OPTIMIZING DATA COMPRESSION VIA DATA REORDERING STRATEGIES

QINXIN DU

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF ARTS

GRADUATE PROGRAM IN INFORMATION SYSTEMS AND
TECHNOLOGY
YORK UNIVERSITY
TORONTO, ONTARIO

APRIL 2024

© QINXIN DU, 2024

Abstract

To improve the efficiency and cost-effectiveness of handling large tabular datasets stored in databases, a range of data compression techniques are employed. Among these, dictionary-based compression methods such as Lz4, Gzip, and Zstandard are commonly utilized to decrease data size. However, while these traditional dictionary-based compression techniques can reduce data size to some degree, they are not able to identify the internal patterns within given datasets. Thus, there remains substantial potential for further data size reduction by identifying repetitive data patterns.

This thesis proposes two novel approaches to improve tabular data compression performance. Both methods involve data preprocessing using an advanced data encoding technique called locality-sensitive hashing (LSH). One approach utilizes clustering for data reordering, while the other employs a heuristic-based solver for the Travelling Salesman Problem (TSP). The data encoding process enables the identification of internal repetitive patterns within the original datasets. Records with similar features are grouped together and compressed into a much smaller size after reordering.

Furthermore, a novel table partitioning strategy based on the number of

distinct values in each column is designed to further improve the compression ratio of the entire table. Extensive experiments are then conducted on one synthetic dataset and three real datasets to evaluate the performance of the proposed algorithms by varying parameters of interest. The data encoding and reordering methods show significant efficiency improvements, resulting in reduced data size and substantially increased data compression ratios.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Professor Yu for his continuous guidance and mentorship throughout my research journey. His unwavering support has enabled me to develop a profound understanding of the research domain, providing me with valuable insights and inspiration for my thesis. I am grateful for the numerous opportunities he has bestowed upon me, enriching my graduate studies immeasurably.

I extend my heartfelt thanks to Professor An for her invaluable contributions, as she has been a source of inspiration and assistance, guiding me to effectively bridge theoretical knowledge with practical applications.

Special appreciation is extended to Dairuiz from IBM, whose professional insights have significantly influenced my thinking and contributed to the continual refinement of my research methodologies.

Finally, I would like to acknowledge my parents, whose understanding and support in all aspects of my life have been instrumental in ensuring that I persevered through this challenging yet profoundly meaningful phase of my academic journey. Their steadfast encouragement has been a constant source of strength.

Table of Contents

Abstract **ii**

Acknowledgements **iv**

Table of Contents **v**

List of Tables **viii**

List of Figures **ix**

1 Introduction **1**

1.1 Motivation 1

1.2 Methodology 4

1.3 Contributions and Outline 5

2 Literature Review **8**

2.1 Data Compression Algorithms 8

2.1.1 Lossy Compression 9

2.1.2 Dictionary-based Compression 9

2.1.3 Entropy-based Compression 12

2.1.4 Hybrid Compression 15
2.2 Data Reordering Techniques and Applications 17
2.2.1 Sorting Algorithms 17
2.2.2 Graph-based Algorithms 19
2.2.3 Other Algorithms 20
2.3 Data Reordering for Data Compression 20

3 Preliminaries and Problem Description 22
3.1 Preliminaries 22
3.1.1 Clustering 23
3.1.2 Traveling Salesman Optimization Algorithm (TSP) . . 23
3.1.3 Shingling & MinHashing 25
3.1.4 Run-Length Encoding (RLE) 27
3.1.5 Zstandard Compression (Zstd) 29
3.1.6 MessagePack Binary Serialization (MsgPack) 30
3.2 Problem Description 30

4 Methodologies 32
4.1 Data Representation Algorithm 32
4.2 Clustering-based Methods 35
4.3 TSP-based Methods 37
4.3.1 DVP-Partitioned Table Compression 41

5 Experiments 45
5.1 Experimental Settings 45
5.1.1 Environment 45

5.1.2 Datasets 46
5.1.3 Evaluation 46
5.2 LSH and K-prototype clustering 47
5.2.1 Sliding Window Size Q for LSH 48
5.2.2 Dimensional Level M for LSH 52
5.2.3 Number of Clusters N for K-prototype 54
5.3 TSP-based vs Clustering-based 55
5.4 DVP-Partitioned Table Compression with heuristic-based TSP solver 57

6 Conclusions **61**

Bibliography **64**

List of Tables

3.1 Notations used in the Traveling Salesman Problem objective function 24
3.2 Notations in minHashing Algorithm 26
3.3 Notations used in the Problem Description 31

5.1 Overview of Dataset Information 46
5.2 Compression Ratios and Execution Times for Bank Marketing Dataset using LSH combined with Heuristic-based TSP solver / K-prototype clustering. 56

List of Figures

2.1 Tree Diagram for Data Compression 8

2.2 Tree Diagram for Data Reordering Algorithms 17

4.1 Data Representation Flow Diagram 33

4.2 An LSH Representation Example for Column *Car Name* . . . 34

4.3 Clustering-based Table Reordering Diagram 36

4.4 Example of Square Matrix Formation 38

4.5 Example of Chunk-wise TSP Solution 40

4.6 DVP-Partitioned Table Compression Workflow Diagram . . . 43

5.1 Compression Ratio for `mileage` When Varying Q (M=2/3, N=10) 50

5.2 Compression Ratio for `mileage` When Varying M (Q=2/3, N=10) 53

5.3 Compression Ratio for `Customerkey` When Varying N (Q=3, M=5) 54

5.4 Overall Compression Ratio for Vehicle Dataset 58

5.5 Overall Compression Ratio for Census Income Dataset 59

5.6 Overall Compression Ratio for Bank Marketing Dataset 60

Chapter 1

Introduction

1.1 Motivation

Recent developments in information technology contributed to the creation of enormous quantities of data every second [24]. The need to access, store, and update data from diverse databases and applications continues to escalate. Simultaneously, there is an enormous flow of data over communication links due to the proliferation of computer communication networks [38]. As a result, more and more companies and researchers pay attention to the concept of data compression which aims to reduce redundancy in stored or communicated data, thus increasing effective data density [38].

Data compression can be achieved either by lossless (exact) or loss (inexact) compression [15]. Using a lossless compression technique, the original data can be recovered from the compressed data by decompression while maintaining the integrity of the original data. Text files and numeric files are usually

compressed using lossless techniques because programs that handle these types of files cannot withstand errors in the data [49]. In contrast, lossy compression techniques aim to acquire higher compression ratios by sacrificing some amount of data precision or quality. Lossy algorithms are best suited for static photos, videos, and audio data since either the brain compensates for the loss or humans are unable to notice it [15].

Even though data can be maintained in various formats, the most prevalent form of data in real-world applications is tabular data, which consists of samples (rows) with identical feature sets (columns) [69]. A regular table can include numerical data, ordinal data, categorical data, or a combination of the previously mentioned which requires being precisely retrieved from the compressed form [69]. Hence, the study concentrates on compressing tabular data using lossless compression algorithms.

In the database, tabular data compression can have a positive effect not only on the storage and transfer of data but also in the areas of database security, backup and recovery procedures, as well as enhancing database performance [62]. By compressing data before storage or transfer, significant reductions in communication and storage costs can be achieved [38]. Decreasing the amount of data to be transferred can improve the capacity of communication channels. For instance, halving the size of a file corresponds to doubling the capacity of the storage medium. Additionally, reducing data volume can alleviate pressure on computer systems' input and output channels, enabling data to be stored at higher, faster levels of the storage hierarchy.

Applying standard lossless compression techniques alone for tabular data

may not consistently obtain optimal results due to the presence of patterns, repetitions, and localized structures that these methods may struggle to capture efficiently. Therefore, researchers propose a sequential approach where rows are first reordered, followed by the application of standard compression methods.

Lemire et al. introduced a row-reordering approach derived from traveling salesman heuristics and integrated it with run-length encoding [39]. Another study describes an incremental heuristic for selecting columns to compress and determining row order for improved compression ratios in in-memory columnar databases using run-length encoding [67]. Both papers exclusively use run-length encoding after data reordering. However, each column in the table possesses unique characteristics, including distinct values within the same column. Run-length encoding tends to yield superior results when applied to columns with a limited number of unique values.

Moreover, specific repetitive patterns may remain undetectable through data reordering techniques, as they are limited in arranging items solely based on complete similarity. For example, to a human observer, the words "apply" and "apple" appear similar, but the data reordering technique might not effectively group these two words.

The challenge is how to employ appropriate data reordering and compression strategies for different columns in the table to achieve an optimal compression ratio. Therefore, this study focuses on preprocessing the data before reordering it to discover similar patterns among data items, gather them more effectively, and select appropriate data compression strategies specific to each column based on their inherent traits.

1.2 Methodology

We design two alternative data reordering methodologies and one innovative table partition strategy to optimize the compression ratio of tabular data. Both methods initiate with a data transformation stage intended to facilitate subsequent data reordering procedures. The concept of clustering and the traveling salesman problem(TSP) are employed to reorder data. The table partition strategy involves grouping the columns in a table based on the percentage of distinct values for each column and applying appropriate compression strategies for different groups.

The primary data reordering method is based on a clustering algorithm, where the raw tabular data undergoes encoding into high-dimensional vectors using locality-sensitive hashing (LSH). This encoding aims to mimic the sliding window characteristic observed in dictionary-based compression, fostering the identification of intricate features during the compression process. The clustering algorithm subsequently groups similar tuples into clusters, facilitating the reordering of the table based on these groupings. To be specific, the clusters are concatenated together to form a reordered table. This strategy's final stage involves applying dictionary-based compression methods, such as Lz4, Gzip, and Zstandard, to the reordered table.

The clustering process can be time-consuming, particularly when dealing with high-dimensional vectors. We introduce a second data reordering method derived from the Traveling Salesman Problem (TSP) to address this practical concern. Following the same LSH data encoding procedure as the clustering approach, the high-dimensional vectors are reordered chunk by chunk using a

heuristic-based solver for TSP. The compression ratios achieved by dictionary-based compression for the reordered columns using clustering and heuristic-based TSP solver are comparable.

An enhancement is introduced to improve the overall table compression ratio through a table partitioning strategy and the heuristic-based TSP solver. Considering the varying number of distinct values in individual columns of a table, optimizing strategies for column-wise compression becomes crucial. For columns with a limited number of distinct values, a direct application of TSP reordering and run-length encoding proves to be more efficient, mitigating the computational cost associated with data transformation. Conversely, columns with a larger number of distinct values benefit substantially from data representation and TSP reordering with dictionary-based compression techniques. Consequently, a threshold is defined as a column's percentage of unique values. This threshold determines whether the column undergoes LSH encoding before TSP reordering and is subsequently compressed using either run-length encoding or dictionary-based compression methods. This dynamic methodology contributes to a comprehensive solution applicable to the whole table compression, optimizing the compression process by tailoring it to the unique characteristics of each column in the tabular data, thereby enhancing the overall compression ratio.

1.3 Contributions and Outline

In summary, we make the following contributions:

- Inspired by the sliding window scheme inherent in dictionary-based compression, we employ Locality-Sensitive Hashing (LSH) to segment the data into fixed-size slices and transform these slices into high-dimensional vectors. This process enhances the computer's ability to discern similarities akin to human perception.

- The utilization of clustering for data reordering entails grouping similar items followed by rearranging the data based on these identified clusters. We opt for the k-prototype clustering method as it allows for a comprehensive treatment of both numerical and categorical data types.

- We rearrange data in chunks adopting the heuristic-based solver for TSP to handle large datasets in less time.

- We devise a data partitioning method to enhance the generality of our approach via the unique value percentage in each column. Different groups of columns adopt distinct compression strategies.

- We experimentally quantify the effectiveness of our proposal and present the results of a thorough evaluation testing the impact of different dataset parameters using both synthetic and real datasets demonstrating impressive performance improvement.

The rest of the thesis is organized as follows: In Chapter 2, the background for the research topic is reviewed. Additionally, a range of research work and approaches relevant to our topic are introduced. They are organized into distinct categories and some works are highlighted and given more details.

Chapter 3 provides essential background material and context that facilitate understanding the proposed methodology. Moreover, we define the problems which are aimed to be addressed in the thesis. In Chapter 4, we propose our algorithm for the mentioned issue and the methodology we employ towards it. In terms of Chapter 5, Experiments, all the experiments are implemented to assess the proposed procedures. The analysis of the experiments' results is presented with baselines. The thesis concludes with Chapter 6, which provides an in-depth review and summary of our work as well as directions for future studies.

Chapter 2

Literature Review

2.1 Data Compression Algorithms

This section provides relevant literature on data compression methods. Based on compression quality, two main categories are lossy and lossless compression. We emphasize lossless compression due to its close ties to our proposed approach. Figure 2.1 illustrates the section's structural layout for reference.

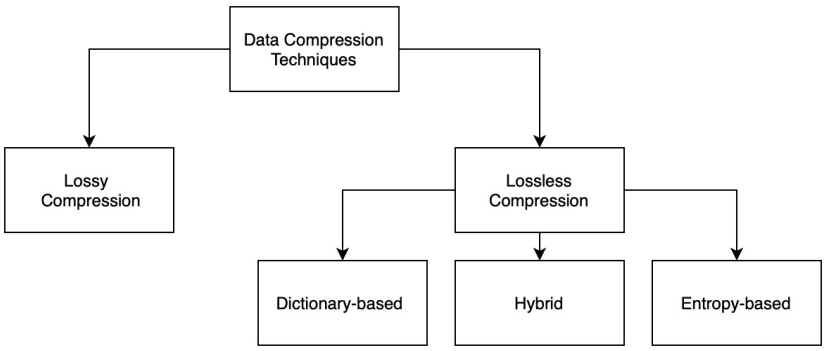


Figure 2.1: Tree Diagram for Data Compression

2.1.1 Lossy Compression

In general, lossy data compression aims to find efficient approximate representations for relatively large amounts of data [35]. In contrast to lossless compression, which avoids data loss during the decompression process, lossy compression allows for a certain amount of data loss. It is a widely used technique in various fields, including image, audio, and video compression, enabling smooth data transmission over networks [82].

The most common application for image lossy compression is JPEG which has been conceived to reduce the file size of natural, photographic-like true-color images as much as possible without affecting the quality of the images [20]. JPEG compression owns a lot of parameters to help users make trade-offs between compressed image size against reconstructed image quality. Similarly, for audio compression, lossy algorithms like MP3 employ perceptual coding schemes to reduce file sizes while maintaining audio quality [58]. In video compression, lossy algorithms used in MPEG standards aim to achieve greater compression than lossless schemes by sacrificing exact representation [27].

2.1.2 Dictionary-based Compression

Dictionary-based compression is a lossless data compression technique that relies on creating and utilizing a dictionary to reduce the size of data files. The idea is to identify repeating patterns or sequences within the data and replace them with references to entries in the dictionary. It can achieve significant compression ratios, particularly when dealing with data that contains repeated patterns or sequences. In 2.1.2, three well-known dictionary-based compression

algorithms are presented: Lz77, Lz4, and Zstandard.

2.1.2.1 Lz77

Lz77 was created by Ziv and Lempel [36]. It is a lossless compression algorithm based on dictionary-based compression that replaces recurring sequences in data streams with references to previous occurrences [25]. The mechanics of Lz77 involve a sliding window that retains recent data, endeavoring to locate the lengthiest matches within this window concerning the uncompressed data stream [68]. This characteristic endows Lz77 with the ability to scale space consumption proportionate to the number of modifications, making it an ideal choice for compressing repetitive data collections [46].

Lz77's adaptability has led to a wide range of applications in a variety of fields. It is a critical tool in computational biology for storing and retrieving highly repetitive sequence collections [46]. To supplement Lz77 in the context of string matching, a pseudo-optimal compressed matching algorithm has been developed [12]. Furthermore, Lz77 plays an important role in the compression of DNA sequences and genome data [44].

The Lz77 method was modified and optimized over time, leading to the release of the Lz78, an enhanced version [41]. The combination of Lz77 and Lz78 strengths has led to the proposal of hybrid compression algorithms. Furthermore, using convolutional neural networks, researchers have investigated the precise blind parameter estimation for the Lz77 algorithm [36].

2.1.2.2 Lz4

Lz4 is a fast, lossless compression algorithm that originated from the Lz77 compression algorithm. It is known for its high-speed compression and decompression capabilities. Various modifications and optimizations have been implemented to enhance the performance of the Lz4 algorithm for specific applications.

Liu et al. implement two hardware architectures of the modified Lz4 algorithm on an FPGA evaluation kit, achieving a higher compression ratio than all previous Lz algorithm designs [43]. Its application is an example that makes the Lz4 algorithm a hardware accelerator for compression. Haghighatkhah et al. work on continuous integration environments and propose a Test case prioritization approach facilitated by Lz4 compression [18]. In addition, Lz4 compression is integrated into cloud computing, providing flexibility and scalability for resources to lessen data transmission cost [63].

Compared to Lz77 compression, Lz4's speed, efficiency, and stability have made it a preferred choice for many applications, leading to the development of specialized algorithms and systems that leverage its core principles.

2.1.2.3 Zstandard

Zstandard is a modern and efficient compression algorithm known for its ability to achieve a good compression ratio while maintaining high speeds [10]. It offers tunable compression levels, allowing for significant trade-offs between compression speed and factors [34].

Zstandard has been integrated into nanopore sequencing data analysis

systems, providing a fast and efficient compression/decompression scheme alongside other established libraries like zlib and StreamVByte [14]. In the Internet of Things (IoT) for healthcare applications, Zstandard is being applied to reduce network overloading by compressing sensor data and minimizing the quantity of data broadcast by IoT devices [29]. Moreover, Chen et al. introduce the architecture of a new hardware compression kernel for Zstandard that allows the algorithm to be used for real-time compression of big data streams [7].

While both Zstandard and Lz4 algorithms offer high compression speeds, Lz4 is tailored for real-time applications, whereas Zstandard is versatile across a range of use cases, offering adaptability and advanced features at the expense of potentially higher memory usage in some scenarios.

2.1.3 Entropy-based Compression

Entropy-based compression, also known as entropy coding, is a type of lossless data compression technique that takes advantage of the statistical properties of the input data to achieve compression. It operates on the principle that highly predictable or repetitive data can be represented using fewer bits, reducing overall storage space. This is achieved by analyzing the probability distribution of symbols within the data and encoding them to minimize the average length of the encoded representation. We introduce two prominent examples of entropy-based compression techniques: Run-Length Encoding (RLE) and Huffman coding in 2.1.3.

2.1.3.1 Huffman Coding

Huffman coding was proposed by David A. Huffman in 1952 [83]. The algorithm works by assigning shorter codes to more frequently occurring symbols in the input data, and longer codes to less frequently occurring symbols. This results in a variable-length prefix code, where no code is a prefix of another code. The prefix property allows for efficient decoding of the compressed data.

The Huffman coding algorithm has been extensively studied and analyzed in the field of computer science. It has been shown to achieve optimal compression for a given set of symbol probabilities [80]. The algorithm has been applied in various domains, including image compression, logging data compression, and secure image compression [33, 71, 30]. It has also been used in the JPEG image compression standard [33].

Several variations and improvements to the Huffman coding algorithm have been proposed over the years. For example, canonical Huffman coding is a variant that assigns codewords in a canonical order, which simplifies the decoding process [13]. Adaptive Huffman coding is another variant that dynamically updates the code tree as symbols are encoded, allowing for better compression of data with changing statistics [51].

Efforts have been made to parallelize the Huffman encoding and decoding processes to improve their efficiency. GPU implementations of Huffman encoding and decoding have been developed, taking advantage of the parallel processing capabilities of graphics processing units [74]. Parallel decoding algorithms have also been proposed for specific applications, such as decoding JPEG files [33].

In general, Huffman coding continues to be a fundamental and efficient method for compressing data. It has undergone thorough examination and practical implementation in diverse fields. The algorithm attains the most efficient compression for a specific set of symbol probabilities and has been employed in the JPEG image compression standard. Several modifications and enhancements to the algorithm have been suggested, such as canonical Huffman coding and adaptive Huffman coding. Different coding schemes have been proposed for memory efficiency, and efforts have been made to parallelize the encoding and decoding processes.

2.1.3.2 Run-Length Encoding(RLE)

Run-length encoding (RLE) is regarded as one of the most straightforward lossless compression algorithms, both in terms of comprehending its concepts and implementing it in software [45]. It is frequently employed as a fundamental method to compare it with more sophisticated compression strategies [56]. According to Sven and Wolf, RLE can be seen as the process of replacing consecutive repeated symbols with a count of the number of repetitions and the symbol itself [73]. This encoding technique is highly efficient when the input data consists of extended sequences of identical symbols.

RLE is utilized in several fields such as data compression, image processing, signal processing, and test data compression [55, 23, 66]. It is a popular choice because of its ability to effectively compress images with lengthy sequences of similar pixels [23]. In signal processing, RLE is used to compress biomedical data, for example, electrocardiograms, by encoding longer sequences with

similar values [1, 37]. Furthermore, researchers employed the RLE method to compress test data, aiming to reduce the size of test patterns in integrated circuits [81].

A variety of scholarly publications proposed multiple iterations and enhancements of the Run-Length Encoding technique. Specifically, two code-based strategies, namely Golomb and frequency-directed run-length encoding, are designed to encode the lengths of consecutive blocks, concluding at the occurrence of a specified symbol [55]. Additionally, an alternative approach involves the application of an adaptive 2D run-length encoding method, which autonomously curtails the aggregate count of run-lengths [78]. Furthermore, RLE can be seamlessly integrated with other compression methodologies, exemplified by Huffman encoding, thereby yielding improved compression ratios [66].

2.1.4 Hybrid Compression

Hybrid compression combines multiple compression algorithms to achieve optimal compression results. For instance, we introduce Deflate and Gzip algorithms which combine two fundamental compression techniques: Lz77 for dictionary-based compression and Huffman coding for entropy encoding.

2.1.4.1 DEFLATE

DEFLATE compression combines the Lz77 and Huffman coding algorithms [2, 22, 77]. The DEFLATE compression method begins by applying the Lz77 algorithm to locate recurrent sequences within the input data. Subsequently, these sequences are substituted with references to past instances, leading to

a decrease in the total size of the data. To obtain more efficient compression ratios, the generated data is additionally compressed through the application of Huffman coding [2, 77].

The DEFLATE compression algorithm has achieved extensive adoption and is compatible with numerous file formats, including the commonly used ZIP format [2]. It is also employed in online technologies such as HTTP and PNG, where it helps to minimize the size of web pages and images for faster transmission and enhanced user experience [76]. Additionally, DEFLATE compression has been incorporated in hardware and software solutions to enable effective compression and decompression of data [8].

2.1.4.2 Gzip

The Gzip algorithm is based on the DEFLATE algorithm, which is a hybrid algorithm combining Lz77, a dictionary-based encoding, and Huffman coding, an entropy-based algorithm. It expands on the Deflate algorithm by incorporating checksums, file packaging capabilities, and increased compatibility.

Gzip is a widespread lossless compression algorithm and a foundation for various applications and research endeavors. For instance, it serves as a benchmark for comparing the performance of other compression methods in studies related to lossless compression and its effects on data integrity [57]. Additionally, Gzip has been extended and improved upon in research works like "The Improvised Gzip" that focus on enhancing its capabilities for real-time lossless data compression [64].

Despite Gzip being prevalent, challenges exist, such as the inability to

decompress files at arbitrary locations due to the nature of the DEFLATE compression algorithm [32].

2.2 Data Reordering Techniques and Applications

Data reordering is a significant operation in computer science, especially in the fields of database management systems and algorithm design and analysis. Various methodologies are proposed for rearranging data matrices to discover patterns and enhance computational efficiency. There are diverse types of algorithms that implement data reordering and aim to achieve particular objectives. Three main categories are discussed in this section, as shown in Figure 2.2.

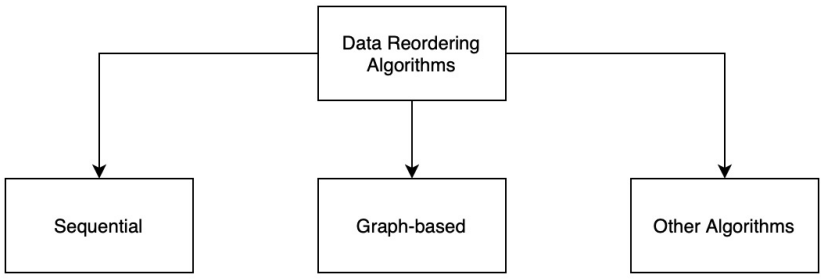


Figure 2.2: Tree Diagram for Data Reordering Algorithms

2.2.1 Sorting Algorithms

Sorting algorithms are essential in data reordering tasks to optimize the arrangement of elements in a dataset for specific purposes. Various algorithms

have been developed to tackle the challenges associated with data reordering.

Quicksort is a fundamental sorting algorithm known for its efficiency and simplicity. It follows the divide-and-conquer strategy, selecting a pivot element from the array and partitioning the other elements into two sub-arrays based on their relationship with the pivot. Researchers have developed practical quicksort algorithms tailored for graphics processors, enhancing sorting performance in highly parallel environments [6].] Quicksort algorithm has been extended and optimized for GPU architectures, demonstrating its versatility and applicability in modern computing systems [47]. While quicksort is a flexible and effective sorting algorithm with many benefits, it is crucial to take into account its sensitivity to pivot selection and possible worst-case time complexity.

Mergesort is another classic sorting algorithm known for its stability and consistent performance. Efficient implementations of mergesort have been explored in various contexts. For instance, researchers have developed efficient hardware implementations of MergeSort using high-level synthesis, showcasing the algorithm's adaptability to hardware acceleration and optimization [26]. Even if both mergesort and quicksort follow the divide-and-conquer strategy, mergesort obtains a consistent and stable time complexity of all cases while quicksort has the worst-case scenario.

Moreover, Silva et al. present a feature vector-based sort(FVS) method [70]. Their approach extracts feature vectors from a data matrix and uses them to calculate row and column permutations of the data matrix. It can detect simplex and equi patterns in pre-simplex and pre-equi matrices with 40 percent noise.

2.2.2 Graph-based Algorithms

Graph-based algorithms reorder data by leveraging the inherent relationships or similarities between data elements represented as nodes in a graph. The goal is often to find an ordering of the nodes that reflects meaningful patterns, structures, or groups within the data.

Minimum Spanning Tree(MST) seeks to connect all nodes in a graph with the minimum possible sum of edge weights. It is often used for reordering data to emphasize pairwise relationships. Dellen et al.'s work focuses on analyzing the human connectome using MSTs, demonstrating how MSTs can be utilized to understand neural pathways and optimize data organization within the brain [11]. Additionally, Mayala et al. present a method for brain tumor segmentation based on MSTs, showcasing how MSTs can be employed to rearrange data for medical imaging analysis [50].

Seriation is often considered a graph-based technique as it involves representing the relationships or similarities between elements in a dataset as a graph. In seriation, the elements (e.g., objects, artifacts, or data points) are treated as nodes in a graph, and the edges between nodes represent the pairwise relationships or similarities. The work of Liiv demonstrates the role of seriation as an exploratory combinatorial data analysis technique for reordering objects to reveal regularity and patterning [42]. This technique has been widely used in various fields, including computer science and biology. Furthermore, the study by Perin et al. discusses the relevance of seriation in crafting tabular visualizations, indicating its significance in revealing patterns among the whole series [59].

2.2.3 Other Algorithms

Many other algorithms also provide the ability to reorder data and make contributions in broad fields.

In the field of mathematics and computational applications, Kågström and Poromaa describe a reordering technique that relies on the periodic Schur decomposition [28]. They specify its capability to calculate eigenspaces with predetermined eigenvalues. Gordon-Ross et al. examine the integration of code reordering and cache configuration, emphasizing the practical significance of reordering techniques in enhancing cache utilization [16]. Behrisch et al. provide a comprehensive survey of algorithms for reordering visual matrices of tabular data and adjacency matrices of networks, emphasizing the importance of these methods in data visualization [4].

Collectively, the literature accentuates the importance of data reordering across many domains, encompassing data visualization, computational efficiency, and code optimization. These data reordering techniques play a crucial role in revealing patterns, optimizing data structures, and improving computational performance.

2.3 Data Reordering for Data Compression

Multiple compression techniques used in databases can be affected by data reordering, particularly when it comes to rearranging rows [39].

Poss and Potapov proposed an innovation compression technique in Oracle [60]. Their article highlights that reordering data before loading can enhance

the compression ratio of a database table, as columns arranged sequentially may possess identical content. Sorting the data on fields with fewer unique values in a database object will likely enhance the compression factor.

Data compression in the column-oriented Database Management System is improved through the process of data reordering, where data is stored by column rather than by row. Stonebraker et al. discovered that in C-Store, a column-oriented database, sorting a column results in grouping the same values together [72]. This grouping facilitates the compression of these values using techniques like run-length encoding.

The research of Lemire et al. in 2012 evaluates several row reordering heuristics, including the Lexicographic Order heuristic, the Vertex heuristic, the Multiple Lists heuristic, and the TSP heuristic to improve compression rate for database tables [39]. The Lexicographic Order heuristic involves sorting the rows of the table in lexicographic order, while the Vertex heuristic orders the rows based on their similarity to a reference row. The Multiple Lists heuristic involves partitioning the rows into multiple lists based on their values and then ordering the rows within each list. The TSP heuristic uses the Traveling Salesman Problem (TSP) to minimize the number of runs in the table.

Chapter 3

Preliminaries and Problem Description

3.1 Preliminaries

This chapter aims to provide the foundational understanding and prior knowledge necessary to fully grasp the subsequent research. Both shingling and minHashing are data preprocessing methods applied to raw data sourced from the origin site during the data migration process. The concept of clustering and TSP are served as data reordering techniques. Compression algorithms such as Messagepack binary serialization (MsgPack), run-length encoding (RLE), and Zstandard compression (ZSTD) are implemented after data transformation and reordering. Finally, a formal problem description is presented to delineate the scope and objectives of the research.

3.1.1 Clustering

Clustering is considered a fundamental task of unsupervised learning which involves organizing data structures within an unfamiliar domain, providing a foundation for further learning processes [79]. Although there is a lack of consensus on its precise definition, the classical view emphasizes the essential criteria: instances within the same cluster should display a high level of similarity, whereas instances in distinct clusters should manifest substantial dissimilarity. Additionally, the metrics employed to gauge similarity and dissimilarity should be transparent and practically meaningful.

The conventional clustering process adheres to a logical sequence: Feature extraction and selection entail identifying and selecting the most pertinent features from the original dataset [79]. It is necessary to tailor the clustering algorithm's design to the features of the problem. To evaluate the effectiveness of the approach, a thorough examination of the clustering results is essential, involving the practical interpretation of the acquired clustering outcomes.

3.1.2 Traveling Salesman Optimization Algorithm (TSP)

The Travelling Salesman Problem (TSP) is a widely studied combinatorial optimization problem in the fields of computer science and mathematical optimization [48]. It involves finding the shortest possible route that visits a set of cities and returns to the original city. The practical applications of this problem across multiple domains have led to its extensive investigation [54]. TSP is a challenging problem since its computational complexity increases exponentially when the number of cities increases [19].

The problem can be stated as follows: given a list of cities and the distances between each pair of cities, the task is to find the shortest possible tour that visits each city exactly once and returns to the original city. Table 3.1 presents the notations used in the TSP.

Table 3.1: Notations used in the Traveling Salesman Problem objective function

Notation	Description
N_{city}	Number of cities
a, b	Indices representing cities
ℓ_{ab}	Distance between city a and city b
x_{ab}	Variable indicating inclusion of edge from city a to city b
Z	Objective function (total distance of the tour)

The objective is to minimize the total distance of the tour. If ℓ_{ab} represents the distance of traveling from city a to city b, the objective function is often expressed as:

$$\text{Minimize} \quad Z = \sum_{a=1}^{N_{\text{city}}} \sum_{b=1, b\neq a}^{N_{\text{city}}} \ell_{ab} \cdot x_{ab}$$

Given the constraints as follows:

- Each city must be visited exactly once:

$$\sum_{a=1, a\neq b}^{N_{\text{city}}} x_{ab} = 1 \quad \forall b$$

- There is exactly one departure from each city:

$$\sum_{b=1,b\neq a}^{N_{\text{city}}} x_{ab} = 1 \quad \forall a$$

3.1.3 Shingling & MinHashing

3.1.3.1 Shingling

Shingling is employed in the analysis of text and the assessment of similarity. Its purpose is to represent a data entry as a collection of q-grams, or shingles, to facilitate various text analysis tasks, such as measuring text similarity. The process involves dividing a data entry into a series of shingles, which are essentially substrings of length q. This is achieved by sliding a window of length q over the string of data and capturing each q-sized segment. The resulting shingles form the vocabulary, spanning all entries in the dataset.

3.1.3.2 MinHashing

MinHashing is a data analysis and information retrieval technique for estimating the similarity of datasets. It involves applying a hashing function to each data entry within a dataset and then comparing the resulting hash values to approximate similarity. This approach is particularly beneficial when dealing with large datasets, as it helps alleviate the computational burden associated with comparing every pair of datasets. [5].

We employed minHashing to generate a signature matrix in our work.

The notations used for explanation are shown in Table 3.2.

Table 3.2: Notations in minHashing Algorithm

Notation	Description
$S_1, S_2, \ldots, S_n$	Collection of sets
U	Universal set of elements
$D(i, j)$	minHash signature matrix cell
$h_1, h_2, \ldots, h_k$	Hash functions
$\min\{h_j(u) : u \in S_i\}$	Minimum hash value for a set

The specific procedures are described as follows:

1. **Initialize the Signature Matrix:**

   Create an empty $n \times k$ matrix where each cell $D(i, j)$ will eventually store the minimum hash value for the j-th hash function applied to the i-th set:

$$D = \begin{bmatrix} \emptyset & \emptyset & \ldots & \emptyset \\ \emptyset & \emptyset & \ldots & \emptyset \\ \vdots & \vdots & \ddots & \vdots \\ \emptyset & \emptyset & \ldots & \emptyset \end{bmatrix}$$

2. **Hashing and Signature Generation:**

   For each set S_i in the collection and for each hash function h_j, compute the hash values of all elements in S_i using h_j and record the minimum hash value in the corresponding cell $D(i, j)$ of the signature matrix. Mathematically, for a set S_i and hash function h_j:

$$D(i, j) = \min\{h_j(u) : u \in S_i\}$$

This process is repeated for all sets and hash functions.

3. **Resulting Signature Matrix:**

   When finishing the previous procedure for all sets and hash functions, the signature matrix D will be fully populated with the minimum hash values. Each column $D(:, j)$ represents the signature of a hash function h_j applied to all sets, and each row $D(i, :)$ represents the signature of all hash functions applied to the i-th set.

3.1.4 Run-Length Encoding (RLE)

Run-length encoding (RLE) is a straightforward compression technique for representing consecutive identical data values as a single value and count pair. It's commonly employed to decrease data size by condensing extended sequences of the same value into a single representation.

In Chapter 2, we extensively discussed the research background of RLE. In this chapter, our focus is to introduce its fundamental encoding and decoding procedures.

3.1.4.1 Encoding Notations

count : Number of consecutive occurrences of a value.

value : Data value being counted.

3.1.4.2 Encoding Example

The RLE encoding process involves counting consecutive occurrences of the same value and encoding them as a pair $(\text{count}, \text{value})$. Its encoding example is demonstrated as follows:

Applying RLE to the input data: 00111111110000011111111000011111.

- Start with an empty output string.
- Begin at the first character: 0. Count = 1, Value = 0.
- Continue to the next character: 0. Since it's the same as the previous character, increment the count to 2.
- Move to the next character: 1. Since it's different from the previous character, append the current count and value to the output string: $(2, 0)$.
- Reset the count to 1 and update the value to the new character: 1.
- Continue this process for the entire input string.

3.1.4.3 Encoded Output

The RLE-encoded data for the input 00111111110000011111111000011111 would be:

$$2,0 \quad 8,1 \quad 5,0 \quad 9,1 \quad 4,0 \quad 5,1$$

3.1.4.4 Decoding

To decode the RLE-encoded data, reverse the process. Parse the encoded pairs $(\text{count}, \text{value})$ and repeat the value count times in the output sequence. For example, $2, 0$ means repeat 0 two times.

3.1.5 Zstandard Compression (Zstd)

Zstandard (Zstd) is a lossless compression algorithm designed to efficiently reduce the size of data by leveraging various techniques such as dictionary-based compression, match finding, and reference encoding. It operates by maintaining a sliding window mechanism to detect and replace recurring patterns within the input data. Unlike the Lz77 compression algorithm discussed in the literature review, Zstd offers a larger search window and a more efficient entropy-coding stage. Additionally, Zstd provides adjustable compression levels, allowing users to optimize the balance between compression ratio and speed according to their specific requirements.

The implementation of Zstd involves several key procedures. Initially, the input data is divided into blocks, and within each block, entropy encoding techniques are utilized to identify repeating patterns and determine their frequencies. Context modeling is employed to adapt to the specific characteristics of the data, while a dynamic dictionary containing frequently occurring sequences is utilized to enhance compression effectiveness. Huffman encoding is then applied to assign shorter codes to more frequent symbols, further optimizing compression efficiency. The compressed data is organized into frames, each containing metadata for compression settings and data structures. Different compression

levels can be selected to achieve the desired balance between compression ratio and speed, and optional dictionaries can be utilized to improve compression in specific scenarios. Additionally, Zstd may leverage multi-threading to enhance compression and decompression speeds.

3.1.6 MessagePack Binary Serialization (MsgPack)

MessagePack is a binary data serialization format known for its compactness and efficiency compared to text-based formats like JSON and XML. By encoding data in binary format, MessagePack achieves smaller message sizes and faster serialization and deserialization processes. It supports a wide range of data types, including strings, arrays, maps, integers, and floats.

Unlike JSON, MessagePack is not designed to be human-readable, as its primary focus is on efficient data serialization rather than human inspection or editing.

MessagePack finds common use cases in scenarios requiring space-efficient and performant data transmission or storage, particularly in interprocess communication and data exchange between different systems.

3.2 Problem Description

The primary objective of this section is to provide a clear and formal definition of the research problem.

Let $\mathbf{T}$ denote a relational table comprising multiple columns $\mathcal{C}$, where each specific column c_i stores an attribute. We use r_j to denote the j-th record in T

as per the physical order of records in T.

Our research goal is to optimize the compression ratio of the relational table **T** and improve its storage efficiency. Specifically, we apply the dictionary-based compression algorithm to each c_i stored in **T** to achieve column-wise compression.

Table 3.3: Notations used in the Problem Description

Notation	**Description**
T	A relational table
$\mathcal{C}$	The set of columns within **T**
c_i	The ith column in original **T**
r_j	The jth record in original **T**
d_{ij}	The data entry at ith column and jth row in **T**

Chapter 4

Methodologies

4.1 Data Representation Algorithm

If the clustering algorithm is applied directly to all the r_j which is stored in **T**, the preliminary compression results are not promising. Because every r_j contains different data types and lengths for each c_i, the clustering algorithm cannot compare the similarity among d_{ij} and easily define the centroid of clusters. With the help of LSH, data in each c_i is standardized into the same format, normally a dense vector. LSH representation (*signature*) will be employed in the following clustering process instead of the data itself to facilitate the centroid calculation for the clusters.

Two main steps are taken to implement the LSH scheme demonstrated in Figure 4.1 [40].

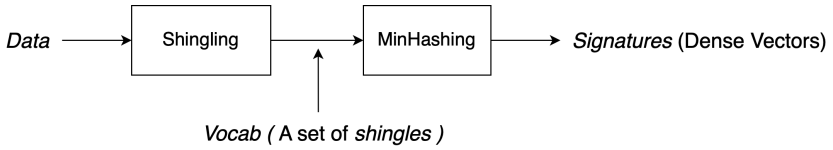


Figure 4.1: Data Representation Flow Diagram

1. **Shingling** is the process of transforming a string of data into a set of *shingles*. It's akin to sliding a window of length Q down the data string and capturing a snapshot at each step. These snapshots are saved to create a set of *shingles*. The vocabulary *vocab* comprises all the shingles spanning all d_{ij}. The vector representation of each d_{ij} is formed using this *vocab*. An empty vector made entirely of zeros and having the same length as the *vocab* is often constructed for each d_{ij}. Each time a *shingle* appears in d_{ij}, its position is detected in the *vocab*, and the corresponding location in the empty zero-vector is set to 1.

2. **MinHashing** enables us the conversion of sparse vectors into dense vectors. Sparse vectors are generated by shingling, which results in high-dimensional levels. Minhash functions are randomly generated for every position in the *signature* (i.e., the dense vector). If a M-dimension signature is required, M minHash functions are applied. MinHash functions shuffle the order of numbers. The *signature* values are constructed by first taking a randomly permuted count vector (ranging from 1 to len(vocab)+1) and identifying the minimum number that corresponds to a 1 in the sparse vector.

   As an illustrating example for LSH, each data entry in the column *Car*

Name is transformed into a four-dimensional vector with sliding window size Q=2 for shingling and M=4 for minhashing.

Car Name	D1	D2	D3	D4
Hyundai Grand i10 CRDi Asta	74	27	454	1246
Hyundai Grand i10 1.2 CRDi Sportz	74	1101	454	1246
Mahindra TUV 300 T4	860	1180	454	1246
Mahindra Bolero SLX 4WD BSIII	860	1180	454	1095
Mahindra XUV300 W8 Option Diesel BSIV	860	1180	454	790

Figure 4.2: An LSH Representation Example for Column *Car Name*

Algorithm 1 Locality Sensitive Hashing

Input: d_{ij}, Q, num_hashes, $vocab$, M
$shingles \leftarrow \emptyset$
for $i \leftarrow 1$ to $|d_{ij}| - Q + 1$ **do**
    $shingle \leftarrow \text{tuple}(d[j : j + Q])$
    $shingles \leftarrow shingles \cup \{shingle\}$
end for
$hashFunctions \leftarrow \emptyset$
for $i \leftarrow 1$ to M **do**
    $a \leftarrow \text{random.randint}(1, |vocab|)$
    $b \leftarrow \text{random.randint}(1, |vocab|)$
    $hashFunction \leftarrow \lambda x : (a \cdot x + b) \mod |vocab|$
    $hashFunctions \leftarrow hashFunctions \cup \{hashFunction\}$
end for
$signatures \leftarrow \emptyset$
for $hashFunction$ **in** $hashFunctions$ **do**
    $minValue \leftarrow +\infty$
    for $shingle$ **in** $shingles$ **do**
        $hashValue \leftarrow \text{applyHashFunction}(shingle, hashFunction)$
        $minValue \leftarrow \min(minValue, hashValue)$
    end for
    $signatures \leftarrow signatures \cup \{minValue\}$
end for
Output: $signatures$

4.2 Clustering-based Methods

Data compression involves encoding information using fewer bits than the original representation. Clustering can aid in this process by identifying patterns or similarities within the data and grouping similar elements.

Clustering is the process of grouping object attributes and features such that the data objects in one group are more similar than data objects in another group with no prior knowledge of data structure [17]. Compared to the sorting algorithms discussed in the literature review, utilizing clustering for data reordering involves considering all columns within a table, rather than depending exclusively on a subset of columns for the reordering process. While some clustering algorithms, like k-means clustering, treat all digital data as numerical data and determine the center point (mean) using Euclidean distance, this approach may not be suitable for dictionary-based compression settings. For example, the similarity between textual data such as '6666' and '66' is higher than the similarity between '6666' and '5555', even though the Euclidean distance between '6666' and '66' is greater. Thus, it is more appropriate to consider every d_{ij} as categorical data, irrespective of the data format.

K-prototype clustering is an unsupervised machine learning algorithm that extends the k-means algorithm to handle mixed numerical and categorical variables [21]. It combines the standard k-means algorithm for numerical data with a mode-based method for categorical data. The distance between data points is calculated using a combination of Euclidean distance for numeric attributes and a dissimilarity measure, which is often referred to as simple matching for categorical attributes [31]. In our proposed technique, the input

data for k-prototype clustering is represented as M-dimensional *signature* vectors that represent d_{ij}. The number of clusters, N, is set as a parameter for the experiments before the clustering process. After k-prototype clustering, $\mathbf{T}$ is partitioned into N groups. As shown in Figure 4.3, all the d_{ij} are grouped into three clusters. Each cluster exhibits a significant degree of internal similarity. These three clusters are then sequentially merged to form the reordered $\mathbf{T}$. Subsequently, dictionary-based compression techniques are expected to be applied to each reordered c_i."

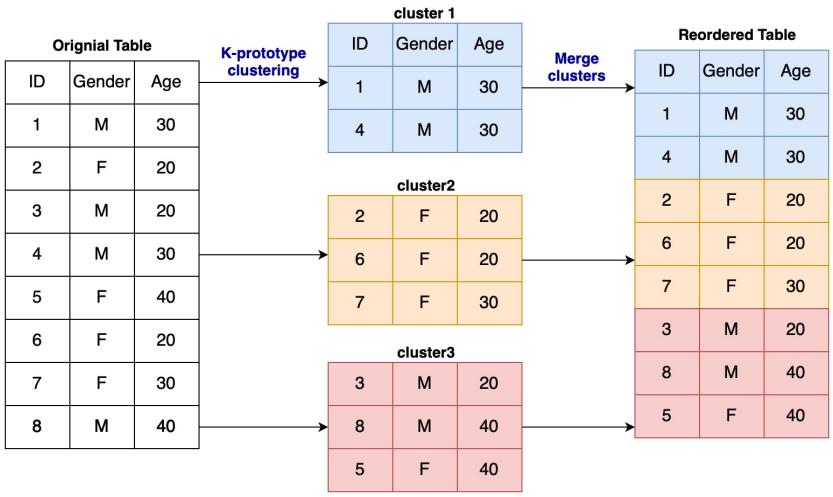


Figure 4.3: Clustering-based Table Reordering Diagram

While k-prototype clustering is effective in improving compression ratios, its application to extensive datasets presents a time-consuming challenge. In real-life scenarios, the extended duration required for reordering data using clustering methodologies renders it impractical. Hence, there is an urgent need to explore alternative, more time-effective approaches for rearranging data and

identifying similar patterns suitable for dictionary-based compression.

4.3 TSP-based Methods

Chapter 3 presents a comprehensive overview of the general application of TSP solutions, primarily focusing on determining the shortest route within a closed loop of cities. However, in our methodology, TSP solutions serve a distinct purpose: they function as a data reordering technique, aiming to optimize the arrangement of data items based on their similarity. Unlike the traditional TSP formulation, which conceptualizes cities as physical entities with spatial relationships, data elements are abstract entities typically representing data points or vectors in data reordering. As a result, the distance calculation is adjusted to assess the similarity between data points rather than computing physical spatial distances. In contrast to graph-based reordering algorithms, which heavily rely on clear hierarchical or relational structures present in the given data, TSP solutions offer greater adaptability to general cases, as they do not necessarily require prior knowledge of the data structure.

We consider the open loop Traveling Salesman Problem (TSP) where the salesperson does not return to the starting city and find the optimal order for data items using a greedy local-search heuristic, which is referred to as a heuristic-based TSP solver in our approach [61]. To initiate the process of applying this heuristic-based TSP solver to data reordering, we first construct a square matrix representing the distances between data points, denoted as d_{ij}. For clarity, let's consider an example illustrated in Figure 4.4, featuring three

data points: X, Y, and Z. The distance from point X to itself is 0, the distance from X to Y is 2, and the distance from X to Z is 3. As a result, the first row of the square matrix displays the distances originating from X, denoted as (0, 2, 3). Following the same procedure for Y and Z, we can derive the complete square matrix encompassing all three data points.

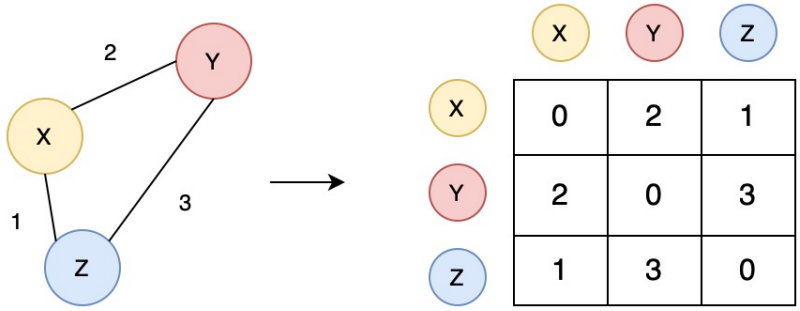


Figure 4.4: Example of Square Matrix Formation

The distance calculation between d_{ij} is made by Jaccard distance measurement in our work. The Jaccard index, also referred to as the Jaccard similarity coefficient, serves as a statistic for assessing the similarity and dissimilarity of sample sets [65]. This coefficient quantifies the similarity between sample sets by evaluating the ratio of the intersection to the union of the sets. Conversely, the Jaccard distance, representing dissimilarity between sample sets, complements the Jaccard coefficient. Equation 4.1 illustrates the calculation of the Jaccard distance:

A and B are two sets, $|A \cap B|$ denotes the cardinality of the intersection of sets A and B, and $|A \cup B|$ denotes the cardinality of the union of sets A and B.

$$J(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|} \tag{4.1}$$

Utilizing the distance matrix generated from the Jaccard distance, the heuristic-based TSP solver starts to construct an initial solution greedily aiming to minimize the total distance traveled. Subsequently, the local search heuristic is applied to optimize this initial solution. This heuristic iterates through pairs of edges in the solution, attempting to replace them with more optimal pairs whenever possible. This iterative process continues until no further improvement can be achieved. Finally, the data is reordered based on the optimal path derived from the heuristic-based TSP solver.

TSP is a well-known "hard" optimization problem that challenged mathematicians and computer scientists for decades to come up with the best search algorithm, and it is time-consuming to solve large data sets for TSP [9]. Hence, our approach involves partitioning the $\mathbf{T}$ into chunks, solving TSP within each individual chunk. Based on resolved tours for TSP, we reorder the tuples within each chunk. As illustrated in Figure 4.5, all the d_{ij} of $\mathbf{T}$ are reordered by chunk. Finally, we concatenate all the chucks of $\mathbf{T}$ with the optimal order together.

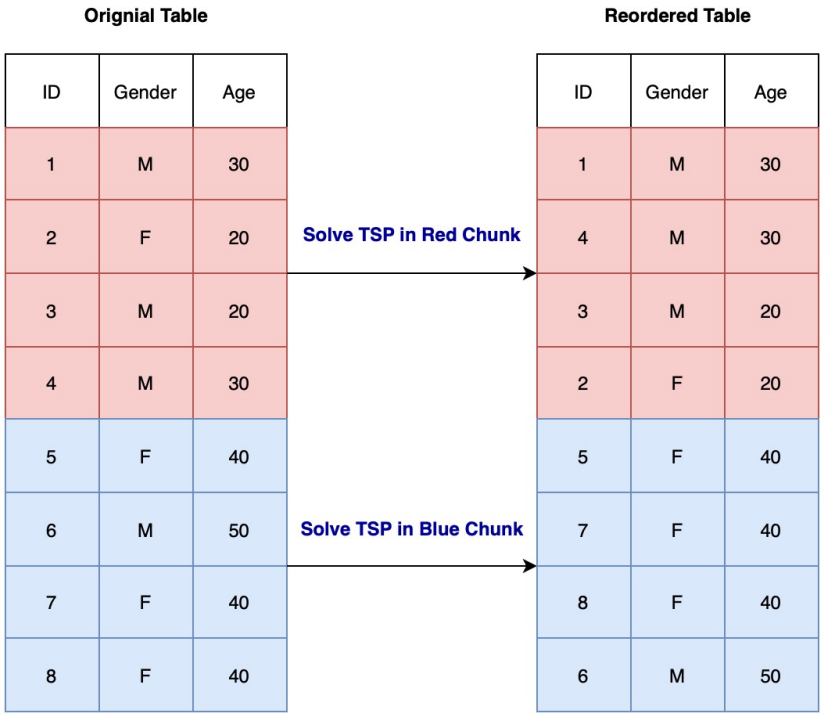


Figure 4.5: Example of Chunk-wise TSP Solution

Reordering the data through a heuristic-based TSP solver based on similarity facilitates the detection of repetitive patterns, particularly through the application of dictionary-based compression techniques. Compared to the previously mentioned k-prototype clustering method, the heuristic-based TSP solver reorders data by chunks, proving to be more time-efficient while providing excellent reordering capability.

For the compression process, column-wise compression will be implemented on individual c_i of the reordered table to attain a higher compression ratio, given that the data within c_i typically shares similar traits in tabular data.

4.3.1 DVP-Partitioned Table Compression

Inspired by the work from Lemire et al. which combines the TSP heuristics with run-length encoding, we not only discover their method's effectiveness for compressing the column with a smaller number of distinct values but also the limitation for the column with a larger number of distinct values [39]. Columns with a higher distinct value percentage may exhibit repetitive patterns that computers might struggle to identify. For example, a customer ID column with numerous distinct values, such as "12345" and "12346," may share the substring "1234". The sole application of data reordering via a heuristic-based TSP solver and run-length encoding is insufficient for detecting such similarities and grouping them together.

To further address the problem mentioned above and achieve an optimal compression ratio for the entire table, we introduce a novel table-partitioned compression strategy based on the distinct value percentages in individual columns. This approach is designated as DVP (Distinct Value Percentage)-Partitioned table compression.

The Distinct Value Percentage (DVP) is calculated as follows:

$$\text{DVP} = \frac{\text{the number of unique values within the column}}{\text{total number of tuples}} \tag{4.2}$$

DVP-Partitioned table compression aims to categorize $\mathcal{C}$ into different groups based on DVP. The same compression strategy is applied to the $\mathcal{C}$ within the same group. Additionally, each group is reordered by the heuristic-based TSP solver together, and a corresponding row ID for each r_j is assigned

before reordering. This is used to ensure the recovery process of the original order after decompression. If **T** contains only a few columns and rows, dividing it into too many groups will lead to a large extra storage cost for row IDs, which is not favorable for calculating the overall compression ratio.

As a result, the groups typically fall into two categories in my proposed strategy:

1. DVP $>$ DVP threshold
2. DVP $\leq$ DVP threshold

The DVP threshold here is adjustable, and based on the results of numerous experiments, we typically set the distinct value percentage threshold in the range of 0.01 to 0.1. This range is chosen to identify the optimal table partition.

The comprehensive procedures for DVP-Partitioned table compression are illustrated in Figure 4.6. Assuming that a **T** is required to be compressed, we attempt to split and compress it via DVP-Partitioned table compression. The DVP for every c_i in **T** is calculated to facilitate the following column grouping process. After comparing each column's DVP with the DVP threshold, all the columns are divided into two groups

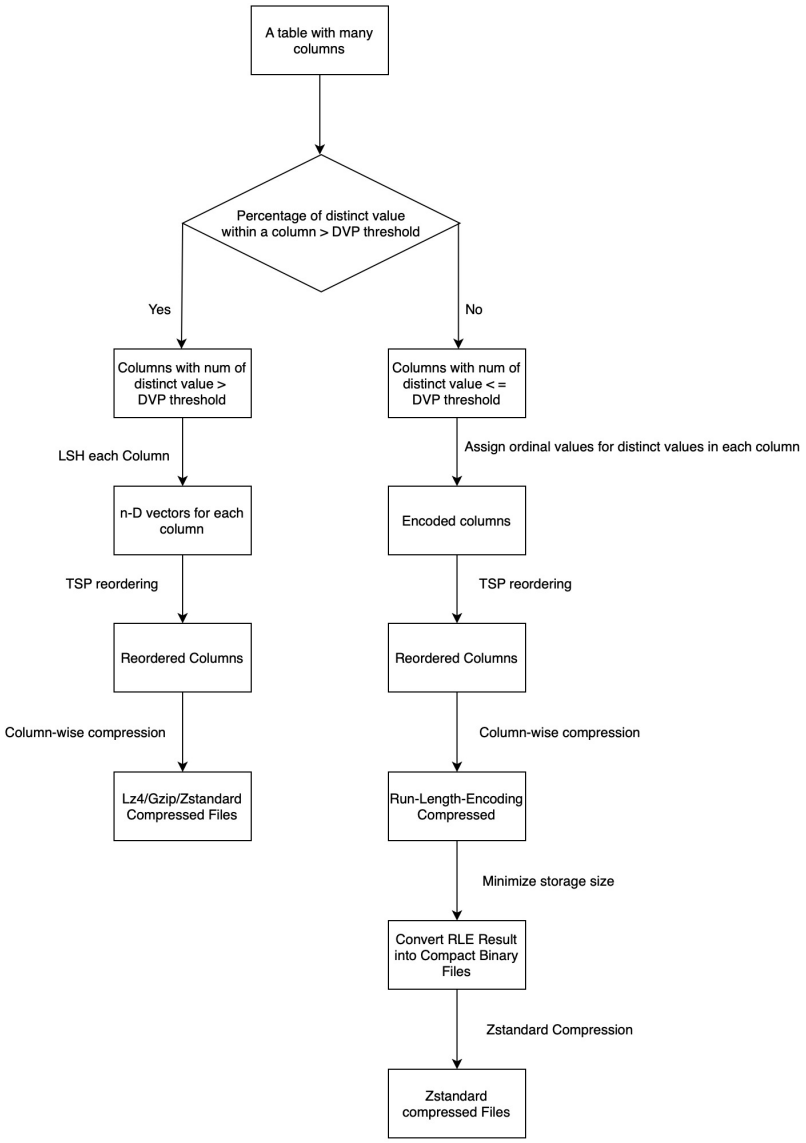


Figure 4.6: DVP-Partitioned Table Compression Workflow Diagram

The LSH algorithm initially preprocesses the group of c_i with DVP larger than the DVP threshold. As illustrated in the left path of Figure 4.6, each c_i undergoes LSH once and is transformed into high-dimensional vectors. The reordering via the heuristic-based TSP solver is then performed utilizing the

vectors generated from all c_i within the group. Finally, the dictionary-based compression algorithm is applied to each reordered column.

The other group of c_i with DVP smaller than the DVP threshold undergoes a simple data preprocessing stage. The distinct values in each c_i are assigned ordinal values to aid the following data reordering and run-length encoding procedures. The compression process for this group involves three compression algorithms performing different functions. First, run-length encoding is applied to compress each reordered c_i as it is used to handle repetitive sequences efficiently. Then, the RLE result is further compressed and converted into compact binary files using the messagepack algorithm, which aims to optimize serialization to reduce overhead and represent data compactly. Finally, Zstandard compression, which provides adaptive compression for different data types, is applied to compact binary files to further reduce the compressed files' size and enhance compression performance. As the three compression algorithms work differently but corporately to exploit the duplicate patterns in smaller DVP columns, there is no existing single compression technique that can achieve similar functions and compression performance.

Chapter 5

Experiments

In this chapter, the proposed methodologies are assessed using one synthetic and three real datasets. Additionally, we vary the parameters of interest to explore the underlying factors that impact our proposal.

5.1 Experimental Settings

5.1.1 Environment

The experiments are conducted on a server with the hardware configurations as shown below. The codes for all the experiments are created using Python 3.9. The primary libraries utilized include pandas, NumPy, and SciPy.

- CPU Model: AMD Ryzen 24-Core Processor
- CPU MHz: 2200
- CPU max MHz: 4568.2

- RAM: 125GB

5.1.2 Datasets

Both numerical and categorical data are present throughout every dataset. DS001 dataset is generated from the TPC-H benchmark [75]. The Bank Marketing and Census Income datasets are sourced from the UC Irvine Machine Learning Repository [52, 3]. The Vehicle dataset is obtained from Kaggle [53]. These datasets are chosen to represent small, medium, and large datasets based on the number of columns and tuples, ensuring the versatility of DVP-partitioned table compression. An overview of the datasets' information is presented in Table 5.1. The *Cols* represents the total number of columns in datasets. In addition, we present the number of columns with numerical, categorical, and mixed (numerical and categorical) data for each dataset.

Table 5.1: Overview of Dataset Information

Dataset	**Rows**	**Cols**	**Numerical**	**Categorical**	**Mixed**
DS001	386671	7	4	3	0
Bank Marketing	45212	17	7	10	0
Census Income	32462	15	6	7	2
Vehicle	8129	13	4	5	4

5.1.3 Evaluation

The data compression ratio, calculated as the ratio of the uncompressed file size to the compressed file size, is used to evaluate the result. The main equation for the data compression ratio is shown in Equation 5.1:

$$\text{Compression Ratio} = \frac{\text{Uncompressed Files Size}}{\text{Compressed Files Size}} \tag{5.1}$$

The compression ratio of the random arrangement of columns serves as the benchmark for the compression ratio of individual columns.

For the evaluation of DVP-Partitioned table compression, the overall compression ratio after reordering is calculated as indicated in Equation 5.2. As the columns are reordered by groups, individual group obtains a sequence of row IDs which is used to recover the original order after decompression. The uncompressed files are reordered columns and two sets of row IDs as the table is divided into two groups based on each column's DVP. The compressed files are compressed columns and compressed sets of row IDs.

$$\text{Overall Compression Ratio} = \frac{\sum \text{Uncompressed Files Size and Row IDs Size}}{\sum \text{Compressed Files Size and Row IDs Size}} \tag{5.2}$$

To ensure a fair comparison, we compress columns separately in random order. Thus, the baseline is calculated as the sum of all the compressed random-order column sizes divided by the sum of all the uncompressed random-order column sizes.

5.2 LSH and K-prototype clustering

To investigate the impacts of two LSH parameters and the number of clusters on the column-wise compression ratio when reordering via k-prototype clustering.

We introduce three major parameters employed in the LSH and k-prototype clustering experiments:

- Q is the size of sliding windows for shingling
- M is the dimension level for *signiture*
- N is the number of clusters for k-prototype clustering

The ultimate reordered file is formed through the concatenation of all clusters. Specifically, cluster 2 is appended to cluster 1, and this process extends sequentially to include cluster 3 and beyond. The comparison is between one compressed original file and one compressed reordered file.

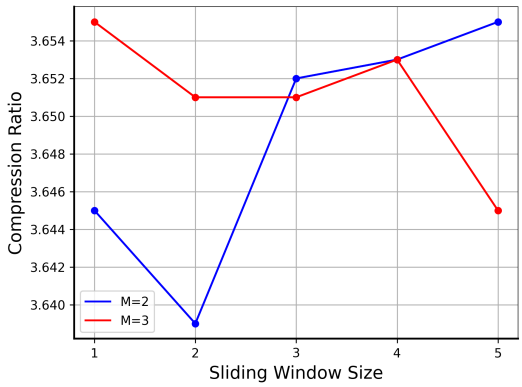
We showcase the `mileage` column from the Vehicle dataset for LSH parameters' experiments due to its inclusion of multiple recurring patterns, a characteristic that is easily discernible for humans but poses a challenge for computational analysis. Furthermore, the tuples within `mileage` consist of mixed types of data (categorical and numerical data) instead of solely one type of data (categorical or numerical data) in other columns, reflecting a scenario akin to those encountered in real-life situations.

Regarding the clustering parameter N, the column `Customerkey` from the DS001 dataset is employed, given its substantial number of tuples, which can significantly influence the determination of the optimal number of clusters.

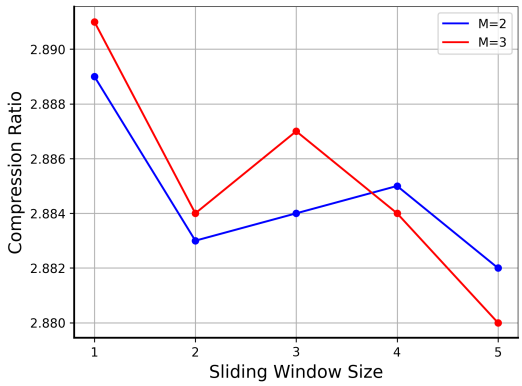
5.2.1 Sliding Window Size Q for LSH

In this experiment, we aim to investigate the impact of the LSH parameter Q on compression results. We select a column `mileage` to perform the experiments

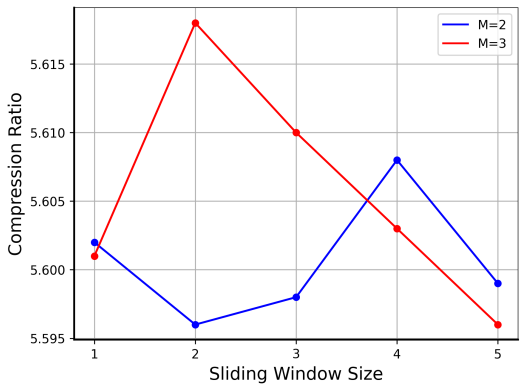
as it contains numerical and textual data. To maintain consistency, we employ k-prototype clustering as the method for data reordering, with a fixed cluster number of 10. Another crucial LSH parameter, M, is set to a constant value of 2 or 3. Subsequently, we vary the size of Q to obtain reorganized columns and apply three commonly used dictionary-based compression algorithms to assess the compression outcomes.



(a) Lz4



(b) Gzip



(c) Zstd

Figure 5.1: Compression Ratio for `mileage` When Varying Q (M=2/3, N=10)

As depicted in Figure 5.1, we observe that, under the influence of three different compression algorithms, the tendencies of compression ratios for the reordered column are different when increasing Q.

Figure 5.1(a) shows that the Lz4 compression ratio increases when Q increases from 2 to 5, with M=2 and N=10. However, when M is set to 3, the tendency is distinct when Q ranges from 2 to 5. The highest compression ratio is 3.655, achieved when Q=1 and M=3, or when Q=5 and M=2. This indicates that the parameters M and Q are highly relevant for the Lz4 compression method.

Figure 5.1(b) illustrates that increasing Q leads to a generally decreasing compression ratio for M=2 and M=3 in Gzip scenarios. The highest compression ratio is obtained when Q=1.

Figure 5.1(c) demonstrates a distinct tendency when M=2 and M=3. The settings of Q and D are highly relevant for Zstd. The highest compression ratio is achieved when Q=2 and M=3.

In summary, optimal compression results are achieved for all three compression methods when Q is set to a relatively small value. A smaller sliding window size Q usually leads to a smaller vocabulary (*vocab*) and better string matching when the column contains many repetitive patterns. However, the effect of Q might be distinct when the setting of M changes. Notably, each column possesses distinct properties and features, resulting in varying impacts of Q on compression for different columns.

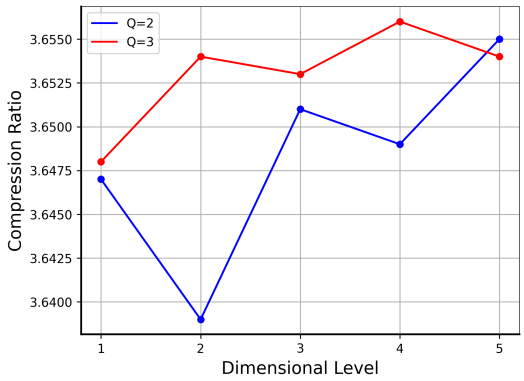
5.2.2 Dimensional Level M for LSH

Similar to the experimentation with the parameter Q mentioned above, the experiment on the parameter M employed the method of controlling variables to investigate its impact on compression effectiveness. The cluster number for k-prototyping is fixed at 10, and the value of parameter Q is set to 2 or 3. By varying the value of M, we aimed to observe its influence on the compression of the reordered column "mileage". The compression algorithms utilized in this experiment are Lz4, Gzip, and Zstd.

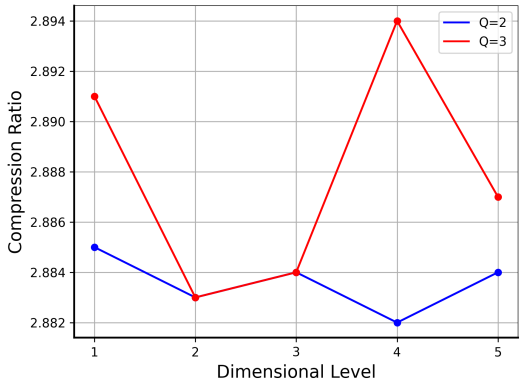
Figure 5.2 shows that different values of Q lead to nearly opposite tendencies for all three compression methods, especially for Lz4 and Zstd scenarios. This emphasizes that parameter M is significantly related to parameter Q when it comes to the compression ratio of the reordered column.

The highest compression ratio for Lz4 is achieved when M=4, as shown in Figure 5.2(a). Similarly, in the case of Gzip, the compression ratio in Figure 5.2(b) reaches its optimal when the value of M is 4. Figure 5.2(c) indicates that the peak point of the Zstd compression ratio is obtained when M=3.

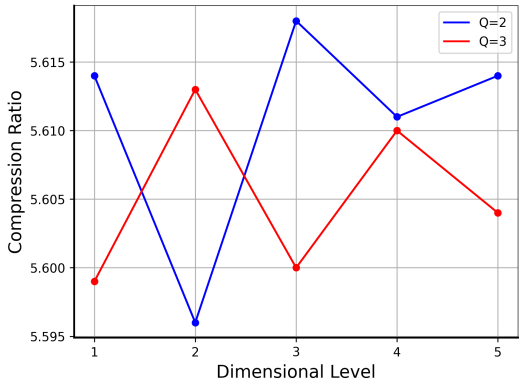
In conclusion, the impact of parameter M on the reordered column's compression ratio is sensitive to the setting of parameter Q. The optimal compression ratio is more likely to be achieved when M is a relatively large value. M represents the dimensional level of the signature. A higher dimensional level of the signature can typically reveal more details and patterns of data than a lower dimensional level of the signature. It is noticed that the impact of parameter M on the reordered column's compression ratio may vary across columns due to differences in data types and lengths for each column.



(a) Lz4



(b) Gzip



(c) Zstd

Figure 5.2: Compression Ratio for `mileage` When Varying M (Q=2/3, N=10)

5.2.3 Number of Clusters N for K-prototype

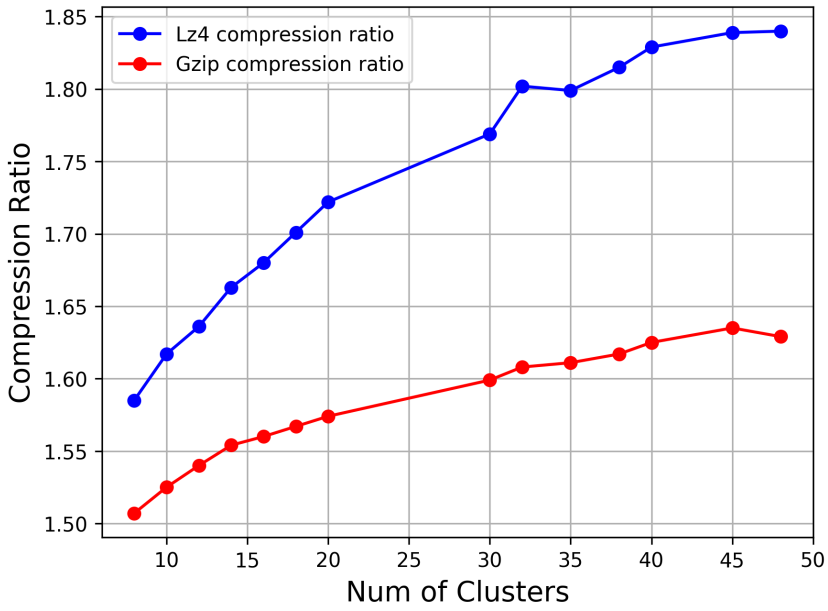


Figure 5.3: Compression Ratio for `Customerkey` When Varying N (Q=3, M=5)

After conducting numerous experiments with various parameter settings, the optimal LSH parameters for the "CustomerKey" column were determined to be $Q = 3$ and $M = 5$. Consequently, to investigate the impact of the cluster number N, we set Q and M as 3 and 5, respectively. The rearranged column is compressed using the Lz4 and Gzip algorithms. As illustrated in Figure 5.3, with N varying from 10 to 48, the compression ratio exhibits a continuous increase. On average, the compression ratio increases by 17.01% for Lz4 and 12.80% for Gzip as N varies. When N is set to 30, the compression ratio obtains a 25.6% improvement for Lz4 and a 16.1% improvement for Gzip compared to baselines of 1.408 and 1.377, respectively, which were generated from the

random split of the column for Lz4 and Gzip.

Also, taking into account the diverse data types, lengths, and record counts of distinct columns, the influence of parameter N on the overall compression ratio may vary across different columns.

5.3 TSP-based vs Clustering-based

Both the heuristic-based TSP solver and k-prototype clustering serve as data reordering methods in our approach. Experiments on whole table compression using the Bank Marketing dataset are conducted to assess their respective effects on enhancing overall compression performance. In real-life scenarios, the execution time of data reordering algorithms may be a limiting factor. Therefore, we record not only the overall compression ratio of both methods but also their processing time.

To guarantee an unbiased comparison, the heuristic-based TSP solver and k-prototype clustering undergo identical data preprocessing stages. All columns are concatenated into a single column, and LSH is applied to this consolidated column. Both LSH parameters, Q and M, are set to 6 to ensure the revelation of more detailed patterns within the consolidated column.

The heuristic-based TSP solver is executed via a greedy algorithm, solving the TSP chunk by chunk to expedite the reordering process. The chunk size is consistently set to 5000 for all experiments.

Note: The time format in the "Time for execution" column is represented as `hh:mm:ss`.

Method	**Lz4 ratio**	**Gzip ratio**	**Time for execution**
Heuristic-based TSP Solver	3.798	5.863	11:30
K-prototype (N=5)	3.613	5.963	28:00
K-prototype (N=10)	3.696	6.048	54:25
K-prototype (N=15)	3.765	6.122	1:17:16

Table 5.2: Compression Ratios and Execution Times for Bank Marketing Dataset using LSH combined with Heuristic-based TSP solver / K-prototype clustering.

In Table 5.2, it is evident that the execution time for all k-prototype clustering experiments is considerably longer than that for the heuristic-based TSP solver. The compression ratio for k-prototype clustering shows a slight increase with the rise of parameter N. The heuristic-based TSP solver achieves a higher compression ratio than all clustering methods when using Lz4. However, it attains the lowest compression ratio when using Gzip. Overall, the compression ratios for the heuristic-based TSP solver and k-prototype clustering show minimal differences, indicating that they can achieve comparable performance for the Bank Marketing dataset. The shorter execution time for the heuristic-based TSP solver stands out as a significant advantage over clustering methods. Consequently, the heuristic-based TSP solver is employed for the data reordering procedure in the subsequent DVP-partitioned table compression.

5.4 DVP-Partitioned Table Compression with heuristic-based TSP solver

To ensure the universality of this strategy, we conducted evaluations using three datasets derived from real-world scenarios. Figure 4.5 outlines the general procedures for processing a tabular dataset.

Initially, we conducted experiments on the Vehicle dataset. After determining the DVP for each column, we iteratively grouped columns ten times based on DVP thresholds ranging from 0.01 to 0.1. The values in high DVP columns (columns with fewer unique values) were transformed into signatures through LSH.

The impact of parameters Q and M on the compression ratio depends on the unique properties of individual columns, as explained in section 5.2. Therefore, following a rule-of-thumb approach, we selected Q=2 and M=4 as suitable LSH parameters to generate signatures for all columns in the high DVP group. The other group of columns with low DVP underwent the process outlined in the right path of Figure 4.6.

The calculation of the overall compression for the reordered table accounted for the inclusion of row IDs, ensuring accurate recovery of the original sequences after decompression, as illustrated in Equation 5.2.

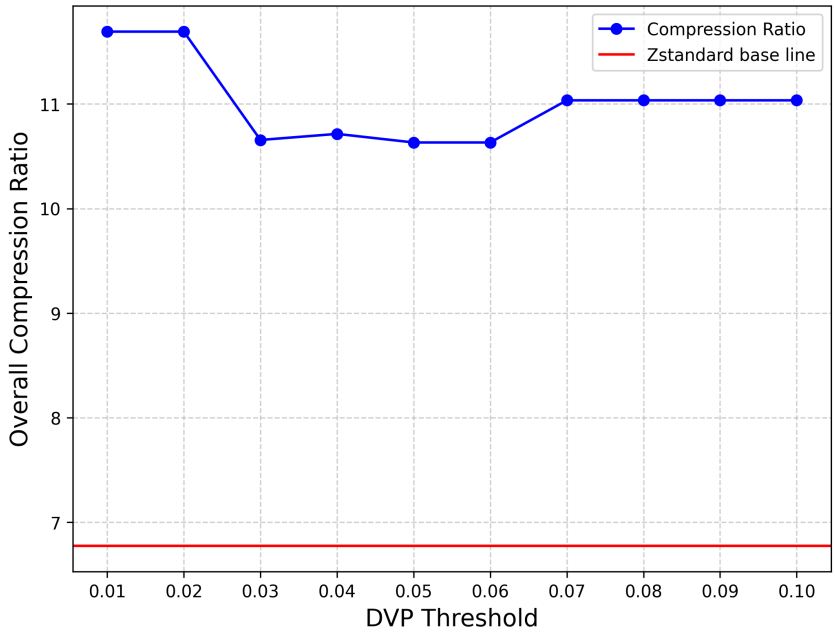


Figure 5.4: Overall Compression Ratio for Vehicle Dataset

In Figure 5.4, the blue line depicts the overall compression ratio for the reordered table across DVP thresholds ranging from 0.01 to 0.1. The red line serves as the baseline, calculated as the sum of all compressed random column sizes divided by the sum of all uncompressed random column sizes. Both the blue and red lines are compressed using Zstd, which was chosen for its superior performance compared to Lz4 and gzip.

The peak point of the blue line, reaching an overall compression ratio of 11.692 in Figure 5.4, corresponds to the table partitioned by DVP thresholds of 0.01 and 0.02. This represents a notable improvement compared to the red line, which registers a compression ratio of 6.774. The compression ratio for the Vehicle dataset experiences a substantial increase of 72.6 percent.

Similar procedures involving appropriate LSH parameter selections are

applied to both the Census Income dataset and the Bank Marketing dataset. Specifically, for the Census Income dataset, the parameters Q and M are set to 2 and 3, respectively, while for the Bank Marketing dataset, they are set to 1 and 2.

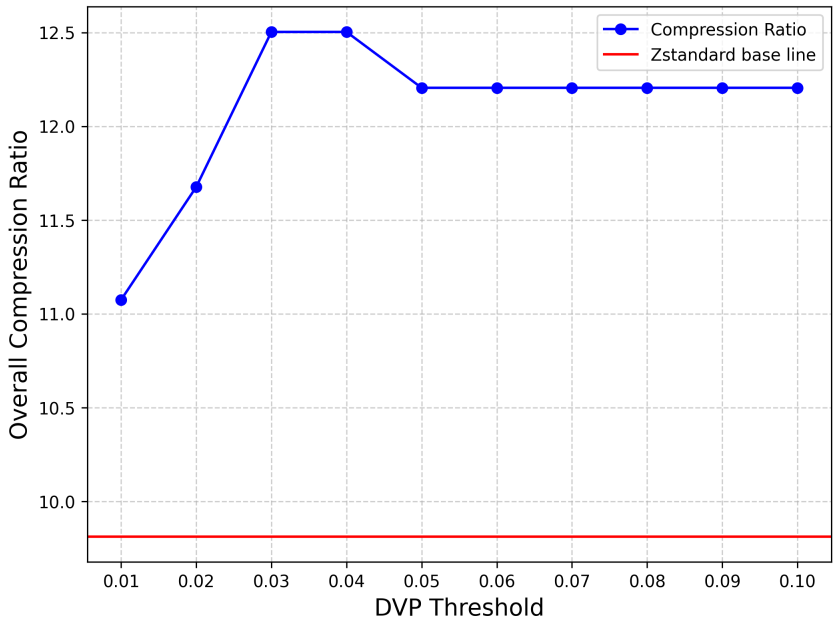


Figure 5.5: Overall Compression Ratio for Census Income Dataset

Figure 5.5 represents the experiment results for the Census Income dataset using DVP-Partitioned table compression. The value of the red line is 9.811. When the DVP threshold is set to 0.03 and 0.04, the optimal table partition is achieved, contributing to an overall compression ratio of 12.503. Compared to the baseline represented by the red line, the reordered table consistently demonstrates a significant improvement in the overall compression ratio, achieving an increase of 27.4%.

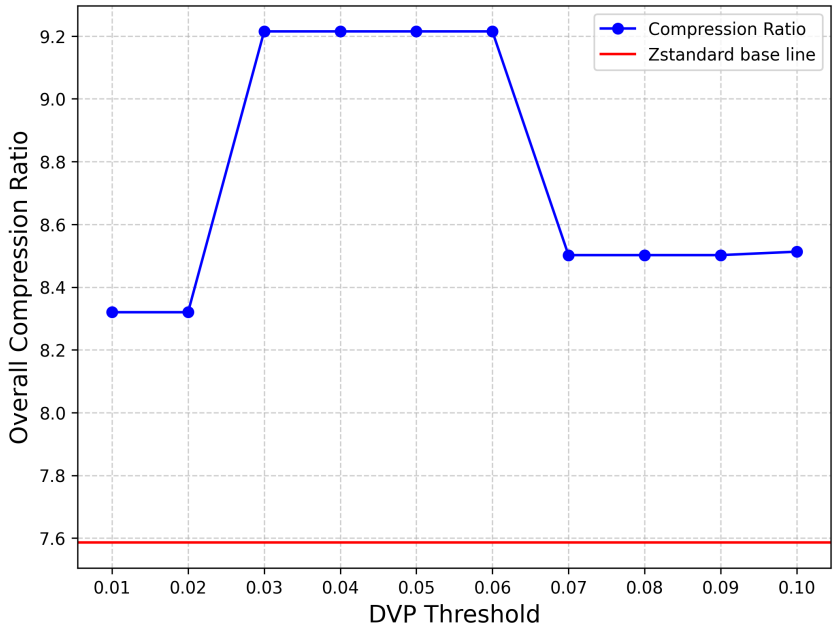


Figure 5.6: Overall Compression Ratio for Bank Marketing Dataset

The baseline compression ratio is recorded in the Bank Marketing dataset at 7.586, as illustrated in Figure 5.6. The maximum overall compression ratio is observed at 9.215, achieved when the DVP threshold ranges from 0.03 to 0.06. With the implementation of DVP-Partitioned table compression, the overall compression ratio significantly increases, notably reaching 21.5

Chapter 6

Conclusions

This thesis is dedicated to enhancing data compression performance for tabular data. We introduce two primary data reordering techniques, complemented by a data representation algorithm, to leverage recurring patterns that often elude computer detection. A novel table partition strategy is devised based on distinct value percentages for each column, further optimizing the overall compression ratio.

Initially, we employed clustering methods for data reordering. Acknowledging the varied data types inherent in tabular data, k-prototype clustering emerged as the preferred technique for reordering tuples. However, exclusive reliance on k-prototype clustering fails to achieve an optimal improvement in compression ratio, as it may overlook many similarities discernible to human perception.

To enhance the identification of repeated data patterns, we integrated Locality-Sensitive Hashing (LSH) into our methodology. This entailed adjusting

the sliding window size and subsequently employing shingling to create a dictionary encompassing all existing data. This dictionary was then utilized to represent each data point, generating vectors. Subsequently, minHashing was applied to reduce the dimensionality of these vectors, ultimately producing high-dimensional vectors, termed signatures.

When the *signature* replaces the original data in the reordering process conducted by k-prototype clustering, many previously disparate yet similar data points are effectively clustered together. This leads to a considerable reduction in the size of files compressed using dictionary-based compression. However, the inherent time complexity of k-prototype clustering poses a significant challenge, rendering its practical application in real-world scenarios challenging. Therefore, an alternative approach to data rearrangement is required to overcome the time-consuming aspect of k-prototype clustering in practical settings.

A heuristic-based TSP solver is employed as another data reordering technique in the thesis to arrange data items based on their similarity. Unlike traditional TSP, which deals with physical entities and spatial relationships, data reordering involves abstract entities representing data points or vectors. Distance calculation is adapted to measure the similarity between these data points rather than physical spatial distances. Historically, TSP is a challenging optimization problem requiring substantial time for large datasets with more data items than cities. To address this, the proposed approach involves partitioning datasets into chunks and solving TSP within each chunk, aiming to improve efficiency.

To address challenges in original order recovery after data reordering and

limitations in combining heuristic-based TSP solver with run-length encoding, we introduce a novel compression strategy, DVP (Distinct Value Percentage)-Partitioned compression. This strategy categorizes columns based on distinct value percentages (DVP), applying a unified compression strategy within each DVP group. Columns in the same group undergo data reordering together using a heuristic-based TSP solver, ensuring row order recovery during decompression. The DVP threshold is adjustable (typically 0.01 to 0.1), and columns are divided into two groups. The larger DVP group undergoes LSH-based preprocessing and data reordering, followed by dictionary-based compression. The smaller DVP group undergoes simpler preprocessing, including ordinal value assignment, followed by run-length encoding and Zstandard compression. This approach aims for optimal compression ratios in tabular data by considering distinct value percentages in individual columns.

Finally, to evaluate the efficiency of all methodologies presented and investigate the factors influencing the advantages of our proposed algorithms, we conduct a variety of experiments on three real datasets. The reordered columns demonstrate a significant improvement in compression ratio with the proposed data representation method (LSH) and reordering techniques (k-prototype and heuristic-based TSP solver). When compared to compression obtained with random column-wise approaches, the DVP-partitioned table compression strategy yields a significant boost in compression ratio, ranging from 20% to 70%.

Bibliography

[1] C. M. Agulhari, I. S. Bonatti, and P. L. D. Peres. "An adaptive run length encoding method for the compression of electrocardiograms". In: *Medical Engineering Amp; Physics* 35 (2 2013), pp. 145–153. DOI: 10.1016/j.medengphy.2010.03.003.

[2] Madyan Alsenwi et al. "Hybrid compression technique with data segmentation for electroencephalography data". In: *2017 29th International Conference on Microelectronics (ICM)*. IEEE. 2017, pp. 1–4.

[3] Barry Becker and Ronny Kohavi. *Adult*. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5XW20. 1996.

[4] M. Behrisch et al. "Matrix reordering methods for table and network visualization". In: *Computer Graphics Forum* 35 (3 2016), pp. 693–716. DOI: 10.1111/cgf.12935.

[5] S. Buettcher. "Information retrieval: implementing and evaluating search engines". In: *Kybernetes* 40 (9/10 2011), pp. 1555–1555. DOI: 10.1108/k.2011.40.9_10.1555.5.

[6] Daniel Cederman and Philippas Tsigas. “A practical quicksort algorithm for graphics processors”. In: *European Symposium on Algorithms*. Springer. 2008, pp. 246–258.

[7] Jianyu Chen, Maurice Daverveldt, and Zaid Al-Ars. “FPGA Acceleration of Zstd Compression Algorithm”. In: *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 2021, pp. 188–191. DOI: 10.1109/IPDPSW52791.2021.00035.

[8] S. Chen and J. Wang. “Vlsi implementation of low-power cost-efficient lossless ecg encoder design for wireless healthcare monitoring application”. In: *Electronics Letters* 49 (2 2013), pp. 91–93. DOI: 10.1049/el.2012.3505.

[9] Suash Deb et al. “Finding approximate solutions of NP-hard optimization and TSP problems using elephant search algorithm”. In: *The Journal of Supercomputing* 72 (2016), pp. 3960–3992.

[10] Xavier Delaunay, Aurélie Courtois, and Flavien Gouillon. “Evaluation of lossless and lossy algorithms for the compression of scientific datasets in netCDF-4 or HDF5 files”. In: *Geoscientific Model Development* 12.9 (2019), pp. 4099–4113.

[11] Edwin van Dellen et al. “Minimum spanning tree analysis of the human connectome”. In: *Human brain mapping* 39.6 (2018), pp. 2455–2471.

[12] M. Farach and M. Thorup. “String matching in lempel—ziv compressed strings”. In: *Algorithmica* 20 (4 1998), pp. 388–404. DOI: 10.1007/pl00009202.

[13] R. G. Gallager. "Variations on a theme by huffman". In: *IEEE Transactions on Information Theory* 24 (6 1978), pp. 668–674. DOI: 10.1109/tit.1978.1055959.

[14] Hasindu Gamaarachchi et al. "Fast nanopore sequencing data analysis with SLOW5". In: *Nature biotechnology* 40.7 (2022), pp. 1026–1029.

[15] Athira Gopinath and M Ravisankar. "Comparison of lossless data compression techniques". In: *2020 International Conference on Inventive Computation Technologies (ICICT)*. IEEE. 2020, pp. 628–633.

[16] A. Gordon-Ross, F. Vahid, and N. Dutt. "Combining code reordering and cache configuration". In: *ACM Transactions on Embedded Computing Systems* 11 (4 2012), pp. 1–20. DOI: 10.1145/2362336.2399177.

[17] Manisha Goyal and Shruti Aggarwal. "A Review on K-Mode Clustering Algorithm." In: *International Journal of Advanced Research in Computer Science* 8.7 (2017).

[18] Alireza Haghighatkhah et al. "Test prioritization in continuous integration environments". In: *Journal of Systems and Software* 146 (2018), pp. 80–98.

[19] A. A. R. Hosseinabadi et al. "Gels-ga: hybrid metaheuristic algorithm for solving multiple travelling salesman problem". In: *2014 14th International Conference on Intelligent Systems Design and Applications* (2014). DOI: 10.1109/isda.2014.7066271.

[20] Mohammad Hosseini. “A survey of data compression algorithms and their applications”. In: *Network Systems Laboratory, School of Computing Science, Simon Fraser University, BC, Canada* (2012).

[21] Zhexue Huang et al. “Clustering large data sets with mixed numeric and categorical values”. In: *Proceedings of the 1st pacific-asia conference on knowledge discovery and data mining,(PAKDD)*. Citeseer. 1997, pp. 21–34.

[22] Abir Jaafar Hussain, Ali Al-Fayadh, and Naeem Radi. “Image compression techniques: A survey in lossless and lossy algorithms”. In: *Neurocomputing* 300 (2018), pp. 44–69.

[23] R. J. S. Al-janabi, S. J. Al-janabi, and Z. H. Toman. “New method for increasing watermarked image quality and security”. In: *Journal of Al-Qadisiyah for Computer Science and Mathematics* 9 (2 2017). DOI: 10.29304/jqcm.2017.9.2.316.

[24] Uthayakumar Jayasankar, Vengattaraman Thirumal, and Dhavachelvan Ponnurangam. “A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications”. In: *Journal of King Saud University-Computer and Information Sciences* 33.2 (2021), pp. 119–140.

[25] H. Jiang and S. Lin. “A rolling hash algorithm and the implementation to lz4 data compression”. In: *IEEE Access* 8 (2020), pp. 35529–35534. DOI: 10.1109/access.2020.2974489.

[26] Yomna Ben Jmaa et al. "An efficient hardware implementation of timsort and mergesort algorithms using high level synthesis". In: *2017 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE. 2017, pp. 580–587.

[27] Nick Johnston et al. "Improved lossy image compression with priming and spatially adaptive bit rates for recurrent networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4385–4393.

[28] B. Kågström and P. Poromaa. "Computing eigenspaces with specified eigenvalues of a regular matrix pair (a, b) and condition estimation: theory, algorithms and software". In: *Numerical Algorithms* 12 (2 1996), pp. 369–407. DOI: 10.1007/bf02142813.

[29] Ahmed Najah Kahdim and Mehdi Ebady Manaa. "Design an efficient internet of things data compression for healthcare applications". In: *Bulletin of Electrical Engineering and Informatics* 11.3 (2022), pp. 1678–1686.

[30] K. Kasmeera, S. P. James, and K. Sreekumar. "Efficient compression of secured images using subservient data and huffman coding". In: *Procedia Technology* 25 (2016), pp. 60–67. DOI: 10.1016/j.protcy.2016.08.081.

[31] Leonard Kaufman and Peter J Rousseeuw. *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, 2009.

[32] Maël Kerbiriou and Rayan Chikhi. "Parallel decompression of gzip-compressed files and random access to DNA sequences". In: *2019 IEEE*

International Parallel and Distributed Processing Symposium Workshops (IPDPSW). IEEE. 2019, pp. 209–217.

[33] S. T. Klein. "Parallel huffman decoding with applications to jpeg files". In: *The Computer Journal* 46 (5 2003), pp. 487–497. DOI: 10.1093/comjnl/46.5.487.

[34] Milan Klöwer et al. "Compressing atmospheric data into its real information content". In: *Nature Computational Science* 1.11 (2021), pp. 713–724.

[35] Ioannis Kontoyiannis. "Pointwise redundancy in lossy data compression and universal lossy data compression". In: *IEEE Transactions on Information Theory* 46.1 (2000), pp. 136–152.

[36] B. Kwon, H. C. Song, and S. H. Lee. "Accurate blind lempel-ziv-77 parameter estimation via 1-d to 2-d data conversion over convolutional neural network". In: *IEEE Access* 8 (2020), pp. 43965–43979. DOI: 10.1109/access.2020.2977827.

[37] D. M. B. Larrotta, D. M. M. Enciso, and A. E. G. Barrera. "Compression of biomedical signals on fpga by dwt and run-length". In: *2010 Ieee Andescon* (2010). DOI: 10.1109/andescon.2010.5633621.

[38] Debra A Lelewer and Daniel S Hirschberg. "Data compression". In: *ACM Computing Surveys (CSUR)* 19.3 (1987), pp. 261–296.

[39] Daniel Lemire, Owen Kaser, and Eduardo Gutarra. "Reordering rows for better compression: Beyond the lexicographic order". In: *ACM Transactions on Database Systems (TODS)* 37.3 (2012), pp. 1–29.

[40] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of massive data sets*. Cambridge university press, 2020.

[41] Y. Li et al. "A nearly loss-less compression technology based on ctp and partial sample points calibration for ofdm signal". In: *Proceedings of the 2015 International Conference on Computer Science and Intelligent Communication* (2015). DOI: 10.2991/csic-15.2015.16.

[42] I. Liiv. "Seriation and matrix reordering methods: an historical overview". In: *Statistical Analysis and Data Mining: The ASA Data Science Journal* 3 (2 2010), pp. 70–91. DOI: 10.1002/sam.10071.

[43] Weiqiang Liu et al. "Data Compression Device Based on Modified LZ4 Algorithm". In: *IEEE Transactions on Consumer Electronics* 64.1 (2018), pp. 110–117. DOI: 10.1109/TCE.2018.2810480.

[44] A. I. Maarala et al. "Distributed hybrid-indexing of compressed pan-genomes for scalable and fast sequence alignment". In: *Plos One* 16 (8 2021), e0255260. DOI: 10.1371/journal.pone.0255260.

[45] B. Madoš, Z. Bilanova, and J. Hurtuk. "rle". In: *Journal of Information and Organizational Sciences* 45 (1 2021), pp. 329–349. DOI: 10.31341/jios.45.1.15.

[46] V. Mäkinen et al. "Storage and retrieval of highly repetitive sequence collections". In: *Journal of Computational Biology* 17 (3 2010), pp. 281–308. DOI: 10.1089/cmb.2009.0169.

[47] Emanuele Manca et al. “CUDA-quicksort: an improved GPU-based implementation of quicksort”. In: *Concurrency and computation: practice and experience* 28.1 (2016), pp. 21–43.

[48] R. Matai, S. P. Singh, and M. L. Mittal. “Traveling salesman problem: an overview of applications, formulations, and solution approaches”. In: *Traveling Salesman Problem, Theory and Applications* (2010). DOI: 10.5772/12909.

[49] Dipti Mathpal, Mittal Darji, and Sarangi Mehta. “A research paper on lossless data compression techniques”. In: *International Journal for Innovative Research in Science & Technology* 4.1 (2017), pp. 190–194.

[50] Simeon Mayala et al. “Brain tumor segmentation based on minimum spanning tree”. In: *Frontiers in Signal Processing* 2 (2022), p. 816186.

[51] A. Moffat, R. M. Neal, and I. H. Witten. “Arithmetic coding revisited”. In: *ACM Transactions on Information Systems* 16 (3 1998), pp. 256–294. DOI: 10.1145/290159.290162.

[52] S. Moro, P. Rita, and P. Cortez. *Bank Marketing*. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5K306. 2012.

[53] Nishant Verma Nehal Birla and Nikhil Kushwaha. *Vehicle Dataset*. Kaggle. DOI: https://www.kaggle.com/datasets/nehalbirla/vehicle-dataset-from-cardekho. 2023.

[54] Y. Niu and D. Zhang. “A randomness ant colony algorithm for solving tsp”. In: *DEStech Transactions on Computer Science and Engineering* (cnsce 2017). DOI: 10.12783/dtcse/cnsce2017/8904.

[55] J. Park and S. Kang. "A twin symbol encoding technique based on run-length for efficient test data compression". In: *ETRI Journal* 33 (1 2011), pp. 140–143. DOI: 10.4218/etrij.11.0210.0154.

[56] Himali Patel et al. "Survey of lossless data compression algorithms". In: *International Journal of Engineering Research and Technolgy* 4.4 (2015), pp. 926–929.

[57] WD Pence, Robert Seaman, and RL White. "Lossless astronomical image compression and the effects of noise". In: *Publications of the Astronomical Society of the Pacific* 121.878 (2009), p. 414.

[58] Barbara Penna et al. "Transform coding techniques for lossy hyperspectral data compression". In: *IEEE Transactions on Geoscience and Remote Sensing* 45.5 (2007), pp. 1408–1421.

[59] C. Périn, P. Dragicevic, and J. Fekete. "Revisiting bertin matrices: new interactions for crafting tabular visualizations". In: *IEEE Transactions on Visualization and Computer Graphics* 20 (12 2014), pp. 2082–2091. DOI: 10.1109/tvcg.2014.2346279.

[60] Meikel Pöss and Dmitry Potapov. "Data compression in oracle". In: *Proceedings 2003 VLDB Conference*. Elsevier. 2003, pp. 937–947.

[61] Mohammad Harun Rashid and Miguel A. Mosteiro. "A Greedy-Genetic Local-Search Heuristic for the Traveling Salesman Problem". In: *2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous*

Computing and Communications (ISPA/IUCC). 2017, pp. 868–872. DOI: 10.1109/ISPA/IUCC.2017.00132.

[62] Mark A. Roth and Scott J. Van Horn. "Database compression". In: *SIGMOD Rec.* 22.3 (Sept. 1993), pp. 31–39. ISSN: 0163-5808. DOI: 10.1145/163090.163096. URL: https://doi.org/10.1145/163090.163096.

[63] Sanket Salvi et al. "An encryption, compression and key(ECK) management based data security framework for infrastructure as a service in Cloud". In: *2015 IEEE International Advance Computing Conference (IACC)*. 2015, pp. 872–876. DOI: 10.1109/IADCC.2015.7154830.

[64] Ahmad Saeed Shah and Muhammad Athar Javed Sethi. "The improvised GZIP, a technique for real time lossless data compression". In: *EAI Endorsed Transactions on Context-aware Systems and Applications* 6.17 (2019), e5–e5.

[65] Mushfeq-Us-Saleheen Shameem and Raihana Ferdous. "An efficient k-means algorithm integrated with Jaccard distance measure for document clustering". In: *2009 First Asian Himalayas International Conference on Internet*. 2009, pp. 1–6. DOI: 10.1109/AHICI.2009.5340335.

[66] Y. Shan et al. "An enhanced run-length encoding compression method for telemetry data". In: *Metrology and Measurement Systems* 24 (3 2017), pp. 551–562. DOI: 10.1515/mms-2017-0039.

[67] Jia Shi. "Column partition and permutation for run length encoding in columnar databases". In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2020, pp. 2873–2874.

[68] S. Shu and Y. Shu. "A two-stage data compression method for real-time database". In: *2012 3rd International Conference on System Science, Engineering Design and Manufacturing Informatization* (2012). DOI: 10.1109/icssem.2012.6340844.

[69] Ravid Shwartz-Ziv and Amitai Armon. "Tabular data: Deep learning is not all you need". In: *Information Fusion* 81 (2022), pp. 84–90.

[70] C. G. d. Silva et al. "A fast feature vector approach for revealing simplex and equi-correlation data patterns in reorderable matrices". In: *Information Visualization* 16 (4 2016), pp. 261–274. DOI: 10.1177/1473871616666392.

[71] S. Song et al. "A lossless compression method for logging data while drilling". In: *Systems Science Amp; Control Engineering* 9 (1 2021), pp. 689–703. DOI: 10.1080/21642583.2021.1981478.

[72] Mike Stonebraker et al. "C-store: a column-oriented DBMS". In: *Making Databases Work: the Pragmatic Wisdom of Michael Stonebraker*. 2018, pp. 491–518.

[73] F. Sven and P. Wolf. "Improving run length encoding by preprocessing". In: *2021 Data Compression Conference (DCC)* (2021). DOI: 10.1109/dcc50243.2021.00051.

[74] D. Takafuji et al. "Gpu implementations of deflate encoding and decoding". In: *Concurrency and Computation: Practice and Experience* 35 (19 2022). DOI: 10.1002/cpe.7454.

[75] *TPC-H Benchmark*. http://www.tpc.org/tpch/. Accessed: May 2nd, 2022.

[76] M. Weinberger, G. Seroussi, and G. Sapiro. "The loco-i lossless image compression algorithm: principles and standardization into jpeg-ls". In: *IEEE Transactions on Image Processing* 9 (8 2000), pp. 1309–1324. DOI: 10.1109/83.855427.

[77] B. A. Wijaya et al. "Application of huffman algorithm and unary codes for text file compression". In: *SinkrOn* 7 (3 2022), pp. 1000–1007. DOI: 10.33395/sinkron.v7i3.11567.

[78] P. Wu et al. "An improved two-dimensional run-length encoding scheme and its application". In: *International Journal of Signal Processing, Image Processing and Pattern Recognition* 9 (4 2016), pp. 339–346. DOI: 10.14257/ijsip.2016.9.4.30.

[79] Dongkuan Xu and Yingjie Tian. "A comprehensive survey of clustering algorithms". In: *Annals of Data Science* 2 (2015), pp. 165–193.

[80] H. Yamamoto, M. Tsuchihashi, and J. Honda. "Almost instantaneous fixed-to-variable length codes". In: *IEEE Transactions on Information Theory* 61 (12 2015), pp. 6432–6443. DOI: 10.1109/tit.2015.2492961.

[81] B. Ye et al. "Test data compression using alternating variable run-length code". In: *Integration* 44 (2 2011), pp. 103–110. DOI: 10.1016/j.vlsi.2010.11.004.

[82] Mehrdad Hosseini Zadeh, David Wang, and Eric Kubica. “Perception-based lossy haptic compression considerations for velocity-based interactions”. In: *Multimedia Systems* 13 (2008), pp. 275–282.

[83] H. Zhan. “Image compression and reconstruction based on gui and huffman coding”. In: *Journal of Physics: Conference Series* 2580 (1 2023), p. 012025. DOI: 10.1088/1742-6596/2580/1/012025.