A New Algorithm for Data Compression

Philip Gage - 1994

Zaranaben Savani
Marina Merzliakova
Tulasi rajgopal
Om Borda

Meeting the Data Compression Challenge

How evolving needs drive innovative algorithms

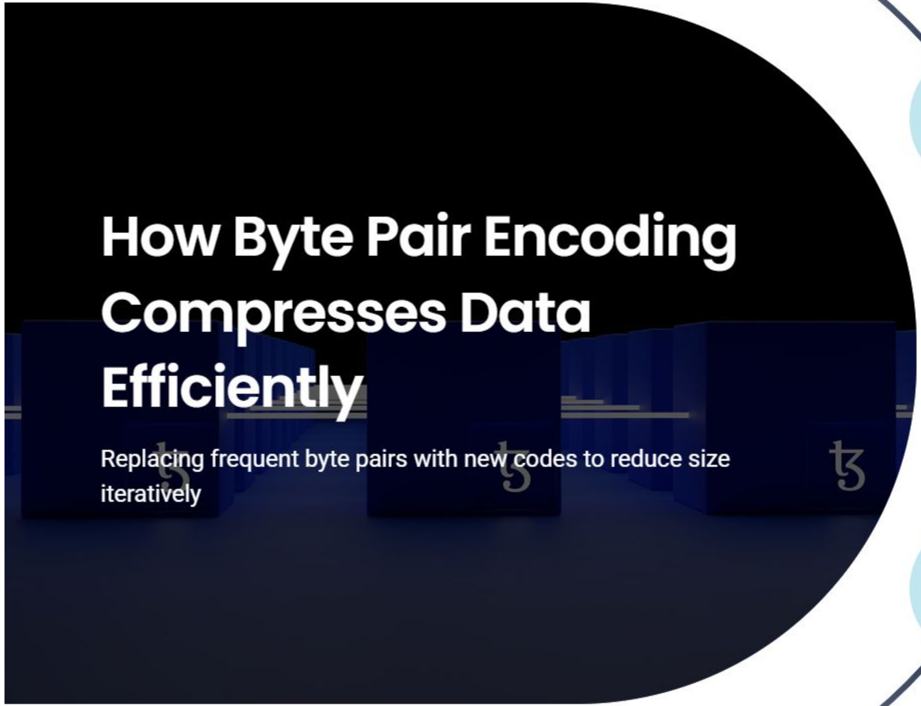
In 1990, growing information volumes created urgent demand for efficient data compression

Traditional methods like **Huffman coding** and **Lempel-Ziv-Welch (LZW)** effectively reduce data size

Increasing data complexity requires newer algorithms to boost compression without losing speed or efficiency

This presentation introduces **Byte Pair Encoding (BPE)**, a novel algorithm simplifying compression while maintaining effectiveness

Understanding this background highlights BPE’s innovative contribution to data compression



How Byte Pair Encoding Compresses Data Efficiently

Replacing frequent byte pairs with new codes to reduce size iteratively

Puzzle

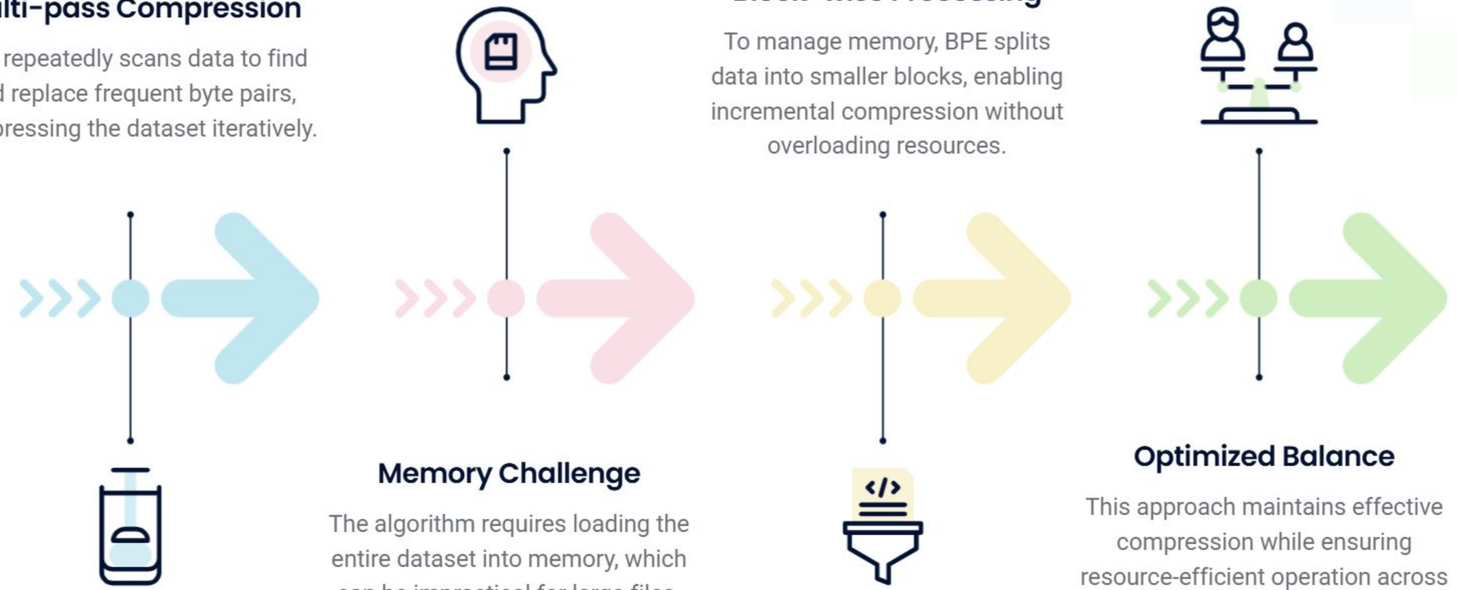
Byte Pair Encoding fits repeated byte pairs together like puzzle pieces, simplifying data by substituting frequent patterns with new codes.

Frequency

BPE leverages frequency insights to innovatively compress data through straightforward substitution, illuminating efficient compression methods.

Understanding Byte Pair Encoding (BPE)

How BPE balances compression power and memory management



Multi-pass Compression

BPE repeatedly scans data to find and replace frequent byte pairs, compressing the dataset iteratively.

Memory Challenge

The algorithm requires loading the entire dataset into memory, which can be impractical for large files.

Block-wise Processing

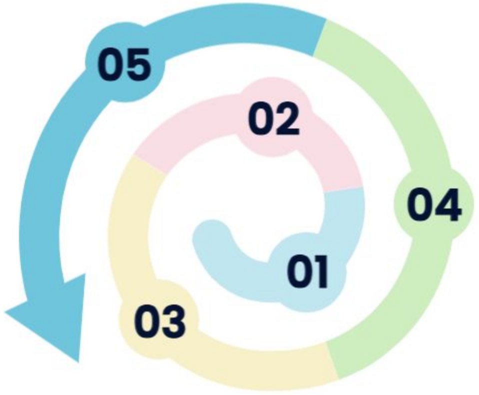
To manage memory, BPE splits data into smaller blocks, enabling incremental compression without overloading resources.

Optimized Balance

This approach maintains effective compression while ensuring resource-efficient operation across diverse applications.

Streamlining Data Compression

Step-by-step process for efficient byte pair encoding



Read Input Data

Load the data block to be compressed.

Count Byte Pairs

Identify and tally occurrences of byte pairs.

Replace Frequent Pairs

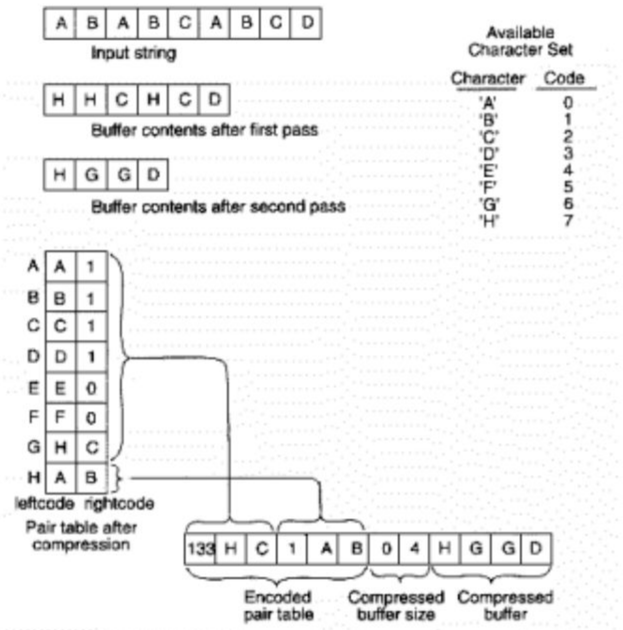
Substitute the most common pairs with new codes.

Update Counts

Adjust byte pair counts to reflect recent substitutions.

Write Compressed Output

Save the transformed data to the output stream.



Optimize Compression with Key Data Structures and Parameters

Understand how arrays and configurable settings drive BPE performance

Arrays left, right, and count track byte pair occurrences and substitutions for efficient compression

BLOCKSIZE defines the size of data blocks processed, impacting memory and speed

HASHSIZE sets the hash table size for rapid byte pair lookup and substitution

THRESHOLD determines the minimum frequency a byte pair must reach to be substituted

Adjusting these parameters balances compression speed and ratio to fit resource constraints and goals

How the Pair Table Boosts Compression Efficiency

Understanding byte pair substitutions and run-length encoding in BPE

01

Stores Byte Pair Substitutions

Holds all byte pair substitutions created during compression to track transformation rules.

02

Run-Length Encoding with Counts

Uses positive and negative counts to efficiently represent repeated sequences, reducing redundancy.

03

Space-Saving Encoding

Stores substitution codes using two bytes per code instead of three, optimizing memory usage.

04

Enables Faster, Smaller Compression

The pair table's efficient structure drives BPE's speed and smaller compressed file sizes.

Expansion

A single-pass stack-based algorithm simplifies decompression with minimal memory use

Input Sources

Use the compressed file and a stack holding pending expansions as inputs.

Direct Output of Literals

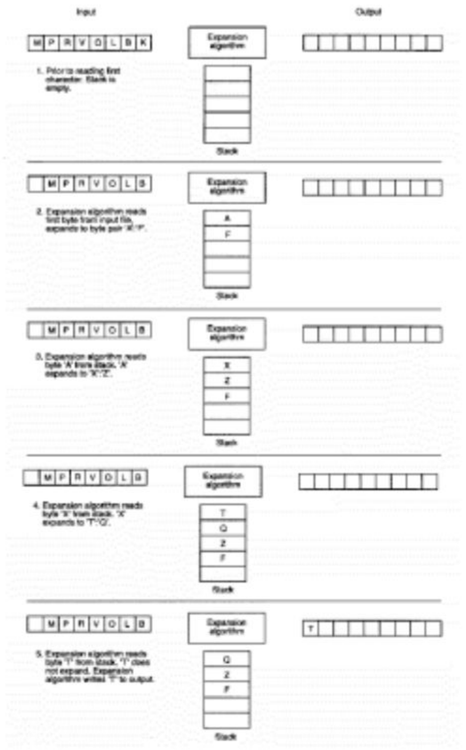
Literals are output immediately without further processing.

Expand Pairs via Stack

Pairs are decomposed by pushing their components onto the stack for further expansion.

Efficient Decompression

This single-pass algorithm reverses compression efficiently while minimizing memory consumption.


Input
Output
Expansion algorithm
1. Prior to reading first character. Stack is empty.
Stack
2. Expansion algorithm reads first byte from input file, expands to byte pair 'A','F'.
3. Expansion algorithm reads byte 'A' from stack. 'A' expands to 'X','Z'.
4. Expansion algorithm reads byte 'X' from stack. 'X' expands to 'T','Q'.
5. Expansion algorithm reads byte 'T' from stack. 'T' does not expand. Expansion algorithm writes 'T' to output.

Advantages of BPE

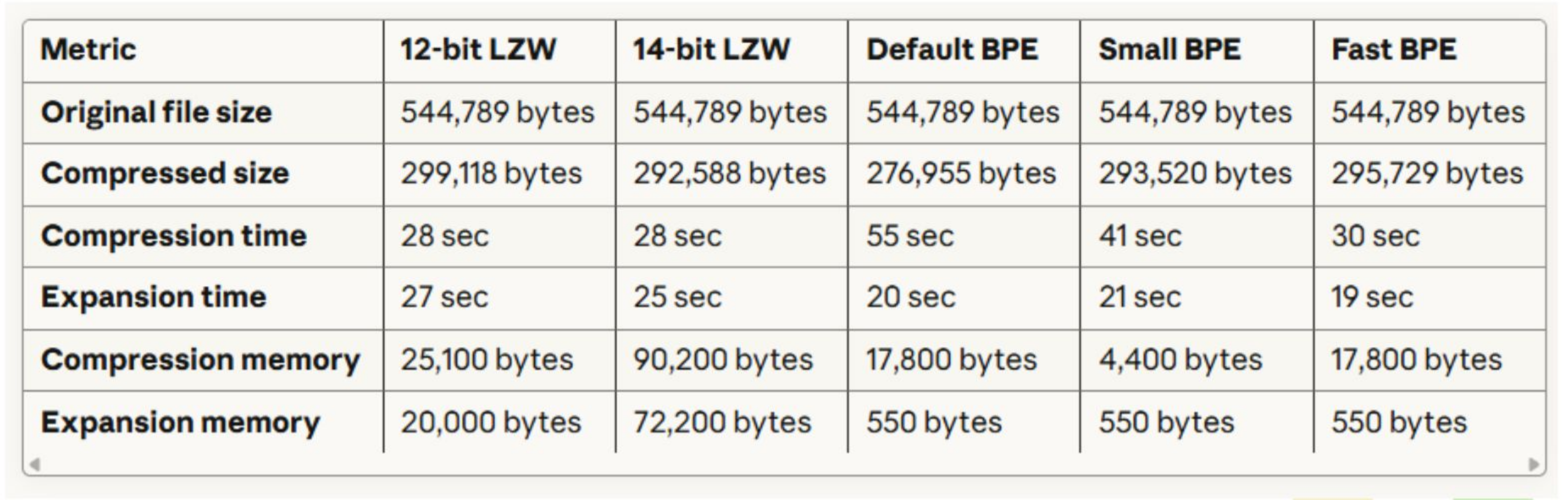
Fast, lightweight, and ideal for resource-limited environments

Never Increases Data Size If Compression Isn't Possible, BPE Just Passes Data Through (Almost) Unchanged Only Adds A Few Header Bytes Per Block

Exponential Adaptation (Vs LZW's Linear)
LZW Builds Patterns One Character At A Time
BPE Can Nest Pairs Within Pairs, Creating Much Longer Strings Quickly

Tiny Memory Footprint
Compression: 5-30KB
Expansion: Only 550 Bytes The Expansion Code Is So Simple It Could Fit In 2KB Total (Code + Data) In Assembly

Comparing Compression Algorithms:
BPE vs LZW



Metric	12-bit LZW	14-bit LZW	Default BPE	Small BPE	Fast BPE
Original file size	544,789 bytes	544,789 bytes	544,789 bytes	544,789 bytes	544,789 bytes
Compressed size	299,118 bytes	292,588 bytes	276,955 bytes	293,520 bytes	295,729 bytes
Compression time	28 sec	28 sec	55 sec	41 sec	30 sec
Expansion time	27 sec	25 sec	20 sec	21 sec	19 sec
Compression memory	25,100 bytes	90,200 bytes	17,800 bytes	4,400 bytes	17,800 bytes
Expansion memory	20,000 bytes	72,200 bytes	550 bytes	550 bytes	550 bytes

Evaluating efficiency and speed trade-offs for optimal data handling

Enhancing BPE

could be enhanced by block size optimization

Listing 2 Expansion algorithm (pseudocode)

```
While not end of file
   Read pair table from input
   While more data in block
      If stack empty, read byte from input
      Else pop byte from stack
      If byte in table, push pair on stack
      Else write byte to output
   End while
End while
```

The block size is critical to both the compression ratio and speed

Large blocks = better for text files Small blocks = better for binary files

Conclusion: Why BPE is Surprising and Useful

Key takeaways on Byte Pair Encoding's strengths and limitations

01 **Works well despite simplicity**

02 **No fancy techniques needed**

03 **Best for self-extracting programs, image display, communication links, embedded systems**

04 **Advantages: tiny expansion code, low memory, adjustable performance, handles worst-case data**

05 **Disadvantages: slow compression, less compression than some algorithms**

06 **Final thought: valuable for fast expansion with minimal memory**

The End