Commenius University
Faculty of Mathematics, Physics and Informatics
Department of Computer Science

# Practical Data Compression

master's thesis

**Viktor Štujber**

Bratislava, 2008

# Practical Data Compression

MASTER'S THESIS

**Viktor Štujber**

COMMENIUS UNIVERSITY
FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS
DEPARTMENT OF COMPUTER SCIENCE

Informatics

Thesis advisor
doc. RNDr. Pavol Ďuriš, CSc.

BRATISLAVA, 2008

I hereby declare that the work presented in this thesis is my own, written only with the help of the referenced literature and internet resources.

Viktor Štujber

## ACKNOWLEDGEMENTS

First and foremost, I would like to give my thanks to Matt Mahoney. Without his research this thesis would never have existed. I also thank my thesis advisor, doc. Pavol Ďuriš, for giving me motivation and useful suggestions. Finally, a big thanks to my parents and siblings for bearing with me and supporting me during the period of writing.

## ABSTRACT

The main focus of this thesis is to examine the concept of compressors that use multiple algorithms or data models. This approach is the core of today's top-ranking compression software, yet there doesn't seem to be any mention of it in data compression literature. We intend to fill in this gap by explaining the technique, introducing a suitable multi-compressor model and describing its structure in detail.
This thesis also covers a closely related area of interest, data compression metrics. We show how apply it to our model to improve its decision-making process.

*Keywords:* lossless data compression, multiple algorithms, parallel processing, metrics

# FOREWORD

All data consists of two components: information and redundancy. *Lossless data compression* is the concept of reducing its size without losing any information. This is done by recognizing and eliminating the redundancy part. And the better the compressor is at finding redundancy and the more of it is present in the data, the shorter the output will be once the data compression process completes.

There are several main reasons why we would want to use data compression.

First, reducing the *size* of data lets us extend the effective capacity of storage devices past their physical specifications. This can provide significant savings, especially in situations where an alternative approach would be economically or technically prohibitive.

Whenever *transferring* data over a communication channel whose capacity is smaller than the computing power of its endpoints, employing data compression can increase the real transfer rate above the limit imposed by the available bandwidth. This means a reduction in time spent waiting for the transfer to complete. Depending on the characteristics of the data, the improvement can be by a factor of two or more.

Finally, the concept of data compression is closely tied to *information theory*. If some string of data can be compressed, it contains redundancy. If it contains redundancy, it is not random. This is of importance in the field of cryptology.

Due to the significance of data compression, we believe that if there is possibility of improvement, we should definitely investigate it. Such was the case with the material presented in this thesis. To the best of our knowledge this is the first publication that mentions the area of our research. Therefore we hope that our ideas will be helpful in achieving better compression results.

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# 1. INTRODUCTION

## *1.1 Background*

From the beginning, data compression was centered around the principle of a single universal compression algorithm doing all the work. It would consist of a model to analyze the data, and an entropy coder to process its output. The resulting software would provide very few configuration options and offer little space for improvement beside the algorithm itself.

But recently, compression tools have appeared which utilize more than one compression algorithm. By understanding this concept, a whole area of research opens up - one that offers new possibilities to improve on existing data compression techniques, and also to develop brand new ones.

Implementations of this idea have already existed for some time now, in a very basic form. These either depended entirely on manual configuration, or used some simple heuristic to select an appropriate strategy. There was usually only a single primary coder, while the others were only utilized in some special cases which could not be handled well enough by the core.

This situation changed dramatically after M. Mahoney introduced his *context mixing* model [Mah02] and the open-source PAQ[1]series of compressors. In his approach, multiple specialized models process the input independently and in parallel. Each model has its own assumptions on how the input is structured, and after processing a bit of input, each model produces a prediction and confidence rating. These are then gathered and combined to produce the final prediction.

Shortly after its introduction in 2002, PAQ became number one in terms of compression, beating the Calgary Challenge [Bro96] and remaining on the top of compression benchmark charts ever since.

But even with various sophisticated input preprocessing steps, the PAQ algorithm had by design the worst compression/decompression performance from among all the other algorithms. It could be considered as a sort of bruteforcing approach that relied on its models to identify patterns and make accurate predictions. This made it interesting in theory but nearly unusable in practical applications. Nonetheless, its structure and mode of operation became a great source of inspiration for this thesis.

[1] PAQ is available at `http://www.cs.fit.edu/~mmahoney/compression/#paq`.

## 1.2 Goals

Our goal was to explain the concept of utilizing multiple compression algorithms and show that it is good enough to be used in practice, as a superior alternative to the classic single-compressor approach. We also wanted to find out whether metrics other than compression ratio (or equivalently, bits per character) could be used for comparisons, or even inside compression algorithms themselves.

We can say that we achieved the first goal. We have designed a model for the so-called *multi-compressor* and explained its components in detail. We have investigated multiple resources to find suitable metrics, and present them in a separate chapter. We have also shown how these metrics can applied to our model to produce an interesting compressor that doesn't focus on compression ratio only.

Sadly we could not provide empirical comparisons against classic compressors. This would require a full implementation based on the model and that was not possible due to time constraints. Nonetheless, weaker implementations of this idea already exist and are very successful at ranking high on various compression benchmarks. Therefore we believe that any more tests would only serve to prove the point.

## 1.3 Overview

This thesis is structured as follows.

Chapter 2 introduces a small set of *terms and conventions* that will be used frequently in the following text, to reduce unneccessary repetition that would otherwise accompany any mention of oompression or decompression.

Chapter 3 gives a quick *overview* of the currently accepted basic data compression model. It mentions various input preprocessing techniques, some well-known universal data compression algorithms and entropy coding schemes. For each of these, we give a short description and references to relevant publications.

In Chapter 4 we provide information on compressor benchmarking and the various *metrics* involved. We then describe the notion of a *scoring* function and list some known examples found in literature.

In Chapter 5 we present an unified *model* for compressors with multiple algorithms. We describe its structure and explain the purpose of its three components. We show some interesting properties and propose an implementation that uses the results of the preceding chapter.

Finally, Chapter 6 summarizes the important points of this thesis and offers suggestions for future work.

# 2. TERMS AND DEFINITIONS

Here are some basic basic terms and conventions which will be used throughout the document.

**Definition 2.1.** *Data compression* is the process of transforming data into a representation of smaller size (compression), in a way that allows an inverse transformation to reconstruct the original data (decompression).

**Definition 2.2.** *Lossless*[1]data compression is data compression where decompression of compressed data always yields output identical to the original input.

**Definition 2.3.** A compression *algorithm* (*technique*) is a specific method of performing data compression.

**Definition 2.4.** A compression algorithm is *adaptive* if it dynamically adjusts to the data being processed.

**Definition 2.5.** A compression *program* (*compressor*, *encoder*, *packer*) is an implementation of a compression algorithm.

In the rest of the text, any mention of *compression* should be understood as *adaptive lossless data compression*. Also, since an algorithm and its implementation represent the same concept, we will be using them interchangeably unless explicitly stated otherwise. The same thing applies to the pair *compression* and *decompression*, since they are very closely tied.

These conventions are introduced to reduce repetition and make the text easier to read and understand. It should be clear what the text is referring to from the surrounding context.

This thesis is written in a way that closely relates to practice, as opposed to the usual, highly abstract terminology (hence the title). We believe that this will make the content more intuitive to the reader.

[1] In contrast to *lossy* data compression

# 3. RELATED WORK

In data compression literature, the notion of a *basic compressor* is pretty well defined. Thanks to [RL81] we know that such a compressor can be separated into two components - the *model* and the *entropy coder*. There can be an optional *preprocessing* stage where input is transformed to a form more suitable for data compression.

The process itself is a loop where input is fed to the model, the model outputs a symbol and a probability, and the entropy coder encodes the symbol into a sequence of bits based on the given probability. The model then updates itself and proceeds with the next piece of input.

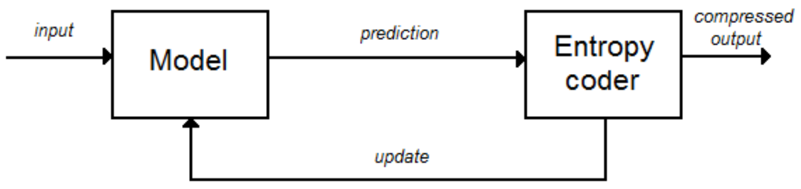

*Fig. 3.1:* The classic compressor model

For each of these three stages there exist various approaches and implementations. Since the components are standalone, it is possible to interchange them, thus allowing a certain degreee of flexibility when designing a compressor.

In the rest of the chapter we will briefly mention some popular techniques in each category. A much more extensive approach can be found for example in [Ski06] or [Say02].

## *3.1 Preprocessing*

Preprocessing means transforming and restructuring the input in a way that's expected to improve its compressibility. These algorithms are only applicable to specific types of data and therefore need some sort of input recognition scheme.

*Text-based* transforms exploit well-known characteristics of written languages, like the fact that text consists of sentences and sentences consist of delimited words. Techniques range from simple substitution of words from a prepared dictionary, to sophisticated schemes like Star encoding and Word Replacing Transformation. The details of these approaches are explained well in [SGD05] and [AT05].

*Audio* stream preprocessing works by decorrelating the audio channels present in the stream. Then, linear prediction (or some other form) is applied to approximate the original signal and compute the differences. The resulting sequence can then be compressed by conventional means.

*Graphics* data is two-dimensional, and thus uses special transformations to “linearize” it. This is coupled by filters that perform spatial decorrelation of neighbouring pixels, which a normal compression algorithm would be unable to do.

Specialized techniques also exist for many *other* input types. These include XML-like markup, executable code and video streams. Some compressors have even demonstrated preprocessing that performs partial decompression of already compressed graphics data to let more powerful algorithms do the work better. But these are currently in experimental stage and require that the decoder perfectly reconstructs the original data, which some of them fail to guarantee.

## 3.2 Entropy Coding

Entropy coding is the process of encoding symbols into sequences of bits so that the code lengths correspond to the probabilities of occurence of these symbols and the most frequent symbols get the shortest codes. This is used as a final stage of the data compression process, where the symbols and probabilities are provided by the part that does input modelling.

The principle dates back to [Sha48] and this area is well-researched. Several schemes are available: Huffman coding, Arithmetic coding and Range coding.

*Huffman* coding is based on [Huf52]. The algorithm decides on which codes to use by organizing the individual symbols into a binary tree according to their probabilities, the most frequent being at the root. The code for a particular symbol is then obtained by traversing the tree from root to the symbol’s node and noting the path taken. The simplicity of this approach makes it popular for teaching purposes and hardware implementation. Its drawback is that it only achieves the best possible result if all probabilities are a equal to some negative power of two, because it has to encode each input symbol individually and into an integral number of bits.

*Arithmetic* coding, published in [Ris76], solves the problem by encoding input as one fractional number. Since its entire output is one long codeword, the abovementioned effect does not occur. Furthermore, it has been proven that for any given set of probabilities, the scheme used by the encoder guarantees that the size of the produced output will be very close to their actual entropy. The algorithm itself works by dividing the interval [A..B], initially [0..1] into subintervals of length proportional to the symbol probabilities, choosing the interval corresponding to the symbol being currently encoded, and mapping A and B to the endpoints of this interval. Then this process continues for the next input symbol until the entire input is exhausted. The result is a fractional number encoded in binary that uniquely represents that particular input.

*Range* coding ([Mar79]) uses the same concept as arithmetic coding, but instead of fractions, it works with integral numbers big enough to represent the range. There are also some minor differences in processing that increase the output size slightly, but in turn allow much faster processing.

## 3.3 Modelling

Modelling input means trying to recognize the structure and redundancy in the data. The better the algorithm understands the input, the better it can estimate what its following contents will be. This estimate will in turn affects the performance of the followup entropy coding process.

In this category we recognize these four approaches: Dictionary compression (also called Lempel-Ziv compression), Block-sorting compression (also called Burrows-Wheeler compression), Prediction by Partial Matching, and Context Mixing.

*Dictionary*-based compression operates by tracking a list (*dictionary*) of sequences that were seen in the already seen input, and replacing newly found occurences of the same sequences by a reference to the dictionary. After its publication in [ZL77] it became the first widely-used scheme for compressing text and similar data. LZ compressors are the fastest-performing of all, but at the expense of compressing power.

*Block-sorting* compression is a solution that is heavily based on a preprocessing step called the Burrows-Wheeler Transform [BW94]. This is an intriguing input permutation algorithm that moves characters with similar context closely together, no matter what their original position was. If the input contained repetitive phrases, then the result will contain long runs of repetitive characters. Specialized algorithms like Move-to-front coding and Run-length coding are then used to remove this redundancy.

*Prediction by Partial Matching* (shorthand: PPM) bases its algorithm on the theorems of Shannon [Sha48], which say that the entropy of a source can be better approached if we track symbol occurences along with their context. This will give a closer estimation of the real probabilities of symbol occurence in the input and the entropy coder can therefore produce a more precise codeword.

*Context mixing* is a very recent method, first documented in [Mah02]. It started as an extension to PPM by using more than one model. Each one would have its own specifications on how to match the input, and during processing of each bit, each model would give a prediction of the next bit, and a confidence level of this prediction. A mixer would then compute a weighed average and pass the result to an entropy coder. By using over 20 models, each specialized for certain patterns of data, it is able to perform significantly better than other compressors, but at the expense of huge memory requirements and very slow processing speed.

## 3.4 Summary

In table 3.1 we list the four compression algorithms described earlier, along with their general characteristics like compression speed and memory usage. In figures 3.2 and 3.3 we plot the distribution of results of the Maximum Compression Benchmark ([Ber08]) for over 100 compressors. The more a value is to the lower-right part the better. From here we can clearly see the effects of the time-space tradeoff, where the best performing compressors require significantly more time to perform both compression and decompression.

[1] according to [Ski06]

| Class | Introduced in | Compr. effectivity | Compr. speed | Decompr. speed | Memory usage |
|---|---|---|---|---|---|
| LZ | 1977 | Average | High | Very high | Low |
| BWT | 1994 | Good | Average | Average | Average |
| PPM | 1984 | Very good | Low | Low | High |
| CM | 2002 | Best | Very low | Very low | Very high |

*Tab. 3.1:* Overview of described compression algorithms[1]

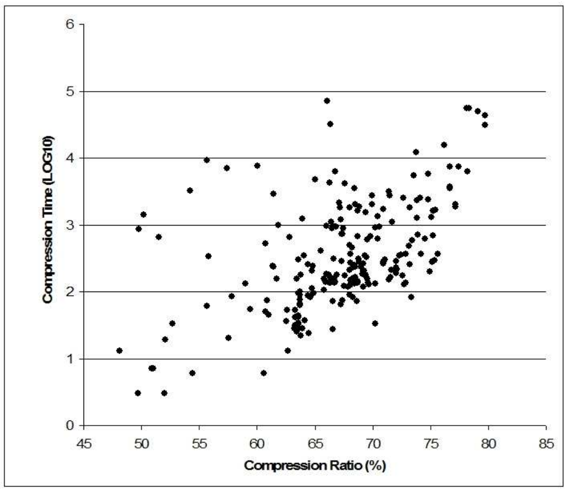

*Fig. 3.2:* Compression time and ratio comparison

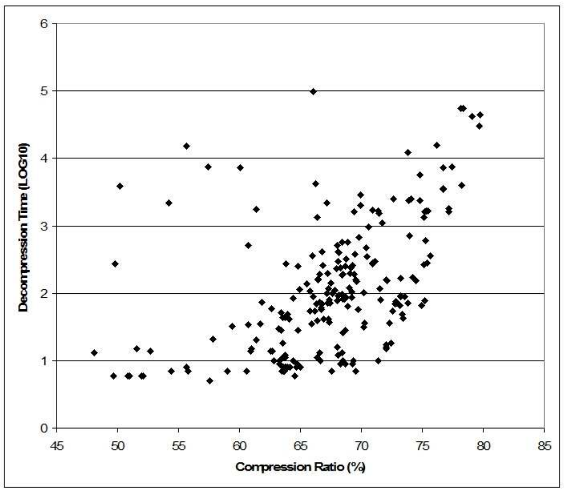

*Fig. 3.3:* Decompression time and ratio comparison

# 4. METRICS

This section is organized as follows. First we provide some background on compressor benchmarking. Afterwards we describe which metrics and factors are present during the data compression process, define the notion of a scoring function and finally list some relevant instances of this function.

If we have two or more compressors at hand, naturally we would want to compare them. This can be done by either theoretical analysis of their algorithms (and perhaps implementations), or by empirical tests and measurements.

This section will focus on the latter approach. For the theoretical part, we suggest taking a look at some already achieved results, like the the establishment of a relationship between LZ and DMC (mentioned in [How93]). We also recommend [Cha97] which presents some very good impossibility results related to data compression. These give us the motivation to focus on a compressor's runtime behavior instead.

The ultimate purpose of data compression research is to have its results applied in practice. This implies that to give measurements any significance, they need to be made on real-world data. To make individual measurements comparable, some level of consistency needs to be maintained. If the results are meant to be used only for demonstrative purposes, it is sufficient to provide identical conditions to all executions that are to be measured. If they are to be published so that others can reproduce the results and perform comparisons of their own, then it is crucial that all of the data used in the tests be made available. This can become a problem if the data set is prohibitively large, or for some reason may not be released to public.

To this end, an overall of several data sets were proposed, the content of each chosen carefully as a representative of usual data of that time. The initial and most accepted set that appears in data compression literature is of course the Calgary Corpus [BCW90].

Sadly, there is no universal testing platform, which only leaves authors the possibility of specifying the hardware configuration which they used for their experiments. Due to the multitude of hardware available today, this means that the results would probably be only approximate. For this reason, the values published in data compression literature will most likely be only in bits per character, which represent the compression factor achieved by the test.

## 4.1 Measuring

Let us now identify the measurable variables that are present during a single execution of a compression program.

- $s_i$ size of the input data (bits)
- $s_o$ size of the resulting output (bits)
- $t_c$ compression time (seconds)
- $t_d$ decompression time (seconds)
- $mem$ peak memory usage

There are also several external factors that are constant for a given system, but have influence on the results.

- $cpu$ processing power
- $bw$ available bandwidth

*Note.* The compression and decompression time is in fact itself a function of processing power, the program's code and the input, but we will omit such low-level decomposition because they cannot be measured.

## 4.2 Ranking

The metrics mentioned in the previous section are *basic* - their values come directly from measurements. But neither of these alone can really be used to objectively rank results. As an example, comparing results of coders A and B using $s_o$ might show that they are equally good, even though A is several times faster and requires less memory. Similarly, A and B could have comparable compression times, but A's asymmetric algorithm would make decompresion way faster than B's.

For this reason we introduce *derived* metrics. These are functions of the basic metrics, and some bring in various external factors. Their common point is that they all give scalar values, meaning that the values can be totally ordered and ranked. What we are looking for is a good metric to serve as a ranking function.

More formally, let $\overline{v} = (v_1, ..., v_k)$ be basic metrics of our choice. We want a function $score : \overline{v} \to \mathbb{R}$ that would allow the comparison of vectors of measured values. All we require is that obeys a basic sanity constraint

$$(\forall i : u_i \gtrdot_i v_i) \implies score(\overline{u}) \gtrdot score(\overline{v})$$

where $\gtrdot$ represents a "better than" comparison over the corresponding domain for each pair of arguments. The reason this extra step is needed is because each metric can be using a different ordering.

Table 4.1 lists the metrics that are not affected by any external factor (explicit and implicit).
Table 4.2 containts the metrics that are based on time, and therefore on the $cpu$ factor.
Table 4.3 adds several metrics that work with $bw$, to be used where communication is involved.

| | |
|---|---|
| compressed output size | $s_o$ |
| compression ratio | $\frac{s_o}{s_i}$ |
| bits per character | $\log \lvert L \rvert * \frac{s_o}{s_i}$ |
| compression gain[1]1 | $100 \log_e \frac{s_i}{s_o}$ |

*Tab. 4.1:* absolute metrics

| | |
|---|---|
| compression time | $t_c$ |
| compression rate | $\frac{s_i}{t_c}$ |
| efficiency[2]2 | $\sqrt[16]{\frac{(s_i - s_o)^{16}}{t_c}}$ |
| efficiency[3]3 | $t_c * 2^{\frac{\frac{s_o}{S_o}-1}{0.1}}$, $S_o = min\{s_{i_o}\}$ |

*Tab. 4.2:* time-based metrics

| | |
|---|---|
| total transmission time | $t_c + t_t + t_d$, $t_t = \frac{s_o}{bw}$ |
| transmission acceleration | $\frac{t_u}{t_c+t_t+t_d}$, $t_u = \frac{s_i}{bw}$ |
| average streaming rate | $min(\frac{s_i}{t_c}, bw, \frac{s_i}{t_d})$ |

*Tab. 4.3:* communication-based metrics

[1] by [How93]
[2] by M. Ashland : `http://www.monkeysaudio.com/comparison.html`
[3] by [Ber08]

# 5. THE MULTI-COMPRESSOR MODEL

Here we introduce the model for compressors with multiple cooperating algorithms. We describe its components and their purpose. Then we show some interesting properties of this model, and finally sketch an implementation that makes its decisions based metrics described in Chapter 4.

**Definition 5.1.** The *multi-compressor* is a compressor capable of processing its input using more than one compression technique.

As was mentioned in the earlier chapters, the classic compressor's mode of operation is simple and requires no special decision-making process. But once we add multiple choices on how to accomplish its task, the outcome of which is not yet known, we suddenly require a way to choose a suitable strategy. And even after that, there is still the need to interpret the results and combine them to produce the final output.

Our model handles these problems by clearly defining which components are responsible for dealing with them and which approaches can be used to solve them. This structure lets us separate the problems and focus on each one individually.

*Note.* This chapter only discusses the compressor's model. Its decompressor counterpart has a similar but much less complex architecture, due to the fact that decompression is straightforward and no choices have to be made except for looking up the correct module(s) to decompress the input.

## 5.1 Structure

Our model specifies three distinct components: the *selector*, the set of *modules*, and the *mixer*.

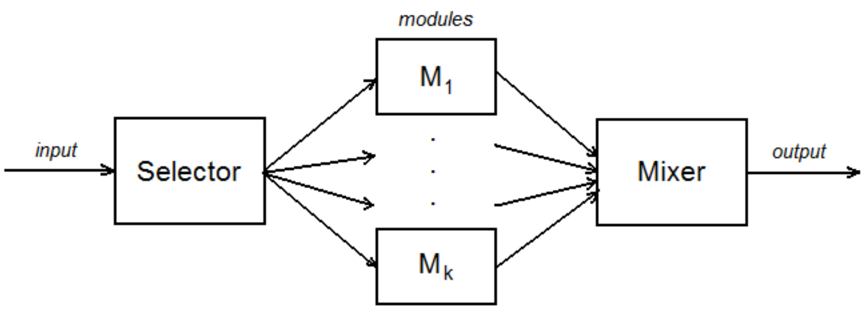

*Fig. 5.1:* The multi-compressor model

### 5.1.1 Selector

The *selector* focuses on input handling. Its task is to analyze the input and understand its structure, then choose the most suitable subset from the set of modules. It will then activate them and make the input available for processing.

A classic universal data compression algorithm makes no explicit assumptions on the structure of its input. We can see from [Ber08] that specialized techniques give results superior to generic algorithms. This happens because understanding the structure means being able to make more accurate predictions.

The downside is that such techniques can have very strict requirements on the input's contents, and will not be applicable if the requirements are not fulfilled. This is why a good selector implementation needs to know what the type of the input is and which modules are able to handle it. In the worst case, this would require scanning the input for patterns before doing any work, because the input might come as a single cosecutive stream of foreign structure. Type recognition schemes do exist (see [MH03] or [HLR05]), but they have a very varying success rate which makes them unsuitable as a primary mechanism.

Luckily for us, such problems do not occur often in practice. This is because the need to communicate information between various systems gave rise to a data format standardization process. As a consequence, frequently used data will most likely use a well known and standardized format. For example, MIME[1]is a very well-known standard that registers and categorizes these formats.

Compare this approach to the PAQ compression scheme, where the models are forced to do input recognition themselves. This allows them to process data of arbitrary structure, but significantly degrades processing speed. Also, compression rate might suffer if one or more models make incorrect assumptions about the input.

Once the input has been recognized, a subset needs to be chosen from the list of candidate modules. This process will vary depending on how the rest of the compressor is implemented. An example strategy might be to select all of them and let the *mixer* deal with the rest. A complementary approach would be to choose only a single module which is "best" for that particular type of input, based on static ranking tables.

The overall sequence of activities that a selector performs is given in fig.5.2.

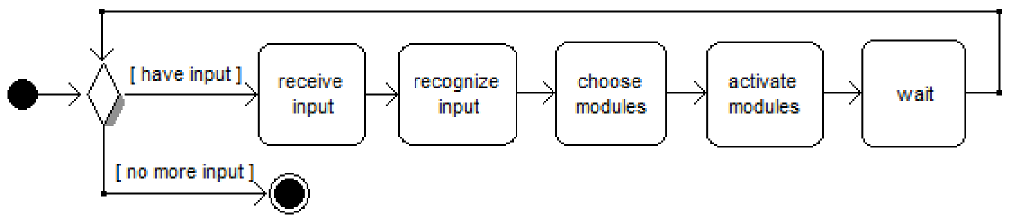

*Fig. 5.2:* Selector activity diagram

[1] MIME media types are described in `http://tools.ietf.org/html/rfc2046`

### 5.1.2 Modules

A *module* in the loosest sense is an arbitrary data compressor, adjusted slightly to fit into the multi-compressor model. Its purpose is straightforward - it has to accept input and produce compressed output when activated by the selector.

Along with the compressed data itself, every module shall also send additional metadata. These will not be part of the output, but will instead be used to assist in the mixer's decision-making process. Some possible values to send are

- module's unique identifier
- original input size
- compression statistics

The set of *modules* represents the multi-compressor's "power". The more modules there are, the more options the compressor has to improve the compression rate. But the modules themselves aren't enough: the selector needs to be able to choose them appropriately.

### 5.1.3 Mixer

The *mixer* is the final stage of the multi-compressor model. Its task is to receive data and metadata from active modules, to choose the results of one of them and to write them in some appropriate form to the output.

The way to decide which input to accept depends solely on the implementation. Normally this would be a simple comparison to rank the inputs according to some chosen metric, and then choose the best one. A default choice would be *compressed size*, but we will in a latter section demonstrate how other metrics can be used as well.

A basic multiplexing scheme that we use in this document for simplicity is to prefix the chosen module's output with its unique identifier. This is the minimum required to let the decompressor work properly. However, without knowing the size of the compressed stream, it has to wait for the modules themselves to signal end of processing. Conversely, providing the length adds more overhead to the output, but allows advanced techniques like random acccess and parallel decoding.

## 5.2 Properties

**Lemma 5.1.** The compressed output of a classic compressor $A$ and a multi-compressor module $m_A$ which uses $A$ is identical, for every input $I$.

*Proof.* By definition, a module is merely a wrapper for a classic data compressor. Thus their outputs will always be the same. □

**Theorem 5.2.** For each classic data compressor $A$ there exists a multi-compressor which for the same input produces identical output.

*Proof.* All we need to do is define the selector to simply forward all of its input, define a single module that uses $A$, and define the mixer to output the data received from the module, unchanged. From lemma 5.1 it follows that this system will behave just like the original compressor. □

*Note.* This observation shows that the multi-compressor model is a natural generalization of the classic model.

**Theorem 5.3.** Let $A$ be a classic compressor and $I$ its input. Let $M$ be a multi-compressor that contains $A$ as a module $m_A$, whose mixer decides using the metric $s_o$[2]and whose selector will choose at least $m_A$ when given the input $I$. Then the size of $M$'s output will be no higher than the size of $A$'s output plus a constant.

*Proof.* If the selector chooses the module $m_A$, it means that the mixer will subsequently receive the compressed output of $m_A$. From lemma 5.1 we get that this output, and therefore its size will be identical to $A$'s. The constant factor is just the unique identifier required for the decompressor. Combined with the fact that there might be a second module that performs better than $m_A$, we get the desired inequality. □

*Note.* This shows that multi-compressors will not perform worse[3]than classic compressors.

**Corollary 5.4.** Adding more modules does not significantly degrade the output size of a multi-compressor (only by $\log k$, the space needed to store a module's unique identifier).

**Theorem 5.5.** The operations done by the modules can be simply performed in parallel, thus reducing the running time from $\sum_{i=1}^{k} t_{m_i}$ to $\min\{t_{m_i} | i = 1..k\}$.

*Proof.* See fig.5.3. This is possible because each module works independently of others. □

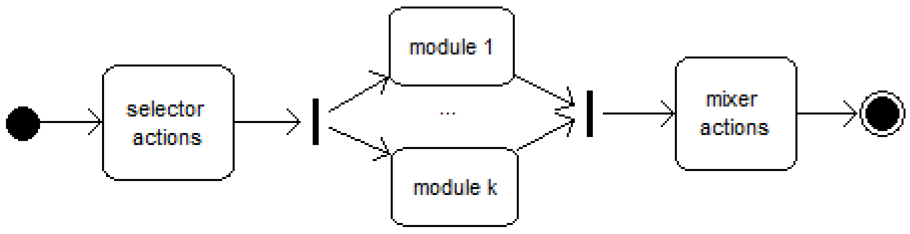

*Fig. 5.3:* Parallelism in the multi-compressor model

[3] Compressed size, lower is better.
[3] W.r.t. compressed size.

## 5.3 Metric-driven mixer

In the previous text we used a metric that's exclusively used in data compression literature - compressed size. This is understandable, since this metric is the only one that guarantees perfect reproducibility and allows comparing measurements straight away.

What isn't considered is that the most powerful compressors consume the most CPU time, both during compression (fig.3.2) and decompression (fig.3.3). The author of PAQ himself states in [Mah05] that

> *"Context mixing algorithms allow compressors to push the three way tradeoff between compression, speed, and memory to the extreme."*

This has severe consequences when such algorithms are deployed, because even though ranking "best", they are unsuitable for use in practical applications.

In the previous text we mentioned that a multi-compressor's mixer can base its decisions on the measured values provided by models after they finish processing their input. There we maintained the convention of using the compression time metric, but as was said here, this is *not* the best indicator anymore. In a hypothetical run of a multi-compressor, one powerful algorithm would dominate all the weaker but much faster ones. This would defeat the whole point.

To this end, we chose a sub-category of multi-compressors whose selector is based on a more suitable *efficiency* metric. Some such metrics we already listed in Chapter 4. Their main advantage of these compressors is that they introduce a new, very much needed level of flexibility to the data compression process.

As an example, let us take two modules $m_1$ and $m_2$ and two blocks of input, $I_1$ and $I_2$. Assume that $m_1$ is more powerful than $m_2$ but takes significantly longer to compress and decompress data. Assume that $I_1$ contains very little redundancy compared to $I_2$. Then neither of these modules alone would be able to perform efficiently - $m_1$ would waste time working with $I_1$'s poorly compressible data and $m_2$ wouldn't compress $I_2$ thoroughly. If both were active and the compressor used a good efficiency metric, the result would both be smaller in size and decompress faster.

What this example demonstrates is a situation that can happen easily in practice, thanks to the wide-spread use of compressed formats. Our solution shows that it is possible to avoid it and ensure the best[4]decompression performance possible, at the cost of increased compression time. And by theorem 5.5 this time cost can be converted into parallel work, which makes it suitable for implementation on current multi-core systems.

[4] W.r.t. a chosen metric and a set of modules.

# 6. CONCLUSION

In this thesis we introduced the concept of a compressor that processes its input using multiple data compression algorithms. We specified a model, gave a detailed overview of its components and showed several interesting properties.

We also investigated the measurable aspects of compressor execution. We stressed the need for a good scoring function and listed some candidates. Finally we proposed an instance of our model that bases its choices on scoring functions.

To the best of our knowledge there is no previous publication that deals with this subject. Therefore further exploration of this area of data compression might yield new findings.

From the practical side, the first thing needed is a reference implementation. It would help clarify technical details of the model and let us run comparative benchmarks. Also it would be interesting to see how the metric-based compressor described at the end of Chapter 5 performs against conventional compressors.

# BIBLIOGRAPHY

[AT05] J. Abel and B. Teahan. Universal text preprocessing for data compression. *IEEE Transactions on Computers*, 54(5):497–507, May 2005.

[BCW90] Timothy C. Bell, John G. Cleary, and Ian H. Witten. *Text compression*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1990.

[Ber08] W. Bergmans. Maximum compression, 2008. `http://www.maximumcompression.com/`.

[Bro96] L. Broukhis. The calgary corpus compression challenge, 1996. `http://mailcom.com/challenge/`.

[BW94] M. Burrows and D. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, 1994.

[Cha97] G.J. Chaitin. Information, randomness & incompleteness - papers on algorithmic information theory - second edition, 1997.

[CW84] J. Cleary and I. Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, COM-32(4):396–402, April 1984.

[HLR05] Douglas J. Hickok, Daine Richard Lesniak, and Michael C. Rowe. File type detection technology. In *38th Midwest Instruction and Computing Symposium April 8 - 9, 2005. University of Wisconsin-Eau Claire, Eau Claire, WI*, 2005.

[How93] P. Howard. The design and analysis of efficient lossless data compression systems. Technical Report CS-93-28, Brown University, June 1993.

[Huf52] D. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[Mah02] M. Mahoney. The paq1 data compression program, 2002.

[Mah05] M. Mahoney. Adaptive weighing of context models for lossless data compression. Technical Report CS-2005-16, Florida Institute of Technology, 2005.

[Mar79] G. Martin. Range encoding: an algorithm for removing redundancy from a digitised message. conference, 1979.

[MH03] Mason McDaniel and M. Hossain Heydari. Content based file type detection algorithms. In *HICSS '03: Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 9*, page 332.1, Washington, DC, USA, 2003. IEEE Computer Society.

[Ris76] J. Rissanen. Generalized kraft inequality and arithmetic coding. *IBM J. Res. Devel.*, 20:198–203, 1976.

[RL81] J. Rissanen and G. Langdon. Universal modeling and coding. *IEEE Transactions on Information Theory*, 27(1):12–23, January 1981.

[Say02] K. Sayood, editor. *Lossless Compression Handbook*. Academic Press, 1st edition, August 2002.

[SGD05] P. Skibiński, Sz. Grabowski, and S. Deorowicz. Revisiting dictionary-based compression. *Software - Practice & Experience*, 35(15):1455–1476, December 2005.

[Sha48] C. Shannon. A mathematical theory of communication. *Bell Sys. Tech. J.*, 27:379–423, 623–656, 1948.

[Ski06] P. Skibiński. *Reversible data transforms that improve effectiveness of universal lossless data compression*. PhD thesis, University of Wrocław, October 2006.

[ZL77] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

## ABSTRAKT

Hlavným cieľom tejto práce je preskúmať koncept bezstratovej dátovej kompresie ktorá využíva viacero algoritmov alebo modelov súčasne. Tento prístup je jadrom súčasného špičkového kompresného softvéru, no dostupná literatúra sa o ňom vôbec nezmieňuje. Náš zámer je vyplniť tento priestor. Priblížime spomenutý koncept, zavedieme pre neho vhodný model a detailne popíšeme jeho štruktúru.
Taktiež pokryjeme úzko súvisiacu oblasť, metriky dátovej kompresie. Ukážeme ako aplikácia na náš model upraví jeho proces rozhodovania.

*Kľúčové slová:* bezstratová dátová kompresia, integrácia algoritmov, paralelné spracovanie, metriky