Evolutionary Synthesis of Lossless Compression Algorithms with GP-zip3

Ahmed Kattan and Riccardo Poli

Abstract— **Here we propose GP-zip3, a system which uses Genetic Programming to find optimal ways to combine standard compression algorithms for the purpose of compressing files and archives. GP-zip3 evolves programs with multiple components. One component analyses statistical features extracted from the raw data to be compressed (seen as a sequence of 8-bit integers) to divide the data into blocks. These blocks are then projected onto a two-dimensional Euclidean space via two further (evolved) program components. K-means clustering is applied to group similar data blocks. Each cluster is then labelled with the optimal compression algorithm for its member blocks. Once a program that achieves good compression is evolved, it can be used on unseen data without the requirement for any further evolution. GP-zip3 is similar to its predecessor, GP-zip2. Both systems outperform a variety of standard compression algorithms and are faster than other evolutionary compression techniques. However, GP-zip2 was still substantially slower than off-the-shelf algorithms. GP-zip3 alleviates this problem 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.**

I. Introduction

In 2007 IBM researchers predicted that by 2010 the digital contents available on the Internet would double every 11 hours. While this has clearly not happened, the growth in demand for storage is still phenomenal. One of the paradoxes of the evolution of technology is that, despite the rapidly increasing need for storage and transmission of information, there has been a lack of development in the area of compression techniques.

Two principles are commonly accepted in the field of data compression: *a) there is no algorithm that is able to compress all files, even by 1 byte*, and *b) less than 1% of all files can be compressed losslessly by 1 byte* [18]. In other words, it is difficult to find a universal compression algorithm that performs well on any type of data [18]. Thus, researchers in the compression field tend to develop algorithms that work with specific types of data, taking advantage of available knowledge about the data, effectively devoting less attention to the development of generic compression algorithms. Nevertheless such algorithms are extremely useful when the nature and regularities of a given data file are not predictable. For example, in archive systems, the users need to compress huge amounts of different data, such as text, music, pictures, video and so forth. A single universal compression model would be preferable in this case.

In recent research we proposed a system called GP-zip2 [11]. GP-zip2 is a lossless compression system based on Genetic Programming (GP) (e.g., see [12], [13], [17]). The main idea behind GP-zip2 is to divide the data into fragments based on their statistical characteristics and then match them with different compression models in such a way as to minimise the total size of a compressed file. GP-zip2 has achieved superior compression ratios in comparison with state of the art compression algorithms. The main disadvantage of GP-zip2 was the long training time required by the system: 13 hours with the training set in [11]. This is largely due to the costly fitness evaluation adopted in GP-zip2 which requires compressing data fragments using multiple compression algorithms.

In this paper we propose a substantial improvement of this system, called *GP-zip3*. This evolves and then uses decision trees to predict the performance of the population's individuals. As we will see, predicting the performance of individuals, rather than actually evaluating them, reduces training time significantly and further improves the system's stability.

The structure of the paper is as follows. In Section II some related work is briefly reviewed. Section III provides a description of GP-zip2 (the system's predecessors). In Section IV a detailed description of the proposed improvement is presented. This is followed by experimental results in Section V. Finally, some conclusions are given in Section VI.

II. Related Work

The problem of heterogeneous file compression has been tackled by Hsu in [6]. Hsu's system segmented data into blocks of a fixed length (5 KB) and then compressed each block individually. The system passed the blocks to an appropriate compression model by using a file-type detector which could recognise ten different types of data. The approach also used a statistical method to measure the compressibility of the data. However, due to various restrictions, results were not impressive.

Data compression requires highly specialised procedures. Therefore, evolving data compression algorithms is far from easy. One approach involves the use of GP in order to

Ahmed Kattan and Riccardo Poli are with the School of Computer Science and Electronic Engineering, University of Essex, UK, email: {akatta,rpoli}@essex.ac.uk.

find parameters for a compression algorithm, with the aim of maximising the compression ratio (e.g., [20]). Another approach is based on the use of GP for programmatic compression [15]. This is quite powerful, at least in principle, as was demonstrated by Nordin and Banzhaf, who used GP to achieve lossy compression for images and sounds [15].

Other researchers have also used GP in developing compression models. For example, [3] used GP for string compression. Fukunaga and Stechert [4] developed a nonlinear predictive model to compress grey scale images. Parent and Nowe [16] used GP to evolve transformation programs which reduced the entropy of the data thereby increasing the compression ratios obtained when compressing a file using lossless compression algorithms.

Our prior work with GP in the area of compression is summarised in the next section.

III. The GP-zip Family

The core idea of our research is to identify statistical regularities in the data to be compressed, divide up the data based on such regularities and match them with different compression algorithms, with the aim of maximising the lossless compression of the data.

A. *GP-zip and GP-zip**

Our first attempts in this direction were performed with two systems based on GP called GP-zip [8] and GP-zip* [9]. In both systems, the compression process for a file took place during evolution. These systems learned the data patterns of the file to be compressed and matched them with appropriate compression models. The difference between the two systems was in the representation for solutions: both were linear, but the second one was more flexible and efficient. The approach worked well providing very competitive compression performance in comparison to a variety of standard techniques.

The big disadvantage of these systems, however, is that the result of evolution is a recipe for compression, which is appropriate only for the specific file or archive of files used during evolution. In other words, the outcome of evolution is not reusable. Also, both approaches are extremely slow.

We attempted to overcome these limitations in a more recent system, called GP-zip2 [11]. The aim there was to evolve compression algorithms which would work well with a variety of data files, including unseen ones. This required a complete re-design of the decision making and learning strategies used in our systems.

Since the GP-zip2 system shares several components with GP-zip3, below we describe it in some detail.

B. *GP-zip2*

GP-zip2 uses the following five compression algorithms in its compression pool: Arithmetic Coding (AC) [19], Lempel-Ziv-Welch LZW [14], unbounded Prediction by Partial Matching (PPMD) [2], Run Length Encoding (RLE) [5], and Boolean Minimisation [7]. Each individual in GP-zip2's population (and, thus, also the outcome of evolution) is a program which processes the raw byte-series (seen as a digital signal) to be compressed and performs two major functions: *segmentation* of the signal based on its statistical features and *classification* of the identified segments based on their compressibility with a particular compression algorithm. To do so, GP was provided with a language that allows extracting statistical features out of the byte-series. Table I illustrates the primitive set of GP-zip2.

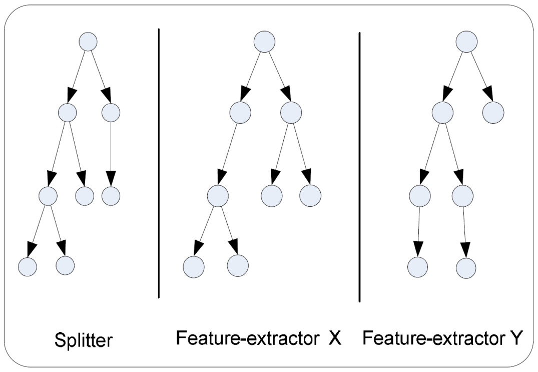

Fig. 1. GP-zip2 and GP-zip3 individual.

The same compression algorithms and primitive sets are used also in GP-zip3. GP-zip2 and GP-zip3 also use the same representation for individuals. Each individual has multi-tree representation composed of one splitter tree and two feature-extraction trees (without automatically defined functions, see [12], [17]), as depicted in Figure 1. These all use the same primitive set (see Table I), although the size of the array passed to the List terminal for the splitter trees is fixed, while it is variable for the feature-extraction trees, as we will clarify below.

The main job of splitter trees is to split the given raw byte-series into meaningful segments. By *meaningful* we mean that each segment can be effectively compressed using one of the compression algorithms available. An effective splitter tree should be able to detect the statistical differences within the data to be compressed and divide the file into different segments based on them. For example, if the given data was a document file that contained text and graphical charts, a good splitter tree would notice the change in the byte-series values from the text to the pictures and *vice versa*. Moreover, an ideal splitter tree might even detect different fragments within the same data type (e.g., a page full of blank lines within the text or white area in a picture).

To decide where to split, the system moves a sliding window of size L over the given byte-series with steps of S bytes. In our experiments we used $L = 300$ bytes and $S = 50$ bytes. At each step, the data in the window are stored in the List terminal and the splitter tree is evaluated. Its output is a number which is an abstract representation of the features of the signal in the window. The system then splits the byte-series at a particular position if the difference between the output of the splitter tree in two consecutive windows is more than θ (a predefined threshold). In our implementation $\theta = 10$.

TABLE I
GP-ZIP2 AND GP-ZIP3 PRIMITIVE SET

Primitive	*Arity*	*Input type(s)*	*Output type*
Median, Mean, Avg. deviation, Stdev, Variance, Signal size, Skew, Kurtosis, Entropy	1	Array of integers(0–255)	Real number
Plus, Minus, Div, Mul	2	Real number	Real number
Sin, Cos, Sqrt	1	Real number	Real number
List	0	NA	Array of integers(0–255)

The main job of the two feature-extraction trees in our GP representation is to re-represent the segments identified by the splitter tree and project them into a two-dimensional Euclidian space using a composition of primitives from Table I. To this end, each segment identified by the splitter tree is stored in the List terminal and the two feature-extraction trees are evaluated. Their numeric outputs can be seen as composite features. They are used as the coordinates of a point in a two-dimensional space, which can be seen as a re-representation of the corresponding segment of the data.[1] In principle, segments that share similar statistical features will form dense groups.

After this transformation of the segments, during training/evolution an unsupervised pattern classification approach is applied to them in order to discover regularities in the training data files. In particular, K-means clustering is used to organise segments (as represented by their two composite features) into groups in such a way that each group is best compressed with the same compression algorithm. With this algorithm, objects within a cluster are similar to each other but dissimilar from objects in other clusters. The system labels each cluster according to the dominant algorithm used to optimally compress the segments (more details provided in section IV-B). This is costly as it requires trying every available compression algorithm on each segment (as we will see, this is where GP-zip2 and GP-zip3 differ). In this phase, the best compression ratio achieved for the whole dataset is also computed since the contribution of the splitter tree to the fitness of an individual is determined by such a compression ratio.

After evolution is complete, we save the clusters found by K-means for the best program in the population together with its splitter and feature extraction trees.

In normal operations or in a testing phase, unseen files can be compressed using the trees and clusters identified during the training phase. When a test file is processed, the file is divided into segments by the splitter tree. These are then fed into the feature-extraction trees and projected into a two-dimensional space. A segment is assigned to the cluster whose centroid is closest (in terms of Euclidian distance) to the corresponding two-dimensional feature vector. The segment is then compressed with the compression algorithm associated to closest centroid.

[1]We tested the use of one, two and three feature-extraction trees, and found that two provided the best compromise between evolution speed and accuracy.

GP-zip2 significantly outperformed its predecessor as well as most other compression algorithms. It exceeded the performance of all the compression algorithms tested on heterogeneous files and produced competitive results in relation to other types of data. In addition, the division of data files into blocks presents the advantage that one can decompress a section of the data without processing the entire file. This is particularly useful, for example, if the data are decompressed for streaming purposes (such as music and video files). Also, the decompression process can be parallelised.

While GP-zip2 presents all these interesting properties, the system suffers from one main disadvantage: its long training time (approximately 13 hours for the training files used in [11]). This is due to the heavy computations associated with the fitness evaluation which requires compressing segments — a lengthy process — multiple times (with all available compression algorithms).

IV. GP-ZIP3

The aim of GP-zip3 is to significantly reduce the long training time required by GP-zip2 and take us one step further towards an ideal compression system. This would be a system that can rapidly adapt to the needs of users by producing compression systems that can quickly identify incompatible data fragments (both at the file level and within each file) in an archive, and to allocate the best possible compression model for each, in such a way as to minimise the total size of the compressed version of the archive.

In GP-zip2, the system optimally compresses the divided segments individually (trying every available algorithm on each segment) and measures the total compression ratio. Thus, in order to evaluate individuals, the system has to compress the training set many times with multiple compression models. This is the bottleneck of the training phase.

An idea to accelerate this step would be to use some form of prediction of the compression ratio that would be achieved by different compression models when applied to each data segment. If the prediction was accurate and faster than the actual compression with a model, presumably one could speed up the fitness evaluation in GP-zip2 by using such a prediction, without modifying the course of evolution too much.

However, predicting compression ratios is a very challenging task. So, before attempting to employ this strategy within an evolutionary compression system, we decided to test whether the problem could be solved at all to an acceptable degree. This was done in [10], where we tested an approach based on GP to predicting data compression ratios when applying different compression algorithms. The approach was successful and forms the basis for the new fitness evaluator of GP-zip3. We describe this in detail below.

A. Estimating Compression Ratios

To estimate compression ratios we used GP. The system evolves predictors which are specialised for a particular compression algorithm. Programs have three components. One component analyses statistical features extracted from the byte frequency distribution of a file in order to come up with a compression ratio prediction for that file. Another component does the same but by analysing statistical features extracted from the files raw ASCII representation instead. A further (evolved) component acts as a decision tree to determine the overall output (compression ratio estimation) returned by an individual. The decision tree produces its result based on a series of comparisons among statistical features extracted from the files and the outputs of the two prediction components. The evolved decision tree has the choice to select either the outputs of the two compression-prediction components or, alternatively, to integrate them into an evolved mathematical formula.

In [10], experimentation with this technique has produced accurate estimations for different files when applying different compression models. Also, the estimation of compression was much faster than performing the actual compression itself. Naturally, the evolved prediction models work well only for the particular compression algorithm they have been trained to predict. Thus, the user has to evolve a prediction model for each compression algorithm of interest.

In GP-zip3, prediction programs evolved with this approach were used to determine the best way to compress the segments produced by the splitter tree, rather than trying every available algorithm on each segment. This has accelerated the training phase significantly. Naturally, the evolved estimation programs are not 100% accurate. Thus, the system could occasionally allocate a sub-optimal compression model to some of the segments. We expected this to reduce GP-zip's ability to produce high performance compression algorithms. However, as we will see, this did not happen, although GP-zip3 has an inferior generalisation ability than its predecessor.

Naturally, the system proposed in [10] was designed and used for the prediction of the compression ratio of entire files. So, we had to customise this system in such a way as to make it work well with the segments of files produced by GP-zip's splitter tree. To achieve this, the training set was divided into blocks of predefined lengths, namely 200, 350, 450, 800, 1000, 2000, 3000 and 5000 bytes. These blocks sizes were selected so as to approximately match the typical segments produced by splitter trees. These blocks were treated by GP as independent files for which we wanted to estimate, as accurately as possible, the compression ratio achieved by a specific technique. Compression estimation modules were evolved for all the compression models available to GP-zip3, i.e., AC, PPMD, BooleanM, LZW and RLE.

These five pre-evolved estimation modules were then used in GP-zip3 as explained in the next section.

B. Fitness Measurement

The calculation of the fitness is divided into two parts. Each part contributes with equal weight to the total fitness, which is the sum of the two.

The fitness contribution of the splitter tree is measured by estimating the total compression ratio on the training set. This is computed by evaluating the pre-evolved estimation functions (see previous section) for the segments identified by the splitter. The system will label the segments with the algorithm providing the highest estimated compression ratio.

The estimated compression ratios obtained in the previous phase are then used to evaluate the fitness contribution of the splitter tree. (Segments' labels are retained, since they are also used in evaluating the quality of the feature-extraction trees, as we will see shortly.)

More specifically, let $ES_x(S_i)$ be the output of the estimation function for algorithm x, where $x \in \{\mathrm{AC}, \mathrm{PPMD}, \mathrm{LZW}, \mathrm{RLE}, \mathrm{BooleanM}\}$ and let segment i be denoted as S_i. Furthermore, let n be the total number of segments. Then, the fitness of the splitter tree can be expressed as:

$$F_{Splitter} = \frac{1}{2} \times [100 - (\frac{\sum_{i=1}^{n} \max_x(ES_x(S_i))}{\textit{File Size}} \times 100)].$$

Note that this approach is computationally much less expensive than actually compressing each segment in the training set as was required by the fitness function used in GP-zip2.

The second part of an individual's fitness is the classification accuracy provided by the feature-extraction trees. After performing the clustering using K-means, the quality of the clustering is evaluated by measuring clusters *homogeneity* and *separation*.

The homogeneity of the clusters is calculated in the following manner. Since the (estimated) optimal compression algorithm for each segment is already known from the previous step, the clusters are labelled according to the dominant algorithm. The fitness function rates the homogeneity of clusters in terms of the proportion of data points – segments – that are being optimally compressed with the algorithm that labels the cluster.

Using the information that was obtained concerning the segments and their optimal compressions, it is easy to find the total number of segments that should be compressed with a particular compression model. Any deviations from this optimal value, due to clusters containing extra members, should be discouraged. Thus, a penalty term was used within the fitness function in order to penalise extra members in the clusters.

The Davis Bouldin Index (DBI) [1] was used to measure cluster quality. DBI is a measure of the nearness of the clusters members to their centroids and the distance between clusters centroids. A small DBI index indicates well separated and grouped clusters. Therefore, the negation of the DBI index was added to the total feature extraction fitness in order to push the evolution to separate clusters (i.e., minimise the DBI). Consequently, the DBI here is treated as a penalty value; the lower the DBI the lower penalty applied to the fitness.

More formally, the clusters' homogeneity can be expressed as follows. Let H be a function that calculates the homogeneity of a cluster and C_i be the i^{th} cluster. Furthermore, let K be the total number of clusters and λ the penalty term. Then,

$$F_{\text{Homogeneity}} = \frac{\sum_{i=1}^{K} H(C_i) - \lambda}{K}.$$

Thus, the fitness of the feature-extraction trees is

$$F_{Feature-extration} = \frac{1}{2} \times (F_{\text{Homogeneity}} - DBI)$$

and the total fitness can be expressed as:

$$Fitness = F_{Feature-extration} + F_{Splitter}.$$

C. Search Operators

GP-zip3 uses the same search crossover and mutation operators as GP-zip2. If the i^{th} individual of the population is denoted as I_i and T_c^i is the c^{th} tree of individual i, where $c \in \{splitter, feature{-}extractor_x, feature{-}extractor_y\}$, the system selects an operator with a predefined probability for each T_c^i. In crossover, a restriction is applied so that splitter trees can only be crossed over with splitter trees. However, the system is able to freely crossover feature-extractions trees at any position.

V. Experiments

Experiments were carried out on various archives that contain both homogeneous and heterogeneous sets of data. The main aim of the experiments was to investigate the performance and speed of GP-zip3 in comparison with its predecessors. Some of the archives that were used for testing were composed of data types that are similar to those used in the training set, while others contained different data types, to which the algorithm had not been exposed during training.

Table II illustrates the contents of the training set. Files were chosen so as to be representative. In other words, they are likely to be compressed by users, and their sizes are within the normal range (they are neither not too big nor too small). Our training archive contained 12 different data types for a total size of 332KB. To ease the comparison of GP-zip3 against other members of the GP-zip family, the same 12 large archives used to test previous methods in [8], [9], [11] were used for testing. The files within the archives are grouped in such a way to ensure each archive contains a unique combination of file types that did not exist in the training set. Table IV reports the contents and the size of each archive.

GP-zip3 was compared to the standard compression algorithms listed previously, as well as GP-zip, GP-zip* and GP-zip2. In addition, Winzip and WinRar – two of the most popular compression algorithms in regular use – were included in the comparison.

TABLE II
Training files for GP-zip3

File types	*Total size of training set*
pdf, exe, CPP code, gif, jpg, xls, ppt, mp3, mp4, txt,xml	332 KB

TABLE III
Parameter settings for our GP runs.

Parameter	*Value*
independent runs	16
population size	100
maximum number of generations	30
crossover probability	50%
mutation probability	50%
selection type	tournament selection (size 5)
termination criterion	max number of generations

The experiments presented here were performed using the parameter settings in Table III. Note that it is practically impossible to determine the best fitness level that could be achieved with our training set. So, there is no special terminating condition for GP-zip3. Simply, the system runs until the maximum number of generations is reached.

Since GP search is stochastic, GP-zip3 performance has been measured through 16 different runs, each of which trains the system and uses the output of the training to compress the 12 different test archives. The aim is to obtain a reasonably general compression algorithm that performs well, on average, for all test files.

Table V summarises the 16 GP runs. The first row reports the average compression ratio in all runs, archive by archive. The second row illustrates the standard deviation of the achieved compression ratios in all runs for each file, to show the system's robustness when dealing with different archives. The third row provides the results of the best evolved compression program. By way of comparison, the fourth row reports the results of the worst evolved compression program, in order to show the performance of GP-zip3 in its worst cases. The last two rows report the best and worst achieved compression ratio respectively for each file in any run.

The 16 programs evolved by GP-zip3 were reviewed and the best program for each test archive was selected in order to compare it against the other compression algorithms.

Table VI reports the best achieved compression ratio for each file and compression system.[2] In all cases GP-zip3 produced extremely competitive compression results, outperforming traditional compression algorithms with considerable margin in most of the cases. Nonetheless, it is clear that GP-zip2 is slightly superior to GP-zip3. This is no surprise; the reason is that GP-zip2 is getting accurate information during the training phase while GP-zip3 has to work with a noisy fitness evaluation. Actually, it is remarkable that GP-zip3 managed to stay this close to GP-zip2.

[2]Reporting the best performance achieved by programs evolved in different runs may seem unfair. However, this level of performance is achievable, albeit with some computational cost: all one needs to do is to compress a file with each of the 16 evolved programs and pick the shortest compressed version.

TABLE IV

TEST FILES FOR GP-ZIP3

Archive	Files	Size (KB)
Text	English translation of The Three Musketeers by Alexandre Dumas, Anne of Green Gables by Lucy Maud Montgomery, 1995 CIA World Fact Book	4,822
Exe	DOW Chemical Analysis program,Windows95/98NetscapeNavigat,Linux 2.x, PINE e-mail program	4,824
Archive1	Mp3 Music, Excel sheet, Certificate card replacement form PDF `http://www.padi.com/`, Anne of Green Gables by Lucy Maud Montgomery (text file)	1,474
Archive2	PowerPoint slides, JPEG file, C++ source code, Mp4 Video (5 seconds)	2,458
Archive3	GIF file ,Unicode text file (Arabic language), GP-zip* executable file, Xml file	1,384
Archive4*	GP-symbolic regression system, MS Access database file, Text file.	34,069
Archive5*	JPG (picture of faces), Word file (rsum), Tif (Fax cover), WMV movie 6:25 minutes	19,309
Archive6	Windows95/98 Application, JPG picture of sea and sky, Text book, Tif picture lena, Resume.xml	2,518
Archive7	Exe application (file splitter program), JPG picture, Text file (book), XML database	694
Archive8*	Java code (Tiny_GP), Mov (high definition file movie), PS file (Journal paper)	56,883
Archive9*	PDF file, Docx Word 2007, FLV video 3:09 minutes	2,793
Canterbury corpus*	English text, fax image, C code, Excel sheet, Technical writing, SPARC exe, English poetry, HTML, lisp code, GUN Manual Page, play text.	2,276

* Contains data types to which the algorithm has not been exposed during training.

TABLE V

SUMMARY OF RESULTS IN 16 INDEPENDENT GP RUNS

File	Archive1	Archive2	Archive3	Archive4	Archive5	Archive6	Archive7	Archive8	Archive9	Canterbury	Text	Exe
Compression Avg.	40.93	30.84	60.79	78.76	34.32	46.48	55.86	36.4	31.03	70.01	59.8	49.79
Std	7.27	19.19	9.39	17.11	18.25	5.48	17.51	14.45	19.27	8.54	22.02	6.98
Best Run	58.34	47.67	61	90.12	51.12	51.12	71.24	50.11	47.93	80.61	80.3	52.89
Worst Run	32.53	3.12	41.93	50.29	7.44	48.16	70.76	18.42	2.17	67.64	79.83	59.91
Best Compression in all runs	58.34	53.86	71.34	90.87	55.12	55.61	72.48	52.56	53.63	81.17	80.3	62.11
Worst Compression in all runs	28.29	3.12	41.93	45.8	6.88	32.4	32	18.39	2.17	52.72	26.88	37.68

GP-zip3 required only two and a half hours on average to perform the learning process, while GP-zip2 required 13 hours, which corresponds to a fivefold speedup. Both GP-zip2 and GP-zip3 take between 12 and 15 minutes to perform the actual compression on a modest 2.21GHz AMD PC system.

The GP-zip family (including GP-zip3) has been slightly outperformed by WinRar-Best on the Canterbury corpus. This dataset is often used as a reference for comparison of compression algorithms, and hence it might seem disappointing to see the GP-zips come second in this particular comparison. However, because this corpus is used so frequently, parameters in highly optimised compression software, such as WinRar, are often tuned to maximise compression on such a dataset, with potentially deleterious consequences (more on this below) for other data types. Indeed, the high compressibility of the dataset indicates that, despite it being heterogeneous, effectively the entropy of the binary data it contains may be atypically low (making it similar to a text archive). Consequently, it is felt that the overall performance of GP-zips, in realistic situations, is better represented by its behaviour on our collection of archives rather than simply the Canterbury corpus.

The estimation approach used to accelerate the evaluation of the splitter trees acted as noise (i.e., produced errors in the estimated values) on the training set. While in small amounts, noise can improve a learning process by increasing the algorithm's generalisation, beyond a certain level noise may result in deleterious consequences on the learning process and limit the algorithm's abilities to generalise its knowledge beyond the training set. In order to compare the generalisation performance of GP-zip3 against its direct predecessor, the average of averages of the achieved compression in all runs for file types in and outside the training set for both GP-

TABLE VI

PERFORMANCE COMPARISON (NEGATIVE VALUES MEAN THAT THE COMPRESSED VERSION OF A FILE WAS LARGER THAN THE ORIGINAL).

File	WinZip-bzip2	WinRar-Best	PPMD	BooleanM	LZW	RLE	AC	GP- zip	GP-zip*	GP-zip2	GP-zip3
Archive1	32.93	34.036	33.32	3.77	1.13	-10.2	9.98	49.49	**63.43**	61.12	58.34
Archive2	3.9	3.19	3.9	2.81	-43.98	-11.48	0.69	N/T	**58.7**	54.19	53.86
Archive3	64.49	65.99	64.36	23.28	43.62	9.16	27.62	N/T	**75.05**	74.79	71.34
Archive4	90.96	**93.13**	90.73	49.21	-24.74	-372.81	58.24	N/T	N/T	90.96	90.87
Archive5	7.97	11.17	8.64	8.64	-36.17	-5401.27	2.72	N/T	N/T	**55.39**	55.12
Archive6	50.61	56.32	49.07	4.99	-3.65	-5002.04	11.88	N/T	N/T	**71.05**	55.61
Archive7	68.64	64.21	70.76	11.34	33.69	-4377.29	33.98	N/T	N/T	**73.96**	72.48
Archive8	15.67	14.41	18.74	5.77	-47.74	-7003.42.	5.39	N/T	N/T	**52.82**	52.63
Archive9	2.94	3.3	2.28	2.877	-41.54	-6333.82	0.3	N/T	N/T	**53.66**	53.63
Canterbury	80.48	**85.14**	81.19	38.01	15.61	6.549	41.4	N/T	81.58	81.18	81.17
Exe	57.86	**64.68**	61.84	11.42	35.75	-4.66	17.46	61.82	62.02	61.85	62.11
Text	77.88	**81.42**	79.95	24.24	56.47	-11.33	37.77	79.95	80.23	80.29	80.3

N/T: not tested

Bold numbers are the highest

TABLE VII

PERFORMANCE COMPARISON: TRAINED VS. UN-TRAINED FILES TYPES

Algorithm	*Average compression for trained files types in all runs*	*Average compression for untrained files types in all runs*
Gp-zip2	47.78%	48.16%
Gp-zip3	50.10%	49.21%

zip2 and GP-zip3 were measured. The untrained files' are the archives that consist of data types to which the algorithm had not been exposed during training. Those files in particular are: *Archive4, Archive5, Archive8, Archive9 and Canterbury*. As illustrated in Table VII, GP-zip3 has a slightly better average than its predecessor. This indicates the algorithm stability with the provided test archives. However, GP-zip3 may perform less well when dealing with archives that are completely different from the training set. This aspect will be further investigated in future research.

The experimental results of 16 runs for GP-zip2 and GP-zip3 were compared. Figure 2 shows them as histograms. The first column reports the average of averages of the compression ratio for all test files in all runs. The second and the third columns show the average compression for all test files for the best and worst evolved program, respectively. The last column reports the standard deviation for all runs. It is clear from the figure that GP-zip3 has more stability and a higher average in most cases.

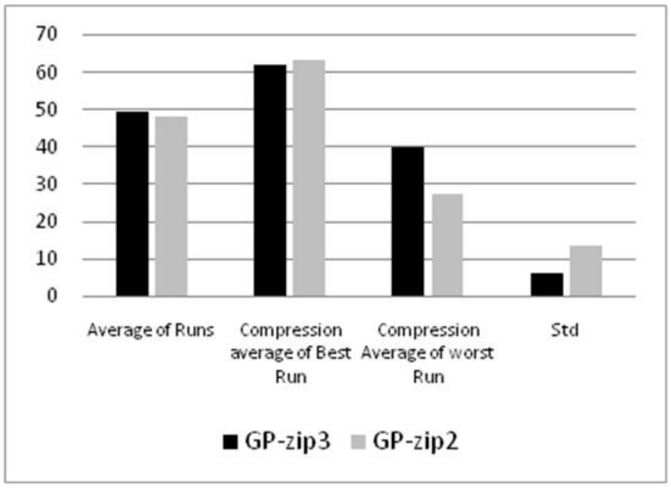

Fig. 2. GP-zip2 vs. GP-zip3 generalisation comparison

VI. CONCLUSIONS

In this paper an improvement to GP-zip2 was proposed; the new improvement is referred to as GP-zip3. In GP-zip3, the aim was to overcome the problem of the extended training time required by GP-zip2. To achieve this we eliminated the need for repetitive compression of the training set with multiple compression algorithms in the fitness evaluation. We proposed to use evolved estimation functions to predict the compression ratio achieved on a segment of data with different compression algorithms without applying the actual compression algorithms. It was found that doing so reduced the training time significantly and improved the system's stability.

Experimentation with GP-zip3 demonstrated that despite the much shorter learning time in comparison to GP-zip2, GP-zip3 achieves almost the same level of performance in terms of compression ratios.

The disadvantage of GP-zip3 is that it depends on evolved estimation functions. Thus, the user must spend extra time evolving accurate estimation functions. Also, the user has to evolve an estimation function for each of the compression algorithms available to GP-zip3. This, however, needs to be done only once. When estimation functions are available, GP-zip3 can efficiently be applied to evolve general compression algorithms as well as compression algorithms tailored to specific domains of application.

This research could be extended in many different ways. In the future the use of a form of hyper-GP-zip will be investigated. This would choose the best program from a pool of solutions for each unseen archive, based on statistical features from the data.

Clustering data fragments using more sophisticated classification techniques is likely to provide further improvements to the systems performance. It would also be beneficial to explore whether the use of additional primitives such as the geometric mean, the mode, the quartiles, etc. could further improve GP-zip3's performance. Treating each component of the fitness function as a separate objective, rather than adding them up, might also provide significant benefits.

Moreover, currently, each compression model the system can use is treated as a black box. A promising extension for this research would be to decompose each model into more elementary entities and allow the system to use compositions of multiple such elements. Such a lower-level language could express both classical and novel compression algorithms.

Another interesting research avenue concerns the idea of storing the state of GP runs in such a way as to make it possible to restart evolution if it becomes apparent that the current best compression model is unsuitable for new data files.

Acknowledgements

The authors would like to thank the anonymous referees for their helpful and constructive comments.

References

[1] J. Bezdek and N. Pal. Some new indexes of cluster validity. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 28(3):301–315, 1998.

[2] J. G. Cleary, W. J. Teahan, and I. H. Witten. Unbounded length contexts for PPM. In *Data Compression Conference*, pages 52–61, 1995.

[3] I. De Falco, A. Iazzetta, E. Tarantino, A. Della Cioppa, and G. Trautteur. A Kolmogorov complexitybased genetic programming tool for string compression. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, pages 10–12.

[4] A. Fukunaga and A. Stechert. Evolving nonlinear predictive models for lossless image compression with genetic programming. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, 1998.

[5] S. W. Golomb. Run-length encodings. *IEEE Trans. Inform. Theory*, IT-12:399–401, 1966.

[6] W. H. Hsu and A. E. Zwarico. Automatic synthesis of compression techniques for heterogeneous files. *Softw. Pract. Exper.*, 25(10):1097–1116, 1995.

[7] A. Kattan. Universal lossless data compression with built in encryption. Master's thesis, School of Computer Science and Electronic Engineering, University of Essex, 2006.

[8] A. Kattan and R. Poli. Evolutionary lossless compression with GP-ZIP. In J. Wang, editor, *2008 IEEE World Congress on Computational Intelligence*, Hong Kong, 1-6 June 2008. IEEE Computational Intelligence Society, IEEE Press.

[9] A. Kattan and R. Poli. Evolutionary lossless compression with GP-ZIP*. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, 2008.

[10] A. Kattan and R. Poli. Genetic programming as a predictor of data compression saving. In P. Collet, editor, *Evolution Artificielle, 9th International Conference*, Lecture Notes in Computer Science, pages 13–24, 26-28 Oct. 2009.

[11] A. Kattan and R. Poli. Evolutionary synthesis of lossless compression algorithms with GP-zip2. In *Submitted to GPEM*. Springer, 2010.

[12] J. Koza. *Genetic programming: on the programming of computers by means of natural selection*. The MIT press, 1992.

[13] W. B. Langdon and R. Poli. *Foundations of Genetic Programming*. Springer-Verlag, 2002.

[14] A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978.

[15] P. Nordin and W. Banzhaf. Programmatic compression of images and sound. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 345–350, Stanford University, CA, USA, 28–31 July 1996. MIT Press.

[16] J. Parent and A. Nowe. Evolving compression preprocessors with genetic programming. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*.

[17] R. Poli, W. B. Langdon, and N. F. McPhee. *A Field Guide to Genetic Programming*. Published via `http://lulu.com`, 2008. (With contributions by J. R. Koza).

[18] I. M. Pu. *Fundamental Data Compression*. Butterworth-Heinemann, Newton, MA, USA, 2005.

[19] I. Witten, R. Neal, and J. Cleary. Arithmetic coding for data compression. 1987.

[20] M. J. Zaki and M. Sayed. The use of genetic programming for adaptive text compression. *Int. J. Inf. Coding Theory*, 1(1):88–108, 2009.