Universal Intelligent Data Compression Systems: A Review

Ahmed Kattan
School of Computer Science and Electronic Engineering
University of Essex, United Kingdom
akatta@essex.ac.uk

***Abstract*—Researchers have classically addressed the problem of universal compression using two approaches. The first approach has been to develop adaptive compression algorithms, where the system changes its behaviour during the compression to fit the encoding situation of the given data. The second approach has been to use the composition of multiple compression algorithms. Recently, however, a third approach has been adopted by researchers in order to develop compression systems: the application of computational intelligence paradigms. This has shown remarkable results in the data compression domain improving the decision making process and outperforming conventional systems of data compression. This paper reviews some of the previous attempts to address the universal compression problem within conventional and computational intelligence techniques.**

I. Introduction

Data compression [1], [2], [3] is one of many technologies that enables today's information revolution. High-quality digital TV would not be possible without data compression: *one second* of video transmitted or stored without compression using the traditional CCIR 601 standard (625 lines per frame, 720 samples per line) would need over 20 MB, i.e., 70 GB for a two-hour movie! Also, the clarity we experience in our phone calls and in the music produced by our MP3 players would not be possible without data compression: 2 minutes of a uncompressed CD quality (16 bit per sample) music file would require more about 80 MBs. Without compression uploading or downloading online videos and music would consume prohibitive amounts of bandwidth and/or time. Even faxing documents over ordinary phone lines would have not been possible without data compression.

So, what exactly is data compression? Data compression is the process of observing regularities in the data and trying to eliminate them with the aim of reduce the total size of the data. Traditional compression techniques involve analysing the data and identifying regular patterns within them using a single model. Such techniques are designed to capture particular forms of regularities and, therefore, they are not practical when dealing with board range of data types. Thus, new methods are needed to advance the data compression domain.

Data compression is a sophisticated process and a good universal compression algorithm would need to take intelligent actions to allow for the encoding of different types of data. Researchers have classically addressed the problem of universal compression using two approaches. The first approach has been to develop adaptive compression algorithms, where the system changes its behaviour during the compression to fit the encoding situation of the given data. The second approach has been to use the composition of multiple compression algorithms [4]. Recently, however, a third approach has been adopted by researchers in order to develop compression systems: the application of computational intelligence paradigms. This has shown remarkable results in the data compression domain improving the decision making process and outperforming conventional systems of data compression. However, researchers are facing difficulties that are limiting the practicality of these approaches.

This paper reviews some of the previous attempts to address the universal compression problem within conventional and computational intelligence techniques.

978-1-4244-9030-1/10/$26.00 © 2010 IEEE

II. Conventional Universal Data Compression

Most universal compression techniques collect statistical information to identify the highest frequency patterns in the data. In this section some of the previous attempts to address the universal compression problem by utilising statistical approaches are reviewed.

A. Adaptive Compression Algorithms

Typically data compression algorithms have been implemented through a two-pass approach. In the first pass, the algorithm collects information regarding the data to be compressed, such as the frequencies of characters or substrings. In the second pass, the actual encoding takes place. A fixed encoding scheme is required to ensure that the decoder is able to retrieve the original message. This approach has been improved through so called *adaptive compression*, where the algorithm only needs one pass to compress the data [5]. The main idea of adaptive compression algorithms is that the encoding scheme changes as the data being compressed. Thus, the encoding of the n^{th} symbol is based on the characteristics of the data until position $n-1$ [5]. A key advantage of adaptive compression is that it does not require the entire message to be loaded into the memory before the compression process can start.

Adaptive Huffman coding [6] is a statistical lossless compression where the code is represented using a binary tree structure. This algorithm gives the most frequent characters that appear in the file to be compressed shorter codes than those characters which are less frequent. Top nodes in the tree store the most frequent characters. To produce the compressed version of a file, the algorithm simply traverses the tree from the root node to the target character. At each step the algorithm will encode 1 if it has moved to the left branch and 0 otherwise. The Huffman tree creates a unique variable length code for each character. The novelty of the Adaptive Huffman compression is that nodes swap locations within the tree during the compression. This allows the algorithm to adapt its compression as the frequency of the symbols changes throughout the file.

Adaptive Arithmetic Coding (AC) is a statistical compression that learns the distribution of the source during the compression process [3]. AC has historic significance as it was the best alternative to Huffman coding after a gap of 25 years [3]. Adaptive AC encodes the source message to variable-length code in such a way that frequently used characters get fewer bits than infrequently used ones. Unlike other compression techniques which replace blocks of the data with smaller code words, AC encodes the entire message into a single real number. Hence, the AC is based on the mathematical fact that the cumulative probability of a particular sequence of characters has a unique and small subinterval within $[0, 1)$. The algorithm simply calculates the cumulative probability of each symbol within the message at a time. The count for each encountered symbol increases after it has been encoded. Then, the cumulative count table is updated accordingly. Each symbol leads the algorithm to a new smaller subinterval based on its probability. The compression process starts with the initial interval $[0, 1)$, and keeps iterating until it reads the entire source. The final output is a single real number selected from the smallest identified subinterval. In the decompression process, the algorithm receives the encoded real number as input and starts from the initial interval. The algorithm divides the main interval into smaller subinterval according to which section the input falls in. The new subinterval become the new main interval and output the corresponding symbol. This process keeps iterating until the entire message is decoded.

Another adaptive compression is Prediction by Partial Matching (PPM) which was proposed by Cleary and Witten in 1984 [1]. This algorithm is classified as statistical compression. The main ideas of this algorithm are context modelling and prediction. PPM tries to predict the probability of a particular character being in a specific location from the previous n symbols previously occurred. To this end, a table of statistical information is created, which stores strings of size o with the probabilities of their following characters. The number of previous symbols o is called the order of the PPM model. If the symbol to be encoded has not been previously encountered in the context then an escape symbol is encoded and the o is reduced. In the next iteration the algorithm uses a table of size $o - 1$. This process keep iterating until the algorithm encounters the target symbol or it concludes that the symbol has never been seen before. In this case (i.e., the symbol was never seen before), the algorithm updates the statistical tables with a new enrty (i.e., add the new symbol with its probability and its o previous symbols). The probability of the newly added symbol is $1/M$, where M is the size of the source alphabet [1]. The actual encoding of the symbols is done with Arithmetic Coding compression. A variant of the algorithm called PPMD [7] where it increments the probability of the escape symbol every time it is used.

Cormack and Horspool introduced Dynamic Markov Compression (DMC) in 1987 [5]. DMC is another statistical compression algorithm using on context modelling and prediction. Similarly to PPM, it uses Arithmetic Coding to encode predicted symbols. The difference, however, is that the model operates at the bit level rather than on bytes. Although this model has similar performance to PPM it is not widely used.

Lempel and Ziv developed a universal compression system in 1977 known as LZ77 [8]. Later, in 1978, an improved version was presented which is referred to as LZ78 [1]. These algorithms are classified as dictionary-based compression. The main idea is to replace frequent substrings in the data with references that match the data that has already have been passed to the encoder. The encoder creates a dictionary for the common substring in the file and uses it as reference for the data. Both encoder and decoder must share the same dictionary to perform the compression/decompression process. In LZ77 a sliding window of predefined length is used to maintain the dictionary. Data are scanned via a look-ahead buffer and compared with the previous data within the sliding window. Once a match is found the dictionary is updated. LZ78 eliminated the need for a fixed size window and builds the dictionary out of all previously seen symbols. LZ78 works very well where the frequent symbols are distributed in isolated locations in the file. Unlike LZ77, strings in LZ78 can be longer which gives flexibility and higher performance. The adaptability of the LZ algorithms comes from the fact that the algorithms update the dictionary contents during the compression process to tune the algorithm's behaviour as the frequency of the symbols change throughout a file.

In some sense adaptive compression algorithms represent a form of intelligent systems that adapt their behaviour based on the given encoding situation. The primary advantage of adaptive compression is that it uses a single pass to compress data. This accelerates the compression process and makes it appealing for use in the cases where speed is favoured over performance. The use of a single pass has the disadvantage of sacrificing the ability to see ahead in the file to determine future fluctuations which may contain valuable information for the encoding scheme. In adaptive compression once the algorithm makes the decision to encode the data in a particular way it will not be able to change it, even if the algorithm later learns a better way of encoding this information. Another problem with adaptive compression algorithms is that they are sensitive only to changes in relation to the particular redundancy type which they are specialised to detect, hence, they are not able to exploit different forms of redundancy [4]. Also, because they learn the alphabetic distribution during the compression time, they cannot handle drastic changes in the files streams as, for example, is the case with changes from text to picture. Consequently, adaptive schemes are not recommended for heterogeneous files where multiple types of data are stored in a single file and their constituent parts have different degrees of compressibility.

B. Composite Compression Algorithms

In many real world applications, it is ineffective to use a single compression model to adapt the regularities of the data. Therefore, a practical approach is to apply a composite model, which can be defined as a combination of several models where only one model can be active at any given time [1]. This is another approach to develop universal compression algorithms and to achieve higher compression. It has been reported that this approach show superior performance to standard compression algorithms. A successful composition of compression algorithms can be achieved in two ways. Firstly, by running a number of compression algorithms successively. Secondly, by combining a number of simple compression algorithms and heuristically selecting them where they are expected to perform best.

A well-known composite compression system is Bzip2. Bzip2 is a free, open-source, composite and lossless compression algorithm developed in 1996 [9]. It has been widely used in many commercial compression applications, such as WinZip [10]. In this algorithm, the data are compressed through several compression and transformation techniques in a particular order. The reverse order is used in the decompression process. Bzip2 compresses files using the Burrows-Wheeler transform and the Huffman coding. This technique has been found to be better than LZ77 and LZ78 [9]. Bzip2 is known for being slow in the compression process and much faster in decompression.

Katz in [11] proposed an algorithm that combines LZ77 with Huffman coding as an improvement for the PKZIP archiving tool. This algorithm is referred to as *Deflate*. In the original implementation of LZ77, the algorithm tries to match strings within a sliding window with a look-ahead buffer. Matched strings are used to build a dictionary. Each repeated string is replaced each with a triplet (pointer [1], length and next symbol). The next symbol element is needed in case there is no exact match in the dictionary (e.g., William and Will). In Deflate a variant of LZ77 is used, which eliminates the third element and encodes a pair (pointer, length). Unmatched characters are written in the compressed stream [2]. In the original implementation of the Deflate algorithm, compressed data consisted of a sequence of blocks corresponding to successive blocks of input data. These blocks can be of different lengths based on the various prefix codes used and the memory available to the encoder. The algorithm has three options for each input block: *i) Apply no compression*, which is used if the data is already compressed, *ii) Compress with fixed Huffman code* and *iii) Compress with dynamic Huffman code*. Each uncompressed block is individually compressed using the previously described modification of LZ77 and then Huffman coding. Thus, each block is composed of two parts: *i) a Huffman tree* that describe the data and *ii) the compressed data itself*. Deflate has been widely implemented in many commercial compression applications, such as gzip, the HTTP protocol, the PPP compression protocol, PNG (Portable Network Graphics) and Adobe's PDF (Portable Document File) [2].

Another composite compression system is the Lempel-Ziv Markov-chain Algorithm (LZMA) [2]. LZMA uses a similar approach to Deflate. The difference is that it uses range encoding instead of Huffman coding (the range encoder is an integer-based version of Arithmetic Coding) [2]. This enhances the compression performance, but at the expense of increasing the encoder's complexity. In LZMA, the input stream is divided into blocks, each block describing either a single byte, or an LZ77 sequence with its length and distance.

The main disadvantage of standard composite compression algorithms is that they are unreliable when dealing with heterogeneous files, i.e., files that are composed of multiple data types such as archive files (e.g., ZIP and TAR). This is because most standard composite compression schemes follow deterministic procedures that involve applying several compression models in a particular order. These procedures are selected based on the designer experience or experimental evidence which demonstrated their superior performance under certain conditions. Whilst these methods have proven to be successful in achieving high compression ratios, the use of deterministic steps to perform compression entails the disadvantage of making the encoding decision fixed and unable to deal with unpredictable types of data.

Another approach to developing composite compression system is allowing the system to heuristically select and apply compression algorithms where they are expected to perform best. This idea has been explored by Hsu in [4]. Hsu's system segmented the data into blocks of a fixed length (5 KB) and then compressed each block individually with the best compression algorithm. Four compression algorithms were used in Hsu's system, namely, Arithmetic Coding, Run-length encoding, LZW and JPEG for image compression. The system works in two phases. In the first phase, the blocks are scanned to determine the compressibility and the contents of each block. The compressibility of the blocks is calculated by measuring three different quantitative metrics: alphabetic distribution, average run length and string repetition ratio. The system considers the blocks already compressed if these measures were under a predefined threshold. Consequently, already compressed blocks are skipped. The files contents are determined using a modified Unix ***file*** command. This command is able to classify ten different types of data. The modified file command works by examining the first, middle and last (if it exists) 512 bytes and thereafter comparing their patterns with collections of known patterns from the Unix operating system. In the second phase, the actual compression takes place, where the system passes the blocks to the appropriate compression model based on the gathered information from the first phase. Experimentation with 20 heterogeneous test files revealed that

[1]pointer is an index in the dictionary

the proposed system was able to outperform other commercial compression systems with 16% saving on average. The main disadvantage of Hus's system is that the size of the blocks is fixed to 5KB, which limits the algorithm's ability to identify the true boundaries of heterogeneous fragments within the data. Moreover, the modified Unix command used detects ten file types only and is not reliable enough to guarantee that the blocks are passed to the optimal compression algorithm. This is because the system assumes that one particular compression algorithm is suitable for all files of a particular type. This assumption, however, is flawed as the compressibility of the data depends on the forms of regularities within the data and whether the compression algorithm is designed to capture them. These regularities are not necessarily to be correlated with a particular file type. Thus, at least in principle, two text files may be better compressed with two different compression algorithms depending on their contents.

III. Data Compression with Computational Intelligence Paradigms

Computational Intelligence (CI) researchers attempt to understand and simulate intelligent behaviour through the modelling of natural intelligence, such as evolution, insects swarms, neural systems and immune systems. This has resulted in a variety of paradigms including Artificial Immune Systems, Neural Networks, Particle Swarm Optimisation, Ant Colony, Fuzzy systems and Genetic Algorithms [12]. The main problems solved by CI paradigms include but are not limited to optimisation, classification, prediction and pattern recognition. Researchers have achieved significant successes in solving real world problems using these techniques.

Due to the great potential of current CI paradigms in solving complex problems, researchers have tended to apply some of these techniques to develop new intelligent data compression systems. However, thus far, only a little has been done to address this problem. Current research has focused mainly on investigating the applications of neural networks and genetic algorithms in order to explore the future of data compression. Applications of neural networks and genetic algorithms illustrate that CI paradigms in this area may be ready to play a significant role in assisting and complementing traditional techniques. In this section, we will review some of the previous attempts to use neural networks and genetic algorithms techniques to address this problem.

A. Neural Networks Applications in Data Compression

Artificial Neural Networks (ANN) have the potential to extend data compression algorithms beyond the standard methods of detecting regularities within the data. Although, they have often been avoided because they considered too slow for practical use. Nevertheless, neural networks have been applied to data compression problems. Existing research on neural network applications in compression can be summarised into three categories: transform coding, vector quantization, and predictive coding [13].

1) Code Transformation: In the first category (code transformation), the neural network is asked to identity a mapping between input and output which is then used for compression. Neural networks fit well with image compression because they have the ability to process input patterns to produce simpler patterns [14]. The network is composed of three layers; input, hidden and output [14]. Here, the desired output is to be identical to the input itself. Compression is achieved when the number of neurons in the hidden layer is smaller than the dimensionality of the input and output layers [14]. The input images are divided into blocks of $4 \times 4, 8 \times 8$ or 16×16 pixels [14]. The size of the input layer is equal to the number of pixels in the blocks. The neural network is trained to scale an input of N dimensions into narrower outlets of M dimensions at the hidden layer. It then produces the same input at the output layer. The quality of the network is calculated by measuring the difference between input and output. The idea is in the activiation of the neurons in the hidden layer will be saved or transmitted as the compressed version of the image. The original image will be reconstructed using the output layer to achieve the decompression process. However, it should be noticed that the activities of the hidden layer are real numbers between -1 and 1 which will most likely require greater storage space than the image itself. Therefore, the outputs of the hidden layer are encoded (quantised) to reduce the size of the data. This approach has been implemented in [15], where experimentation with three black and white images was conducted to prove its practicality.

This approach has been improved by Hassoun in [16] who proposed an alternating Hebbian algorithm in order to improve the training approach for the decompression part in the network. Also, the learning rule was extended to capture nonlinear relationships among the component in the training patterns. In [14] a cumulative distribution function was estimated for the images and used to map images pixels. Experimentation showed that this improves the compression ratio with back-propagation networks and accelerates convergence.

Namphol *et al.* [17] proposed a hierarchical architecture in order to improve the basic back-propagation network. In this work, two more hidden layers were added to the overall network. The three hidden layers are referred to as the combiner layer, compressor layer and decombiner layer respectively. All hidden layers are fully connected. The aim is to exploit the correlation between pixels via the combiner layer and the correlation between blocks of pixels via the decombiner layer. Thus, compression is achieved by dividing the target image into a number of blocks and each block is further divided into smaller blocks. The proposed system used nested training algorithm to improve the training time. Experimentation with computer generated images and real world images demonstrated the stability and generlisation of the system, however, the compression rate achieved by the system was not significant (1 bit/pixel) to that of standard data compression techniques.

An adaptive back-propagation neural network has been proposed to overcome the limitations in the basic back-propagation in [18]. The basic idea is that different neural networks have different performance and only certain types of networks work best with a particular set of images. As a result, a group of neural networks with an increasing number of hidden layers in the range (h_{min}, h_{max}) are used in a single system. The aim is to match the input image blocks with the most appropriate network based on their complexity. Several training schemes have been proposed to train the new adaptive architecture, including parallel training, serial training, activity-based training and direction-based training [18].

2) Vector Quantisation: In the second category (vector quantisation), neural networks are used to encode the input image into a set of codewords. The input image is divided into blocks and then fed into the network as input. The input layer is K dimensional and $M < K$ output neurons are designed to calculate the vector quantisation codebook [18]. Each neuron represents one codeword. The learning algorithm aims to adjust the weights of the network in such a way as to associate the i^{th} neuron from the input layer with the c^{th} codeword [18]. Around this basic architecture, many learning algorithms have been developed to optimise the learning process, such as competitive learning [19], [20], fuzzy competitive learning [21], and predictive vector quantisation [22].

Laskaris and Fotopoulos in [23] proposed a modified learning scheme to improve codebook design, which describes the mapping from the input data to their compressed representation. In this work, the sequential presentation of the training patterns is controlled via an external, user-defined criterion. The proposed modification considers blocks with spatial structure (e.g., well-formed edges) more important than homogenous regions. Therefore, they require more information in the codebook in order to represent them. The *roulette-wheel* [24] sampling technique (borrowed from the genetic algorithm literature) has been applied to favour some of the training samples. A technique was also used to maintain the diversity of the training samples. The proposed training scheme was found to be specialised in the representation of edges while the standard training scheme is better tailored to luminance variations. Comparisons with the standard training procedure based on the quality of reconstructed image showed that roulette-wheel-based training achieved significant improvements in codebook design.

3) Predictive Coding: In the third category (predictive coding), neural networks were used to improve existing compression technology. Schmidhuber and Heil [25] combined 3-layered predictive neural networks with statistical coding schemes to compress text. In this work, neural networks trained by back propagation were used to approximate characters probabilities when given n previous characters. The predictor input comes from a time-window of size t that scans the data with steps of size p. The outputs of the network are then fed to standard statistical coding algorithms to encode the data. Two different coding methods were introduced in conjunction with the predictor network: Huffman coding and Arithmetic Coding. In [26], the same authors have extended their work and introduced a third method. In this work, prediction networks have been used in a similar manner as in PPM (see section II-A). Thus, a time-window corresponding to the predictor input scans the data sequentially. The output produced by the prediction network is compared with the actual data. Wrong predictions are stored in a separated file (Meta file) to be used in the decoding process. Then, Huffman coding is applied to the Meta file in order to reduce the total size of the compressed data. The decoding process starts from n default characters to be fed to the predictor network. The predictor will sequentially predict the next character. Information in the Meta file is used to correct wrong predictions and restore the original file without loss. Experimentation with these three methods revealed that they outperformed LZW and Huffman coding. The major disadvantage, however, is that the algorithm was too slow for practical use. Training on 10KB to 20KB consumed 3 days of computational time.

Mahoney [27] proposed a faster text compression method based on neural networks that could compress the same amount of data in about 2 seconds. In this work, a 2-layers predictive neural network that predicts one bit at a time was used. The proposed predictive network learns and predicts in a single pass. Experimentation with the proposed method showed better compression ratios, but slower than LZW. However, results were almost identical to PPM both in terms of speed and achieved compression ratios.

B. Genetic Algorithms Applications in Data Compression

Data compression algorithms often require a large search space to be explored in order to find some forms of redundancy within the data or to select an optimal encoding scheme based on the given files. Evolutionary Algorithms (EAs) can be used as a search engine for this purpose or even as a method to evolve new compression algorithms. Nevertheless, data compression is a highly sophisticated procedure and evolving a data compression algorithm is not an easy task. Yet, few attempts have been made to use EAs to evolve data compression models. Existing research mainly focuses on investigating the applications of Genetic Algorithm (GA) and Genetic Programming (GP) in order to explore their potentials in solving data compression problems. As we will see, the good compression ratios achieved by early attempt with these techniques in comparison to other conventional compression algorithms show the importance of further exploring the applications of EAs in the field of data compression.

Koza [28] was the first to use GP to perform compression. He considered, in particular, the lossy compression of images. The idea was to treat an image as a function of two variables (the row and column of each pixel) and to use GP to evolve a function that matches the original as closely as possible. Small, 30×30 pixel, images were treated as symbolic regression problems with just the basic arithmetic operators in the functions set. The evolved function can be then considered as a lossy compressed version of the image. The technique, which was termed *programmatic compression*, was tested just on one small synthetic image with good success.

Programmatic compression was further developed and applied to realistic data (images and sounds) by Nordin and Banzhaf [29]. In this research the authors presented the target data as a continuous sequence of numbers (fitness cases) and asked GP to evolve a program that could fit these data. A problem, however, appears with very large fitness cases (which is the case with most images) where GP may fail to find a solution of acceptable quality. As a result, an alternative approach was presented where the fitness cases were divided into equally sized subsets and solutions were evolved for each of them individually. The main disadvantage of the system is that the time required to compress a picture was up to 10 days.

In [30] the use of programmatic compression was extended from images to natural videos. A program was evolved that generates intermediate frames of a video sequence. A sequence of gray-scale frames was considered. Each frame is composed of a series of transformed regions from adjacent frames. The possible motions within a single frame can be expressed as a combination of scaling, shearing, rotating and change in luminance. The function set of the system was composed of one function **(Transform)** that encapsulates all these operations. The **Transform's** output is controlled by several parameters that set the object to be moved, the start location, the end location, and the luminance change. Programs are evaluated by measuring the difference between the approximated frames and the target frames. If a program achieves satisfactory approximation in one frame the system uses it for the following frame, utilising the fact that subsequent frames usually share similar characteristics. The results were encouraging as a good approximation to frames was achieved. Naturally, although a significant improvement in compression was achieved, programmatic compression was very slow in comparison with other methods, the time needed for compression being measured in hours or even days. In [31] an optimal linear predictive technique was proposed. Thanks to the use of a simpler fitness function, acceleration in GP image compression was achieved.

GAs have been used for image compression. In [32] Mitra *et al.* proposed a new GA-based method for fractal image compression. The proposed method utilises the GA for finding self similarities in the given image. Then, the system divides the given images into blocks and tries to find functions that approximate the target block. Here, the squared mean error was used as fitness function to evaluate the quality of the individuals.

Iterated Functions Systems (IFS) are important in the domain of fractals and the fractal compression algorithms. [33] and [34] used GP to solve the inverse problem of identifying a mixed IFS whose attractor is a specific binary image of interest. The evolved program can be taken to represent the original image. In principle this can be further compressed. The technique is lossy, since the reverse problem can rarely be solved exactly. No practical application or compression ratio results were reported in [33] and [34]. Using similar principles, Sarafoulous [35] used GP to evolve affine IFSs whose attractors represent a binary image containing a square (which was compressed exactly) and one containing fern which was achieved with some error in the finer details).

The application of GAs to accelerate fractal coding time has been explored in several works. In [36], [37] a GA has been used to compress fractal images by finding Local Iterated Functions Systems that encode a single image and reducing the time needed to 30% compared with other methods. In [38] a GA implementation to speed up the compression without significant loss of the image quality was proposed. The GA searches the optimal parameters for domain block coordinates and isometric flip. The luminance and contrast parameters are computed with standard equations. In this implementation, chromosomes include 5 genes, from which only 3 genes are submitted for genetic modification and the other 2 are computed with a standard equation. This was found to improve compression time, in the reported experiments. The time needed to perform compression ranged from 9 seconds to 23 minutes depending on the settings used for the system.

A first lossless compression technique was reported in [39], where GP was used to evolve non-linear predictors for images. These are used to predict the gray level a pixel will take based on the gray values of a subset of its neighbours (those that have already been computed in a row-by-row and column-by-column scan of the image). The terminals used for the GP system were the values of four neighbouring pixels, constants, arithmetic operators, and Min/Max functions. The prediction errors together with the model's description represent a compressed version of the image. These were further compressed using Huffman encoding. The system required several hours to compress a single image. Results on five gray images from the NASA Galileo Mission database were very promising, with GP compression outperforming some of the best human-designed lossless compression algorithms.

Text compression via text substitution has been investigated in several applications using both GA and GP. Ucoluk and Toroslu [40] used a GA to evolve a dictionary of syllables for the Huffman coding. The GA was used to search for a combination of the alphabet in such a way as to improve the Huffman coding performance. The chromosomes of each individual represent a selected combination of a predefined set of text. Individuals were evaluated by measuring the entropy of the encoded text. Experimentation with Turkish text corpora revealed that the achieved compression ratios when using evolved dictionaries were superior to those provided by standard methods. This work has been extended in [41] with Czech and English text where the approach has been tested with the LZW algorithm. Also, in [42] the same method has been tested, where GA was used to construct dictionaries for text substitution. Experiments demonstrated that GA always outperforms standard text dictionary-based compression methods.

Zaki and Sayed [43] used GP to improve the standard Huffman coding. The proposed system utilised GP to find the most repetitive substrings within the text to be compressed. In this work, the search population was a collection of nodes that describe different substrings. Then, the Huffman tree is generated to encode high frequency substrings with shorter references. For the proposed representation, the authors used specialised search operators. Reported results demonstrated that the Huffman tree generated with this system was able to achieve higher compression than the standard Huffman coding algorithm in some cases by small margins. However, due to various restrictions on the system, standard Huffman coding was not significantly improved.

In [44] Oroumchian *et al.* proposed an online text compression system for web application based on a GA. The system utilises a GA to find the most repetitive N-grams within the text and replaces them with shorter references. N-gram tables are stored to be used in the decompression process. In the first step, the system calculates the frequencies of 2-, 3-, 4- and 5- grams. The aim is to allow the GA to find the best combination of N-grams replacements in such a way as to achieve a high compression ratio while minimising the total number of N-grams used. The compression process is carried out on the HTTP server side. The decompression process is performed during the loading of the web page. The proposed method was tested on Persian text. The best result reported by the proposed system was 52.26%. However, no comparisons with other compression techniques were reported.

In many compression algorithms some form of preprocessing or transformation of the original data is performed before compression. This often improves compression rates. In [45] Parent and Nowe evolved pre-processors for lossless compression using GP. The objective of the pre-processor was to reduce losslessly the entropy in the original image. The function set of the GP system was divided into two categories. Functions that read the input data into a buffer and functions that process the data read from it. All used operators are reversible in order to guarantee applicability for lossless compression. In tests with five images from the Canterbury Corpus [46] GP was successful in significantly reducing image entropy. As verified via the application of Bzip2, the resulting images were markedly easier to compress.

Many conventional compression systems have a number of parameters, e.g., window size, prediction order, dictionary limits, etc. Such parameters typically influence the compression performance. The ideal setting for these parameter is depending on the structure of the file to be compressed in most cases. For this purpose, GAs and GP have been used to optimise the parameters of existing compression systems (e.g., as previously reviewed in [40], [41], [43]). Recently, Burtscher and Ratanaworabha [47] proposed a system based on a GA to tune the parameters of the FPC compression [48]. FPC divides the data to be compressed into blocks and processes each block individually. The GA was used to initialise the population of settings (i.e., each individual represents a different configuration). Each individual in the first population applies its settings to the FPC and compresses the first block. The designed fitness function favoured individuals that yield highest compression ratios. The next block is compressed with the new generation. Thus, the size of the blocks is directly related to the number of generations. Experimentation with several versions of the FPC algorithm revealed that evolution was able to find optimal settings for each block, which led the algorithm to achieve higher compression ratios in comparison with the standard versions of the algorithm. Typically, this evolutionary process slows the compression phase significantly.

Klappenecker [49] used GP to find optimal parameters to optimise the performance of conventional wavelet compression schemes, where internal nodes represented conjugate quadrate filters and leaves represented quantisers. The aim was to minimise the rate-distortion while maintaining high compression ratios. Results on a small set of real world images were impressive, with the GP compression outperforming JPEG at all compression ratios.

Wavelets are frequently used in lossy image and signal compression. In [50] Grasemann and Miikkulainen used a coevolutionary GA to find wavelets that are specifically adapted to fingerprint images. The evolved wavelets were compared with human designed wavelets that were used by the FBI to compress fingerprint images and standard JPEG2000. In this work, the GA initialises several populations of lifting steps [2] in parallel and then randomly combined to form new wavelets. Each sub-population is evolved in isolation. At each generation, the best individuals are selected from each population to join the new wavelet. Results showed that the GA was able to outperform its competitors in terms of compressed image quality by a factor of 15% to 20%. The authors reported that evolution required approximately 45 minutes to complete the learning process.

Feiel and Ramakrishnan [52] proposed a new vector quantisation scheme using a GA (GAVQ) to optimise the compression of coloured images. In this work, the GA was used to find optimal, or more precisely near optimal codebooks that describe the mapping from the input data to their compressed version. Experiments showed that the results obtained are better than the LBG algorithm by of 5% to 25%.

Keong *et al.* [42] used GA to optimise the a Generalised Lloyd Algorithm (GLA). [3] GLA receives the input vectors and initialises the codebook randomly, such that it maps the input sequence to a compressed digital sequence. GLA refines the codebook though an iterative process in order to reduce the average distortion. The authors introduced a Genetic GLA (GGLA), which uses a GA to find optimal codebooks. Each chromosome in the GA population represents a codebook and the set of genes corresponds to the codewords. Simulations with the three versions of the proposed system have been conducted with first-order Gaussian-Markov processes. This showed that GGLAs outperformed GLA in most cases.

Intel patented a method in [53] to compress microcode based on GAs. The proposed method utilises a GA to find common patterns in the microcode's bit strings and store them in a table with a unique ID for each pattern. To achieve

[2]The lifting scheme introduced by Sweldens in [51] offers an effective way to construct complementary filter pairs. A finite filter called a Lifting step and can be used to construct a new filter pair. Thus, new wavelets can be constructed starting from a known complementary filter pair.

[3]GLA is a lossy image compression method based on vector quantisation.

this task, the system represents the microcode's bit strings in a matrix format and groups similar columns of microcode storage into clusters to minimise the total size.

Evolvable hardware (EHW) has been explored in the practical applications of data compression. In [54], [55] Salami *et al.* applied EHW to data compression applications. In this work, a new type of hardware evolution was proposed which is referred to as function-level EHW [56]. EHW was used as a predictive function for image compression. In this approach, the images to be compressed are divided into blocks and then EHW finds a function for each block. Thus, by finding a different predictive function for each region of the image EHW applies an adaptive predictive technique for the image. The system can be treated as lossy if it ignores errors, or lossless if it tracked the errors for the decompression process. Sekanina [57] extended this work and proposed a technique to allow balancing between achieved compression ratio and image quality. Four threshold values were added to the terminal set (randomly initialised). Each value corresponds to a quarter of the image. The system transfers any particular pixel (represented as position and value) to the compressed image if its prediction error is greater than the relevant threshold. Thus, transferred pixels are ignored during the decompression process. Threshold values change during evolution. The system only allows the transfer of pixels up to a maximum limit. To maintain a balance between compression and quality, the number of transferred pixels was treated as a penalty value in the fitness calculations. Experimentation with the proposed technique revealed that the quality of the images increase with the size of the population. However, JPEG still outperformed EHW in terms of compression achieved ratio. Furthermore, the time needed to complete the evolution process was considerably larger than for standard image compression methods.

A GA system for the optimisation of combined fuzzy image compression and decompression was introduced in [58]. Here, an image is divided into squares and a fuzzy system replaces the whole square with a single pixel of particular colour decided according to its neighbours. The square is a fuzzy set defined through a membership function that describes to what degree each neighbour pixel belong to the square. Thus, the content of the square is mapped into a single pixel in the compressed image. The reverse process is used in the decompression phase. The pixels of the compressed squares are restored by calculating their values with the appropriate membership function according to the value of the single pixel in the relevant square. Generally, some pixels may belong to more than one square with different degrees depending on their position. A GA has been used to optimise the parameters of the membership functions for both the compression and the decompression process. The Mean Square Error (MSE) between original and compressed images was used to guide evolution. Experimentation demonstrated that the system has good generalisation capabilities when tested with images outside of the training set. However, one of the main problems with restored images is the blurring of contours.

C. The GP-zip Family

Recently, Kattan and Poli proposed a series of intelligent universal compression systems (the GP-zip family) in [59], [60], [61], [62]. The main idea of which is to evolve programs that attempt to identify what is the best way of applying different compression algorithms to different parts of a data file so as to best match the nature of such data. He presented four members of the GP-zip family, namely, *GP-zip, GP-zip*, GP-zip2* and *GP-zip3.* Each new version addresses the limitations of previous systems and improves upon them.

With GP-zip the aim was to understand the benefits and limitation of combining existing lossless compression algorithms in a way that ensures the best possible match between the algorithm being used and the type of data it is applied to. To explore this idea GP-zip implemented a simple decision making mechanism. GP-zip uses a linear GP representation for its individuals. The system divides the given data file into segments of a certain length and asks GP to identify the best possible compression technique for each segment. GP-zip was executed multiple times until it could find the best size for the segments. The function set of GP-zip is composed of primitives that are themselves basic compression and transformation algorithms. These algorithms are treated as black boxes that receive input and return output, although some algorithms may use another learning mechanisms and/or multiple layer of compressions themselves.

In GP-zip*, the second member of the family, a new method for determining the length of the blocks was presented, which completely removes the need of a staged search for an acceptable fixed length typical of GP-zip. GP-zip*, also, uses a linear GP representation for its individuals. The system evolves the length of the blocks within each run, rather than use the staged evolutionary search, possibly involving many GP runs, of GP-zip. Each individual is divided onto blocks of different sizes to each of which a compression and/or transformation algorithm is allocated. The system changes the blocks' sizes and their allocated algorithms via new intelligent crossover and mutation operators which targeted hotspots in the selected individuals. The intelligence of these operators comes from the fact that, instead of acting on random blocks in the parents, they select blocks with the lowest compression ratio, which arguably are the most promising places where variations can be beneficial. This approach was found to achieve higher compression ratios and to require less computational effort than its predecessor.

Both GP-zip and GP-zip* outperformed state-of-the-art compression systems in terms of achieved compression ratios. However, the compression process is very slow and impractical for large files. This is a common problem for compression systems based on CI paradigm (see section III). In GP-zip2, the system aimed at addressing this particular weakness. GP-zip2 uses a tree-like representation for its individuals. The system treats the data to be compressed as digital signals represented using 8-bits per sample. GP evolves programs with multiple components. One component analyses statistical features extracted from the raw byte series and divides the data to be compressed into blocks. These blocks are projected onto a two-dimensional Euclidean space via two further (evolved) program components. K-means clustering is then applied to group similar data blocks into clusters. Each cluster is labelled with the optimal compression algorithm for its member blocks. After evolution, evolved programs can be used to compress unseen data. Experimental results showed that GP-zip2 compares very well with established compression algorithms in terms of the compression ratios achieved. Also, once compression programs are evolved, they can be used over and over again to perform compression. This compression process is much faster than other evolutionary compression techniques. These features make GP-zip2 appealing for practical use.

The major disadvantage of GP-zip2 that it requires on average 6.4 hours to evolve a practical solution. This is largely due to the costly fitness evaluation adopted in GP-zip2 which requires compressing data fragments using multiple compression algorithms. GP-zip3 tried to solve this particular problem in the system by using a novel fitness evaluation strategy. More specifically, GP-zip3 evolves and then uses decision trees to predict the performance of GP individuals without requiring them to be used to compress the training data. As shown in a variety of experiments, this speeds up evolution in GP-zip3 considerably over GP-zip2 while achieving similar compression results, thereby significantly broadening the scope of application of the approach. The major disadvantage of GP-zip3 is that its fitness evaluation depends on evolved estimation functions. Thus, the user must spend extra time evolving accurate estimation functions for each of the compression algorithms available to GP-zip3. This, however, needs to be done only once and then it can be reused many times.

In this research the author have produced systems that outperform most other compression algorithms. They come top of all compression algorithms tested on heterogeneous files and are never too far behind the best with other types of data. Reported results shows that GP-zip family evolved solutions are human-competitive based on Koza's 8 criteria for human-competitiveness [63].

IV. Conclusion

The main limitations of the previous techniques are twofold: *practicality* and *generality*. The implementation of CI techniques in the data compression domain has proven to smarten compression programs by allowing them to make educated decisions with regard to how to encode the data. These have often consequently outperformed conventional compression schemes. Naturally, this gain in performance comes at a fairly high cost in termes of increased program complexity and intensive load of computations. This makes CI-based compression algorithms slow and inapplicable for practical use in most cases. Nevertheless, some applications in the CI literature have been reported to be suitable for practical use (e.g, [14], [27], [40], [41], [44]). Yet, all of these applications are tailored to handle a single type of data, such as wavelets, images, text, ...etc, or are designed for a few specific classes of data. Hence, the second limitation is that none of the previously reviewed CI-based methods is a reliable universal compression technique.

There is an urgent need to have smart compression systems that can deal effectively with any regular data type and can tailor different encoding techniques based on the given situation. While GP-zip systems are perhaps present few steps toward a practical solution, they still suffer from some disadvantages as reported by the authors. More research in this area is still required to develop intelligent reliable compression systems.

References

[1] Khalid Sayood, *Introduction to Data Compression*, Morgan Kaufmann Publishers, San Francisco, CA, USA, second edition, 2000.

[2] David Salomon, *Data Compression: The Complete Reference*, Second edition, 2004.

[3] Ida Mengyi Pu, *Fundamental Data Compression*, Butterworth-Heinemann, Newton, MA, USA, 2005.

[4] William H. Hsu and Amy E. Zwarico, "Automatic synthesis of compression techniques for heterogeneous files", *Softw. Pract. Exper.*, vol. 25, no. 10, pp. 1097–1116, 1995.

[5] Gordon V. Cormack and R. Nigel Horspool, "Data compression using dynamic markov modelling.", *Comput. J.*, vol. 30, no. 6, pp. 541–550, 1987.

[6] Jeffrey Scott Vitter, "Design and analysis of dynamic huffman codes.", *J. ACM*, vol. 34, no. 4, pp. 825–845, 1987.

[7] John G. Cleary, W. J. Teahan, and Ian H. Witten, "Unbounded length contexts for PPM", in *Data Compression Conference*, 1995, pp. 52–61.

[8] Jacob Ziv and Abraham Lempel, "A universal algorithm for sequential data compression", *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 337–343, May 1977.

[9] Julian Seward, "Bzip2", Website, 2009, Nov, http://www.bzip.org/.

[10] WinZip, "The compression utility for windows", Website, 2009, Nov, http://www.winzip.com/.

[11] Peter Deutsch, "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951 (Informational), May 1996.

[12] Andries Petrus Engelbrecht, "Computational intelligence: An introduction.", *J. Artificial Societies and Social Simulation*, vol. 7, no. 1, 2004.

[13] Robert D. Dony and Simon Haykin, "Neural network approaches to image compression", *Proceedings of the IEEE*, vol. 83, no. 2, pp. 288–303, 1995.

[14] S. Anna Durai. and E. Anna Saro, "Image Compression with Back-Propagation Neural Network using Cumulative Distribution Function", *International Journal of Applied Science, Engineering and Technology*, vol. 3, no. 4, pp. 185–189, 2006.

[15] B. Verma, M. Blumenstein, and S. Kulkarni, "A Neural Network based Technique for Data Compression", in *Proceedings of the IASTED International Conference on Modelling and Simulation, MSO*. IASTED, vol. 97, pp. 12–16.

[16] Mohamad Hassoun and Agus Sudjianto, "Compression net-free autoencoders", in *Workshop on Advances in Autoencoder/Autoassociator-Based Computations at the NIPS*, 1997, vol. 97.

[17] Aran Namphol, Steven H. Chin, and Mohammed Arozullah, "Image compression with a hierarchical neural network", *IEEE Transactions on Aerospace and Electronic Systems*, vol. 32, no. 1, pp. 326–338, 1996.

[18] J. Jiang, "Image compression with neural networks- a survey", *Signal Processing: Image Communication*, vol. 14, no. 9, pp. 737–760, 1999.

[19] Stanley C. Ahalt, Ashok K. Krishnamurthy, Prakoon Chen, and Douglas E. Melton, "Competitive learning algorithms for vector quantization.", *Neural Networks*, vol. 3, no. 3, pp. 277–290, 1990.

[20] Syed A. Rizvi and Nasser M. Nasrabadi, "Finite-state residual vector quantization using a tree-structured competitive neural network", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 7, no. 2, pp. 377–390, 1997.

[21] Fu-Lai Chung and Tong Lee, "Fuzzy competitive learning.", *Neural Networks*, vol. 7, no. 3, pp. 539–551, 1994.

[22] Nader Mohsenian, Syed A. Rizvi, , and Nasser M. Nasrabadi, "Predictive vector quantization using a neural network approach (Journal Paper)", *Optical Engineering*, vol. 32, no. 07, pp. 1503–1513.

[23] Nikolaos A. Laskaris and Spiros Fotopoulos, “A novel training scheme for neural-network-based vector quantizers and its application in image compression.”, *Neurocomputing*, vol. 61, pp. 421–427, 2004.

[24] Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee, *A Field Guide to Genetic Programming*, Published via `http://lulu.com`, 2008, (With contributions by J. R. Koza).

[25] Jrgen Schmidhuber and Stefan Heil, “Predictive coding with neural nets: Application to text compression.”, in *NIPS*, Gerald Tesauro, David S. Touretzky, and Todd K. Leen, Eds. 1994, pp. 1047–1054, MIT Press.

[26] Jrgen Schmidhuber and Stefan Heil, “Sequential neural text compression”, *IEEE Transactions on Neural Networks*, vol. 7, no. 1, pp. 142–146, 1996.

[27] Matthew V. Mahoney, “Fast text compression with neural networks.”, in *FLAIRS Conference*, James N. Etheredge and Bill Z. Manaris, Eds. 2000, pp. 230–234, AAAI Press.

[28] John R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, MA, USA, 1992.

[29] Peter Nordin and Wolfgang Banzhaf, “Programmatic compression of images and sound”, in *Genetic Programming 1996: Proceedings of the First Annual Conference*, John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, Eds., Stanford University, CA, USA, 28–31 July 1996, pp. 345–350, MIT Press.

[30] Thomas Krantz, Oscar Lindberg, Gunnar Thorburn, and Peter Nordin, “Programmatic compression of natural video”, in *Late Breaking Papers at the Genetic and Evolutionary Computation Conference (GECCO-2002)*, Erick Cantú-Paz, Ed., New York, NY, July 2002, pp. 301–307, AAAI.

[31] Jingsong He, Xufa Wang, Min Zhang, Jiying Wang, and Qiansheng Fang, “New research on scalability of lossless image compression by GP engine”, in *Proceedings of the 2005 NASA/DoD Conference on Evolvable Hardware*, Jason Lohn, David Gwaltney, Gregory Hornby, Ricardo Zebulum, Didier Keymeulen, and Adrian Stoica, Eds., Washington, DC, USA, 29 June-1 July 2005, pp. 160–164, IEEE Press.

[32] Suman K. Mitra, C. A. Murthy, and Malay Kumar Kundu, “Technique for fractal image compression using genetic algorithm.”, *IEEE Transactions on Image Processing*, vol. 7, no. 4, pp. 586–593, 1998.

[33] Evelyne Lutton, Jacques Levy-Vehel, Guillaume Cretin, Philippe Glevarec, and Cidric Roll, “Mixed IFS: Resolution of the inverse problem using genetic programming”, *Complex Systems*, vol. 9, pp. 375–398, 1995.

[34] Evelyne Lutton, Jacques Levy-Vehel, Guillaume Cretin, Philippe Glevarec, and Cidric Roll, “Mixed IFS: Resolution of the inverse problem using genetic programming”, Research Report No 2631, Inria, 1995.

[35] Anargyros Sarafopoulos, “Automatic generation of affine IFS and strongly typed genetic programming”, in *Genetic Programming, Proceedings of EuroGP’99*, Riccardo Poli, Peter Nordin, William B. Langdon, and Terence C. Fogarty, Eds., Goteborg, Sweden, 26-27 May 1999, vol. 1598 of *LNCS*, pp. 149–160, Springer-Verlag.

[36] Lucia Vences and Isaac Rudomin, “Fractal compression of single images and image sequences using genetic algorithms”, *The Eurographics Association*, 1994.

[37] L. Vences and I. Rudomin, “Genetic algorithms for fractal image and image sequence compression”, *Proceedings Computacion Visual*, pp. 35–44, 1997.

[38] Faraoun Kamel Mohamed and BOUKELIF Aoued, “Optimization of fractal image compression based on genetic algorithms”, in *2nd International Symposium on Communications, Control and Signal Processing*, Marrakesh, Marocco, 2006.

[39] Alex Fukunaga and Andre Stechert, “Evolving nonlinear predictive models for lossless image compression with genetic programming”, in *Genetic Programming 1998: Proceedings of the Third Annual Conference*, John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, Eds., University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998, pp. 95–102, Morgan Kaufmann.

[40] Gktrk oluk and I. Hakki Toroslu, “A genetic algorithm approach for verification of the syllable-based text compression technique”, *Journal of Information Science*, vol. 23, no. 5, pp. 365, 1997.

[41] Tomas Kuthan and Jan Lansky, “Genetic algorithms in syllable-based text compression”, in *DATESO*, Jaroslav Pokorný, Václav Snásel, and Karel Richta, Eds. 2007, vol. 235 of *CEUR Workshop Proceedings*, CEUR-WS.org.

[42] Wee Keong, Sunghyun Choi, and Chinya Ravishankar, “Lossless and Lossy Data Compression”, *Evolutionary algorithms in engineering applications*, pp. 173 – 188, 1997.

[43] Mohammed Javeed Zaki and M Sayed, “The use of genetic programming for adaptive text compression”, *Int. J. Inf. Coding Theory*, vol. 1, no. 1, pp. 88–108, 2009.

[44] Farhad Oroumchian, Ehsan Darrudi, Fattane Taghiyareh, and Neeyaz Angoshtari, “Experiments with persian text compression for web”, in *WWW Alt. ’04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, New York, NY, USA, 2004, pp. 478–479, ACM.

[45] Johan Parent and Ann Nowe, “Evolving compression preprocessors with genetic programming”, in *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, New York, 9-13 July 2002, pp. 861–867, Morgan Kaufmann Publishers.

[46] Ross Arnold and Tim Bell, “A corpus for the evaluation of lossless compression algorithms”, in *DCC ’97: Proceedings of the Conference on Data Compression*, Washington, DC, USA, 1997, p. 201, IEEE Computer Society.

[47] Martin Burtscher and Paruj Ratanaworabhan, “gFPC: A Self-Tuning Compression Algorithm”, *Data Compression Conference,*, vol. 3, no. 4, 2010.

[48] Martin Burtscher and Paruj Ratanaworabhan, “High throughput compression of double-precision floating-point data.”, in *DCC*. 2007, pp. 293–302, IEEE Computer Society.

[49] Andreas Klappenecker and Frank U. May, “Evolving better wavelet compression schemes”, in *Wavelet Applications in Signal and Image Processing III*, Andrew F. Laine, Michael A. Unser, and Mladen V. Wickerhauser, Eds., San Diego, CA, USA, 9-14 July 1995, vol. 2569, SPIE.

[50] Uli Grasemann and Risto Miikkulainen, “Effective image compression using evolved wavelets.”, in *GECCO*, Hans-Georg Beyer and Una-May O’Reilly, Eds. 2005, pp. 1961–1968, ACM.

[51] Wim Sweldens, “The lifting scheme: A custom-design construction of biorthogonal wavelets”, *Applied and Computational Harmonic Analysis*, vol. 3, no. 2, pp. 186–200, 1996.

[52] Harald Feiel and Sub Ramakrishnan, “A genetic approach to color image compression.”, in *SAC*, 1997, pp. 252–256.

[53] Wu Youfeng and Msuricio Breternitz Jr, “Genetic algorithm for microcode compression”, Nov 2008, US Patent 7,451,121.

[54] Mehrdad Salami, Masahiro Murakawa, and Tetsuya Higuchi, “Data compression based on evolvable hardware.”, in *ICES*, Tetsuya Higuchi, Masaya Iwata, and Weixin Liu, Eds. 1996, vol. 1259 of *Lecture Notes in Computer Science*, pp. 169–179, Springer.

[55] Mehrdad Salami, Masaya Iwata, and Tetusya Higuchi, “Lossless ImageCompression by Evolvable Hardware”, in *Proc. Fourth European Conference on Artificial Life, MIT Press, Cambridge, MA*, 1997, pp. 28–31.

[56] Tetsuya Higuchi, Masahiro Murakawa, Isamu Iwata, Masayaand Kajitani, Weixin Liu, and Mehrdad Salami, “Evolvable hardware at function level”, in *IEEE International Conference on Evolutionary Computation*, 1997, pp. 187–192.

[57] Lukás Sekanina, “Evolvable hardware as non-linear predictor for image compression”, 1999.

[58] Mauro L. Beretta, Gianni Degli Antoni, and Andrea G. B. Tettamanzi, “Evolutionary synthesis of a fuzzy image compression algorithm”, 1996.

[59] Ahmed Kattan and Riccardo Poli, “Evolutionary lossless compression with GP-ZIP”, in *2008 IEEE World Congress on Computational Intelligence*, Jun Wang, Ed., Hong Kong, 1-6 June 2008, IEEE Computational Intelligence Society, IEEE Press.

[60] Ahmed Kattan and Riccardo Poli, “Evolutionary lossless compression with GP-ZIP*”, in *GECCO ’08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, Atlanta, GA, USA, 12-16 July 2008, pp. 1211–1218, ACM.

[61] Ahmed Kattan and Riccardo Poli, “Evolutionary synthesis of lossless compression algorithms with GP-zip2”, in *Submitted to GPEM*. Springer, 2010.

[62] Ahmed Kattan and Riccardo Poli, “Evolutionary synthesis of lossless compression algorithms with GP-zip3”, in *2010 IEEE World Congress on Computational Intelligence*, Barcelona, 18-23 July 2010, IEEE Computational Intelligence Society, IEEE Press.

[63] John R. Koza, Martin A. Keane, Matthew J. Streeter, William Mydlowec, Jessen Yu, and Guido Lanza, *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*, Kluwer Academic Publishers, 2003.