Exploration of MSC Trees Using Genetic Algorithms

Jakub Rada*, Tomas Musil, and Vit Fabera

Faculty of Transportation Sciences, CTU in Prague, Konviktska 20, Praha 1 110 00, Czech Republic

Abstract. Multistream Compression has proven to be competitive in terms of compression ratio. It utilizes a binary tree structure which follows practically the same rules for its construction as the Huffman method. This tree structure is ideal in terms of compression ratio for creation of codewords by Huffman compression.
Comparing to the Huffman method the Multistream Compression is based on different idea of how to compress data. This fact raises the question whether the same tree structure provides the best performance. The presented paper investigates possible improvements of the tree structure.
A genetic algorithm application was created for this purpose. The design of this application is presented in the paper as well as the obtained results. The whole application is set to the context of the MSC trees.

Keywords: MSC algorithm · MSC tree · Genetic algorithm · Binary tree optimization

1 Introduction

The Multistream Compression (MSC) is a statistical lossless compression method invented by Jiri Kochanek. The method is based on the idea of splitting data into parts [1]. For each part of the data, a coding method that gives the best compression ratio is chosen.

The algorithm is distinctive by a special binary tree structure, similar to Huffman tree, which is used throughout the compression process. Each of the tree nodes contains its own data for compression which are arranged in streams. This method differs from other compression algorithms based on splitting data into streams by the fact that in this case the streams do not contain symbols but counters.

Although the MSC algorithm is not very well known, it has proven its qualities in terms of compression ratio. It is reported that the original unoptimized version of MSC used in conjunction with Burrows-Wheeler Transform (BWT) and Move-to-Front Transform (MTF) provides results somewhere between gzip and bzip2. When it is compared to simple Huffman and Arithmetic coding, the compression ratio is the highest in case of the MSC [2].

* Corresponding author: radajak3@fd.cvut.cz

This paper shall serve as a demonstration that the compression ratio of the MSC algorithm can be further improved. This is achieved by using a different, more convenient, structure of the MSC tree than the one constructed using the original version of algorithm. The reader shall be warned in advance that this paper does not present a new way of constructing such an improved tree. The paper rather presents an application of genetic algorithms. It searches the space of possible tree solutions and based on the evaluation of candidates it tries to find the best trees possible. However, it is not the purpose to find the single best tree for given input data. The main purpose of the application is to delineate to what extent these improvements can be achieved.

2 MSC Algorithm

The MSC algorithm consists of several steps. Some of these steps are incorporated into the presented application. For this purpose the working principle of the MSC algorithm is described in this chapter.

2.1 Input Data Statistics

The algorithm starts with reading characters from the input data. For each symbol occurring in the input data, its first occurrence as well as number of occurrences are determined. Once the reading of input data is finished, the statistics is sorted with respect to the number of occurrences.

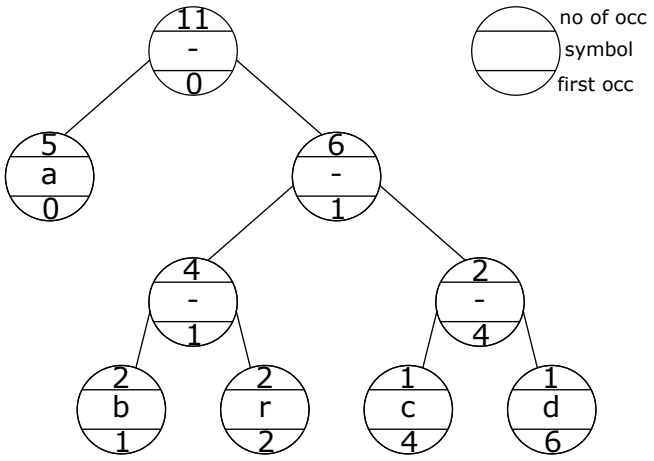

Fig. 1. Example of MSC tree for "abracadabra"

2.2 Creation of Binary Tree

When the statistics are in the desired form, they serve as a basis for building an MSC tree. Similarly to the Huffman tree, the symbols become the leaves of the binary tree. In this stage, parents of two nodes with the lowest number of occurrences are created iteratively until there is only one unconnected node remaining – the root of the tree as shown in Figure 1. The newly created nodes store the same properties as the leaves. The number of occurrences is obtained as the sum of the values of its children. The first occurrence is adopted from the child with earlier occurrence (left child). Comparing to the leaves, the inner nodes do not carry any symbol. As opposed to the Huffman tree, the MSC tree is governed by an additional rule. This rule states that every left child must have earlier occurrence in the data than the right one, even if it has a smaller number of occurrences. Moreover, each node is equipped with a counter and a direction switch.

2.3 Creation of Counter Streams

Initially, before the traversing of the tree starts, the counter of each node is set to zero and each switch position is set to left. The switch determines the active child of a node and thus only one active path is built in the tree from the root to the leaf. During the creation of counter streams the input data are read for the second time. For each read symbol, the tree is traversed from the root node all the way to the leaf node representing this symbol. Each time a node is passed, its counter is incremented. If two consecutive symbols are different, at least one direction switch in the tree needs to be switched to the opposite direction. The rule is that when the position of a switch is swapped, the counter value of the node which loses the active path is written into the stream of this node.

2.4 Statistical Analysis of Streams

This step is performed only when multiple coding methods are available [3]. After the counters are obtained, the length of the compressed data is calculated for different coding methods. The analysis is executed for each node of the tree separately and the best method is selected separately for every single one. In the presented application two coding methods are considered:

Elias Alpha is an unary coding method [4]. It is usually used for short or badly compressible data. If all streams are coded by Elias Alpha, the output data has equal length to the same data compressed by Huffman coding.

ZEBC is a variable-length coding similar to Elias Gamma. This method is more suitable for compression of the streams containing higher values. It was developed specifically for MSC algorithm [1]. It is parametrized by a selected number, base b, which further optimizes the coding of stream. During analysis of the counters,

value of the best base is determined. The coded number consists of unary Zero Ending (ZE) prefix and of a value expressed by the Binary Complement (BC) only if the number to be coded is greater than the base value. If the number is lower than the base, it is expressed only in ZE code. The coding is performed according to ZEBC table of intervals.

2.5 Coding

At this stage, the streams of counters are known as well as the coding methods. For creation of the final compressed stream, the MSC tree is traversed once again. At the beginning, the counter of the root node is set to number of characters in input data, counters of other nodes are reset and switches are set to left. The tree is traversed from the root to leaf nodes using the active path determined by the switches until the counter of root node is equal to zero. In case the counter at entering a node has value 0, next value from the stream of counters is read, coded and written to the final stream of coded values. If a node is accessed for the first time, additional information (leaf node flag, chosen coding method and its parameters and symbol for leaf node) is written to the output before the coded counter value. Each time a node is accessed, its counter is decremented. When it reaches zero, the switch position of the parent node is changed.

3 Background

In the well-known Huffman algorithm the codeword lengths depend on the depths of the leaves (representing a symbol) in the tree. In the MSC algorithm the codeword length directly depends on the values of counters – lower number of higher values can be coded more compactly than higher number of lower values. The counter values are then dependent on the structure of the tree. By changing the order as well as depths of the leaves, the number of occurrences for inner nodes (which is the sum of counters) and the total switching count of the tree switches (which determines how the number of occurrences will be divided) also changes.

As described above, the MSC algorithm incorporates the possibility to select a coding method based on the reached compressed length. Each of these methods has a different relationship between values of the stream counters and the coded length of the counters. As an example we can name the Elias Alpha coding where the coded length depends linearly on the coded value. ZEBC coding has a logarithmic relationship and therefore it is more suitable for coding higher values as shown in Figure 2. It is also worth mentioning that the coded length changes in steps – the insensitivity increases for higher values. For example it makes considerable difference if we divide 120 into 119 and 1 or into 60 and 60, the difference in coded length of the two values is 30%.

Some implementations of the MSC algorithm include also Huffman coding[1] for coding of the counters. Huffman coding does not depend on a single value but the resulting codeword for the counter depends on all values in the considered stream. Moreover, the selection among various coding methods causes that the relationship between the occurrences in input data and the length of the final coded string is rather complex.

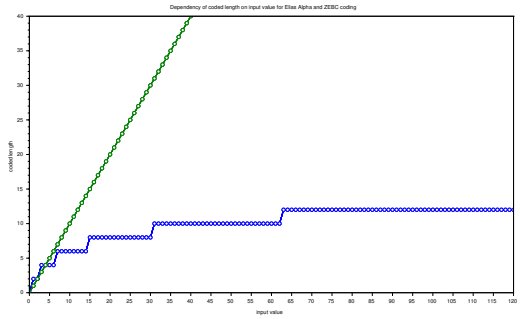

Fig. 2. Dependency of coded length on input values for Elias Alpha and ZEBC(1) coding

Also, in order to increase the compression ratio, various preprocessing transformations of input data can be used in conjunction with the MSC algorithm. The transformation having the highest effect on the compression ratio is the Burrows-Wheeler Transform (BWT) [5]. This transformation reorders the input data so that the same characters often occur consecutively in groups. The result is that the counters in the streams possess higher values. Consequently, the coding methods with the more complex relationships are generally selected.

For unprocessed input data, the majority of the nodes is coded by Elias Alpha. Since Elias Alpha coding produces compressed data length equal to Huffman coding, much improvement is not expected by changing the tree structure. Preprocessing of the input data causes that strong majority of the nodes is coded by ZEBC coding. Since different relations apply for ZEBC, the question is whether in this case the tree constructed according to Huffman produces the best result.

4 Genetic Algorithm - Application

Genetic algorithms are included in the group of algorithms called evolutionary computing algorithms. These algorithms are derived from the principles de-

[1] In this case the Huffman coding is considered as a coding method for MSC coding. In other cases throughout the paper it is considered as a standalone compression algorithm which is compared with the MSC.

scribed by the Darwinian evolution of species. Genetic algorithms work with a population of result candidates. These candidates are represented as chromosomes. Chromosomes are composed of alleles which can be binary or have a different representation.

4.1 Chromosomes

The presented application optimizes the structure of a binary tree. Therefore, the chromosomes shall in some way represent the binary tree. The trees were chosen to be represented by the tuple $\{symbol, depth\}$ for each leaf. This representation is a compact one since only leaves determine the entire tree. Also, the chromosomes allow simple usage of genetic operators and are easily set up after generating initial population. The length of the chromosome is dependent on the number of symbols in input data.

Various tree representations can be found in literature. In [6] the nodes are ordered in a way the trees are usually recursively traversed – left-to-right and top-to-bottom. In contrast, in [7] a linked representation is used. In this case, each non-leaf node stores the link to the left and right node. This representation is not as compact as the used tuple $\{symbol, depth\}$. Also, in our opinion it does not have any added value when using genetic operators.

4.2 Initial Population

The initial population of the MSC trees is generated in a random fashion. No a priori knowledge is incorporated. It could possibly accelerate the finding process. However, the difference rate in the structure between the tree constructed by the original version and the best trees generated by the genetic algorithm is not known. The generation of each tree starts with the root of the tree. In following iterations, two nodes are added to one of the current leaves in the tree. This is performed until the number of leaves is equal to number of symbols in the input data. The current leaf which is appended with the two nodes is selected randomly. When the structure of the tree is finished the symbols are randomly assigned to the leaves.

4.3 Selection of Individuals

The progression of the result candidates is caused by the selection and application of genetic operators. Selection is probabilistic and depends on a function evaluating each candidate-fitness function. For genetic algorithms various selection mechanisms have been reported. For example, standard genetic algorithms (SGA) typically use Roulette Wheel Selection. In SGA each results get a proportionate interval based on its evaluation result. A random number is then selected (figuratively a roulette wheel is turned). The result candidate into whose interval the random number fell is then selected into the next generation.

The presented application uses the Roulette Wheel Selection. The evaluation of the result candidates is based on the calculation of coded data length. To

determine the length, node streams are created and the analysis is performed as described in Sections 2.3 and 2.4 for each result candidate. When using ordinary roulette wheel selection, the best solution is not quite often copied to the next generation. To prevent this loss one of the solutions having the best fitness is automatically transferred to next generation.

The aim of the application is to get a minimal length of the coded data. But for Roulette Wheel Selection better solutions shall get a larger chunk. To convert the best solution with minimal value to the highest number the fitness function is of following form:

$$fitness = (codedLen - minLen)^2 \tag{1}$$

where *minLen* is the calculated length for the best result candidate in the current generation and *codedLen* is calculated length for the examined candidate. To improve the convergence of the results, the fitness function required a tuning. It was empirically determined that the second power of the subtraction gives reasonable balance.

In this implementation two methods are considered for the analysis Elias Alpha and ZEBC. As mentioned above, the length of coded values by Elias Alpha is equal to the coded value itself. Determining the length of the ZEBC coded value is more complicated. Based on the input value and the base, the interval in ZEBC table is found, then lengths of both parts of coded value ZE and BC are determined by (Z+I) and (I+1), where Z is the base and I is the interval number in the ZEBC table.

4.4 Genetic Operators

There are two basic genetic operators to modify the result candidates crossover and mutation. In crossover a random point(s) in chromosome is/are selected and subsequently chromosomes of two result candidates are divided in this point and combined. In this way two children are created as a combination of two parents. In order to perform the crossover stage the whole population is traversed and two candidates are randomly picked.

After the crossover is performed, there is a high probability that some of the characters will be contained more than once in the tree. The leaves are thus traversed and if a second occurrence of a symbol is found, it is removed from the leaf. After the traversal, the characters that have not been assigned are assigned to the blank nodes.

In case of mutation, each allele is mutated with a chosen probability of mutation [9]. The mutation operator changes the depth of a leaf. The decision whether the depth is increased or decreased is based on number of occurrences of the symbol (of the leaf) in the input data. If its number of occurrences is higher than the arithmetical average, its depth is lowered. This causes that the symbols with higher number of occurrences are pushed to lower depths. This technique follows the same idea as the construction of the Huffman tree.

Crossover and mutation are not the only genetic operators used in genetic algorithms. For example, the paper [8] also utilizes other genetic operators - translocation and switch operator. Unlike translocation switch operator does not make sense in our implementation due to the requirement for the left child to have earlier occurrence in input data. For translocation, chromosomes are again selected based on probability. In the selected chromosome, the symbols in two random leaves are exchanged. The difference between the implementation in [8] and ours is that in our case the translocation is performed only on leaves and irrespective of their depths.

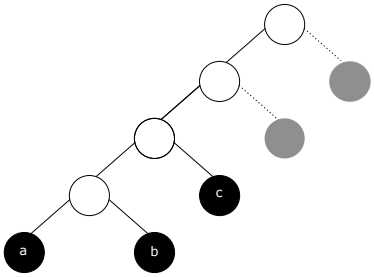

Fig. 3. Tree balancing

4.5 Tree Corrections

A non-negligible part of the execution time belongs to managing the created trees. As mentioned before, the MSC tree is not arbitrary. One of the things that needs to be managed is to force its binary structure. This is not a concern when generating the population considering the way the trees are created - two children are added to a node. However, after applying the genetic operators the trees need corrections. For this purpose a technique called balancing is employed.

When balancing the tree we have to look at the number of indicators. Firstly, it has to be found out how many leaves are remaining to be added to the tree and at least how many of them need to be added to get a valid binary tree. When these two indicators are equal it is required that the remaining nodes are added and the tree does not grow deeper. This is illustrated in Figure 3. It is obvious that when the input data contains 5 symbols and after assigning 3 we have two leaves to add. To get a valid tree at least two nodes need to be added. Therefore the last two nodes are added to these positions and the depth does not grow. Table 1 depicts an example of chromosome correction ({symbol,depth} pairs). In the first row a chromosome after crossover is illustrated whereas in the second row the balanced chromosome that is created from the first chromosome is shown.

Secondly, when a leaf is left child of its parent then the depth of the following leaf can be greater than or equal to the depth of the left child. Table 2 depicts

Table 1. Obtained and balanced chromosome after crossover

Obtained chromosome	a	4	b	4	c	3	d	3	e	3
Balanced chromosome	a	4	b	4	c	3	d	2	e	1

an example where the depth of symbol b is changed to 3 by mutation. In this case, this symbol has to be a right child of the node of depth 3 so symbol b must be placed to depth 4 or higher. The corrections are done in a way to reduce the difference between the obtained and the corrected value. The number 4 is thus selected in the given example.

Table 2. Obtained and balanced chromosome after mutation

Obtained chromosome	a	4	b	3	c	3	d	2	e	1
Balanced chromosome	a	4	b	4	c	3	d	2	e	1

Another correction technique forces the above-mentioned rule which states that each left child has earlier occurrence in the input data than the right child. This correction is performed before the fitness calculation and selection. For this task, a tree representation that contains all tree nodes (not only leaves) is needed. The chromosome is thus converted to the left tree representation as described in [10]. The process of correction is rather straightforward. If the right child has earlier occurrence in the input data then the left child the subtrees are exchanged.

5 Experimental Results

This section presents the achieved results of the designed genetic algorithm. Moreover it delineates how much these results differ from the original version of tree construction.

The genetic algorithm was tested on two sets of data. For this purpose, two random English texts were selected as input data with lengths – 5580 and 11508 characters. These data lengths were enough to obtain majority of nodes in MSC tree coded by ZEBC. If the used input data was too short, the effect of preprocessing by BWT would be minimal and the results would not be much different from results for unprocessed data.

Four experiments were performed – each input data were tested unprocessed and preprocessed by BWT. To obtain results with statistical significance each experiment was repeated ten times. The results can be seen in Table 3. Each experiment is identified by IDL (input data length) and by the usage of BWT.

The remaining columns in the table then contain: AL - average value for all repetitions after 100 generations. ML is the average of the minimum lengths and

Table 3. Obtained results for experiments with 100 generations

IDL	BWT	AL	ML	TML	BAL
5,580	yes	14,490 (96.46%)	14,317 (95.31%)	14,270 (94.99%)	15,022
5,580	no	24,931(104.39%)	24,169(101.20%)	24,154 (101.13%)	23,883
11,508	yes	37,294 (98.04%)	36,985 (97.23%)	36,876 (96.95%)	38,038
11,508	no	52,620 (104.47%)	51,133 (101.52%)	50,994 (101.24%)	50,368

TML is the total minimum length that was achieved in all repetitions of the experiment. BAL value is the length of coded data when using the original tree construction process.

The results in the table show that an improvement of compression ratio can be achieved when preprocessing of input data is performed. The rate of improvement is not insignificant since the genetic algorithm created trees reaching improvement of up to 5%. It can also be seen that in case of unprocessed data the genetic algorithm found the trees which at best produced a coded length of around 1% worse than the original algorithm. Therefore, the improvement for preprocessed data is expected to be even slightly higher than the reported value.

Figures 4 and 5 show the development of average and minimum values in generations. The progressions actually show the average values (of average and minimal length) from all repetitions and the dashed lines depict the standard deviation. The turquoise constant line denotes the length obtained by the original algorithm and the green line shows the total minimum obtained. The plots are shown for cases when preprocessing of input data is used. It clearly demonstrates that from the first generations the trees obtained by the genetic algorithm give better results than the original algorithm.

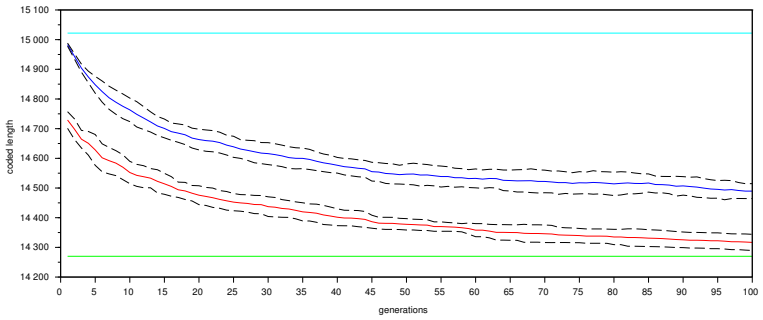

Fig. 4. Progress of average and minimum values in generations for a text with 5580 characters

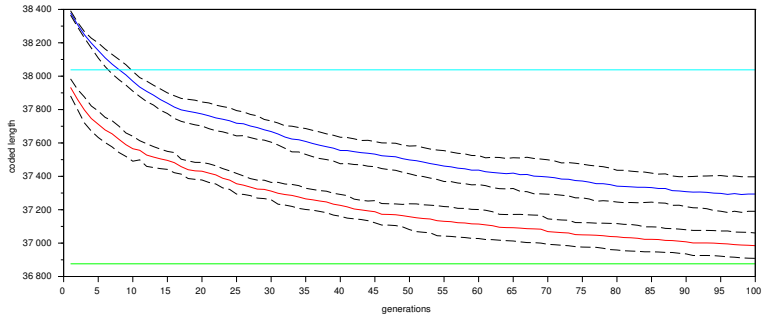

Fig. 5. Progress of average and minimum values in generations for a text with 11508 characters

6 Conclusion

This paper presents a genetic algorithm application for the exploration of possible improvements of the MSC tree. It was shown that when the majority of counters are coded by ZEBC coding, the tree construction process of the original MSC algorithm is not ideal. It was found that the improvements in terms of coded length of the data is up to about 3 - 5%.

Moreover, it was found that in case of unprocessed data, where no or very small improvement is expected, the genetic algorithm finds solutions that in the best case are about 1% worse compared to the original algorithm. This means that further improvements might be achieved if the majority of counters is coded by ZEBC.

The next logical step is to create a new tree construction algorithm which would improve the compression ratio. This can possibly be achieved by genetic programming application that would find this algorithm via optimization. Alternatively, some metrics within the tree can be found that would guide the tree construction based on a determined relationships.

References

1. Kochanek, J.: "Zpusob transformace a bezeztratove komprimace dat v elektronicke podobe, (2007)
2. Kochanek, J., Lansky, J., Uzel, P., and Zemlicka, M.: The new statistical compression method: Multistream compression,First International Conference on the Applications of Digital Information and Web Technologies, Ostrava : IEEE, (2008)
3. Uzel, P.: Entropic coders, Diploma thesis, Charles University, Faculty of Mathematics and Physics, Department of Software Engineering, (2009)
4. Elias, P.: Universal codeword sets and representation of the integers. IEEE Trans. on Information Theory, 21(2):194 - 203, (1975)

5. Unger, L.: Improvements of multistream compression, Diploma thesis, Charles University, Faculty of Mathematics and Physics,Department of Software Engineering, (2010)
6. Gulek, M., Toroslu, I. H.: A genetic algorithm for maximum-weighted tree matching problem. Applied Soft Computing 10 1127 - 1131, (2010)
7. Podgorelec, V., Karakatic, S., Barros, R. C., Basgalupp, M. C.: Evolving Balanced Decision Trees with a Multi-Population Genetic Algorithm. IEEE Congress on Evolutionary Computation (CEC): 54 – 61, (2015)
8. Sorensen, K., Jannsens, G. K.: Data mining with genetic algorithms on binary trees. European Journal of Operational Research 151: 253 - 264, (2003)
9. Marik, V., Stepankova, O., Lazansky, J.: Umela inteligence. Praha: Academia, (2001). ISBN 80-200-0472-6.
10. Fabera, V., Musil, T., Rada, J.: The first hardware MSC algorithm implementation. Neural Network World 27(6): 541 – 555, (2017)