approach can effectively improve compression ratios for data tables. In addition, the decompression procedure runs very fast and thus it is very suitable for lossless compression of data tables in mobile devices with constrained resources.

2 Methodology

In this paper, we propose a novel data table compression approach consisting of three steps: 1. reorder table columns and rows by co-clustering; 2. refine table columns and rows alignment to further expose redundancy; 3. compress the resorted data table by a standard compressor.

2.1 Reorder table columns and rows by co-clustering

Given a data table T with m rows and n columns, co-clustering transforms T into another table T' with k row-clusters and l column-clusters (here k and l are smaller than m and n respectively), where each element $b_{ij}(i \in [1,k], j \in [1,l])$ in T' corresponds to a homogeneous two-dimensional block after reordering columns and rows. The adjacent elements in the block have the same or similar values and they provide potentials to boost compression rates. The co-clustering transform can be illustrated in Figure 1 as below,

$$\begin{array}{ccc} T & \longrightarrow & T' \\ \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} & \rightarrow & \begin{bmatrix} b_{11} & \dots & b_{1l} \\ \dots & \dots & \dots \\ b_{k1} & \dots & b_{kl} \end{bmatrix} \end{array}$$

Figure 1: Co-clustering of a data table

In a statistical way, the original table T can be viewed as a joint probability distribution between two discrete random variables denoting rows and columns respectively. Let R and C be such two discrete random variables that take values in the set $\{r_1, \dots, r_m\}$ and $\{c_1, \dots, c_n\}$ respectively. Co-clustering aims to simultaneously quantize R into k disjoint clusters, and C into l disjoint clusters. In other words, co-clustering will generate mappings M_{row} and M_{col} as below,

$M_{row} : R = \{r_1, r_2, \dots, r_m\} \rightarrow \hat{R} = \{\hat{r_1}, \hat{r_2}, \dots, \hat{r_k}\}$
$M_{col} : C = \{c_1, c_2, \dots, c_n\} \rightarrow \hat{C} = \{\hat{c_1}, \hat{c_2}, \dots, \hat{c_l}\}$
The mappings also can be represented in functional forms:$\hat{R} = M_{row}(R)$ and $\hat{C} = M_{col}(C)$.

For obtaining homogeneous blocks in those clusters and further be applied compression, an optimal co-clustering minimizes the loss in mutual information, that is,

$$Minimize \quad I(R;C) - I(\hat{R};\hat{C}) \quad by \quad subject \quad to \quad the \quad given \quad k \quad and \quad l; \tag{1}$$

Here, the mutual information $I(R;C)$ measures the amount of information random variable R contains about C.

To facilitate the search for the optimal co-clustering, the above objective function can be expressed as the following "distance" of $p(R,C)$ to an approximation $q(R,C)$,

$$I(R;C) - I(\hat{R};\hat{C}) = KL(p(R,C)||q(R,C)) \tag{2}$$

Here, $KL(\cdot||\cdot)$ denotes the Kullback-Leibler divergence, $q(R,C)$ is a distribution of the form

$$q(r,c) = p(\hat{r},\hat{c})p(r|\hat{r})p(c|\hat{c}), where \quad r \in \hat{r}, c \in \hat{c} \tag{3}$$

By following Kullback-Leibler divergence, the following equations (4) and (5) can be derived,

$$\begin{aligned} &KL(p(R,C,\hat{R},\hat{C})||q(R,C,\hat{R},\hat{C})) \\ &= \sum\nolimits_{\hat{r}} \sum\nolimits_{r:M_{row}(r)=\hat{r}} p(r)KL(p(C|r)||q(C|\hat{r})) \end{aligned} \tag{4}$$

$$\begin{aligned} &KL(p(R,C,\hat{R},\hat{C})||q(R,C,\hat{R},\hat{C})) \\ &= \sum\nolimits_{\hat{c}} \sum\nolimits_{c:M_{col}(c)=\hat{c}} p(c)KL(p(R|c)||q(R|\hat{c})) \end{aligned} \tag{5}$$

Here, the distribution $q(C|\hat{r})$ can be defined as "row-clustering prototype". Similarly, the distribution $q(R|\hat{c})$ can be defined as "column-clustering prototype". In this way, the above equations show that the objective function in (1) can solely be expressed in terms of row-clustering or column-clustering.

With this intuition, the co-clustering algorithm is listed in Figure 2.

In the first step, co-clustering algorithm starts with an initial mapping function $M_{row}^{(0)}$ and $M_{col}^{(0)}$, and then computes the approximation distributions $q^{(0)}$, including the initial row-cluster prototype $q^{(0)}(C|\hat{r})$. For every column c, the row-cluster is computed as,

$$q^{(t)}(c|\hat{r}) = q^{(t)}(c|\hat{c})q^{(t)}(\hat{c}|\hat{r}) \quad Where, \hat{c} = M_{col}(c). \tag{6}$$

From line 6 to 16, the algorithm keeps iterative computing of row-clusters and column-clusters until a desired convergence condition is satisfied.

Specifically, from line 7 to 9, we re-assign each row r into a row-cluster by using the mapping function $M_{row}^{(t+1)}(r)$. In line 10, the algorithm recomputes the required marginal of $q^{(t+1)}$ based on the updated row-clustering results. It also recomputes the column-cluster prototype by (7). For every row r, we have,

$$q^{(t+1)}(r|\hat{c}) = q^{(t+1)}(r|\hat{r})q^{(t+1)}(\hat{r}|\hat{c}) \quad Where, \hat{r} = M_{col}(r). \tag{7}$$

From line 11 to 13, we re-assign each column c into a column-cluster by using the mapping function $M_{col}^{(t+2)}(c)$, while keeping the row-cluster fixed.

In line 14, the algorithm recomputes marginals of $q^{(t+2)}$ and the row-cluster prototype $q^{(t+2)}(C|\hat{r})$ by (6).

In line 15, we let the iterative variable $t = t + 2$.

The processes from line 6 to 16 are repeated and the row/column clusters are updated until the change of objective function is very small. Actually, the co-clustering algorithm monotonically decreases the objective function given in (2) [15,16] . It grantees an optimal co-clustering results will be achieved by the iterative procedure in the algorithm.

By co-clustering, similar columns/rows are grouped into one cluster. It facilitates coding and provides compression potentials by using a standard compressor.

```
Algorithm Co-clustering of a Data Table
Input: Table $T$ with a joint probability distribution $p(R, C)$
       The number of row-clusters $k$, The number of column-clusters $l$
Output: The cluster mapping functions $M_{row}$ and $M_{col}$
1: //Initialization:
2: Let iterative variable $t = 0$
3: Start with some initial mapping functions $M_{row}^{(0)}$ and $M_{col}^{(0)}$
4: Compute $q^{(0)}(R|\hat{R})$ , $q^{(0)}(C|\hat{C})$ , $q^{(0)}(\hat{R}, \hat{C})$ and $q^{(0)}(C|\hat{r})(1 \leq \hat{r} \leq k)$ using (6)
5: //Compute new row-cluster index and column-cluster index by iterative rounds:
6: repeat do
7:     for each row $r$ do
8:         $M_{row}^{(t+1)}(r) = argmin_{\hat{r}} KL(p(C|r)||q^{(t)}(C|\hat{r}))$, $M_{col}^{(t+1)} = M_{col}^{(t)}$
9:     end for
10:    Compute $q^{(t+1)}(R|\hat{R})$, $q^{(t+1)}(C|\hat{C})$, $q^{(t+1)}(\hat{R}, \hat{C})$ and $q^{(t+1)}(R|\hat{c})$ $(1 \leq \hat{c} \leq l)$ using (7)
11:    for each column $c$ do
12:        $M_{col}^{(t+2)}(c) = argmin_{\hat{c}} KL(p(R|c)||q^{(t+1)}(R|\hat{c}))$, $M_{row}^{(t+2)} = M_{row}^{(t+1)}$
13:    end for
14:    Compute $q^{(t+2)}(R|\hat{R})$, $q^{(t+2)}(C|\hat{C})$, $q^{(t+2)}(\hat{R}, \hat{C})$ and $q^{(t+2)}(C|\hat{r})$ $(1 \leq \hat{r} \leq k)$ using (6)
15:    $t = t + 2$
16: until convergence of the change in objective function
17: (that is, $|KL(p(R, C)||q^{(t)}(R, C)) - KL(p(R, C)||q^{(t+2)}(R, C))| < \mu$, $\mu$ is a small threshold)
```

Figure 2: Co-clustering algorithm

2.2 Refine table columns and rows by alignment to further expose redundancy

Next, a series of refinements are applied on a co-clustered table T' to further expose information redundancy. For each column, we compute the minimum value Col_min and their standard derivation value Col_std. If Col_min/Col_std is larger than a threshold α, it shows all elements in the column have similar values. Thereby, by deducting Col_min from all column elements, we can only code their differences more efficiently in a smaller range of numbers. In a column-cluster, we sort columns by their column mean values such that neighborhood columns show the maximum similarity. This will help to expose the redundancy among columns in a cluster. In addition, we compute the standard derivation value for each column to obtain the homogenous status in each column. Next, in a row-cluster, rows are sorted by the most similar columns according to the minimum standard derivation. The above transformation helps form the homogenous blocks. Their same or similar values will facilitate a statistical encoder (such as the run-length coding) in a standard compressor to improve compression effects.

Furthermore, for the same reason, if the number of rows is larger than the number of columns (i.e.$m > n$), we transpose the matrix, such that more number of similar neighbor elements are listed in a row and more redundancy can be exposed.

2.3 Compress the resorted data table by a standard compressor

After the above transformation, we use a statistical encoder for data compression. 7-zip is a popular and fast file archiver with a high compression ratio [20]. We select it since it supports several different data compression file formats and encryption algorithms, such as .7Z format with LZMA algorithm and .GZip format with DEFLATE algorithm [21]. LZMA is a variation of LZ77

```
Algorithm Compressing a Data Table
Input:  a Table T with $m$ rows and $n$ columns
        the number of row clusters $k$, the number of column clusters $l$
        a Threshold $\alpha$, a Standard Compressor $SC$
Output: an index $R'$ recording the original order of rows
        an index $C'$ recording the original order of columns
        a Mean Vector $V'$, a Compressed File $F'$
1:  //Co-clustering:
2:  Generate a measure matrix $W$ with $m$ rows and $n$ columns, let all entries in $W$ are $1s$
3:  Setup co-clustering parameters (Options) according to guidance in [15]
4:  Let $[R, C] = co - clusting(T, W, k, l, Options)$
5:  $T' = reorder(T, R, C)$; //Get a Reordered Data Table
6:  // Refine Table Columns and Rows in a co-clustering block:
7:  //If a column has little variance, each element is deducted from the column minimum value
8:  for each column i do
9:      $Col\_min = min(i^{th}$ column of $T')$; $Col\_std$ =standard_derivation($i^{th}$ column of $T'$)
10:     if $Col\_min/Col\_std > \alpha$ then deduct each element in the ith column of $T'$ by $Col\_min$
11:     record $Col\_min$ in $V'$
12: end for
13: //For those columns in a column cluster,sort them by their mean
14: for each column-clustering i do
15:     $Temp = T'(:, C == i)$ //find columns in column cluster i
16:     $Col\_mean = mean(Temp)$, $Temp = sort(T', Col\_mean)$
17:     Add sorted column-cluster Temp into new matrix SortT
18:     Record the updated column index in $C'$
19: end for
20: //For those rows in a row cluster, sort them by the column with the minimum std value
21: $[order\_index] = sort(std(SortT))$
22: for each row-clustering i do
23:     $Temp = SortT(R == i, :)$ //find those rows in row cluster i
24:     //sort rows by the column with the minimum std value
25:     $Temp = sort(Temp(:, order\_index(1)))$
26:     //use the reordered rows in row cluster i to form a matrix
27:     Add sorted row-cluster Temp into new matrix SortT'
28:     Record the updated row index in $R'$
29: end for
30: //Compress the Resorted Table by a Standard Compressor:
31: $F' = Compress(SC, SortT')$
```

Figure 3: Compression algorithm using co-clustering

algorithm and it uses entropy coding with a Markov chain based range coder and binary trees. DEFLATE is a standard algorithm based on LZ77 and Huffman coding. In the above two steps, our approach applies co-clustering to expose redundancy in a data table and the redundancy can be removed by any of these encoding algorithms. Thereby, we use 7-zip to test our approach.

2.4 Complete compression and decompression algorithm

The details of the compression algorithms are shown in Figure 3.

Accordingly, the decompression algorithm consists of two steps: decompression and reorder back to the original table. Its details are shown in Figure 4,

```
Algorithm Decompressing a File to its Original Data Table
Input: a Compressed File $F'$, a Mean Vector $V'$
       an index $R'$ recording the original order of rows
       an index $C'$ recording the original order of columns
       a Standard Compressor $SC$
Output: The Original Table $T$ with $m$ rows and $n$ columns
1: // Decompression:
2: $DT = Decompress(SC, F')$
3: //Reorder Back to the Original Table:
4: Reorder rows in $DT$ according to $R'$, Reorder columns in $DT$ according to $C'$
5: For a column with a $Col\_min$ value in $V'$, add each element in the column with $Col\_min$
```

Figure 4: Decompression algorithm using co-clustering

3 Experimental results

To test the effectiveness of our approach, we designed two experiments. One is the compression on a synthetic dataset. We would like to show how co-clustering can help to reorder and group columns and rows in a data table such that redundancy is exposed. Since compression ratio results are dependent upon different data tables used in mobile applications, we perform the other experiment which is compressing public benchmark datasets. These benchmark datasets are selected from UCI real-life datasets. They cover many typical topics in mobile applications, such as wearable sensors, clinical care, customer reviews, and they have the similar table structures and contents as data in mobile systems. We aim to compare the compression performance before and after applying co-clustering transformation for the benchmark datasets. The co-clustering software we used is Co_Cluster (Version 1.1) [15]. The compression software used is 7-zip. All experiments were performed on a machine with an Intel core i7 2.30GHz processor and 8GB main memory.

For comparing the compression effectiveness, we define a measure called improved compression rate (ICR) based on compression rate (CR) as below,

$$CR = \frac{Original \quad File \quad Size}{Compressed \quad File \quad Size} \tag{8}$$

$$ICR = \frac{CR \quad with \quad Co-clustering \quad - CR \quad without \quad Co-clustering}{CR \quad without \quad Co-clustering} \tag{9}$$

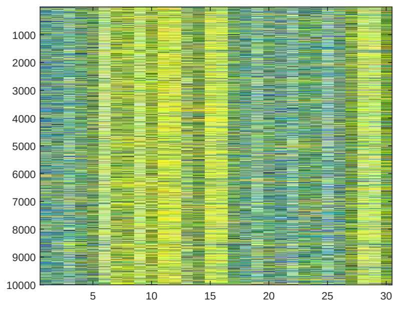
Another two measures are compression time and decompression time.

3.1 Experiment on a synthetic dataset

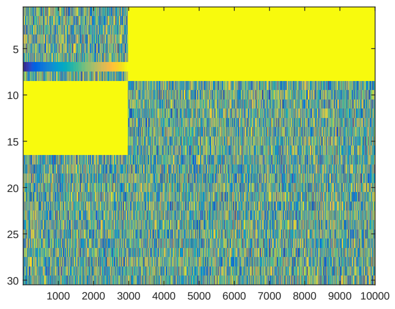
We firstly generate a data table with 10000 rows and 30 columns. It includes 2 row-clusters and 3 column-clusters. Thereby, the data table is designed with six blocks. Two of them are homogenous (entries values all are 1). The other four blocks contain entries with random values ranging in [0, 1]. By random permutation on rows and columns, the data table mixes six blocks together. The image of the designed data table is shown in Figure 5(a) and its original text file size is 946KB.

In compression algorithm, the parameter m is set to 10000, n is 30, k is 2, l is 3, and OPTIONS vector is set to default values.α is configured with 10.

Figure 5(b) shows the reordered data table images after co-clustering. It illustrates clearly that the two homogenous blocks (or equivalently say, redundancy) are discovered by reordering columns and rows. The homogenous blocks are helpful boost the compression effects by a statistical encoder. In addition, in this case $m > n$, so we transpose the transformed table such that the number of consecutive entries with the same or similar values in a row increases for improving the performance of an encoder. By comparing Figure 5(a) with Figure 5(b), we see that the redundancy information has been well exposed by co-clustering. It has been proved by final compression results. In 7-zip, by setting compression format as 7Z, compression level as maximum and compression algorithm as LZMA, other parameters as default values, the compressed file sizes before and after co-clustering are 268KB and 216KB respectively.



(a)



(b)

Figure 5: Synthetic data matrix image: (a) Before co-clustering, (b) After co-clustering and transpose

By (8), CR with Co-clustering and without Co-clustering is 4.38 and 3.53 respectively. Consequently, ICR for the synthetic dataset is 24%, a significant improvement on the original compression rate.

The compression procedure without Co-clustering takes 0.55 seconds, and the counterpart compression procedure with Co-clustering takes 4.77 seconds. For decompression, they both take 0.21 seconds.

3.2 Experiments on ten UCI datasets

We further test our approach on ten real-life benchmark datasets in multiple fields, such as wearable sensors, clinics, ecommerce, et al. They are from the discretized version of UCI Machine Learning Repository [22] and their structures and contents are very similar to data tables in mobile applications. The list of the ten datasets is described in Table 1. For each dataset, we list the number of rows and number of columns in a table, its original file size and its brief description.

Obviously, the ten datasets differ in table rows and columns, from 700 rows to 1000000 rows and from 6 columns to 10000 columns. Subsequently, their file sizes vary from hundreds of KB to tens of MB. The diversified tables are good to test the general compression and decompression performance of our approach.

Table 1: Dataset description

Dataset	Rows*Columns	Original Size	Brief Description
Nursery	12960*8	288KB	Applications for Schools
Waveform	5000*21	304KB	Attributes of waves
ChessKRvK	28056*6	463KB	Chess positions
Gait Freeze	151987*11	6018KB	Data from wearable acceleration sensors
Connect4	67557*42	8775KB	8-ply positions in a game
Diabetes130	101766*48	21482KB	10 years of clinical care data at 130 US hospitals
Arcene_test	700*10000	18560KB	Biomedical Features
Poker_hand	1000000*11	22978KB	A hand of playing cards
Opportunity	51116*250	39022KB	Dataset for Human Activity Recognition
Amazon	1500*10000	29396KB	Customer reviews in a commercial website

In co-clustering settings, m and n are set to the original dimension information of a data table. For simplicity, k is fixed to 4. If a table contains at least 10 columns, l is set to 4, otherwise, l is set to 2. α is configured with 10.

Table 2 compares the compression performance by using LZMA algorithm in 7Z without co-clustering or with co-clustering. In 7Z, the compression level is set to maximum. Other parameters in 7Z are configured with default values. The 3^{rd} and 4^{th} columns in Table 2 (column header "size by 7Z" and "CR by 7Z") show the compressed file sizes and compression rates by using 7Z without co-clustering. And the 5^{th} and 6^{th} columns (column header "size by 7Zco" and "CR by 7Zco") give the compressed file sizes and compression rates by using co-clustering. From the 7^{th} column, we see clearly that co-clustering can significantly improve the compression rates with $ICR \in [21\%, 59\%]$. The 8^{th} and 9^{th} columns compare the compression time difference between 7Z and 7Z with co-clustering. It suggests co-clustering takes more time when a table size increases. The 10th and 11th columns show decompression times between 7Z and 7Z with co-clustering are similar with each other.

For testing the compression performance by using other algorithms in 7Z, Table 3 list the compression results by using GZip algorithm. The compression level is also set to maximum and other parameters are configured with default values. Compared with the results in Table 2, the dataset compression rates by GZip are mostly lower than those by LZMA. However, our co-clustering approach still can improve the compression rates by the range from 23% to 68%.

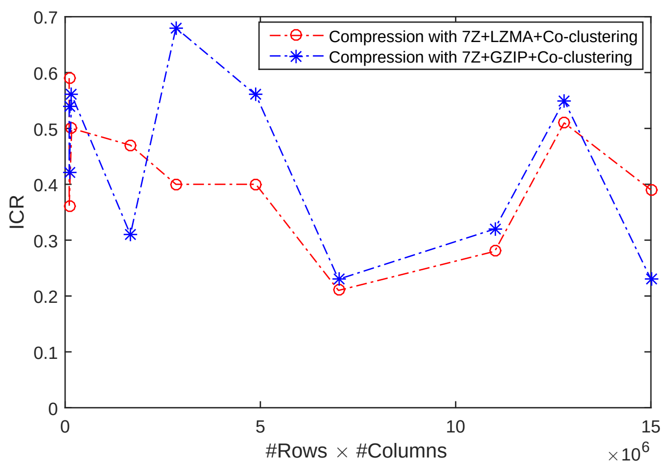


Figure 6: ICRs obtained using co-clustering with LZMA and GZip

Table 2: File compression comparison by using LZMA algorithm in 7Z

Dataset	Original Size (KB)	Size by 7Z (KB)	CR by 7Z	Size by 7Zco (KB)	CR by 7Zco	ICR	CT by 7Z (s)	CT by 7Zco (s)	DT by 7Z (s)	DT by 7Zco (s)
Nursery	288	35	8	22	13	59%	1.88	5.66	0.20	0.20
Waveform	304	30	10	22	14	36%	1.84	3.74	0.23	0.25
ChessKRvK	463	78	6	52	9	50%	3.00	7.13	0.32	0.30
Gait Freeze	6018	1167	5.16	792	7.60	47%	67.64	149.13	0.42	0.41
Connect4	8775	385	23	275	32	40%	46.73	98.70	0.44	0.40
Diabetes130	21482	2177	9.84	1554	13.79	40%	187.69	427.70	3.33	2.85
Arcene_test	18560	5572	3.33	4598	4.04	21%	260.46	539.58	3.81	4.09
Poker	22978	4794	4.79	3749	6.13	28%	444.35	855.87	3.63	3.98
Opportunity	39022	8403	4.64	5573	7.00	51%	522.43	1026.00	4.01	4.56
Amazon	29396	1691	17.38	1220	24.10	39%	170.13	437.01	3.96	4.24

Table 3: File compression comparison by using GZip algorithm in 7Z

Dataset	Original Size (KB)	Size by 7Z (KB)	CR by 7Z	Size by 7Zco (KB)	CR by 7Zco	ICR	CT by 7Z (s)	CT by 7Zco (s)	DT by 7Z (s)	DT by 7Zco (s)
Nursery	288	37	8	24	12	54%	2.00	4.02	0.22	0.23
Waveform	304	34	9	24	13	42%	2.06	3.88	0.24	0.24
ChessKRvK	463	84	6	54	9	56%	2.78	6.24	0.28	0.26
Gait Freeze	6018	1339	4.49	1025	5.87	31%	80.24	178.79	1.78	1.61
Connect4	8775	499	18	297	30	68%	50.01	106.13	2.23	1.74
Diabetes130	21482	2615	8.21	1680	12.79	56%	237.95	490.94	6.71	6.69
Arcene_test	18560	5878	3.16	4792	3.87	23%	307.20	643.88	6.80	6.23
Poker	22978	5342	4.30	4056	5.67	32%	433.88	986.79	7.58	8.27
Opportunity	39022	10256	3.80	6617	5.90	55%	524.55	1124.00	11.51	12.34
Amazon	29396	1908	15.41	1550	18.97	23%	194.45	497.12	8.87	7.42

Figure 6 illustrates that with the increased products of number of rows and columns in the ten tables, the ICRs are waved around 40%. It shows our algorithms keep stable of improved compression rates without much affected by table properties.

Figure 7 visualizes the compression times by using two different compression algorithms in 7Z with co-clustering or without co-clustering. It shows the two algorithms LZMA and GZip have no much difference from compression time view. Except for the last dataset "Amazon", we see that with the product of rows number and columns number increasing, the corresponding compression times increase linearly, without regarding to using co-clustering algorithm. It confirms that the time cost of co-clustering algorithm is just linearly related to the product of rows number and columns number in a table. Thereby, our approach is faster than other approaches where time costs are nonlinearly proportional to a data table size. The last dataset digitalizes customer reviews in Amazon website. By digitalization representation of text reviews, the table is a high dimensional sparse matrix and it contains a lot of consecutive 0s. It leads to a sharp decrease on compression time though it contains a huge number of columns.

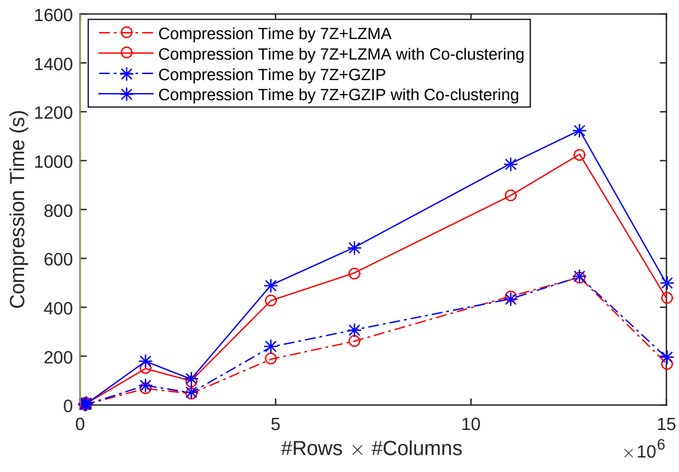


Figure 7: Compression time comparisons

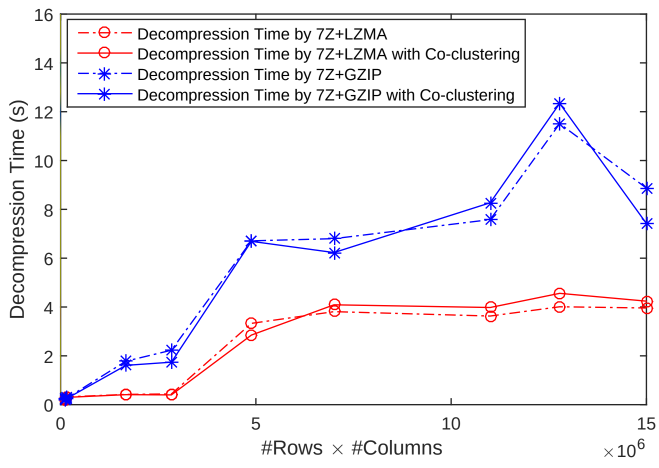


Figure 8: Decompression time comparisons

Figure 8 compares the decompression time costs with different compression settings. It shows LZMA algorithm runs consistently faster than GZIP on decompression for the ten datasets. The co-clustering approaches, either using LZMA or GZIP, take similar time on decompression than those counterparts without using co-clustering. One reason is that the compressed file after co-clustering has a smaller size than the packaged file without co-clustering. In addition, the inverse transform of co-clustering is just reordering rows and columns from recorded indexes in a compression procedure. Thereby, it is very simple and fast in table transformation. By integrating the two factors, we see in Figure 8 that our co-clustering compression approach takes very similar time costs on decompression as original 7Z approach. It is favorable for compressed data tables in a mobile device, which will mostly be sent to a cloud server and be decompressed.

Conclusions

Data tables are widely used in many mobile applications. Lossless compression of a data table can not only reduce storage sizes and network transmission latencies, but also save energy in batteries. In this paper, we propose a novel co-clustering compression approach consisting of three steps: reordering and grouping columns and rows by co-clustering; refine table rows and columns to further expose redundancy; data compression by using a standard compressor. We tested the approach on a synthetic dataset and ten UCI real-life datasets. The experimental results suggest that our approach can significantly improve compression rates at least 21% and up to 68%. The approach is robust both to LZMA and GZip encoders. In addition, the time cost of our approach is linearly proportional to the product of number of columns and rows in a data table. Furthermore, since the inverse transform of co-clustering is just exchange of rows and columns according to records, the decompress procedure is fast and takes similar time on decompression as the original compressor. Therefore, our approach is suitable for compressing a data table in a mobile device where the table requires a limited storage size and reduced communication latency with constrained energy cost.

In next step, we would like to test our co-clustering approach on lossy compression of multimedia files in mobile devices, such as images and videos. We are also interested in incorporating other dimension reduction techniques, such as principle component analysis and independent component analysis, to further improve the effectiveness of multimedia compression in the mobile systems with constrained resources.

Acknowledgment

This work was supported by the National Natural Science Foundation of China under Grants 61272272 and U1531122, and by the Natural Science Foundation of Hubei province under Grant 2015CFA058.

Bibliography

[1] Rao S., Bhat P. (2015); Evaluation of lossless compression techniques, *IEEE International Conference on Communications and Signal Processing (ICCSP)*, 1655-1659.

[2] Kontoyiannis I., Verdu S. (2014); Optimal lossless data compression: Non-asymptotics and asymptotics, *IEEE Transactions on Information Theory*, 60(2): 777-795.

[3] D. Salomon (2007); *Data Compression: The Complete Reference*, 4th ed. New York: Springer, 2007.

[4] K. Sayood (2000); *Introduction to Data Compression*, 2nd ed. San Francisco, CA: Morgan Kaufmann, 2000.

[5] Mielikainen J., Huang B.(2012); Lossless compression of hyperspectral images using clustered linear prediction with adaptive prediction length, *IEEE Geoscience and Remote Sensing Letters*, 9(6): 1118-1121.

[6] Venugopal D., Mohan S., Raja S.(2016); An efficient block based lossless compression of medical images, *Optik-International Journal for Light and Electron Optics*, 127(2): 754-758.

[7] Patauner C. et al. (2011); A lossless data compression system for a real-time application in HEP data acquisition, *IEEE Transactions on Nuclear Science*, 58(4): 1738-1744.

[8] Kolo J.G. , et al. (2012); An adaptive lossless data compression scheme for wireless sensor networks, *Journal of Sensors*, vol. 2012, Article ID 539638, http://dx.doi.org/10.1155/2012/539638, 1-20.

[9] Kolo J.G. et al. (2015); Fast and efficient lossless adaptive compression scheme for wireless sensor networks, *Computers & Electrical Engineering*, 41: 275-287.

[10] A.L. Buchsbaum et al. (2000); Engineering the compression of massive tables: an experimental approach, *Proceedings of the ACM-SIAM Annual Symposium on Discrete Algorithms*, San Francisco, CA, 213-222.

[11] A.L. Buchsbaum, G.S. Fowler, R. Giancarlo (2002); Improving table compression with combinatorial optimization, *Proceedings of the ACM-SIAMAnnual Symposium on Discrete Algorithms*, San Francisco, CA, 175-184.

[12] Q. Yang, S. Lonardi (2005); A compression-boosting transform for 2D data, *Data Compression Conference (DCC'05)*, 492-492.

[13] H. Shan, A. Banerjee (2008); Bayesian co-clustering, *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM'08)*, 530-539.

[14] F. Pan, X. Zhang, W. Wang (2008); CRD: fast co-clustering on large datasets utilizing sampling-based matrix decomposition, *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'08)*, 173-184.

[15] A. Banerjee et al. (2007); A generalized maximum entropy approach to Bregman coclustering and matrix approximation, *Journal of Machine Learning Research*, 1919-1986.

[16] I.S. Dhillon, S. Mallela, D.S. Modha (2003); Information-theoretic co-clustering, *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03)*, 89-98.

[17] R.G. Pensa, J.F. Boulicaut, F. Cordero, M. Atzori (2010); Co-clustering numerical data under user-defined constraints, *Statistical Analysis and Data Mining*,3(1): 38-55.

[18] R.G. Pensa, J.F. Boulicaut (2008); Constrained co-clustering of gene expression data, *Proceedings in Applied Mathematics 130, 8th SIAM International Conference on Data Mining 2008*, 1:25-36.

[19] Wu X., Zurita-Milla R., Kraak M.J. (2015); Co-clustering geo-referenced time series: exploring spatio-temporal patterns in Dutch temperature data, *International Journal of Geographical Information Science*, 29(4): 624-642.

[20] http://www.7-zip.org

[21] http://en.wikipedia.org/wiki/7z

[22] F. Coenen, The LUCS-KDD Discretised/normalized ARM and CARM Data Library, In http://www.csc.liv.ac.uk/~frans/KDD/Software/LUCS-KDD-DN/DataSets/dataSets.html