

GENE EXPRESSION PROGRAMMING APPLIED TO IMAGE COMPRESSION

Samuel Ashworth

426 S 1000 E Apt 508
Salt Lake City, UT 84102

ABSTRACT

We here describe an image compression algorithm that generates a set of mathematical functions capable of approximately reproducing the input image, but collectively smaller than the raw image data. These functions are created using an evolutionary algorithm, gene expression programming. When used with a 256 X 256 grayscale Lena image as input, the algorithm we developed created a compressed image with a root mean squared error of 16.3, and a compression ratio of about 1.5 : 1. Given the low quality of the compressed image, the minimal degree of compression, and the algorithm's long running time, it is not a practical compression tool. However, we propose several methods that might be used to boost the algorithm's performance.

1. INTRODUCTION

The compression algorithm introduced herein uses gene expression programming (GEP) to evolve functions that represent an image. We will therefore briefly summarize GEP to lay a foundation for discussion of our algorithm. Also, the algorithm we present employs several of the techniques found in JPEG compression, and so following our summary of GEP, we will introduce the JPEG algorithm. After this ground work has been put into place, we will move on to explanation of the compression scheme we developed.

1.1. Gene Expression Programming

In GEP, each function is represented by a fixed length character string, termed a chromosome, the characters of which represent either an operators (addition, subtraction, etc) and terminals (operands, constants, etc.). The chromosome represents a level-order traversal of an

abstract syntax tree (AST), and when the chromosome is to be evaluated, it is converted into its tree structure.

The chromosome is divided into several parts of equal length called genes. The genes represent separate functions, that is, at the time of the chromosome's expression, each gene is converted into its own AST. To determine the value of the chromosome as a whole, the values of the individual genes are combined in some way, usually additively. Each gene consists of two parts, the head and the tail. The head, which comes first, contains both operators and terminals, and is of arbitrary length. The tail, which follows after the head, contains only terminals, and has length given by $T = H(n - 1) + 1$ where H is the head length and n is the arity of the highest arity operator used in the chromosome. This tail length, coupled with the fact that the chromosome represents a level-order traversal of an expression tree, ensures that every operator has sufficiently many operands.

To effect evolution, an initial population of chromosomes is created randomly. Individuals from this pool are selected, based on fitness, and the selected group is subjected to mutation, crossover, and transposition, each occurring with a specified frequency. Mutation is the random replacement of a character in a chromosome with another character. Crossover is the swapping of substrings between chromosomes. Transposition is the copying of a substring from one place in a given chromosome to another place in the same chromosome. The chromosomes derived from application of these genetic operations are then compared pair-wise with the initial population (the i^{th} individual in the new population is compared with the i^{th} individual of the initial population), and the better in each comparison is put into a new population. This population then becomes the initial population, and the process is repeated, stopping when a specified number of generations has elapsed or a certain maximum fitness reached.

1.2. JPEG Image Compression

JPEG is a prevalent image compression specification that serves as the basis for the first stage of the compression technique presented in this paper. The JPEG specification defines methods for both lossless and lossy compression, but it is the lossy method that is germane to this work.

The first step of the lossy method is to break the image into 8 X 8 blocks, and to compute the discrete cosine transform (DCT) of each block. These 8 X 8 DCT matrices are then each divided element-wise by an 8 X 8 table, termed the quantization matrix, and the resulting values are rounded. This rounding is a lossy operation, and because the divisors in the lower right of the quantization matrix are larger than those of the upper left, more information is lost in the lower right of the DCT matrix than in the upper left. This is done because the majority of the image data perceivable by the eye is contained in the upper left region of the DCT, and thus, space can be saved without loss of quality by eliminating some of the information in the lower right of the matrix. Following quantization, each matrix is converted into a linear array by following a zig-zag pattern from the upper left to the lower right. From each array, all zeros following the last non-zero element are removed, and the remaining values are encoded, generally using Huffman encoding. These encoded arrays, the quantization matrices, and the Huffman tables are the data necessary to reconstruct the image. The complete JPEG specification, from which the above information is taken, can be found in [1].

2. PROPOSED METHODS

The goal of the algorithm presented herein is to compress an image by evolving functions that can be represented more compactly than the image itself, but that are collectively capable of reproducing the image. To simplify the problem, we considered only 256 X 256 grayscale images (one byte per pixel).

The compression algorithm consists of three principle steps. In the first step various operations are performed on the image to make it more amenable to representation by evolved functions. In the second step, functions are generated to represent the preprocessed images. In the final step, the evolved functions are encoded.

Population size	75
Number of genes	4
Chromosome length	21
Selection method	tournament
Fitness function	$\frac{1}{1+RMSE} * 100$
Operator set	+, -, *, /, round, floor
Constant type	integer
Constant interval	[-50, 150]
Prob. 1 pt crossover	1.0
Prob. 2 pt crossover	0.8
Prob. gene recomb.	0.9
Prob. IS transposition	0.9
Prob. RIS transposition	0.9
Prob. mutation	0.025

Table 1. GEP parameters used

2.1. Preprocessing

The first step of the algorithm is to make the image simpler and thus easier to represent. To accomplish this, the operations of JPEG compression prior to Huffman encoding are applied to the image. The mean of each array is then subtracted from each of its elements to make each mean zero. If this were not done, the function evolution process would have to discover a vertical shift constant, and thus, subtracting the mean makes the function evolution process easier.

Note that in some experiments we performed, the preprocessing step was skipped (except for breaking the image into blocks). In these experiments, the evolved functions took two arguments (a row and column number) rather than just one (an index), and produced matrices rather than arrays.

2.2. Function Evolution

The GEP algorithm that we employ is essentially the one described above in section 1.1 and in greater detail in [2], [3], and [4]. It is therefore unnecessary for us to describe the GEP algorithm here, but rather, we will simply list the GEP parameters we used (see Table 1), and explain the four modifications we made to the GEP algorithm.

First, when randomly choosing a character for the head of a chromosome, as in mutation and population initialization, the probability that an operator will be chosen is greater than the probability a terminal will be cho-



Fig. 1. Compressed Lena with preprocessing not used



Fig. 2. Compressed Lena with preprocessing used

sen. This is so because when a terminal occurs in the head, it often results in the rest of the gene going unused. While sometimes this shortening of the function is desirable, it can slow the progress through the search space by making long functions hard to discover.

Second, rather than additively combining the values of the genes to determine the value of the chromosome, we use the last gene's value, and allow each gene to use the values of the genes preceding it. In the tail of the last gene of the chromosome, the probability that a gene reference character will be chosen is higher than the probability of choosing any other terminal. This is because if a gene is not referenced in the final gene, it goes unused. In moderation, unused genes are important contributors to population diversity, as they can accumulate mutations for many generations before suddenly springing into the population upon being referenced [4]. However, unless gene references are favored in the final gene, most chromosomes will never use the values of their non-terminal genes.

Third, to reduce the number of inviable evolved functions (functions that are not defined for some value in the domain of the image data), we define division by zero as division by one. Also, if the application of a given operator would give a result too big or too small for a double precision variable, we refrain from applying the operator. This way, all individuals in our populations are defined on the whole image domain.

2.3. Encoding

In the encoding step, each evolved function and any other values needed to recreate the image it represents are converted into a bit string. If preprocessing has been used, three values precede the function, namely, the first value of the array from the final step of preprocessing, the number of zeros removed in the penultimate step of preprocessing, and the mean from the final step of preprocessing. Respectively, ten, six, and eight bits are allotted to represent these values.

Next in the bit string is the function itself. Since the genome we use contains 13 elements, we allot 4 bits to represent each operator (this presumes that the compressor and de-compressor agree on the genome, and its ordering, in advance). Also, we use 256 different constants, so 8 bits are used to represent each constant. The last gene of the chromosome is written first, followed by any genes that are referenced by the last gene. A scheme identical to this one is employed in experiments where preprocessing is omitted, except that the initial value, number of zeros, and mean are not written.

3. RESULTS

Two classes of experiments were conducted, one in which preprocessing was used, and one in which it was not (except for dividing the image into 8 X 8 blocks). For both experiment classes, a 256 X 256 grayscale Lena image served as the initial input. Images produced by an ex-

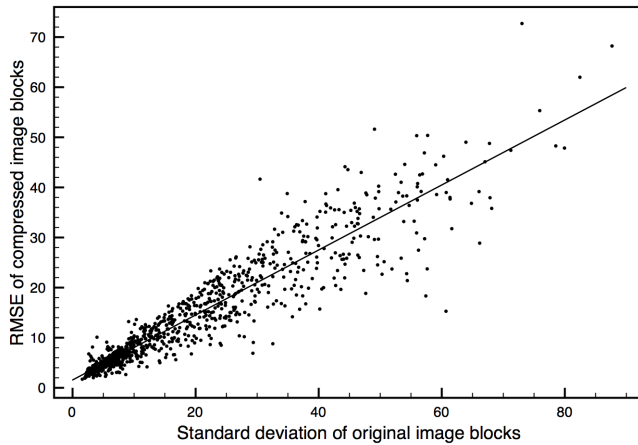


Fig. 3. Plot of the RMSE of the 8 X 8 blocks in Figure 2 versus the standard deviations of the corresponding blocks in the original image

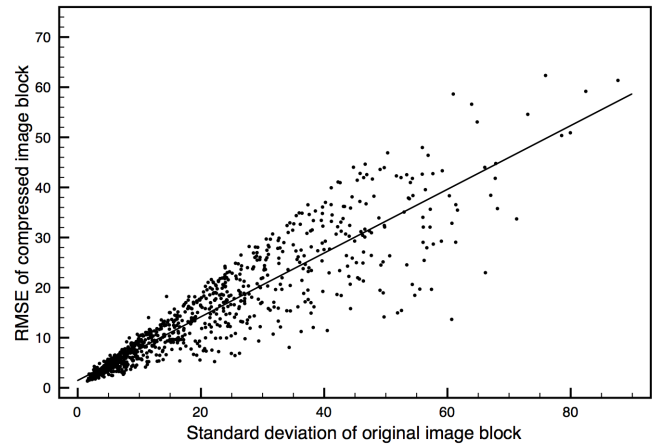


Fig. 4. Plot of the RMSE of the 8 X 8 blocks in Figure 1 versus the standard deviations of the corresponding blocks in the original image

periment from each of the classes can be seen in Figures 1, 2. The root mean squared error (RMSE) of the the images are 16.3 and 16.1 respectively, and the compression ratio for both is approximately 1.5 : 1. In Figure 4 can be seen a plot of the RMSE of each 8 X 8 block of the image in Figure 1 plotted against the standard deviation of the corresponding blocks of the original image. Figure 3 show an analogous plot pertaining to Figure 2.

4. CONCLUSIONS AND FUTURE WORK

The goal of this work was the development of an algorithm that could evolve functions to represent an image. While we were somewhat successful in meeting that goal, it is clear that at present the method herein discussed is not a viable compression technique. Both with and without preprocessing, the RMSE of the compressed image is unacceptably high and the compression is a mere 1.5 : 1. Furthermore, the amount of time required to compress a single image is on the order of eight hours. For a useable GEP based image compression technique to be realized, fidelity must be improved, the size of the block represented by a single function must be increased, and the runtime of the algorithm must be shortened.

In order to improve our algorithm, we must better understand types of image data GEP can and cannot generate functions to represent. In figures 3 and 4, one will

notice that standard deviation in the original image block is positively correlated with RMSE in the corresponding compressed image block. This indicates that there is a positive relationship between the size of the spread of the input data and the difficulty in creating a function to represent the data. However, as standard deviation increases, the degree of correlation between RMSE and standard deviation decreases. Further study should be conducted to ascertain why GEP is able to create functions that closely model some image data with a high standard deviation, yet is unable to do so with others.

In future work, we hope to repeat our experiments with a much larger population size, and to accomplish this, hope to create a distributed architecture for the GEP algorithm. Also, we would like to test new image preprocessing schemes, such as using the discrete wavelet transform and separating the image into bit planes.

5. REFERENCES

- [1] International Telecommunication Union, *Recommendation T.81*, September 1992.
- [2] Cândida Ferreira, "Gene expression programming: A new adaptive algorithm for solving problems," <http://gene-expression-programming.com/webpapers/GEPfirst.pdf>.
- [3] Cândida Ferreira, "Gene expression programming

in problem solving,” <http://www.gene-expression-programming.com/webpapers/GEPTutorial.pdf>, 2001.

- [4] Cândida Ferreira, *Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence*, Springer, 2nd edition, 2006.