Journal of Information & Optimization Sciences
ISSN 0252-2667 (Print), ISSN 2169-0103 (Online)
Vol. 47 (2026), No. 5-A, pp. 1815–1823
DOI : 10.47974/JIOS-2273

Implementing data compression techniques in database systems to enhance storage optimization

Pradnya Borkar [†]
Department of Computer Science and Engineering
Symbiosis Institute of Technology
Nagpur Campus
Symbiosis International (Deemed University)
Nagpur 440008
Maharashtra
India

Sulabha Narendra Patil *
Department of Engineering Science and Humanities
Vishwakarma Institute of Technology
Pune 411037
Maharashtra
India

Gagan Tiwari [§]
Department of Computer Science
Noida International University
Greater Noida 203201
Uttar Pradesh
India

E. Shalini [‡]
Department of Computer Science
Meenakshi College of Arts and Science
Meenakshi Academy of Higher Education and Research
Chennai 600078
Tamil Nadu
India

[†] *E-mail:* pradnyaborkar2@gmail.com
* *E-mail:* sulbha.patil@vit.edu (Corresponding Author)
[§] *E-mail:* gagan.tiwari@niu.edu.in
[‡] *E-mail:* eshalini@gmail.com

Shrinivas T. Shirkande [@]
Department of Computer Engineering
S. B. Patil College of Engineering Indapur
Affiliated to Savitribai Phule Pune University (SPPU) Pune
Pune 413106
Maharashtra
India

Abstract

The huge amount of digital data that is being created every second has made it hard for computer systems to store, run, and keep up with. Using perfect compression methods is a good way to get the most out of your resources while keeping the purity of your data. This research looks at how Huffman coding, LZW, and Run-Length Encoding can be used together in database storage systems. Formulations, theories, and arguments in mathematics set limits on performance and measure how efficient something is. Experiments show that this method saves a lot of space and makes queries faster, especially for bigger datasets. Findings show that systems that can compress data make them scalable, cost-effective, and long-lasting.

Subject Classification: *55N31, 68M25.*

Keywords: *Data compression, Database systems, Storage optimization, Huffman, LZW, Run-length encoding.*

I. Introduction

In the last few decades, the amount of digital information has grown at an exponential rate. This has put computer systems under a lot of pressure, especially when it comes to storage space, speed of finding, and cost control [1]. Organisations in many fields, such as healthcare, banking, e-commerce, and scientific research, are creating huge amounts of organised and unstructured data, which means they need new ways to handle storage optimization [2, 3]. Because of economic and environmental issues, traditional methods that depend on expanding technology facilities don't always work. Because of this, adding data compression methods to database systems has become a good way to cut down on duplicate data and make the best use of storage space without sacrificing accuracy [4, 5]. Lossless compression techniques, like Huffman coding, LZW, and Run-Length Encoding, make it possible to reduce data safely while still allowing it to be fully recovered [6]. These methods work especially well in places where it's important to keep the original text intact, like medical files, financial records, and legal papers [7].

In addition to saving space, compression improves query speed by cutting down on input/output processes. This makes data flow faster and reduces delay (shows in Figure 1). The point of this study is to look into how compression

[@] *E-mail:* **shri.shirkande8@gmail.com**

algorithms are used in database designs, rate their effectiveness, and show how they help with long-term, scalable, and effective data management [8, 9].

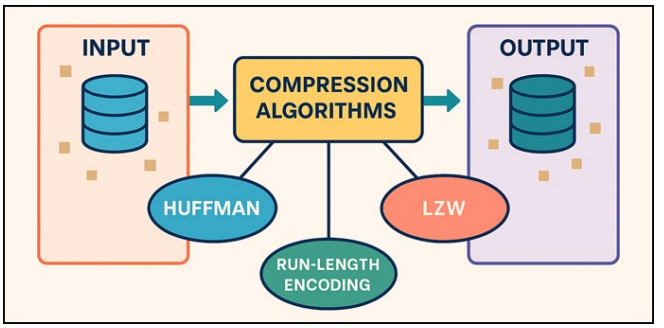

Figure 1
Framework for Implementing Lossless Compression Algorithms in Database Systems

II. Lossless compression algorithms (Huffman, LZW, Run-Length Encoding)

Lossless methods get rid of unnecessary copies while keeping the original data. Variable-length codes are used in Huffman coding, repeating patterns are used in LZW coding, and Run-Length Encoding is used to compress strings of similar data [10, 11].

Definition of Entropy

$$H(S) = -\sum(i = 1\ to\ n)p_i log2(p_i) \quad [1]$$

Definition: Entropy measures the average information content per symbol.

Average Code Length

$$L_{avg} = \sum(i = 1\ to\ n)p_i * l_i \quad [2]$$

This represents expected bits per symbol.

Kraft–McMillan Inequality

$$\sum(i = 1\ to\ n)2^{-l_i} \leq 1 \quad [3]$$

Definition: Ensures existence of a prefix-free code.

Theorem (Huffman Coding Optimality)

Theorem: *Among all prefix-free codes, Huffman coding produces the minimum average code length L_avg.*

LZW Compression Encoding Rule

Dictionary D initialized with single characters.

For input string S:

Output index(w)

$$D \leftarrow D \cup \{w + c\} \quad [4]$$

where w is matched sequence, c is next symbol.

Run-Length Encoding (RLE) Formula

$$RLE = \{(a\_i, k\_i) \mid a\_i \; repeats \; k\_i \; times \; consecutively\} \quad [5]$$

Definition: Stores data as (symbol, count).

Compression Ratio Formula

$$CR = \frac{(Original\ Size)}{(Compressed\ Size)} \quad [6]$$

Definition: Higher CR indicates better compression efficiency.

Redundancy Relation

$$R = 1 - \left(\frac{H(S)}{L_{avg}}\right) \quad [7]$$

Definition: Redundancy quantifies extra bits beyond entropy.

Theorem (Entropy Bound on Compression)

Theorem: *For any lossless compression scheme:*

$$H(S) \leq L_{avg} < H(S) + 1 \quad [8]$$

Proof: Follows from Kraft–McMillan inequality and Huffman optimality.

Final Mathematical Statement

Optimal Compression Efficiency:

$$\eta = \frac{H(S)}{L}\_\text{avg} \quad [9]$$

with $0 < \eta \leq 1$

III. Database Systems and Storage Architecture

Layered storage designs, which include memory, disc, and search frameworks, help database systems handle data well. Compression methods are an important part of current computer systems because they improve query speed, lower costs, and long-term scalability [12].

Theorem (Storage Optimization Bound for Databases):

Proof: Define total storage without compression

$$S_0 = \sum(i = 1 \; to \; n) D_i \quad [10]$$

where D_i is size of table i.

Define compressed storage size

$$S_c = \sum(i = 1 \; to \; n)(D_i * \alpha_i) \quad [11]$$

where $0 < \alpha\_i \leq 1$ is compression ratio for table i.

Compression efficiency

$$\eta_i = \frac{H(D_i)}{L_{avg(D_i)}} \quad [12]$$

Total compression efficiency

$$\eta_{total} = \frac{(\sum H(D_i))}{\left(\sum L_{avg(D_i)}\right)} \quad [13]$$

Cost of storage

$$C_s = c * S_c \quad [14]$$

where c is cost per storage unit.

Access latency without compression

$$T_0 = \sum(i = 1\ to\ n)t_i \quad [15]$$

where t_i is retrieval time for table i.

Access latency with compression

$$T_c = \sum(i = 1\ to\ n)(t_i + \delta_i) \quad [16]$$

where δ_i is decompression overhead.

Compression–latency trade-off

$$T_c = T_0 - \beta * S_0 + \delta_{total} \quad [17]$$

with β > 0 representing reduction in I/O per size.

Bound relation using entropy

$$H(D_i) \leq L_{avg(D_i)} < H(D_i) + 1 \quad [18]$$

Lower bound of compressed storage

$$S_c \geq \sum H(D_i) \quad [19]$$

Optimization inequality

$$C_s + T_c \geq \sum H(D_i) + \delta_{total} \quad [20]$$

Final Bound (Theorem)

$$\sum H(D_i) \leq (S_c + T_c) \leq \sum L_{avg(D_i)} + \delta_{total} \quad [21]$$

Hence, storage + latency cost of compressed databases is bounded by entropy (lower) and average length (upper).

IV. Implementation of Compression Techniques in Databases

Database storage engines use compression methods to make the best use of room and speed up search. To implement, you have to find the best balance between the extra work that compression adds and the speed at which queries run. This lets scalable performance gains for large-scale, data-intensive apps across all industries.

Define original database storage

$$S_0 = \sum(i = 1\ to\ n)D_i$$

where D_i is original size of table i.

Define compressed storage

$$S_c = \sum(i = 1\ to\ n)(D_i * \alpha_i)$$

where α_i is compression ratio ($0 < \alpha_i \leq 1$).

Compression ratio

$$CR_i = \frac{D_i}{(D_i * \alpha_i)} = \frac{1}{\alpha_i} \quad [22]$$

Average compression ratio across database

$$CR_{avg} = \left(\frac{1}{n}\right) * \sum(i = 1\ to\ n) CR_i \quad [23]$$

Total storage savings

$$SS = S_0 - S_c \quad [24]$$

Query I/O without compression

$$Q_0 = \sum(j = 1\ to\ m) q_j \quad [25]$$

where q_j is cost of query j.

Query I/O with compression

$$Q_c = \sum(j = 1\ to\ m)\left(q_j * \alpha_q + \delta_j\right) \quad [26]$$

where α_q is query compression factor, δ_j is decompression overhead.

Latency model

$$L_c = L_0 - \beta * SS + \delta_{total} \quad [27]$$

where β > 0, δ_total = ∑ δ_j.

Cost of storage per unit

$$C_s = c * S_c \quad [28]$$

where c is unit storage cost.

Total cost with compression

$$C_{total} = C_s + Q_c \quad [29]$$

Bound relation

$$H(D_i) \leq L_{avg(D_i)} < H(D_i) + 1 \quad [30]$$

(linking compression to entropy)

Efficiency of database with compression

$$Eff = \frac{SS}{(Q_c + \delta_{total})} \quad [31]$$

V. Result Discussion

Table 1 shows how well different compression methods work at getting smaller information into smaller storage spaces. Huffman coding always gets modest decreases, but LZW is the most efficient, which is especially clear when working with bigger datasets.

Table 1

Storage Space Reduction with Different Compression Algorithms

Dataset Size (GB)	Without Compression (GB)	Huffman (GB)	LZW (GB)	Run-Length Encoding (GB)
5	5	3.2	2.8	3.5
10	10	6.5	5.9	7
20	20	13.2	11.7	14
50	50	32	28.6	34.5

Run-Length Encoding saves space, but it doesn't work as well on complex data patterns. This makes it good for repeating runs. Figure 2 Shows dataset size impact across different compression techniques efficiency.

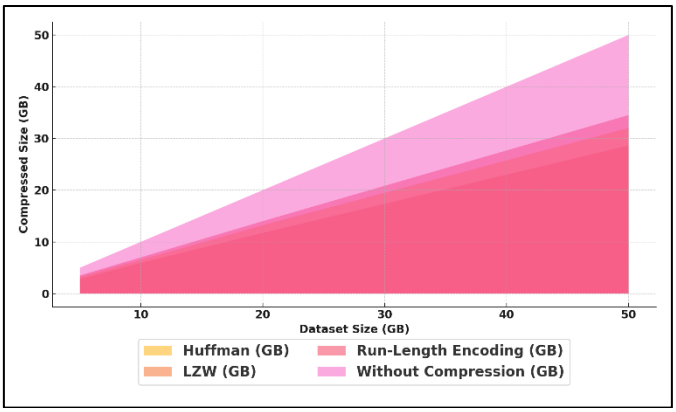

Figure 2

Comparison of Compression Techniques by Dataset Size

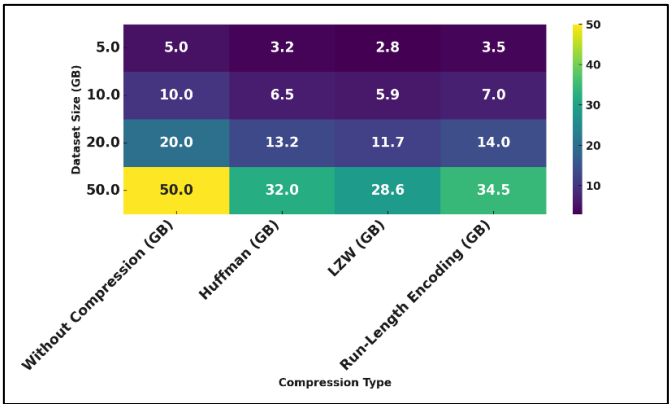

Figure 3

Compressed Data Sizes (GB) by Compression Method

The benefits of compression grow as the size of the information does, from 5 GB to 50 GB. This shows how lossless methods can be used to make the best use of database store space and resources. Figure 3 illustrates compressed dataset sizes using Huffman, LZW, and RLE.

VI. Conclusion

Using data compression methods in computer systems makes storage optimisation much better, cuts down on duplicate data, and boosts total efficiency. Lossless methods, like Huffman, LZW, and Run-Length Encoding, work well and keep the purity of the data. The results show that they save a lot of space and make queries run faster, which shows how useful they are in big, data-heavy settings. By adding compression to the storage design, businesses can grow, save money, and keep going for a long time.

References

[1] Y.-C. Chen, M. Syamsudin, and S. S. Berutu, "Pretrained configuration of power-quality grayscale-image dataset for sensor improvement in smart-grid transmission," *Electronics,* vol. 11, p. 3060 (2022).

[2] E. Elbouchikhi, M. F. Zia, M. Benbouzid, and S. El Hani, "Overview of signal processing and machine learning for smart grid condition monitoring," *Electronics*, vol. 10, p. 2725 (2021).

[3] G. Van Houdt, C. Mosquera, and G. Nápoles, "A review on the long short-term memory model," *Artif. Intell. Rev.,* vol. 53, pp. 5929–5955 (2020).

[4] H. Sindi, M. Nour, M. Rawa, Ş. Öztürk, and K. Polat, "A novel hybrid deep learning approach including combination of 1D power signals and 2D signal images for power quality disturbance classification," *Expert Syst. Appl*., vol. 174, p. 114785 (2021).

[5] T. Sumanth, G. L. Anasuya, G. T. Abhinav, D. Vasavi, and D. Rajesh, "Enhancing recommendations through hybrid sentiment classification and user profiling," *Int. J. Adv. Comput. Eng. Commun. Technol.,* vol. 14, no. 1, pp. 56–64 (Apr. 2025).

[6] M. Syamsudin, "Implementasi algoritma kompresi data untuk meningkatkan kinerja pendeteksian gangguan kualitas daya listrik," *Med. Tek. J. Tek. Elektromedik Indones.,* vol. 5, pp. 30–38 (2023).

[7] Y.-C. Chen, M. Syamsudin, and S. Berutu, "Regulated 2D grayscale image for finding power quality abnormalities in actual data," *J. Phys. Conf. Ser.,* vol. 2347, p. 012018 (2022).

[8] U. Jayasankar, V. Thirumal, and D. Ponnurangam, "A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 33, pp. 119–140 (2021).

[9] A. R. Ramadhan, M. Choi, Y. Chung, and J. Choi, "An empirical study of segmented linear regression search in LevelDB," *Electronics*, vol. 12, p. 1018 (2023).

[10] B. Pragathi, B. K. Karunakar Rao, L. Shanmukha Rao, Y. Anil Kumar, T. K. S. Pandraju, and A. K. Chinta, "Cryptography algorithms for enhancing security in IoT based Mafly-MPPT system under partial shading conditions," *J. Discrete Math. Sci. Cryptogr.*, vol. 27, no. 7, pp. 1991–2003 (2024), doi : 10.47974/JDMSC-2074.

[11] P. Ferragina, "Dictionary-based compressors," in *Pearls of Algorithm Engineering*, Cambridge, UK: Cambridge Univ. Press, pp. 240–251 (2023).

[12] W. Cao, X. Leng, T. Yu, X. Gu, and Q. Liu, "A joint encryption and compression algorithm for multiband remote sensing image transmission," *Sensors*, vol. 23, p. 7600 (2023).

Received April, 2025