Data Compression Technique Report

Assignment: Data Mining and Data Warehousing (DMDW)

GitHub Link: https://github.com/Ashutoshtiwari898/DMDW

Team Members:

- Ashutosh Kumar Tiwari (22052369)
- Jaideep Bose (22052389)
- Shubham Kumar (22054172)
- Divyansh Raj (22052380)

Submission Details:

- Course: Data Mining and Data Warehousing (DMDW)
- Assignment Title: Transaction Data Compression
- Submission Date: 04/09/2024

1. Introduction

This report outlines the data compression technique implemented in the provided C++ code. The code aims to reduce the size of transaction datasets by identifying and replacing repetitive sequences of items with unique indices. This report details the algorithm used, the approach to compression, and how the method ensures lossless compression. Additionally, potential optimizations, future improvements, and scalability considerations are discussed.

2. Algorithm and Approach Used

The algorithm used in the code is a dictionary-based substitution method that involves several key steps: reading transaction data, identifying sequences of different lengths, replacing these sequences with unique indices, removing unused sequences, and finally saving the results. Here's a breakdown of the approach:

a. Reading Transactions:

The code begins by reading transaction data from a file. Each line represents a transaction consisting of item IDs.

b. Sequence Identification:

The code identifies sequences of items within each transaction for sequence lengths of 5, 4, and 3. The sequences are stored in an unordered_map, where the sequence string is the key and an associated unique index is the value.

c. Replacing Sequences:

In the transactions, identified sequences are replaced by their corresponding indices, starting from the longest sequences (length 5) down to the shortest (length 3).

d. Removing Unused Sequences:

Once the replacement is complete, the code removes any sequences from the map that were not actually used in the dataset.

e. Saving Results:

The updated transaction data and sequence map are saved to files. A size reduction analysis is also performed.

3. Compression Technique

The compression technique employed is a dictionary-based substitution method. It identifies repetitive patterns (or sequences) in the data and replaces them with shorter representations (indices), thereby reducing redundancy and compressing the data.

Why is this technique used?

The method is particularly effective for transaction data, where certain combinations of items often repeat. By exploiting this redundancy, the size of the dataset can be significantly reduced. The rolling hash function ensures that sequence identification is efficient and scalable.

4. Lossless Compression

The technique implemented in the code is a lossless compression method. This means that the original data can be perfectly reconstructed from the compressed data, ensuring no loss of information.

Why is it better than lossy compression?

In applications where data integrity is critical, such as financial records or medical data, lossless compression is essential. It ensures that every bit of information is preserved, making the technique highly reliable.

5. Optimization and Future Improvements

While the current implementation is efficient, there are several areas where further optimization and improvements could be made:

a. Dynamic Sequence Length Adjustment:

The code could dynamically adjust sequence lengths based on the characteristics of the data, potentially identifying more efficient patterns.

b. Parallel Processing:

For larger datasets, parallelizing sequence identification and replacement could significantly reduce computation time.

c. Advanced Compression Techniques:

Incorporating techniques such as frequent pattern mining or entropy encoding could further enhance the compression ratio.

6. Compression Results

Using th6 proposed approach, we were able to achieve approximately 69.4792% compression on the small-sized dataset. Here is a comparison of the storage costs:

Original size: 342294 bytes

Reduced size: 104471 bytes

Size reduction: 237823 bytes

Reduction percentage: 69.4792%

This translates to a compression ratio of 69.4792%, significantly reducing the storage requirements while maintaining a lossless reconstruction of the original data.

7. Future Aspects and Scalability

To scale this code for datasets in the order of gigabytes (GBs), the following strategies could be employed:

a. Distributed Computing:

Implementing the code in a distributed computing environment (e.g., Hadoop or Spark) would allow it to handle large-scale datasets.

b. External Storage:

Using efficient storage solutions like NoSQL databases could help manage large sequence maps and datasets.

c. Batch Processing:

Processing the data in batches could reduce the memory footprint and enable handling of larger files.

8. Conclusion

The data compression technique implemented in the code is a robust method for reducing the size of transaction data. Its lossless nature ensures data integrity, making it ideal for critical applications. With further optimization and scalability enhancements, this technique could be adapted to handle much larger datasets, offering significant benefits across various industries.