A PREDICTOR-BASED COMPRESSOR

Costin-Anton BOIANGIU [1*]
Iulia STANICĂ [2]
Vlad NECULAE [3]

ABSTRACT

As technology evolved, more and more information is in circulation. The amount of data is in a continuously expanding phase since the first digital data storage option. For this matter, compression must be applied on, hence the predictor compression is one of the choices used nowadays. The real benefit, not only comes from compressing data for storage but from sending and receiving data over networks as well. Reducing the size of the desired data before sending will ensure the transfer will be faster and network bottlenecks are less likely to occur.

KEYWORDS: *data prediction, data compression, neural network, machine learning, prediction tree, polynomial interpolation*

INTRODUCTION

Message transfer is a widely used form of communication today. Every day more than 150 billion emails are sent over the world and more than 4 billion searches are performed over the Internet. These actions bring with them the need to store large amounts of information in a compressed form.

There is also streaming which requires real-time delivery, meaning data has to be hastily compressed, delivered and decompressed, with the smallest delay possible. To consume as few resources as possible, such as storage space or bandwidth, it is necessary to process the data in order to represent it in a reduced format (compression) but which offers the same quality of information upon decompression.

Due to storage limitation or network bandwidth, data must be compressed to store or to transfer it. In order to do so, redundant information found within data shall be removed. As a solution, a predictor is a high-speed compression algorithm. On what a predictor excels is that it is still one of the fastest algorithms for compression used today, even if the ratio obtained is not the best.

[1*] corresponding author, professor PhD Eng., ”Politehnica” University of Bucharest, 060042 Bucharest, Romania, costin.boiangiu@cs.pub.ro
[2] Teaching Assistant, Eng, PhD Student, ”Politehnica” University of Bucharest, 060042 Bucharest, Romania, iulia.stanica@cs.pub.ro
[3] Student, ”Politehnica” University of Bucharest, 060042 Bucharest, Romania, neculae.vlad@gmail.com

Generally, data compression is preferred when it is rarely used, otherwise, depending on the implementation and on the data type and content, it is possible to spend more time and resources on decompressing the data in order to access the initial information. Compression algorithms are time and resources consuming. For example, streaming would require real-time delivery, meaning data has to be hastily compressed, send through whatever means of communication is used, such as the internet, radio, satellite etc., and fast decompressed at the receiver, with the smallest delay possible. Depending on various factors such as bandwidth, speed, location, storage etc., this would be, or not, preferred in order to obtain the desired result.

In our current paper, we will present various compression methods, their applicability, the data types on which the algorithms perform best, the underlying structure and functionality of each, and the obtained results after tests.

We will present dome data-specific modules featuring neural network prediction, prediction trees, and numerical interpolation. Each predictor has its performance based on the data acquired at a given time. A codec module will make use of the aforementioned predictors to encode and decode data from a file.

This paper will cover the basics of prediction-based compression, the general idea as well as how it is actually working, followed by approaches for implementation and their corresponding results.

ARCHITECTURE OF APPLICATION

Given a data stream denoted DS, containing either a fixed number of bytes, namely N bytes, expressed as $D_{[0]}\ldots D_{[N-1]}$, or an indefinitely continuous number of bytes $D_{[\ldots]}$. Given any DS data stream, the aim is to compress and respectively decompress it using a number of predictors for different sets of data types and to use a mean to select and alternate between the predictors in order to choose the one which best performs on the DS, thus allowing to achieve best compressing and decompressing results.

Components

A theoretical approach to this subject would be having a number of interlinked and dependent modules, each performing a single, indivisible objective, specifically: a predictor module, a codec module, a core encoding module and an after-prediction encoding module, all of which will be detailed in the following paragraphs.

Predictor Module

The predictor module has to encapsulate the total number of predictors used for attempting to make a best-guess of the next DS, looking into the current DS data history and also decide which predictor produces the best result on the given DS. Every Predictor will have to take into account a maximum number of bits and predict a given number of bits, according to its data type. Moreover, the predictor must also be able to produce as well as to receive residues for the encoding and, respectively, decoding process. The aforementioned residue represents the difference, if any, between the data anticipated by the predictor and the actual DS data. This difference can be computed by choosing from a large variety of means and it depends on each predictor to determine, according to the DS data type, the best residue-computing mechanism, be it difference at every byte-level, integer-like difference on multi-byte representation, difference between dictionary entries in the case of dictionary-based prediction, smart heuristics etc., or a combination of the above. The purpose is to decrease the entropy of the residue, to maximize the uniformity of residue data at byte-level and to group similar symbols together, where, by entropy is denoted the randomness of the data, and the unpredictability of the entire system; the more the entropy the lesser the compression ratio will be, thus the entire system will lack performance.

Codec Module

The codec module is responsible for performing the compression and decompression of data contained in different files, by making use of the results obtained after running the predictor module over data chunks. In order to perform the compression operation, the codec module has to receive an archive name to create, if non-existent, and a file name to compress and add to the archive. Correspondingly, the decompression operation will only be executed if an archive name to decompress will be provided. A binary executable, the codec, will receive a parameter which will indicate what action to perform: adding compressed file to the archive, extracting and decompressing the content of the archive, listing the content contained by the archive, as well as metadata information, such as the date and time, the file size, compression ratio, checksums; and testing the integrity of the archive as it is of very crucial importance to create a valid, uncorrupted archive, since the compression needs to be lossless, and thus the data retrieved entirely.

Core Encoding Process Module

A good approach is to use two tables, one containing all the predictors (TP – “Table of Predictors”, index represented on NB bits for an up to 2^{NB} total predictors), and one for containing the most recently used predictors (TRUP – “Table of Recently Used Predictors”, index represented on NBRU bits for an up to 2^{NBRU} positions. TRUP will be updated at every encoding step, using a MRU-type logic, keeping in mind that the lower the index, the better the expectancy of the predictor to be used again, and the last index kept reserved for a special marker (NIT – “Not in Table) used as an “escape” to specify that a newly requested predictor, currently not in the TRUP will be loaded from the TP and the full TP index will be encoded afterward on NB bits. In the logical compressed data stream, the predictor will be followed by the residue resulted as a “difference” between the predicted value and the real data stream values.

The residue will be produced by the selected predictor. In the case of the LAST predictor, no residue value is needed. It is not necessary to code explicitly the number of bits used for every predictor, simply because a limited number of bits values will make sense and, as a result, different bits values for the same predictor may occupy different entries in TRUP and TP tables.

Example of predictors may include:

- NONE(N) – No prediction for the next N bits, the residue will be DS itself;
- LAST(N) – The prediction is exactly the last N bits encountered, the residue will be the difference;
- INT(N) – N-bit integer is expected, compute residue as a difference on N-bit numbers;
- FLOAT(N) – N-bit FP number is expected, compute residue as a difference on N-bit FP numbers;
- TEXT(N) – Text of N-bit characters is expected, predict as textual information then store the residue as N-bit difference from the prediction;

After-Prediction Encoding Module

After using a chosen prediction-based compression algorithm for encoding, to ensure all the data obtained by the previously mentioned modules is compressed it will be used a third-party powerful generic compression library, such as Zlib.

SPECIFIC PREDICTION MODULES

Neural Network Predictor

The idea behind the Neural Network Predictor approach is to use a general multi-layer perceptron neural network to create a text predictor. As input, the network receives N characters from the history and outputs M predicted characters where $M < N$. We tried various architectures for the network, most of which differed based on their number of inputs and outputs.

Firstly, we tried a simple approach, in which the characters were fed into the network as they were, without any preprocessing. It means that the network worked at byte level. A few architectures that we tried are: *64 -> 32 -> 16 -> 8, 32 -> 16 -> 8* or *512 -> 128 -> 32* and so on. The numbers correspond to the size of fully-connected layers in a multi-layer perceptron architecture.

The second approach consists of processing the characters at bit level. That means each byte had to be split in its individual bits and fed into the network. For example, a network with 8-byte input will now have 64-bits input.

The networks were trained on a compilation of texts from *Wikipedia* [1] and on two books from *gutenberg.org*, *Pride and Prejudice*, and *Heart of Darkness*. *Pride and Prejudice* was used as training data while *Heart of Darkness* was used as testing data.

Prediction Tree

Prediction Tree (PT) is an algorithm designed to predict the next symbol in a sequence of symbols, by trying to match a sequence of symbols to the last N symbols from the input history. As an example, the following sequence will be considered: “streamstreamstre”. It represents a history of the last parsed symbols. A good prediction for the following character in the given sequence would be ‘a’, as the ‘a’ symbol is always preceded by ‘e’ in the data stream history. Moreover, it can be noticed that ‘a’ is always preceded by the “re” sequence and so on. The more matches are made, the clearer it is that ‘a’ should be the next character in the given sequence.

An important deficiency of this model is its limitation in terms of the number of patterns that can be learned. Even the most subtle variation within a symbol subsequence will greatly affect the prediction’s outcome, and also the prediction’s accuracy. Therefore, this algorithm does not behave very well if datasets include a high degree of noise, but the tests have shown that this algorithm can behave pretty well for various types of inputs.

The Prediction Tree is a multi-children decision tree. Each node is described by a symbol and contains references to its parent and children nodes. A sequence of symbols within the tree is represented as starting from a child of the root and continuing to any other node down in the tree. In order to build the PT, the training algorithm is fed with a set of sequences from a training dataset.

Given a sequence of symbols, the algorithm verifies whether the current node (initially the root) has a child containing the value of the first symbol of the sequence, in which case the next symbol is being processed. Otherwise, a new child of the current node is created, containing the current symbol, a child that now becomes the new current node. This process is repeated until no symbols are left in the given sequence.

Moreover, each node also contains a table of symbols to predict (prediction table) and the number of occurrences of each symbol because choosing the next symbol will be based on the symbols’ occurrence probabilities. This table of symbols will not keep the entire set of symbols which resulted from training, but only the first N symbols with the highest number of occurrences.

Given a sequence of symbols, the algorithm verifies whether the current node (initially the root) has a child containing the value of the first symbol of the sequence, in which case the current node becomes the child node described by the first symbol in the sequence and so the next symbol is being processed. Each symbol in the sequence is thus processed until there is no child of the current node containing the next symbol to be processed, from the sequence of given symbols.

Thus, if the next sequence symbol to be processed is not contained by a child of the current node, based on the prediction table of the current node, the algorithm will provide the symbol with the highest probability of occurrence, as the predicted symbol. If the current node does not contain any symbol in the table of prediction symbols, then the algorithm will go up to the parent to let it provide a prediction symbol based on its prediction table (and so on). If no parent contains any symbol in its prediction table, the root will always provide a prediction symbol (always the same symbol).

If a prediction table contains two or more symbols with the same highest probability, then the last parsed symbol will be chosen.

Polynomial Interpolation Predictor

A classic mathematical method used to predict data is polynomial interpolation or extrapolation. Polynomial interpolation is used to estimate data inside the range of two known points, while extrapolation estimates data in the future, based on previous data.

The objective of this research is to predict future data based on a data stream. Therefore, using polynomial regression is a valid technique. This method tries to construct a regression model following the general formula:

$$y_j = f(x_j) + e_j,\ 1 \le j \le n,$$

where x_j , $1 \le j \le n$, the index of the data or a measure of one of its traits; y_j , $1 \le j \le n$, the data at point x_j; e_j ,$1 \le j \le n$, the residual errors of the prediction (the difference between what the model predicted using $f(x_j)$ and what the data should be).

Adapting the formula such that the regression model is polynomial, the $f(x_j)$ function can be defined as such:

$$f(x_j) = f(x;\ c_0, c_1, c_2 \ldots, c_d) = c_0 + c_1x + c_2x^2 + \ldots + c_dx^d,$$

where: d – the degree of the polynomial and c_k, $0 \le k \le d$ are constants.

Based on the previous paragraphs, the process of polynomial data prediction must fit a polynomial to the given data stream and then use that polynomial to predict the data that follows. Polynomial fitting requires the constants and degree of the polynomial to be determined. Afterward, the polynomial must be tested to check if it correctly predicts new values.

Alglib is a numerical analysis and data processing library, which was used by this project to implement the polynomial regression method. We used the library's 'polynomialfit' function to estimate a polynomial of a certain degree based on a given number of bytes of the data stream given as input. Using the resulting polynomial, *Alglib*'s 'barycentriccalc' function then predicts the next few bytes of the data stream. The index of the byte data stream represents the *x* variable of the polynomial and the data byte represents $y = f(x)$, the result of the polynomial.

The predictor was written in C++, using the Alglib library and its previously mentioned functions are used to fit 3rd and 4th-degree polynomials and then predict data.

CONCLUSIONS AND FUTURE WORK

As benchmark test files, we used the files from the Maximum Compression website [8]. We found 10 general files for benchmarking the performance of file compressors. We compared our results with the results of the best compression programs known at this moment: PAQ8PX and WinRK 3.1.2.

We benchmarked each predictor individually and we measured the compressed size, the compression ratio and the time it took to compress (which is not of great relevance in this context). The most relevant metric here is the compression ratio which, in the end, was at least comparable to the best solution available.

The implementation of the presented algorithms may be further refined and optimized to obtain even better results through an efficient implementation. There is still work to be done in order to create more complex predictors. For now, the presented algorithms have high potential as each has obtained good results for their given cases.

The Neural Network Predictor shows great promise as it has some improvement. The more training it is applied to it, the more improvement should be made.

Prediction Tree has a very good ratio in most cases and its results prove this.

The Polynomial data prediction is potentially useful if the data stream is mostly numerically-based.

In conclusion, the presented research is just in the beginning phase but displays promising results. There is a lot of room for improvement of the currently existing algorithms as well as for new ideas on how to approach this particular subject. For now, the presented algorithms have high potential as each has obtained good results in their use cases.

ACKNOWLEDGMENTS

The authors would like to thank the following MSc students (in alphabetical order): ARTENE Cristina, ATANASIU Alexandru-Marian, AUNGURENCEI Ștefania, BĂLOI Bogdan-Cristian, BEJAN Andrei Stelian, BELMEGA Diana-Elena, BÎRZU George, CARAGHEORGHIEV Mihai-Traian, DAMIAN Petrișor-Alin, DINU Ovidiu Alexandru, DORCIOMAN Albert-Alexandru, HĂLCEANU Andrei, IONESCU Ionuț-Emil, JINGA Nicolae, MUEDIN Emin, PANAITESCU Cristian, PARASCHIVU Răzvan Mirel, PRĂZARU Anca-Teodora, ȘERBAN Razvan, SIMION Ruxandra, SPOIALĂ Cristian, TUDOR Cosmin-Mihai, UNGUREANU Teodor, VĂDUVA Gabriel Eugen, for their great ideas, support and assistance with this paper.

REFERENCES

[1] S. Merity, “The wikitext long term dependency language modeling dataset”, September 26, 2016, URL: https:// einstein.ai/ research/ the-wikitext-long-term-dependency-language-modeling-dataset, Available online: June 30, 2018.

[2] M. Lavielle, “Polynomial regression model: an example”, URL: http:// sia.webpopix.org/ polynomialRegression1.html, Available online: June 30, 2018

[3] J. Schmidhuber, S. Heil, “Predictive coding with neural nets: application to text compression”, URL: https:// pdfs.semanticscholar.org/ 5297/ 664960407e14b574832d50c4637bf8e49c22.pdf, Available online: June 30, 2018

[4] A. Avramovic, G. Banjac, “On Predictive-Based Lossless Compression of Images with Higher Bit Depths”, Telfor Journal, 2012, URL: http:// journal.telfor.rs/ Published/ Vol4No2/ Vol4No2_A9.pdf, Available online: June 30, 2018

[5] M. Fink, M. Holters, U. Zolzer, “Comparison of various predictors for audio extrapolation”, 2013, URL: http:// dafx13.nuim.ie/ papers/ 42.dafx2013_submission_27.pdf, Available online: June 30, 2018

[6] D. Giacobello, T. Waterschoot, “High-order sparse linear predictors for audio processing”, August 2010, URL: https:// www.eurasip.org/ Proceedings/ Eusipco/ Eusipco2010/ Contents/ papers/ 1569293087.pdf Available online: June 30, 2018

[7] R. Saran, H. B. Srivastava, A. Kumar, “Median Predictor-based Lossless Video Compression Algorithm for IR Image Sequences”, March 2009, URL: http:// docplayer.net/ 62668009-Median-predictor-based-lossless-video-compression-algorithm-for-ir-image-sequences.html, Available online: June 30, 2018.

[8] Maximum Compression (lossless data compression software), URL: www.maximumcompression.com, Available online: March 1, 2019.