Column-wise compression of open relational data

Sebastian Wandelt[a,b], Xiaoqian Sun[a,b,*], Ulf Leser[c]

[a] *School of Electronic and Information Engineering, Beihang University, Beijing, China*
[b] *Beijing Key Laboratory for Network-based Cooperative ATM, Beijing, China*
[c] *Knowledge Management in Bioinformatics, Humboldt-University of Berlin, Berlin, Germany*

Abstract

The recent growth of open data initiatives has led to a tremendous increase in publicly available data resources. This amount of data, together with a rising interest in people to analyze it, poses severe challenges regarding data storage. Data suppliers often compress their resources with standard compressors. The choice of a compression technique, however, has significant impacts on the compression ratio, the compression speed, and the decompression speed.

In this paper, we provide an empirical analysis on the compression of open data provided in a relational format, such as comma-separated value files. We consider several compression tools and parameter settings. Furthermore, we propose using a novel column-wise compression strategy, where items that have similar properties, are compressed together. We perform a comprehensive analysis on 24 datasets from different domains, such as life sciences, governmental data, finance sector, and public transportation, which cover a wide range of file sizes (from a few MB to several GB). Our results show that the traversal strategy is of paramount importance for achieving high compression ratios; with improvements of up to one order of magnitude. This study further highlights a set of issues for future work on compressing open data.

1. Introduction

The Open Data movement is an emerging force, with the idea that certain data should be freely available for everyone to use [3, 7, 9]. There are multiple initiatives at various organizational levels, such as the Europe 2020 Initiative, which strongly encourage a culture of sharing [14]. Another example is the Open Science Initiative [28], which states that scientific data, whether collected directly as part of an experiment or indirectly as a secondary output of downstream analysis, should be made freely available in electronic form and accessible online [36]. Accordingly, many large datasets were provided for various domains, for instance, health care [1], transportation [31], geographical [18], public sector [19], and education [22]. Clearly, reproducibility and efficiency of scientific processes benefit, the more data is made openly available in a useful manner [25]. Therefore, many researchers have advocated for the reduction of barriers to the availability and reusability of scientific data [32, 30, 27]. Furthermore, a robust citation benefit from open data was found, and that, at least for gene expression microarray data, a substantial fraction of archived datasets are reused, and that the intensity of dataset reuse has been steadily increasing since 2003 [29]. In this light, researchers proposed that data should be considered as first class citizens for scientific sharing and publishing [5].

It has been argued that raw data must be formatted in a standard way, together with appropriate metadata attached [32]. In addition, complexity of open data must be reduced in order to increase attraction [20]. Therefore, open datasets are often released in simple file formats, such as comma-separated value (CSV) files. This file format is chosen since parsing is trivial and many tools support processing of CSV files and importing these files into database systems for post-processing. However, the release of complete raw datasets comes at the price of high storage and transmission costs. Hence, data providers often compress their data simply using the standard compression format zip. The choice of a compressor, however, has a tremendous impact on the result of the compression. A comprehensive analysis of which compressor is used (or should be used) for compression of open data is, to the best of our knowledge, not available.

In this paper, we analyse the compressibility of 24 carefully selected datasets. We report on the effectiveness of different standard compression techniques (zip, bzip2, gzip, LZMA, LZMA2, and PPMd) with different parameter

*Corresponding author: sunxq@buaa.edu.cn

setups. Our study analyzes the following three criteria: 1) Compression ratio (the size of the original file in Byte divided by the size of the compressed representation in Byte), 2) Compression speed (the amount of MByte compressed per second), and 3) decompression speed (the amount of MByte decompressed per second).

We find that the results for the three criteria differ significantly for each dataset; there is no single best compressor according to compression ratio, compression speed, and decompression speed. Using Pareto analysis, we find that, if the user is interested in two out of three criteria only, the number of candidate compression tools is significantly reduced. Considering all three criteria at the same time makes the choice of an appropriate standard compressor complicated.

We next analyzed non-standard compression techniques. All standard compressors follow a row-wise compression scheme. For CSV files, this traversal scheme is often a poor choice, since each line consists of several contexts (usually one for each column in the CSV file). We propose to compress CSV files following a column-wise traversal scheme, which avoids context changes and thus often increases the compression ratio significantly. Our traversal strategy is inspired by column-databases, which store each column separately [4, 21, 8, 34]. We extend column-wise traversal by clustering columns together, and show that this increases the compression ratio further. The major contributions of our study are summarized as follows:

1. We provide an extensive analysis on the **compressibility of 24 public datasets**, finding that the compression ratio, compression speed, and decompression speed largely depend on the compression method. Moreover, our analysis reveals that no single competitor serves as a silver bullet for compression.
2. We show that changes to the traversal strategy, from row-wise to **column-wise**, can **boost the compression ratio significantly**, since the latter strategy can exploit value similarities inside columns.
3. Extending the column-wise compression method by **clustering of highly-similar columns**, significantly increases the compression ratio for several datasets, leading towards a novel approach to open data compression.
4. Based on our **comprehensive evaluation**, we propose a list of **future research directions**, which will help addressing current data storage problems.

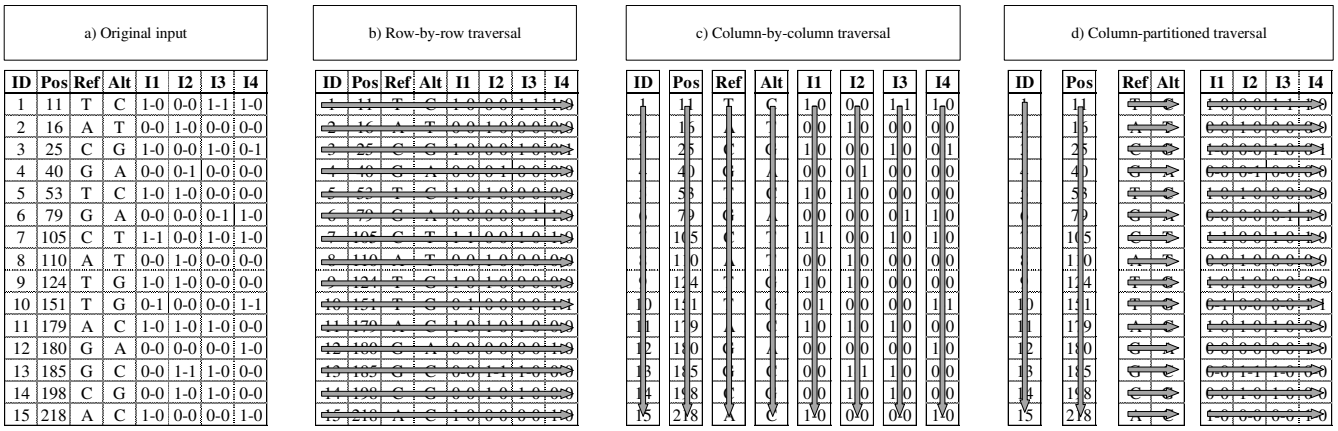
Related work usually addresses the problem of compression in fixed domains or for specific datasets. Much work has been published on the compression of biological sequence data (or post-processed results), together with comprehensive surveys on the problem [13, 16, 37]. Compression techniques have also been compared for image data [33], wireless sensor data [15, 39], video encoding [17, 23], telecommunication data [10], and 4D trajectories of aircraft [38]. Moreover, a comparison of compression tools for typical Linux data was performed [26]. We are not aware of any comprehensive analysis on general open data stored in CSV files. Avoiding changes in compression contexts for increasing the compression ratio has also been proposed in the Bioinformatics community [12], where the authors compress FASTQ-files, which represent information about sequencing of individuals. Besides raw data, these files contain also quality scores denoting uncertainties occurred during sequence identification. These distinct streams in the input file (albeit not being columns as in CSV files) are compressed separately, yielding an improved compression ratio and significantly higher compression speeds. Similar ideas have also been used for storing telecommunication data [10]; note that in telecommunications, compression does not only reduce transmission time, but also induces significant power savings [35].

The remainder of this manuscript is organized as follows. Section 2 introduces the methodology underlying the compression of relational data. Section 3 presents the results of our evaluation against 24 datasets. In Section 4, we discuss the implications of our findings and propose a list of future research directions.

2. Methods

2.1. Standard Compression of CSV Files

A *string* s is a finite sequence of characters from an alphabet Σ. The concatenation of two strings s and t is denoted with $s \circ t$. A string s is a *substring* of string t, if there exist two strings u and v (possibly of length 0), such that $t = u \circ s \circ v$. The length of a string s is denoted with $|s|$ and the substring starting at position i with length n is denoted with $s(i, n)$. $s(i)$ is an abbreviation for $s(i, 1)$. All positions in a string are zero-based, i.e., the first character is accessed by $s(0)$. A CSV file represents data in the relational model, with a fixed number of columns and a fixed number of rows. Each row in the file corresponds to a tuple in the relation, where the items of a tuple are separated by a comma (or any other delimiter). The first line in a CSV file is sometimes reserved for stating the



ID	Pos	Ref	Alt	I1	I2	I3	I4
1	11	T	C	1-0	0-0	1-1	1-0
2	16	A	T	0-0	1-0	0-0	0-0
3	25	C	G	1-0	0-0	1-0	0-1
4	40	G	A	0-0	0-1	0-0	0-0
5	53	T	C	1-0	1-0	0-0	0-0
6	79	G	A	0-0	0-0	0-1	1-0
7	105	C	T	1-1	0-0	1-0	1-0
8	110	A	T	0-0	1-0	0-0	0-0
9	124	T	G	1-0	1-0	0-0	0-0
10	151	T	G	0-1	0-0	0-0	1-1
11	179	A	C	1-0	1-0	1-0	0-0
12	180	G	A	0-0	0-0	0-0	1-0
13	185	G	C	0-0	1-1	1-0	0-0
14	198	C	G	0-0	1-0	1-0	0-0
15	218	A	C	1-0	0-0	0-0	1-0

Figure 1: Different traversal strategies for compressing CSV files. Subfigure a) shows uncompressed relational input with eight columns. Subfigure b) visualizes the row-by-row traversal strategy as performed by standard compression tools. Subfigure c) shows an alternative traversal strategy, where columns are compressed separately. Subfigure d) demonstrates a hybrid solution, where some columns are clustered together.

names of columns. In order to discuss compression and traversal strategies of a CSV file, we define such a file as a matrix, neglecting the optional header line [1].

Definition 1 (CSV matrix)**.** A CSV file with m columns and n rows is modeled as a *CSV matrix*

$$M_{csv} = \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & \dots & x_{1,m} \\ x_{2,1} & x_{2,2} & x_{2,3} & \dots & x_{2,m} \\ \dots & \dots & \dots & \dots & \dots \\ x_{n,1} & x_{n,2} & x_{n,3} & \dots & x_{n,m} \end{bmatrix}$$

The columns of matrix M_{csv} are denoted with $Col_1, \dots, Col_m$. The number of columns is denoted with $colCount(M_{csv}) = m$ and the number of rows is denoted with $rowCount(M_{csv}) = n$. The item at row j and column i, i.e. $x_{j,i}$, is denoted with $M_{csv}[j, i]$.

Example 1. The CSV matrix M_{EX} below represents the original input from Figure 1a).

$$M_{EX} = \begin{bmatrix} 1 & 11 & T & C & 1-0 & 0-0 & 1-1 & 1-0 \\ 2 & 16 & A & T & 0-0 & 1-0 & 0-0 & 0-0 \\ 3 & 25 & C & G & 1-0 & 0-0 & 1-0 & 0-1 \\ 4 & 40 & G & A & 0-0 & 0-1 & 0-0 & 0-0 \\ 5 & 53 & T & C & 1-0 & 1-0 & 0-0 & 0-0 \\ 6 & 79 & G & A & 0-0 & 0-0 & 0-1 & 1-0 \\ 7 & 105 & C & T & 1-1 & 0-0 & 1-0 & 1-0 \\ 8 & 110 & A & T & 0-0 & 1-0 & 0-0 & 0-0 \\ 9 & 124 & T & G & 1-0 & 1-0 & 0-0 & 0-0 \\ 10 & 151 & T & G & 0-1 & 0-0 & 0-0 & 1-1 \\ 11 & 179 & A & C & 1-0 & 1-0 & 1-0 & 0-0 \\ 12 & 180 & G & A & 0-0 & 0-0 & 0-0 & 1-0 \\ 13 & 185 & G & C & 0-0 & 1-1 & 1-0 & 0-0 \\ 14 & 198 & C & G & 0-0 & 1-0 & 1-0 & 0-0 \\ 15 & 218 & A & C & 1-0 & 0-0 & 0-0 & 1-0 \end{bmatrix}$$

[1]The header line can be stored uncompressed, since the size of a CSV file is usually dominated by the data, not the header.

Algorithm 1 STDComp: Standard compression of a CSV matrix

Input: To-be-compressed CSV matrix M_{csv} and compressor *comp*
Output: Compressed string c
$s = ''$
for $1 \leq j \leq rowCount(M_{csv})$ **do**
  for $1 \leq i \leq colCount(M_{csv})$ **do**
    $s = s \circ M_{csv}[j,i] \circ ','$
  end for
end for
c=comp(s)
return c

Compression tools usually work on a stream of bytes, instead of a matrix. Since a tuple of M_{csv} corresponds to one line in the CSV file, standard tools will compress CSV files as a stream of lines. This traversal strategy is described in Algorithm 1. The algorithm compresses a CSV-Matrix M_{csv} with a byte-wise compressor *comp*. Here, we model a compressor as a function *comp* from strings to strings, i.e., $comp(s)$ denotes a byte-wise compressed representation of s. We discuss several instances for *comp* below. In Algorithm 1, M_{csv} is traversed line-by-line and the to-be-compressed string s is constructed. The result of the compression is denoted with $STDComp(M_{csv}, comp)$. Please note that standard compression algorithms do not generate the full s first, but usually compress a file on the fly; yet, the traversal strategy remains line-by-line. The time complexity of Algorithm 1 is linear in the size of the input, as long as the compressor *comp* (a parameter to the algorithm) has a linear time complexity. All compressors evaluated in our study have this property.

We provide a brief review of standard compression techniques next. The six standard compressors evaluated in our study are all implemented in the command line port p7zip (`http://www.7-zip.org/`):

- **zip**: Standard LZ77-based algorithm (Command line switch 7za a -tzip). LZ77/zip-based methods are all table-based: Table entries are kept for storing recurring substrings of data. For most LZ methods, this table is generated dynamically from earlier data in the input.
- **bzip2**: Standard BWT algorithm (Command line switch 7za a -tbzip2). bzip2 compresses a text by performing a burrow-wheeler transformation [11] on the run-length encoding of the input.
- **gzip**: Standard LZ77-based algorithm (Command line switch 7za a -tgzip). Opposed to zip, gzip in general only compresses a single file; multiple files can be compressed by tar-balling these files first.
- **LZMA**: Optimized version of LZ77 algorithm (Command line switch 7za a -t7z -m0=LZMA). In LZMA, compression is improved over LZ77 by using a longer history buffer (up to 4 GB), optimal parsing, shorter codes for recently repeated matches, literal exclusion after matches, and arithmetic coding.
- **LZMA2**: Improved version of LZMA (Command line switch 7za a -t7z -m0=LZMA2). LZMA2 is a successor of LZMA, which can contain both uncompressed data and LZMA data, the latter one with multiple different encoding parameters.
- **PPMd**: Dmitry Shkarin's PPMdH with small changes (Command line switch 7z a -t7z -m0=PPMd). The compressor PPM compresses at byte level: A previous context is kept as a variable number of bytes, and the next byte, is predicted based on the most recent context. PPMd is a refinement that considers additional contexts.

All compressors provide a set of parameters, which influence compression ratio and compression speed. These parameters are controlled by selecting a predefined compression level (between 1 and 9). We analyze the impact of these levels in Section 3. In general, when compressing data, one is interested in finding a small (the smallest) compression of the dataset at hand.

Compression Problem 1 (Finding minimal compression method for row-by-row traversal)**.** Given a CSV matrix M_{csv} and a set of standard compression methods $COMP = \{comp_1, \ldots, comp_n\}$, compression method $comp_i$ is min-

imal w.r.t. row-by-row traversal of M_{csv}, if there exists no $comp_j$ with $i \neq j$ such that $|STDComp(M_{csv}, comp_j)| < |STDComp(M_{csv}, comp_i)|$. Compression Problem 1 is defined as finding the minimal compression method in $COMP$.

Finding a minimal standard compression method, as defined in Compression Problem 1, is a fundamental problem. Depending on the size of $COMP$, the compression of M_{csv} against each $comp_i$ can be computed and the compression method with the minimal compression is chosen. In practice, however, data suppliers often just pick one compression method without trying others. One reason is that the execution of a single compression method is very time consuming for large datasets: In our evaluation we show that compressing a dataset of 11 GB can take several hours to finish. Another reason is that the knowledge about different compression methods together with their strengths and weaknesses is not widespread outside the computer science community. We show in our evaluation that the choice of a standard compression method has a tremendous effect on the compression ratio (as well as compression speed and decompression speed). However, as we discuss in the next subsection, the degree of freedom for the compression of CSV files is much higher than defined in Compression Problem 1.

2.2. Compression guided by Column-partitionings

Effectively, the string s in Algorithm 1 is a linearization of M_{csv}. It is easy to see that there are other types of linearization, and some of them might be significantly better compressible than the linearization obtained in Algorithm 1. The reason is that different linearizations order values from the CSV file differently, such that, ideally, identical values are places next to each other, obtaining for higher compression ratios. In general, if we want to compress a CSV matrix M_{csv}, *any* efficiently reversible linearization of M_{csv} can be considered as input to a standard compression program. For instance, we could define a column-wise traversal strategy or apply a Z-order curve for processing M_{csv}. There exist $(colCount(M_{csv}) * rowCount(M_{csv}))!$ different traversal strategies for M_{csv}. Analysing all of them is clearly intractable. Different compression traversal strategies are visualized in Figure 1. Given a to-be-compressed input (Figure 1a), standard compressors process a file row-by-row (Figure 1b). Alternative compression strategies include column-by-column compression (Figure 1c) and column-partitioned compression (Figure 1d). Below we formalize the idea of using distinct traversal strategies for the compression of CSV files. We discuss a particular type of traversal strategy, which processes M_{csv} column-wise.

The general idea of column-wise data processing has long traditions in the database community [4, 21]; yet systems like MonetDB [8] pioneered its usage in modern column-oriented database systems only recently [2]. We design a novel compression technique based on column-wise compression and further extend the idea of by keeping some columns together, in so-called column clusters. The idea is that some columns can be compressed better together than being compressed separately (we will discuss such examples in Section 3).

Definition 2 (Column Cluster and Column Partitioning)**.** A *column cluster* for a CSV matrix M_{csv} is a non-empty list $clust = [col_1, \ldots, col_x]$, such that for each $1 \leq i \leq x$, we have that $1 \leq col_x \leq colCount(M_{csv})$ and for each $1 \leq i < j \leq x$ we have $col_i \neq col_j$. With M_{csv}^{clust} we denote the projection of M_{csv} to the columns of $clust$, i.e. the first column of M_{csv}^{clust} is column col_1, the second column of M_{csv}^{clust} is column col_2, and so on. A *column partitioning of* M_{csv} is a non-empty list of column clusters $part = [clust_1, \ldots, clust_y]$, such that all clusters are pairwise disjoint and for each $1 \leq i \leq colCount(M_{csv})$, there exists a cluster in $part$, which contains column i.

Example 2. Given the CSV matrix M_{EX} from Example 1 and the column partitioning $part = [[1], [2], [3, 4], [5, 6, 7, 8]]$,

Algorithm 2 COLPARTComp: Compression of a CSV matrix against a column-partitioning

Input: To-be-compressed CSV matrix M_{csv}, column partitioning $part = [clust_1, \ldots, clust_y]$ for M_{csv} and list of compressors $[comp_1, \ldots, comp_y]$
Output: Compressed string c
$c = ''$
for $1 \leq k \leq y$ **do**
$s = ''$
  for $1 \leq j \leq rowCount(M_{csv}^{clust_k})$ **do**
    for $1 \leq i \leq colCount(M_{csv}^{clust_k})$ **do**
      $s = s \circ M_{csv}^{clust_k}[j, i] \circ ','$
    end for
  end for
$c = c \circ comp_k(s)$
end for
return c

the projections of the four column clusters are:

$$M_{EX}^{[1]} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \end{bmatrix}, M_{EX}^{[2]} = \begin{bmatrix} 11 \\ 16 \\ 25 \\ 40 \\ 53 \\ 79 \\ 105 \\ 110 \\ 124 \\ 151 \\ 179 \\ 180 \\ 185 \\ 198 \\ 218 \end{bmatrix}, M_{EX}^{[3,4]} = \begin{bmatrix} T & C \\ A & T \\ C & G \\ G & A \\ T & C \\ G & A \\ C & T \\ A & T \\ T & G \\ T & G \\ A & C \\ G & A \\ G & C \\ C & G \\ A & C \end{bmatrix}, M_{EX}^{[5,6,7,8]} = \begin{bmatrix} 1-0 & 0-0 & 1-1 & 1-0 \\ 0-0 & 1-0 & 0-0 & 0-0 \\ 1-0 & 0-0 & 1-0 & 0-1 \\ 0-0 & 0-1 & 0-0 & 0-0 \\ 1-0 & 1-0 & 0-0 & 0-0 \\ 0-0 & 0-0 & 0-1 & 1-0 \\ 1-1 & 0-0 & 1-0 & 1-0 \\ 0-0 & 1-0 & 0-0 & 0-0 \\ 1-0 & 1-0 & 0-0 & 0-0 \\ 0-1 & 0-0 & 0-0 & 1-1 \\ 1-0 & 1-0 & 1-0 & 0-0 \\ 0-0 & 0-0 & 0-0 & 1-0 \\ 0-0 & 1-1 & 1-0 & 0-0 \\ 0-0 & 1-0 & 1-0 & 0-0 \\ 1-0 & 0-0 & 0-0 & 1-0 \end{bmatrix}$$

From the partitioning in Example 2, potential benefits of a column partitioning for the compression of a CSV matrix become visible. Each cluster contains symbols from a reduced alphabet, compared to the alphabet of the complete CSV matrix M_{EX}. Furthermore, repetitions in the data, e.g. frequently occurring items, often appear closer together and are easier to be identified. Other examples for a column partitioning include $part = [[1], [2], [3], [4], [5], [6], [7], [8]]$ (where each column is compressed separately) and $part = [[1, 2, 3, 4, 5, 6, 7, 8]]$ (where all columns are compressed together, i.e., the CSV matrix is traversed row-wise). Thus, our model is rather general, capturing all such variants with a single definition. Given a partitioning of the CSV matrix, we can now compress each cluster with a different compression method. The intuition is that the effectiveness of a compression method depends mainly on the type of data, which may be different for columns/cluster. The algorithm for compression of a column-partitioning is presented in Algorithm 2: The projection of each cluster is compressed separately using a compression method in $[comp_1, \ldots, comp_y]$. The result of the compression is denoted with $COLPARTComp(M_{csv}, part, compl)$, where $compl$ is the list of compression methods. Similarly to Algorithm 1, the time complexity is linear in the size of the input, as long as all compressors $comp_1$ to $comp_y$ have a linear time complexity. Again, all compressors evaluated in our study have this property.

The number of possible column partitionings for a CSV matrix M_{csv} grows quickly with the number of columns: The number of partitions of a set is computed by the Bell number [6]. For a CSV matrix with 10 columns, the Bell number is already 115,975, which makes it intractable to evaluate all combinations by hand. Therefore, we investigate two simplified compression problems below: First, we start with the compression of separate columns, where each partitions consists of a single column only. Second, we will group similar columns into a compression

stream.

Compression Problem 2 (Finding minimal compression method for column-by-column traversal)**.** Given a CSV matrix M_{csv} with m columns and a set of standard compression methods $COMP = \{comp_1, \ldots, comp_n\}$, compression list $compl$ (over $COMP$) is minimal w.r.t. column-by-column traversal of M_{csv}, if there exists no $compl_2$ with $compl \neq compl_2$ such that $COLPARTComp(M_{csv}, [[1], \ldots, [n]], compl_2) < COLPARTComp(M_{csv}, [[1], \ldots, [n]], compl)$. Compression Problem 2 is defined as finding the minimal compression list for separate columns.

In Compression Problem 2, we simply considered separate columns. Below we discuss an approach to partition columns. We propose a simple heuristic for computing a column partitioning for a CSV matrix M_{csv}. Our partitioning technique is based on combination of statistics derived for each column separately. We exploit these statistics to cluster columns with similar values together. Note that our heuristic is a proof of concept; we propose several improvements of this approach in Section 4. For each column col_i of a CSV matrix M_{csv}, we extract the following five statistics:

1. Σ_i: The alphabet of all items in column i, i.e., the alphabet of the items $x_{i,1}, \ldots, x_{i,n}$. The alphabet of a column can be used for an initial estimation about compressibility: Smaller alphabets can, in general, be better compressed than larger alphabets. Moreover, columns with a largely overlapping alphabet are likely to be well compressible together.
2. $Entropy_{\Sigma_i}$: The entropy of character frequencies for each of the symbols $\sigma \in \Sigma_i$ of the items $x_{i,1}, \ldots, x_{i,n}$. The smaller the entropy, the fewer bits are needed for the amortized representation of a symbol.
3. $ItemEntropy_i$: The entropy of item frequencies for each item in $x_{i,1}, \ldots, x_{i,n}$. Columns with a low item entropy contain a large number of repetitive items, which can be efficiently compressed.
4. $RatioNonEmpty_i$: The ratio of non-empty items to the total number of items in $x_{i,1}, \ldots, x_{i,n}$. Columns with many empty items can often be better compressed than completely filled columns.
5. $Uniqueness_i$: The number of unique items in the column, divided by the number of non-empty items in the column. The less unique items a column has, the better it can be compressed.

The idea for our column clustering heuristic is to keep columns together which have similar properties for all five statistics. We compute the sum of absolute differences between four out of five statistics first.

$$\begin{aligned} diffsum(i_1, i_2) = &|Entropy_{\Sigma_{i_2}} - Entropy_{\Sigma_{i_1}}| + |ItemEntropy_{i_2} - ItemEntropy_{i_1}| + \\ &|RatioNonEmpty_{i_2} - RatioNonEmpty_{i_1}| + |Uniqueness_{i_2} - Uniqueness_{i_1}| \end{aligned}$$

Using this summation, we define the similarity of two columns, taking into account the alphabet of both columns: Columns with non-identical alphabets are considered as dissimilar in our heuristic. Future measures could also consider the degree of overlap between alphabets (see our discussion below).

$$sim(i_1, i_2) = \begin{cases} 0 & \text{if } \Sigma_i \neq \Sigma_j \\ 1 & \text{else if } diffsum(i_1, i_2) = 0 \\ \frac{1}{diffsum(i_1, i_2)} & \text{else.} \end{cases}$$

Given the similarity estimation between all column pairs, we would like to identify highly-similar columns automatically. This is a clustering problem: Given a set of observations about the columns, identify columns with a high degree of similarity and merge them into a column cluster. Unfortunately, we deal with data which is unknown at compile time. Thus, standard clustering algorithms (centroid-based clustering methods or density-based methods) are not applicable, since we have neither an initial estimation for the number of clusters, nor can we give a density threshold. Recently, the parameter-free clustering algorithm PFClust [24] was shown to provide good results with respect to external gold standard clusterings. This algorithm groups items together that share common attributes, such as their minimum expectation value and variance of intra-cluster similarity. For our experiments we use PFClust to cluster the columns based on the similarity function sim between columns (the input of PFClust is a matrix of similarity scores).

Below we state the third compression problem in this paper, where $part^* = [clust_1, \ldots, clust_y]$ is the partitioning obtained by the procedure described above.

Compression Problem 3 (Finding minimal compression method for column-partitioned traversal)**.** Given a CSV matrix M_{csv} with m columns and a set of standard compression methods $COMP = \{comp_1, \ldots, comp_n\}$, compression list *compl* (over $COMP$) is minimal w.r.t. column-partitioned traversal of M_{csv}, if there exists no $compl_2$ with $compl \neq compl_2$ such that $COLPARTComp(M_{csv}, part^*, compl2) < COLPARTComp(M_{csv}, part^*, compl)$. Compression Problem 3 is defined as finding the minimal compression list for the column partitioning $part^*$.

We summarize the relation between the three introduced compression problems as follows:

- Compression Problem 1: Standard row-by-row traversal technique employed by most existing compression algorithms. This works reasonably well for relational data where cell value distributions follow a random process closely and columns have no distinct data types.
- Compression Problem 2: Transposed compression with column-by-column traversal. Solving this problem is more appropriate than Compression Problem 1, if the relational data has columns that have mixed data types, such as number, strings, dates, etc.; and cell values are frequently based on previous cells in the *same* column. Most relational data meet these latter criteria.
- Compression Problem 3: A combination of Compression Problem 1+2, where selected columns are grouped together and cells are traversed row-by-row within each group. Solving this problem yields particularly good results if the relational data contains columns that have highly similar data types, whose values can be nicely compressed together. An example is variant call data from human genome indexing.

3. Results

3.1. Setup

Our experiments were run on a machine with eight AMD FX(tm)-8320 and 32 GB RAM. The operating system on the evaluation machine was Fedora 21 (64-Bit, Linux kernel 3.19.3-200.fc21.x86_64). The column-wise traversal and clsutering strategy for this study were implemented in C++11. All size measures are in bytes, e.g. 1 MB means 1,000,000 bytes. All standard compressors were tested as implemented in the command line port p7zip of 7zip (`http://www.7-zip.org/`). For our experiments, we used 7za in version 9.20, the latest stable release. All compressors provide a set of parameters, which influence compression ratio and compression speed. In general, these parameters are controlled by selecting a predefined compression level between 1 and 9. For each of these six compressors, we tested three different compression levels: 1 (low compression, fastest mode), 5 (normal compression mode), and 9 (ultra compression, usually the slowest mode).

3.2. Datasets

The experiments in this study are performed on 24 real-world datasets. All datasets were provided as CSV files, with different delimiters (comma, tab, colon) and escape characters (single or double quotes). The datasets are summarized in Table 1. Our study analyzed public data from quite different domains, such as, life science (CHR22, PLANTS), health care (NPPES, BSAPUF), transportation (NAPTAN, SO6, BABSTD), climate research (QCLCD, STORM), geographical-information systems (LONDON), high-energy physics (STAR), and public sector reports (POLICE, WQP). Each of these datasets is freely available for download. The largest dataset, according to the size is CHR22, a description of variant calls from the 1000 Genome Project [1] against a reference genome. This uncompressed dataset is approx. 11 GB in size. CHR22 is also the dataset with the largest number of columns (2,513). Most columns present the variant calls of one of the 2,504 individuals, with nine additional columns for metadata describing the variant. The dataset with the largest number of rows is PPD, which is a database tracking residential property sales in England and Wales.

The datasets used in our study are often provided in a compressed form. Most datasets are provided as zip files (BABSTD, BSAPUF, NAPTAN, NPPES, PLANTS, QCLCD ,WQP) and some as gzipped files (CHR22, STAR, STORM); few are provided uncompressed (LONDON, POLICE, PPD). There is no clear correlation between the size of the input dataset and the applied compressor. We are not aware of how the suppliers of these datasets have chosen their compression method, but we think that often the choice is made rather arbitrary. In our evaluation below, we will show that the choice of a compression technique has a critical effect not only on the compression ratio, but also on the compression speed.

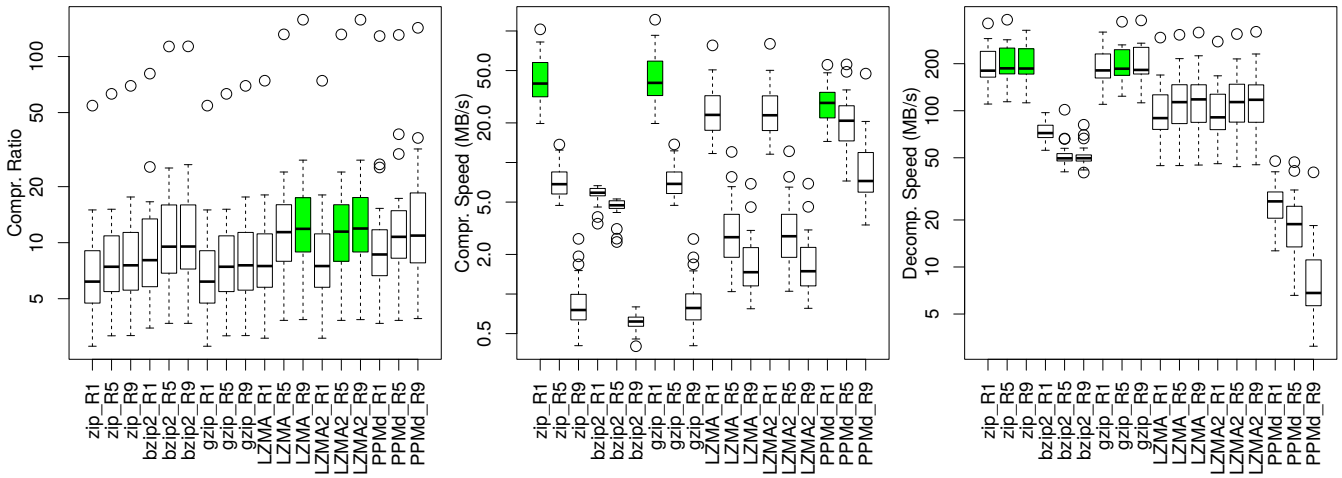


Figure 2: Boxplot for compression ratio (left), compression speed (center), and decompression speed (right) for all 18 compression methods. The Top 3 methods, i.e. the methods with highest median values, are marked in green color.

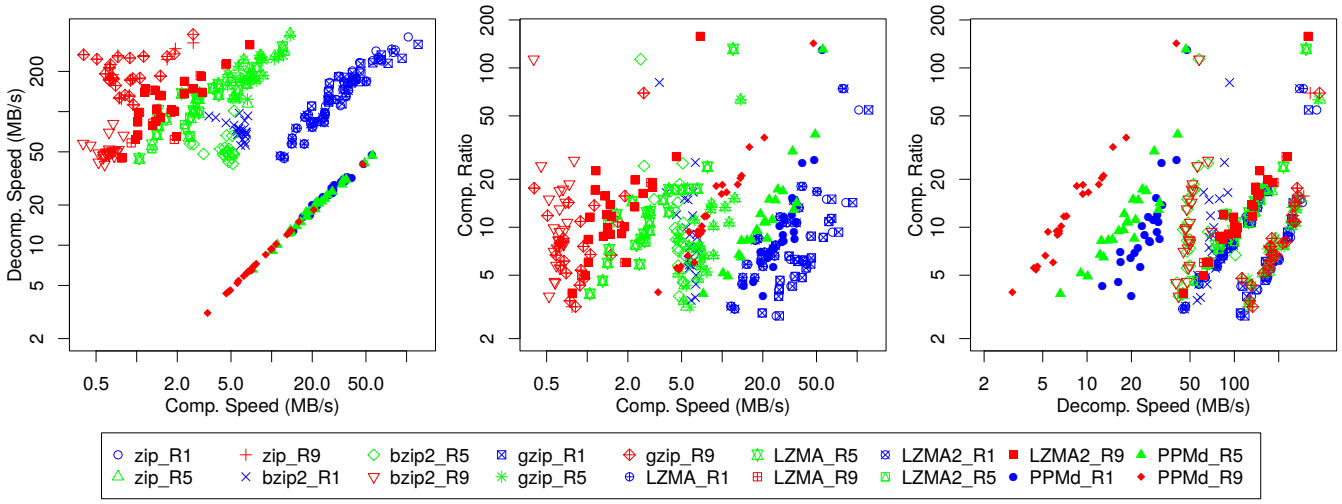


Figure 3: Pair-wise correlation between compression speed, decompression speed, and compression ratio. Compression levels are marked using distinct colors (1=blue, 5=green, 9=red).

3.3. Compression Problem 1: Row-by-Row Compression

We initially evaluated the compression and decompression of the 24 datasets against each of the six compressors with different compression levels (1/5/9). Below, a compressor together with a compression level is referred to as one compression method. The method is denoted as X_RY, where X is the standard compressor and Y is the compression level. For instance, PPMd_R1 is the compressor PPMd with compression level 1. Results for compression ratio and compression speed are shown in Figure 6, in the Appendix. The compression ratio of the best compression method is up to three times higher than for the worst method (for instance, the datasets CHR22 and USPTO). Moreover, the compression speed differs often in order of magnitudes (for instance, between 0.4 MB/s and more than 120 MB/s for CHR22). This supports our claim that the choice of a good compression method is indeed important. On the other hand, our results show that compression methods are often dominated by a few candidates. The dataset BABSTD, for instance, can be compressed well and fast at the same time, using the compression method PPMd_R1. Below we will analyze which compression method is best when considering each of the three criteria (compression ratio, compression speed, and decompression speed), respectively.

First, we report the results for considering a single compression criterion only. The results are shown in Figure 2. The best median compression ratio is obtained by LZMA_R5, LZMA2_R5, and LZMA2_R9. The methods with the

highest median compression speed are zip_R1, gzip_R1, and PPMd_R1. Decompression is fastest for zip_R5, zip_R9, and gzip_R5. It is remarkable, that methods based on PPMd have significantly lower decompression speeds than the other methods in our study. The decompression speed is rather independent of the compressed data for methods based on bzip2. We further analyze the correlation between compression criteria for varying compression levels in Figure 3. It can be seen that the compression level mostly determines the compression speed (left and middle subfigure of Figure 3). Increasing the compression level from 1 to 5 (or from 5 to 9), decreases the compression speed by approx. one order of magnitude. Furthermore, data compressed at high compression speeds can often also be decompressed faster than other data (particularly for compression level 1 and 5).

Next, we analyze whether combinations of compression criteria make the choice of a compressor harder in practice. As mentioned above, compression methods are sometimes dominated by a few candidates regarding multiple criteria (see example for dataset BABSTD above). Therefore, we have computed the Pareto fronts for each dataset and counted how often each compression method is part of the Pareto front for each subset of compression criteria. The results are shown in Figure 7, in the Appendix. For each dataset and compression method, a cross indicates that a compression method belongs to the Pareto front for this dataset. We counted the number of datasets for each compression method, for which it is part of the Pareto front and mark it in the last line of each of the four subtables of Figure 7, respectively. It can be seen that for two-element subsets of the three compression criteria, most of the compressors only occur infrequently in the Pareto fronts. We discuss the details below.

When compression speed and compression ratio are considered (upper left table of Figure 7), four compression methods frequently belong to the Pareto front: gzip_R1, PPMd_1, PPMd_5, and PPMd_9. The method gzip_R1 often provides fast compression, while the methods based on PPMd provide fast compression with high compression ratios, depending on the compression level. For compression speed and decompression speed (lower left table of Figure 7), only two compression methods occur frequently in the Pareto fronts: zip_R1 and gzip_R1. Combining compression ratio and decompression speed (upper right table of Figure 7), again two compressors are dominating: gzip_R9 and LZMA_R9. When all three compression criteria are considered together (lower right table of Figure 7), 15 out of 18 compression methods are frequently occurring in the Pareto fronts. We conclude that if only two out of the three compression criteria are considered, selecting a good compression method is not so difficult, i.e. one does not need to try all combinations of compressors and compression levels. Only once all criteria are important for choosing the best compression method, one needs to exhaustively compress a dataset against all compression methods.

3.4. Column-wise compression

In this subsection, we analyze the compressibility of datasets with column partitioning. First, we look at the partitioning which builds one cluster for each column in the input. This partitioning corresponds to the original idea of column-wise traversal (see discussion above). In Figure 4, the compression ratio of column-wise compression is plotted against row-wise compression (as reported in the previous section). In order to make the results comparable to row-by-row compression, we have used *the same* compressor for each column. The results for exact Compression Problem 2 are presented in the next subsection. For many datasets, the compression ratio improves significantly (entries above the diagonal line). Column-by-column compression, however, also reduces the compression ratio in few cases. Most notably, the dataset CHR22 is compressed much better using row-by-row compression than column-by-column compression. This can be explained as follows. The dataset contains a large number of highly similar columns, each of them indicating the variant information for a particular human. Each line is dominated by this variant information, containing entries such as 1|0 and 0|1. Compressing these columns separately, the compression model has to learn the distribution of symbols and items from the beginning for each column again (and thus compresses inefficiently for large parts of the columns). During row-wise compression, on the other hand, the compression model improved stepwise (by dominating variant information) and thus better compression ratios are achieved.

3.5. Compression Problem 2 and Compression Problem 3

For each dataset, we compare the compression ratio as in the downloaded version to our three novel techniques in Table 2. None of the datasets analyzed in this study was provided optimally compressed (or close to it). Custom compression methods and traversal strategies increase the compression ratio by a factor between 2 and 10. With two exceptions, column-wise compression and clustered-columns compression yield the highest compression ratios. Compression speed and decompression speed for column partitioning compression are quite similar to the other compression techniques, if the file is partitioned offline (see Figure 5 for time spent on partitioning the datasets).

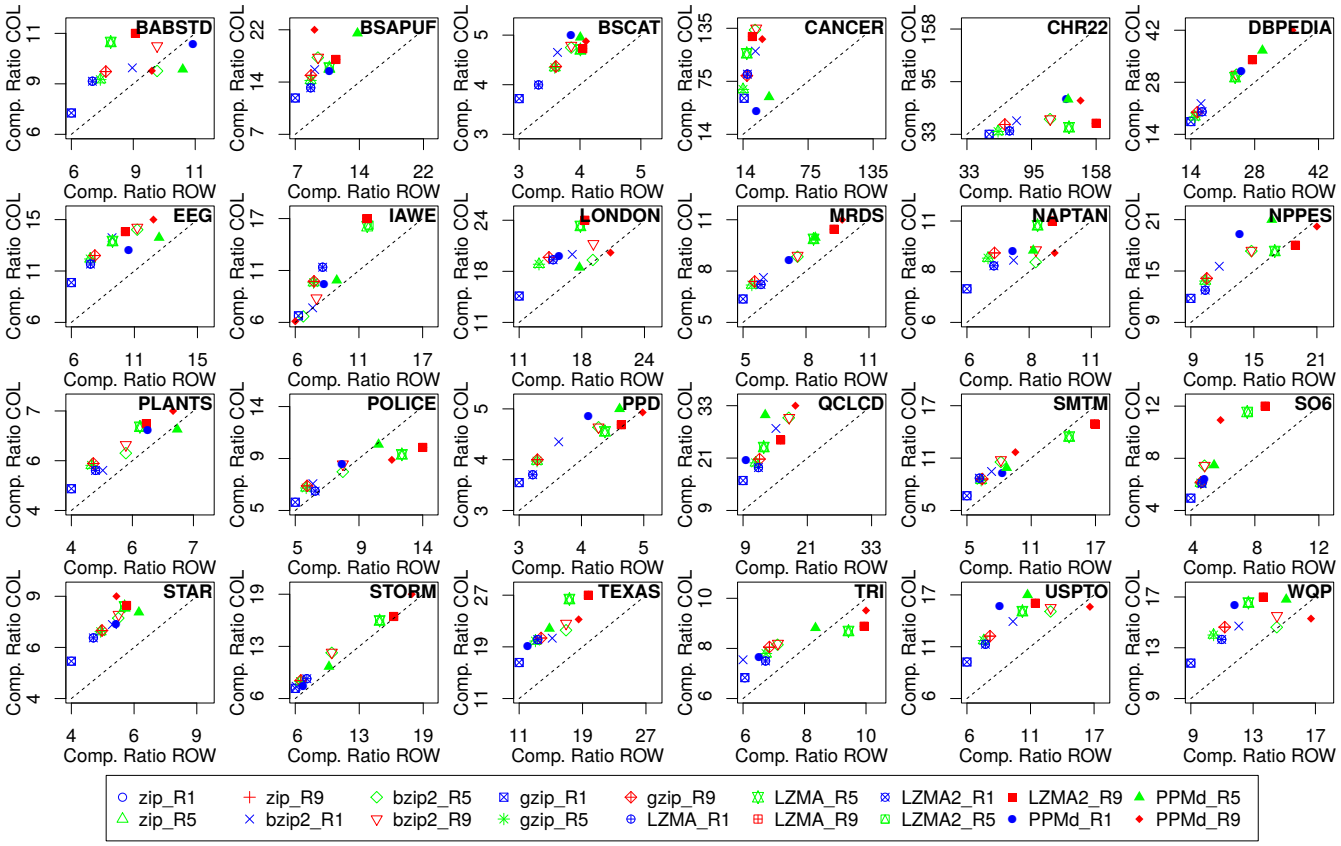


Figure 4: Comparison of compression ratio for row-wise compression (ROW) and column-wise compression (COL).

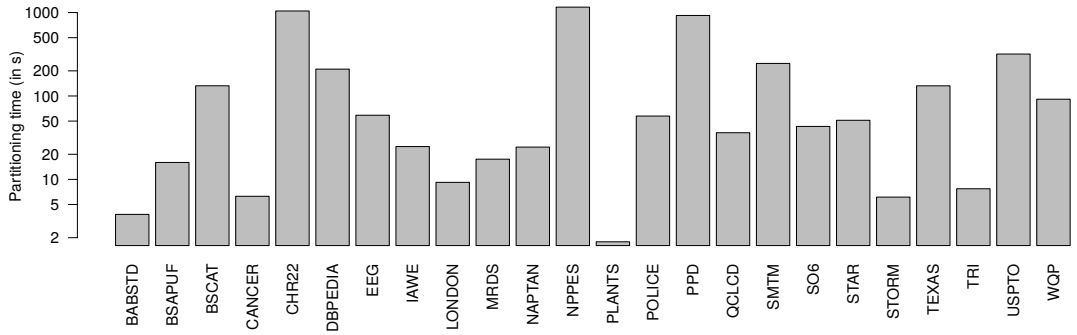


Figure 5: Time (in s) spent on partitioning the CSV file for each dataset.

As an example, we describe the effect of the partitioning for the dataset SO6 (the compression ratio for partitioning-based compression is twice as high as for row-by-row compression). SO6 encodes information about aircraft trajectories in 20 columns. The columns and the computed clusters are shown in Table 3. It can be seen that, with a few exceptions, columns which model similar data (for instance, latitude/longitude information) are clustered together.

4. Discussion and Future Research Directions

The experimental results of our study show that the compression of open data deserves further research. Even with simple modifications, such as changing the traversal strategy or changing the compression method, compression results change significantly; the compression ratio increases up to an order of magnitude, compared to the files as

provided for download by the data suppliers. The problem is that one, in general, wants to avoid trying all possible traversal strategies exhaustively, because of long running times. However, it should be noted that a column-wise traversal strategy is always better than the standard line-by-line processing. There are several directions for future work:

1. **Exploring further traversal strategies**: Our empirical analysis shows that the traversal strategy has a significant impact on the compression ratio. However, there is no single best traversal strategy for all datasets; a good strategy has to be selected based on the to-be-compressed dataset. This poses the challenging problem to design and evaluate traversal strategies at running time. While in this paper we have only looked at traversal strategies induced by column partitionings, other strategies might be developed to capture horizontal functional dependencies between columns. For instance, we have found several datasets where a column represents the differences between two other columns (length of segments defined by latitude/longitude pairs). Another example is the non-normalized storage of key-value pairs. Given an automatic way to extract such relationships between columns, the compression ratios can be further increased.
2. **Automatically selecting a compressor for a column/partitioning**: Given a column (or a cluster of columns) a good compressor needs to be selected, based on a set of compression criteria. We have already shown that for mixed inputs, the set of candidate compression methods is relatively small, if only two out of three compression criteria are relevant. Moreover, we have found that selecting a separate compressor for different columns/partitionings often increases the compression further. The selection process should avoid compressing a whole column, thus sampling-based estimation of compression ratios is another challenge when compressing relational files in a non-standard way. Similarly, the development of custom column compressors, which detect the datatype of a column (e.g., number, date, etc.) and statistical properties (e.g. monotonically increasing, upper/lower bounds) is an interesting direction for future research. For instance, our experiments show that columns with IDs (increasing numbers) cannot be compressed well with standard compressors. Custom compressors, on the other hand, could identify such columns and compress them quite well, and exploit the information using, e.g., delta encoding.
3. **Towards a multi-platform compression tool for open data in CSV files**: With the increasing availability of open data published as CSV files, we believe that the development of a specialized multi-platform, open source compression tool for these kinds of files could help to address storage challenges. Such a tool could even extend the idea of compression levels, which roughly guide the degree of compression, based on user-defined compression criteria. The solution to find a good compression method for a certain dataset is then an optimization problem.
4. **Beyond plain compression**: There are several challenges ahead which go beyond plain compression of files. First, it would be interesting to provide random access to archives (operations should include the extraction of a row or a column). Such an operation should not need to decompress all the data unrelated to the result. On top of such a framework, searching capabilities could be implemented. Such a scenario becomes even more interesting, if the compressed dataset can be kept in main memory, with the option of random access. For instance, it makes a huge difference, whether the CHR22 needs 11.2 GB uncompressed or can be kept in main memory with fewer than 100 MB. Furthermore, data often comes in several versions or is updated regularly. It would be beneficial, if a compressor (compression format) could support the notion of snapshots and difference descriptions natively. In this way, when a dataset is updated, only the difference between the two versions needs to be transferred again.

In conclusion, we showed that the traversal strategy for compressing open relational data has a substantial effect on the compression ratio. We believe that efficient compression of such datasets is a neglected topic in the literature. Our results can have significant impact on the proliferation and usability of large open datasets.

References

[1] 1000 Genomes Project Consortium, "A map of human genome variation from population-scale sequencing," *Nature*, vol. 467, no. 7319, pp. 1061–1073, Oct. 2010. [Online]. Available: http://dx.doi.org/10.1038/nature09534

[2] D. J. Abadi, P. A. Boncz, and S. Harizopoulos, "Column-oriented database systems," *Proc. VLDB Endow.*, vol. 2, no. 2, pp. 1664–1665, Aug. 2009. [Online]. Available: http://dx.doi.org/10.14778/1687553.1687625

[3] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, "Dbpedia: A nucleus for a web of open data," in *Proceedings of the 6th International The Semantic Web and 2Nd Asian Conference on Asian Semantic Web Conference*, ser. ISWC'07/ASWC'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 722–735. [Online]. Available: http://dl.acm.org/citation.cfm?id=1785162.1785216

[4] D. S. Batory, "On Searching Transposed Files," *ACM Trans. Database Syst.*, vol. 4, no. 4, pp. 531–544, Dec. 1979. [Online]. Available: http://doi.acm.org/10.1145/320107.320125

[5] S. Bechhofer, I. Buchan, D. De Roure, P. Missier, J. Ainsworth, J. Bhagat, P. Couch, D. Cruickshank, M. Delderfield, I. Dunlop *et al.*, "Why linked data is not enough for scientists," *Future Generation Computer Systems*, vol. 29, no. 2, pp. 599–611, 2013.

[6] E. T. Bell, "The iterated exponential integers," *Annals of Mathematics*, vol. 39, no. 3, pp. pp. 539–557, 1938. [Online]. Available: http://www.jstor.org/stable/1968633

[7] C. Bizer, "The emerging web of linked data," *Intelligent Systems, IEEE*, vol. 24, no. 5, pp. 87–92, Sept 2009.

[8] P. A. Boncz and M. L. Kersten, "MIL Primitives for Querying a Fragmented World," *The VLDB Journal*, vol. 8, no. 2, pp. 101–119, Oct. 1999. [Online]. Available: http://dx.doi.org/10.1007/s007780050076

[9] K. Braunschweig, J. Eberius, M. Thiele, and W. Lehner, "The state of open data - limits of current open data platforms," in *Proceedings of the 21st World Wide Web Conference 2012, Web Science Track at WWW'12, Lyon, France, April 16-20, 2012*. ACM, 2012.

[10] A. L. Buchsbaum, D. F. Caldwell, K. W. Church, G. S. Fowler, and S. Muthukrishnan, "Engineering the compression of massive tables: an experimental approach," in *Symposium on Discrete Algorithms: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, vol. 9, no. 11, 2000, pp. 175–184.

[11] M. Burrows and D. J. Wheeler, "A block-sorting lossless data compression algorithm," 1994.

[12] S. Deorowicz and S. Grabowski, "Compression of DNA sequence reads in FASTQ format," *Bioinformatics*, vol. 27, no. 6, pp. 860–862, 2011. [Online]. Available: http://bioinformatics.oxfordjournals.org/content/27/6/860.abstract

[13] ——, "Data compression for sequencing data," *Algorithms for Molecular Biology*, vol. 8, p. 25, 2013. [Online]. Available: http://dx.doi.org/10.1186/1748-7188-8-25

[14] European Commission, *Open Access to scientific information*, 2015. [Online]. Available: https://ec.europa.eu/digital-agenda/en/open-access-scientific-information

[15] M. Gaeta, V. Loia, and S. Tomasiello, "Cubic b-spline fuzzy transforms for an efficient and secure compression in wireless sensor networks," *Information Sciences*, vol. 339, pp. 19 – 30, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025515009159

[16] R. Giancarlo, S. E. Rombo, and F. Utro, "Compressive biological sequence analysis and archival in the era of high-throughput sequencing technologies," *Briefings in Bioinformatics*, vol. 15, no. 3, pp. 390–406, 2014. [Online]. Available: http://dx.doi.org/10.1093/bib/bbt088

[17] K. Goswami, J.-H. Lee, and B.-G. Kim, "Fast algorithm for the high efficiency video coding (hevc) encoder using texture analysis," *Information Sciences*, vol. 364-365, pp. 72 – 90, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025516303425

[18] M. Haklay and P. Weber, "OpenStreetMap: User-Generated Street Maps," *IEEE Pervasive Computing*, vol. 7, no. 4, pp. 12–18, Oct. 2008. [Online]. Available: http://dx.doi.org/10.1109/MPRV.2008.80

[19] J. Hendler, J. Holm, C. Musialek, and G. Thomas, "US government linked open data: semantic.data.gov," *IEEE Intelligent Systems*, vol. 27, no. 3, pp. 0025–31, 2012.

[20] M. Janssen, Y. Charalabidis, and A. Zuiderwijk, "Benefits, Adoption Barriers and Myths of Open Data and Open Government," *Information Systems Management*, vol. 29, no. 4, pp. 258–268, 2012. [Online]. Available: http://dx.doi.org/10.1080/10580530.2012.716740

[21] S. Khoshafian, G. P. Copeland, T. Jagodis, H. Boral, and P. Valduriez, "A Query Processing Strategy for the Decomposed Storage Model," in *Proceedings of the Third International Conference on Data Engineering*. Washington, DC, USA: IEEE Computer Society, 1987, pp. 636–643. [Online]. Available: http://dl.acm.org/citation.cfm?id=645472.655555

[22] K. R. Koedinger, R. S. Baker, K. Cunningham, A. Skogsholm, B. Leber, and J. Stamper, "A data repository for the EDM community: The PSLC DataShop," *Handbook of educational data mining*, vol. 43, 2010.

[23] D. Lee, J. Oh, W.-K. Loh, and H. Yu, "Geovideoindex: Indexing for georeferenced videos," *Information Sciences*, vol. 374, pp. 210 – 223, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S002002551630768X

[24] L. Mavridis, N. Nath, and J. Mitchell, "PFClust: a novel parameter free clustering algorithm," *BMC Bioinformatics*, vol. 14, no. 1, p. 213, 2013. [Online]. Available: http://www.biomedcentral.com/1471-2105/14/213

[25] J. C. Molloy, "The Open Knowledge Foundation: Open Data Means Better Science," *PLoS Biol*, vol. 9, no. 12, p. e1001195, 12 2011. [Online]. Available: http://dx.doi.org/10.1371%2Fjournal.pbio.1001195

[26] K. G. Morse Jr, "Compression tools compared," *Linux Journal*, vol. 2005, no. 137, p. 3, 2005.

[27] B. A. Nosek, J. R. Spies, and M. Motyl, "Scientific utopia II. Restructuring incentives and practices to promote truth over publishability," *Perspectives on Psychological Science*, vol. 7, no. 6, pp. 615–631, 2012.

[28] D. Partha and P. A. David, "Toward a new economics of science," *Research policy*, vol. 23, no. 5, pp. 487–521, 1994.

[29] H. A. Piwowar and T. J. Vision, "Data reuse and the open data citation advantage," *PeerJ*, vol. 1, p. e175, 2013.

[30] S.-A. Sansone, P. Rocca-Serra, D. Field, E. Maguire, C. Taylor, O. Hofmann, H. Fang, S. Neumann, W. Tong, L. Amaral-Zettler *et al.*, "Toward interoperable bioscience data," *Nature genetics*, vol. 44, no. 2, pp. 121–126, 2012.

[31] M. Schäfer, M. Strohmeier, V. Lenders, I. Martinovic, and M. Wilhelm, "Bringing Up OpenSky: A Large-scale ADS-B Sensor Network for Research," in *Proceedings of the 13th International Symposium on Information Processing in Sensor Networks*, ser. IPSN '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 83–94. [Online]. Available: http://dl.acm.org/citation.cfm?id=2602339.2602350

[32] P. N. Schofield, T. Bubela, T. Weaver, L. Portilla, S. D. Brown, J. M. Hancock, D. Einhorn, G. Tocchini-Valentini, M. H. de Angelis, and N. Rosenthal, "Post-publication sharing of data and tools," *Nature*, vol. 461, no. 7261, pp. 171–173, 2009.

[33] H.-Y. Shum, S. B. Kang, and S.-C. Chan, "Survey of image-based representations and compression techniques," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 13, no. 11, pp. 1020–1037, 2003.

[34] J. Son, H. Ryu, S. Yi, and Y. D. Chung, "Ssfile: A novel column-store for efficient data analysis in hadoop-based distributed systems," *Information Sciences*, vol. 316, pp. 68 – 86, 2015, nature-Inspired Algorithms for Large Scale Global Optimization. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025515002820

[35] L. Tan and M. Wu, "Data reduction in wireless sensor networks: A hierarchical lms prediction approach," *IEEE Sensors Journal*, vol. 16, no. 6, pp. 1708–1715, 2016.

[36] T. J. Vision, "Open data and the social contract of scientific publishing," *BioScience*, vol. 60, no. 5, pp. 330–331, 2010. [Online]. Available: http://bioscience.oxfordjournals.org/content/60/5/330.short

[37] S. Wandelt, M. Bux, and U. Leser, "Trends in genome compression," *Current Bioinformatics*, vol. 9, no. 3, pp. 315–326, 2014. [Online]. Available: http://dx.doi.org/10.2174/15748936096661405160101 43

[38] S. Wandelt and X. Sun, "Efficient compression of 4d-trajectory data in air traffic management," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 844–853, 2015. [Online]. Available: http://dx.doi.org/10.1109/TITS.2014.2345055

[39] M. Wu, L. Tan, and N. Xiong, "Data prediction, compression, and recovery in clustered wireless sensor networks for environmental monitoring applications," *Information Sciences*, vol. 329, pp. 800 – 818, 2016, special issue on Discovery Science. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025515007124

5. Appendix

Dataset	Size (MB)	|Cols|	|Rows|	Short description of dataset
BABSTD	20.6	11	171,793	Bay Area bike share's trip data, trip data for 2014-08, provided by BAYAREA BikeShare
BSAPUF	93.2	11	2,801,661	Information from Medicare Carrier claims, provided by Centers for Medicare and Medicaid Services
BSCAT	353.4	66	700,886	GALEX catalogs of unique UV sources, provided by Johns Hopkins University.
CANCER	84.1	6	1,329,017	Cancer tumor data: Staining profiles for proteins, provided by The Human Protein Atlas.
CHR22	11,223.1	2,513	1,103,799	Human genome variant calls, provided by 1000 Genome Project
DBPEDIA	9,182.4	514	1,649,650	DBpedia (Release 2014) data for concept Person in tabular form, provided by University Mannheim.
EEG	359.1	23	1,552,808	Map of reneable energy sources,provided by EnergyMap.
IAWE	118.3	9	2,671,802	Energy monitoring and consumption data for a a home in New Delhi India, provided by iawe.github.io.
LONDON	77.6	25	312,010	A complete list of London postcode districts, provided by doogal.co.uk.
MRDS	126.5	44	305,137	Mineral Resources Data System, provided by U.S. Geological Survey.
NAPTAN	131.2	43	422,025	Transportations stops in Great Britain, provided by UK Department For Transport
NPPES	5,474.5	329	4,517,595	Health care provider data, provided by Centers for Medicare and Medicaid Services
PLANTS	7.3	5	90,611	Standardized information about the plants, provided by United States Department of Agriculture
POLICE	300.8	19	1,227,269	Seattle Police Department 911 incident responses
PPD	3,411.2	15	19,794,598	Price Paid Data: tracking of residential property sales in England and Wales, provided by gov.uk.
QCLCD	239.3	44	1,843,814	Quality controlled local climatological data, provided by National Climatic Data Center
SMTM	1,230.4	11	13,924,548	Show me the money: snapshot of the UK peer-to-peer lending market, provided by open data institute.
SO6	183.8	20	1,289,844	Air traffic routes over Europe for a single day, provded by Eurocontrol Demand Data Repository
STAR	194.1	12	2,173,762	Sample data from the high-energy experiment STAR, provided by The STAR experiment
STORM	56.9	51	59,346	Records for the official NOAA Storm Data publication,provided by National Climatic Data Center
TEXAS	770.8	249	1,158,743	2010 Census - Texas Redistricting Data - P.L. 94-171, provided by Texas State Data Center.
TRI	30.1	99	79,395	Toxic Release Inventory information, provided by United States Environmental Protection Agency
USPTO	1,844.1	79	7,381,183	Trademark Case Files Dataset, provided by United States Patent and Trademark Office.
WQP	722.2	35	2,268,093	Water Quality Portal data for the United States, provided by National Water Quality Monitoring Council

Table 1: List of datasets used in this study. In total, 24 datasets have been used, covering a wide range of sizes (from a few MB to a dozen of GB), number of columns (4–2,513), and number of rows (from a few thousands to several millions).

	Compression ratio			
Dataset	**Original**	**Compression Problem 1: Row-by-Row**	**Compression Problem 2: Column-by-Column**	**Compression Problem 3: Column Partitioning**
BABSTD	7.535	11.176	**14.453**	14.073
BSAPUF	7.615	14.203	**22.869**	20.245
BSCAT	1.000	3.910	**4.899**	4.319
CANCER	15.232	38.227	166.383	**198.962**
CHR22	52.334	157.704	77.043	**169.848**
DBPEDIA	15.216	36.529	**43.027**	**43.027**
EEG	7.071	12.112	**16.098**	15.331
IAWE	6.447	12.163	**17.927**	16.695
LONDON	1.000	20.357	26.378	**27.407**
MRDS	5.101	9.493	**11.481**	11.432
NAPTAN	4.703	9.150	**11.341**	11.293
NPPES	10.526	21.090	**21.417**	21.401
PLANTS	1.000	6.726	**7.700**	7.370
POLICE	1.000	**13.839**	12.882	13.610
PPD	1.000	5.421	**5.713**	5.713
QCLCD	9.604	18.456	**33.643**	28.605
SMTM	5.114	**17.143**	16.153	15.748
SO6	8.374	8.379	12.598	**15.891**
STAR	4.535	6.482	**9.365**	8.751
STORM	6.380	18.150	20.068	**20.926**
TEXAS	11.854	19.799	29.249	**32.792**
TRI	1.000	10.130	10.293	**10.304**
USPTO	7.637	16.171	19.548	**19.928**
WQP	1.000	16.566	**18.704**	18.182

Table 2: Comparison of compression ratios. Original represents the compression ratio for the file provided for download (if a file was provided uncompressed, the ratio is set to 1).

Cluster	Column	Column name	Sample 1	Sample 2	Sample 3	AlphabetSize	AlphabetEntropy	ItemEntropy	RatioNonempty	Uniqueness
Cluster 1	0	SegmentID	ENXP_ENXV	$ChiG_$CHiH	AVANT_*5MID	66	5.23485	18.6546	0.999999	0.715738
Cluster 2	1	Origin	ENXP	LFPO	LELC	26	4.18919	7.74494	0.999999	0.000832659
	2	Destination	ENXV	LFML	EGNX	26	4.19549	7.74947	0.999999	0.000828007
Cluster 3	3	Aircraft type	S92	A319	B738	35	3.76134	4.69523	0.999999	0.000192271
Cluster 4	4	Time begin	174200	201317	110420	10	3.13406	16.1104	0.999999	0.0668724
	5	Time end	174221	201337	110435	10	3.14029	16.1577	0.999999	0.0668833
	16	FlightID	1500095102	185473640	185457777	10	3.08615	14.59	0.999999	0.0205513
Cluster 5	6	Flight level begin	1	310	245	10	2.93224	6.31483	0.999999	0.000367486
	7	Flight level end	1	300	240	10	2.93226	6.31502	0.999999	0.000367486
	17	Sequence	1	19	44	10	3.17758	6.12314	0.999999	0.000110091
Cluster 6	8	Status	0	1	2	3	1.58219	1.58219	0.999999	2.32586E-06
Cluster 7	9	Callsign	BHL176A	AF054ZT	RYR81UR	36	4.96429	14.4823	0.999999	0.0194148
Cluster 8	10	Date begin	150412	150412	150412	6	2.27268	0.32961	0.999999	2.32586E-06
	11	Date end	150412	150412	150412	6	2.27351	0.328515	0.999999	2.32586E-06
Cluster 9	12	Latitude begin	3484.700000	2720.483333	3049.200000	12	3.21023	14.9	0.999999	0.0862942
	13	Longitude begin	114.616667	222.616667	-56.300000	12	3.28238	15.1548	0.999999	0.113918
	14	Latitude end	3484.633333	2718.466667	3050.716667	12	3.21025	14.9003	0.999999	0.086298
	15	Longitude end	113.333333	223.583333	-54.283333	12	3.28236	15.1552	0.999999	0.11392
Cluster 10	18	Segment length	0.681852	2.128114	1.980554	11	3.44619	16.9863	0.999999	0.292385
Cluster 11	19	Segment parity	0			1	NaN	NaN	0.999999	7.75288E-07

Table 3: Example of the partitioning for the dataset SO6.

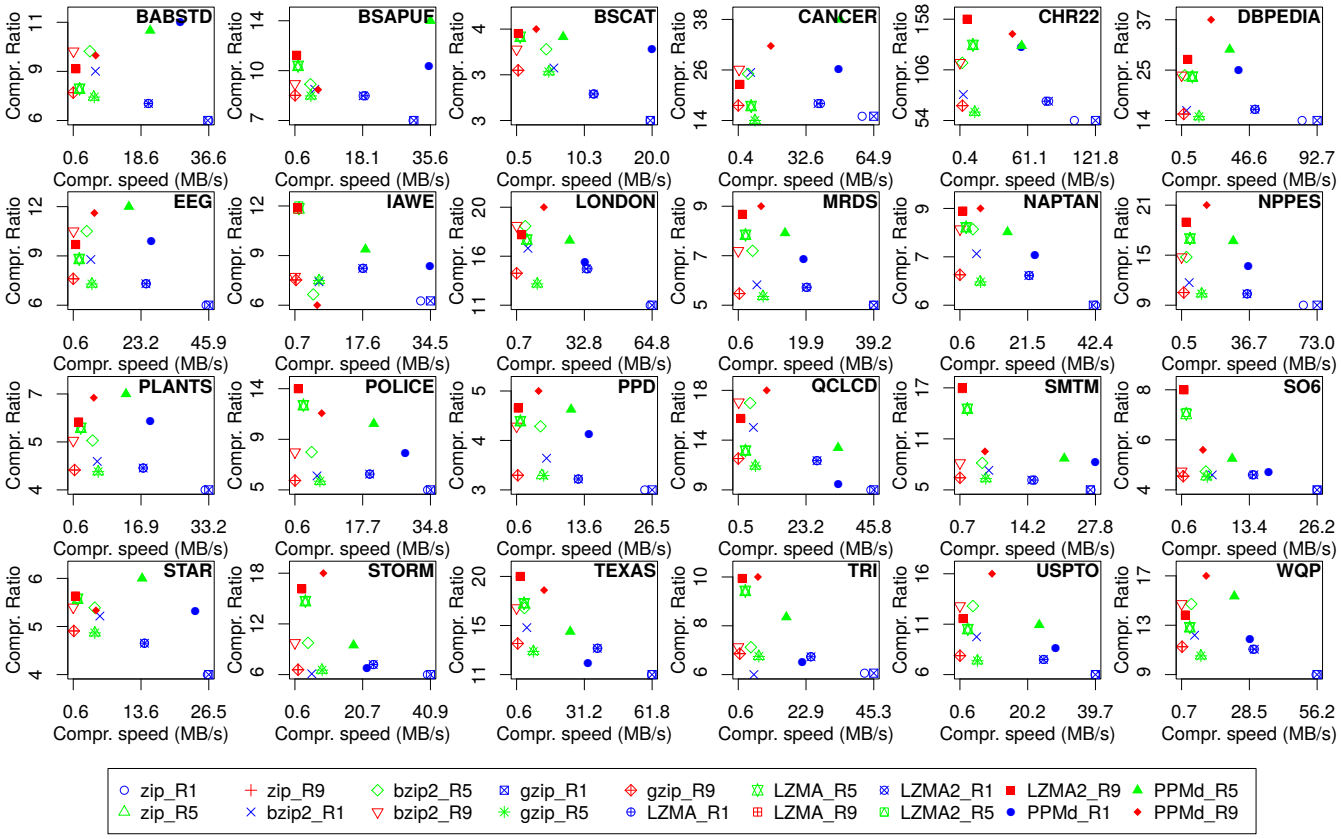


Figure 6: Results for the compression with standard compressor implementations: Compression ratio plotted against compression speed.

Compression speed and compression ratio

Dataset	zip_R1	zip_R5	zip_R9	bzip2_R1	bzip2_R5	bzip2_R9	gzip_R1	gzip_R5	gzip_R9	LZMA_R1	LZMA_R5	LZMA_R9	LZMA2_R1	LZMA2_R5	LZMA2_R9	PPMd_R1	PPMd_R5	PPMd_R9
BABSTD	x						x									x		
BSAPUF																	x	
BSCAT																x	x	x
CANCER							x										x	
CHR22							x			x		x	x		x		x	x
DBPEDIA							x			x						x	x	x
EEG							x									x	x	
IAWE							x				x			x		x	x	
LONDON							x			x						x	x	x
MRDS							x			x						x	x	x
NAPTAN	x						x									x	x	x
NPPES							x									x	x	x
PLANTS							x									x	x	
POLICE							x				x	x		x	x	x	x	x
PPD							x									x	x	x
QCLCD							x									x	x	x
SMTM														x	x	x	x	x
SO6	x											x		x	x	x	x	x
STAR							x									x	x	
STORM							x			x							x	x
TEXAS							x			x		x			x		x	x
TRI							x			x			x				x	x
USPTO							x									x	x	x
WQP							x			x			x			x	x	x
	3	0	0	0	0	0	20	0	0	8	2	4	3	4	5	18	23	17

Decompression speed and compression ratio

Dataset	zip_R1	zip_R5	zip_R9	bzip2_R1	bzip2_R5	bzip2_R9	gzip_R1	gzip_R5	gzip_R9	LZMA_R1	LZMA_R5	LZMA_R9	LZMA2_R1	LZMA2_R5	LZMA2_R9	PPMd_R1	PPMd_R5	PPMd_R9
BABSTD			x			x			x			x				x		
BSAPUF		x	x					x	x					x	x		x	
BSCAT				x					x			x			x			x
CANCER	x		x	x		x			x	x		x			x		x	
CHR22		x							x			x			x			
DBPEDIA			x									x			x		x	x
EEG							x					x					x	
IAWE	x	x					x		x	x				x				
LONDON			x		x	x			x	x	x	x		x				x
MRDS									x			x			x			x
NAPTAN									x			x						x
NPPES								x	x			x						x
PLANTS								x	x		x	x			x	x	x	
POLICE			x						x			x						
PPD									x						x			x
QCLCD					x	x			x			x						x
SMTM			x						x						x			
SO6			x						x			x			x			
STAR				x					x		x	x		x			x	
STORM								x	x		x	x		x				x
TEXAS			x					x				x						
TRI								x	x			x		x				x
USPTO			x		x	x			x			x			x			x
WQP						x			x			x					x	x
	2	3	10	3	3	7	1	6	22	3	4	20	0	6	11	2	7	12

Compression speed and decompression speed

Dataset	zip_R1	zip_R5	zip_R9	bzip2_R1	bzip2_R5	bzip2_R9	gzip_R1	gzip_R5	gzip_R9	LZMA_R1	LZMA_R5	LZMA_R9	LZMA2_R1	LZMA2_R5	LZMA2_R9	PPMd_R1	PPMd_R5	PPMd_R9
BABSTD	x	x	x					x										
BSAPUF	x	x					x	x									x	
BSCAT	x	x					x		x							x		
CANCER	x						x											
CHR22	x	x					x	x										
DBPEDIA	x		x				x											
EEG	x	x					x		x									
IAWE	x						x											
LONDON		x	x				x											
MRDS	x	x					x	x										
NAPTAN	x	x	x					x	x									
NPPES	x	x					x	x										
PLANTS							x	x										
POLICE	x	x	x				x											
PPD							x	x	x									
QCLCD	x		x				x	x	x									
SMTM	x	x	x													x		
SO6	x		x					x										
STAR	x						x	x	x									
STORM							x	x										
TEXAS	x	x					x	x										
TRI	x						x	x										
USPTO	x		x				x	x										
WQP	x	x					x		x									
	20	13	9	0	0	0	20	15	7	0	0	0	0	0	0	2	1	0

Compression speed, compression ratio, and decompression speed

Dataset	zip_R1	zip_R5	zip_R9	bzip2_R1	bzip2_R5	bzip2_R9	gzip_R1	gzip_R5	gzip_R9	LZMA_R1	LZMA_R5	LZMA_R9	LZMA2_R1	LZMA2_R5	LZMA2_R9	PPMd_R1	PPMd_R5	PPMd_R9
BABSTD	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x			
BSAPUF	x	x	x	x	x		x	x	x	x	x	x		x	x		x	
BSCAT	x	x		x	x		x		x	x	x	x		x	x	x	x	x
CANCER	x		x	x		x	x		x	x		x			x		x	
CHR22	x	x	x				x	x	x	x	x	x	x	x	x	x	x	x
DBPEDIA	x	x	x				x			x	x	x		x	x	x	x	x
EEG	x	x		x	x	x	x	x	x	x	x	x	x	x			x	
IAWE	x	x					x	x	x	x	x			x		x	x	
LONDON		x	x	x	x	x	x	x	x	x	x	x		x	x	x	x	x
MRDS	x	x		x	x		x	x	x	x	x	x		x	x	x	x	x
NAPTAN	x	x	x	x	x		x	x	x	x	x	x			x	x	x	x
NPPES	x	x					x	x	x	x	x	x	x	x	x	x	x	x
PLANTS		x		x	x		x	x	x	x	x	x	x	x	x	x	x	
POLICE	x	x	x		x		x	x	x	x	x	x		x	x	x	x	x
PPD				x	x		x	x	x	x	x		x		x	x	x	x
QCLCD	x		x	x	x	x	x	x	x	x	x	x	x		x	x	x	x
SMTM	x	x	x	x	x				x	x			x	x	x	x	x	x
SO6	x		x	x	x			x	x	x	x	x	x	x	x	x	x	x
STAR	x			x	x		x	x	x	x	x	x	x	x			x	
STORM			x		x		x	x	x	x	x	x	x	x	x	x	x	x
TEXAS	x	x	x	x	x		x	x		x	x	x		x	x	x	x	x
TRI	x				x		x	x	x	x	x	x	x	x	x	x	x	x
USPTO	x		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
WQP	x	x		x	x	x	x		x	x	x	x	x	x		x	x	x
	20	16	14	17	19	7	22	19	22	24	21	21	14	19	20	19	23	17

Figure 7: Summary for all 24 datasets and compressors appearing on the Pareto front. A cross indicates that a compressor is included in the Pareto front. The bottom line shows the number of datasets each compressor is an element of the Pareto front. Compression methods which are in the Pareto front of more than $\frac{2}{3}$ of the datasets are highlighted. Only if all three properties are relevant, decision becomes complicated.