*Res. Lett. Inf. Math. Sci.* (2002), **3**, 117-125
Available online at http://www.massey.ac.nz/~wwiims/research/letters/

# Evolution of the Discrete Cosine Transform Using Genetic Programming

Xiang Biao Cui and Martin Johnson
*Massey University*
*M.J.Johnson@massey.ac.nz*

## Abstract

Compression of 2 dimensional data is important for the efficient transmission, storage and manipulation of Images. The most common technique used for lossy image compression relies on fast application of the Discrete Cosine Transform (DCT). The cosine transform has been heavily researched and many efficient methods have been determined and successfully applied in practice; this paper presents a novel method for evolving a DCT algorithm using genetic programming. We show that it is possible to evolve a very close approximation to a 4 point transform. In theory, an 8 point transform could also be evolved using the same technique.

## A Brief Overview of the DCT

The cosine transform translates a set of data points from the spatial domain to the frequency domain using Cosine basis functions[6]. The DCT has found a wide range of application in signal processing, data compression, telecommunication, image processing, feature extraction and filtering. The DCT is a very important translation method in multimedia application. This is because the DCT as an approximation to the Karhunen-Loéve transform for first-order Markov stationary random data. DCT algorithms can be classified into three categories based on their approach. a) indirect computation, b) direct factorization, c) recursive computation. [11]. A two-dimensional DCT can be obtained by first applying a 1-D DCT over the rows followed by a 1-D DCT to the columns of the input data matrix. A great detail research of DCT is introduced in [17].

The expression of a 2-D DCT is based on the following:

$$T(i,j) = c(i,j)\sum_{m=0}^{N-1}\sum_{n=0}^{N-1} S(m,n)\cos\frac{\pi(2m+1)i}{2N}\cos\frac{\pi(2n+1)j}{2N}$$

Where T(i, j) is a real valued output matrix, S(m, n) is the input matrix,
$c(i, 0) = c(0, j) = 1/N$ and $c(i, j) = 2/N$ ($i \neq 0, j \neq 0$)
N is the point size of the DCT.

Many different fast algorithms to compute the DCT have been proposed. The complexity of some well known algorithms for an 8 point 1D DCT are shown below:

| Method | Multiplications | Additions |
|---|---|---|
| W. Chen[5] | 16 | 26 |
| B. G. Lee[14] | 12 | 29 |
| H. S. Hou [11] | 12 | 29 |
| C. Loeffler [15] | 11 | 29 |

**A Short Introduction To Genetic Programming**

Genetic programming as a field was introduced by John Koza in 1992 [12]. Genetic programming is an attempt to answer the question: "can computer solve problems without being explicitly programmed?"[12] GP is based upon the earlier technique of genetic algorithms described by Holland[10][2].

GP runs using a population of possible solutions (individuals). Each individual represents an abstract syntax tree made up with a series of functions and terminals. A fitness value is set up to determine how good an individual is at solving the problem. Since it is possible to calculate an abstract syntax tree for each set of input data, the fitness function can be designed to determine the fitness value based on the result of the abstract syntax tree. Training data is usually a set of input output data pairs. In practice, a distance is used to represent how fit the tree is by comparing the output data of the training set and the result of the abstract syntax tree.

After initializing the population, the GP mechanism will take over and search the problem space. A tree can be created from the function set and terminal set while the fitness function determines how close the tree is to the problem specification. During execution of the GP, new individuals are produced continually and a fitness value for an individual is attached to itself. The commonly used GP operators are reproduction, crossover and mutation.

Reproduction is a method that copies an individual in the population and places it into the new population. Reproduction copies the abstract syntax tree as well as all the information attached to the tree, including the fitness value. The fitness value is used to determine the chance of reproduction. Reproduction results in more useful abstract syntax trees appearing in the population.

The crossover operation takes two parental programs selected based on fitness and generates offspring programs by combining them. The parental programs in genetic programming are typically of different sizes and shapes. The offspring programs are composed of sub-expressions from their parents. Crossover provides a mechanism for exploring the search space by creating a new program that combines advantages of parents with high fitness value[7].

Mutation operates on one program; from this, one or more nodes are selected to be modified. The selected node is replaced by a randomly generated new node[10]. After mutation, the fitness of the program is evaluated and it is placed into the population. The mutation operator helps to avoid local optima in the search space[1][13].

During the run, GP operators operate with given probabilities. Obviously, population members with high fitness values should have a high chance of survival and have a high chance of helping form the next generation than those with a low fitness value. The fitness value will drive a change in population from one generation to next, and like natural selection, only the strong individuals will survive. The average fitness of the next generation should be higher than the previous generation. In such a way, the population can evolve over time[3].

Each individual in the population during the search process represents a point in the search space of all possible programs. Using the fitness measure, the exploration process moves toward the point in space with the largest fitness. When the generation reaches the highest point in the search space, a termination criterion designed to stop the search is satisfied and the genetic programming system is completed.

There are five major steps in applying genetic programming to solve a problem.

- determine the set of terminals.
- determine the set of functions.
- determine the fitness function.
- determine the parameters and variables for controlling the run.
- determine the method of designating a result and the criterion for termination a run.

## 4. Related Work

In Koza [12](10.13), image compression is mentioned and he successfully evolves a model to compress a 30X30 bitmap image using standard GP. Because of the vast computation necessary, only a small image was tested in Koza's work.

In Koza and other standard GP systems, an S-expression is evolved and evaluated. Most of the time is spent on evaluating each S-expression. To increase the speed of evaluating such S-expression, Fukunaga et al [8] have developed a compiler for an efficient GP system. Simple arithmetic operators can be executed with a single instruction at the hardware level instead of many recursive function calls in standard LISP S-expression. In one paper [8], a predictive coder is evolved that can be used for multi-level compression(e.g. a Huffman coder). They worked on a 64X64 image and this proved much faster than the standard GP system.

Nordin and Bnazhaf [16] use the idea of programmatic compression to compress sound and image data. They use a chunking method to process an image that allows the system to converge quickly to a solution. The image was divided into 8X8 or 16X16 pixel blocks. They claim to be the first to first to compress a real full size image(256X256) using GP.

### Implementation

Koza(1989, 1992) describes a genetic programming system using Lisp S-expressions. In his work, he uses a tree–based representation as the genetic algorithm framework. We chose to use C++ for its ease of use and efficiency.

The system must be able to read a data set, which is stored in text format file. The data file contains five columns in which that first four are the training set and the last one is the target. The file is organised in rows where each row represents a training – target pair. The parameters such as population size and program size are predefined in a parameter file. The system must be able to implement genetic operations such as crossover, reproduction and initialisation, in order to search the solution space of the problem domain. At any time, the best solution so far must be recorded so that the best solution can be retrieved even before termination.

Our goal was to evolve a $4 \times 4$ block DCT. This is efficiently implemented in terms of 1 dimensional DCTs using the row – column approach[4]. We follow this approach and try to evolve a 4 point one dimensional DCT. It is possible, theoretically, to evolve a tree in which the entire 4 point DCT is translated. In practice, however, it was found to be almost impossible to evolve such a program. In fact, it is not necessary to perform the entire 4 point DCT using one single translation formula. An easier way is to transform the 4 point DCT one point at a time. This reduces the computational complexity in a parallel way.
Because we are performing the DCT one point in a time, it is obvious that a single tree is a good way of expressing a one point translation. Four trees are required to accomplish the complete 4 point DCT. The nodes are functions that are selected from a function set and the leaves are terminals from a terminal set. The terminals may be input data from a text file or numbers randomly generated by the system.

The system reads training data from a text file in which the 32 set training–target data pairs are located. Each row represents a set of training–target data pairs containing five numbers. The first four numbers are the training set while the last one is the target. Training data is selected randomly from an 8–bit bitmap image. This data is converted to a range of between –128 to + 127. Since we are creating a training data set, translation speed is not an important issue here. Any transform method can be applied to accomplish the DCT. The standard DCT is used for translation in our implementation. That is:

$$T(k) = c(k)\sum_{n=0}^{3} S(n)\cos\frac{\pi(2n+1)k}{8}$$

where $c(0) = \frac{1}{2}$, and $c(k) = \sqrt{\frac{1}{2}}$, $k \neq 0$.

As the four points are translated separately, we have four files after translation. Each file records one point of the data set.
During run time, improving solutions are found continuously. Each time a better individual has been evolved, the system displays its fitness value and the distance between the current value and the target value for each fitness case.

The reason of why we chose a 4 point DCT instead of an 8 point DCT is because of the computational complexity of the genetic programming system. For a DCT, the translation difficulty varies for each point. The first and the fourth point are easier than the others. Genetic programming solves problems by very widespread search in the problem space to find a best solution. The sufficiency requirement must be satisfied when we supply the system with a function set and terminal set. For an 8 point DCT, we tried to supply the terminal set with 8 input data points and 8 random numbers. The terminal set contains 16 elements in total. Because of hardware limitations we found that this problem was just too complex to be solved in a reasonable length of time.

Our fitness function is based on the square of the difference between the target and the actual value .

$$\text{Error} = (C - D)^2$$

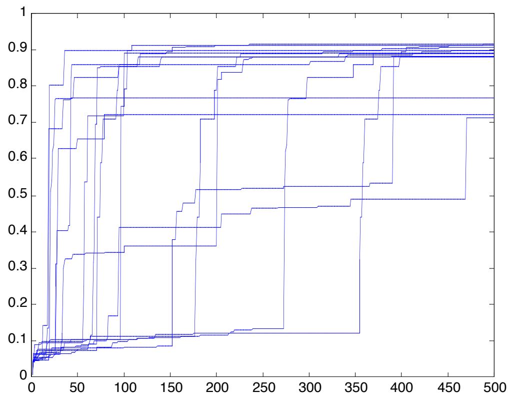
C is the current evaluated value. D is the target value [9].

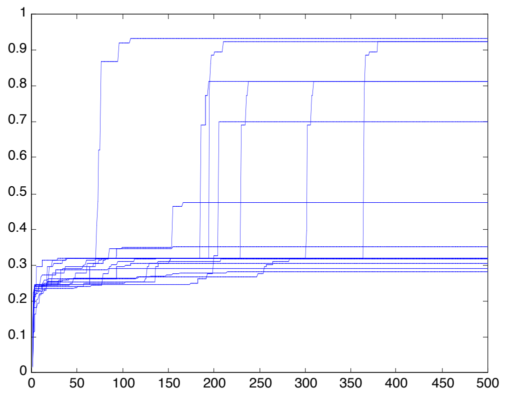
**Results**

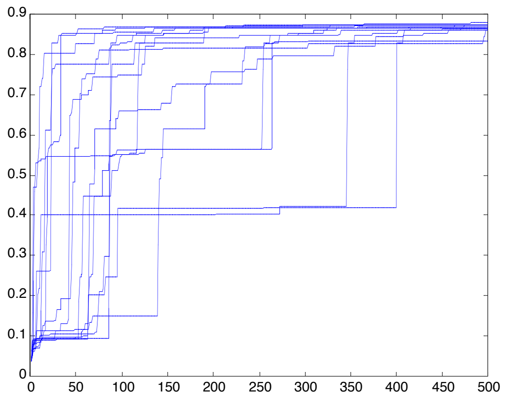
All the runs are based on tournament selection with tournament size of four and the following parameters.

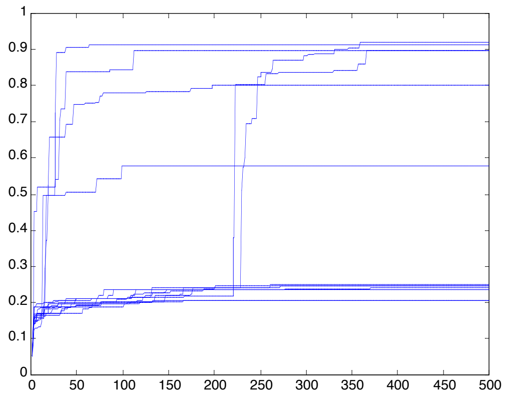
Creation probability: 0.02
Crossover probability: 0.7
Mutation probability: 0.02
Population size: 500
Maximum program size: 50 (for point 0 and point 2)
Maximum program size: 80 (for point 1 and point 3)

The results are summarised as follows. The figures show 15 runs for each point in which point 0 and point 2 are for one million generations


Result for point 0 over 15 trials


Result for point 1 over 15 trials


Result for point 2 over 15 trials


Result for point 3 over 15 trials

Point 1 and point 3 are for five million generations. The run times are between 1 and 6 hours for each run. Point 0 and point 2 take less time than point 1 and point 3 to converge to optimum solutions. This is because of the difference in complexity of the problems and the total number of the elements in the function set and terminal set.
The table below lists the computed data using the best solution and the target data.

| Computed Point 0 | Target Point 0 | Computed Point 1 | Target Point 1 | Computed Point 2 | Target Point 2 | Computed Point 3 | Target Point 3 |
|---|---|---|---|---|---|---|---|
| 64 | 64 | 12 | 12 | -1 | -1 | 5 | 5 |
| 65 | 64 | 1 | 1 | -11 | -11 | -2 | -2 |
| 62 | 62 | 12 | 12 | 3 | 3 | 10 | 10 |
| 66 | 65 | 1 | 1 | 0 | 0 | -14 | -15 |
| 70 | 69 | 10 | 10 | 10 | 10 | -11 | -11 |
| 56 | 55 | 11 | 11 | 7 | 7 | -8 | -8 |
| 70 | 69 | 7 | 7 | 0 | 0 | -9 | -9 |
| 41 | 41 | 19 | 19 | -7 | -6 | 1 | 0 |
| 61 | 61 | 6 | 6 | -1 | 0 | 0 | 0 |
| 39 | 39 | 9 | 9 | -18 | -19 | -1 | -1 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 64 | 64 | 7 | 7 | -1 | -1 | 3 | 3 |
| 45 | 45 | 8 | 7 | -9 | -9 | 1 | 1 |
| 65 | 64 | 8 | 8 | -4 | -4 | -5 | -4 |
| 31 | 31 | 19 | 19 | 6 | 6 | 8 | 8 |
| 55 | 55 | 4 | 4 | -6 | -6 | -1 | -1 |
| 12 | 12 | 7 | 7 | 0 | 0 | 6 | 6 |
| 172 | 172 | 9 | 9 | -4 | -3 | 2 | 2 |
| 130 | 130 | 18 | 18 | -11 | -11 | 0 | 0 |
| 22 | 22 | -17 | -17 | 6 | 6 | 2 | 3 |
| 47 | 47 | -12 | -12 | 2 | 1 | 5 | 5 |
| 3 | 3 | 12 | 12 | -3 | -3 | 0 | 0 |
| -5 | -5 | 2 | 2 | 2 | 2 | -5 | -4 |
| 21 | 21 | -1 | 0 | 1 | 0 | 1 | 1 |
| 7 | 6 | 12 | 12 | -1 | -1 | 2 | 2 |
| 0 | 0 | -5 | -4 | -12 | -13 | 5 | 5 |
| -4 | -4 | 4 | 4 | 6 | 6 | -2 | -1 |
| 4 | 4 | 8 | 8 | 5 | 5 | -5 | -4 |
| 8 | 8 | 13 | 13 | -9 | -8 | -8 | -8 |
| -6 | -6 | 6 | 6 | 1 | 1 | -4 | -4 |
| -8 | -8 | 7 | 7 | 4 | 3 | -4 | -4 |
| 0 | 0 | -5 | -5 | -4 | -4 | 2 | 1 |
| -200 | -199 | 25 | 25 | 0 | 0 | -4 | -4 |

Parts of the test data are randomly selected from an 8-bit black and white bitmap picture while some are the result of the DCT. The raw evolved trees are as follows:

Point 0:

((0.504865)*(((b)-(a))-(c)))-((a)-((d)*(0.504865))+(0.504865)-(div(a,a)))+div(div(div((0.504865)-(d),c),(b)-(a)),(d)-(0.504865))+c+a+a

Point 1:

(0.638957)-((0.638957)*(((0.372204)-((((b)-(c))*((0.638957)*(0.638957)))-(d)))-(a)))

Point 2:

((a)-(b)+(d+0.449478)-(div(b,div(a,0.449478))+c)+div(d,div((b)-((c)*((d)-(c))+(div(a,c))-((b)*(d))),(b)-(0.449478))))*(0.449478)

Point 3:

((div(div(((0.389072)*((0.134196)*((div(0.427715,b))*((0.427715)-(0.389072)))))-(((d+c)-(0.134196))*(d)),d),((b)-((a)-(a)))-(b)))*((0.604979)*(((0.134196+(a)-(d))*(0.427715))-(((b)-(0.604979+(c)-((0.389072)-(0.134196))))-(0.134196)))))-((0.134196)*((0.604979)*(0.427715)))

We simplify the raw programs by hand since they contain redundant code. The simplified programs are as follows:

Point 0:

0.504865*(b-a-c+d)+0.495135+div(div(div(0.504865-d,c),b-a),d-0.504865)+c+a

Point 1:

0.638957-(0.638957*((0.372204-((b-c)*0.408266-d))-a))

Point 2:

(a-b+d+0.449478-(div(b,div(a,0.449478))+c)+div(d,div(b-(c*(d-c)+ (div(a,c))-b*d),b-0.449478)))*0.449478

Point 3:

0.604979*(((0.134196+a-d)*0.427715)-(b-0.484299-c))-0.034724

The fitness of the four programs computed by using the 32 set test data is listed in the following table:

| Point | Fitness |
|---|---|
| Point 0 | 0.9141 |
| Point 1 | 0.9329 |
| Point 2 | 0.8778 |
| Point 3 | 0.9204 |

Note that the total error is the sum of the error squared, which magnifies the actual error value. For point 1, the fitness of the evolved program is 0.9329, this reflects a difference of only 3 between target data and computed data for all the test data. This result is very close to the ideal target.

For point 0, in one run, an interesting result is evolved. It has the form:

(c+b+d+a) * 0.50086

This solution has the fitness value of 0.9022.
For point 2, in one run, a simple solution is also evolved:

0.456939*(0.456939-c-b+d+a)

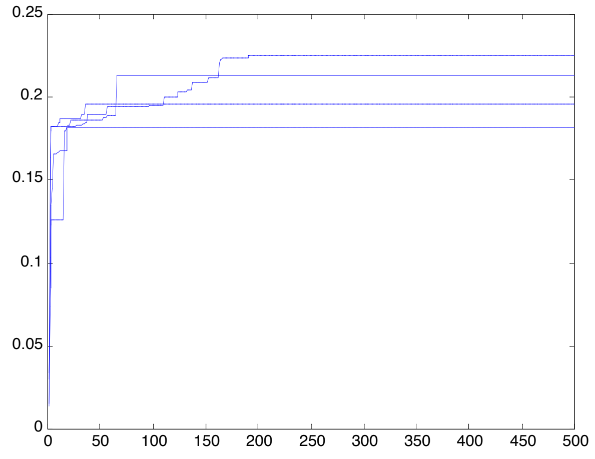
This solution has the fitness value of 0.8635.
For point 1, although we supply 4 random numbers in the terminal set, the best result contains three random numbers only. This result shows that genetic programming converges to a solution that is problem oriented instead of a combination of the elements in function set and terminal set.

In order to compare the evolutionary algorithm and a random search algorithm, we evaluated randomly created individuals for each point. The results are listed as follows. Five million individuals are evaluated for each point. The fitness is between 0 and 1 and is divided into 20 ranges. The first range contains individuals that have the fitness value:
$0<= \text{fitness}<0.05$

| Range | Point 0 | Point 1 | Point 2 | Point3 |
|---|---|---|---|---|
| 0 | 4,999,947 | 4,998,790 | 4,999,137 | 4,935,424 |
| 0.05 | 52 | 1122 | 858 | 64512 |
| 0.1 | 0 | 44 | 3 | 60 |
| 0.15 | 0 | 28 | 1 | 4 |
| 0.2 | 0 | 16 | 1 | 0 |
| 0.25 | 0 | 0 | 0 | 0 |
| 0.3 | 0 | 0 | 0 | 0 |
| 0.35 | 0 | 0 | 0 | 0 |
| 0.4 | 0 | 0 | 0 | 0 |
| 0.45 | 0 | 0 | 0 | 0 |
| 0.5 | 0 | 0 | 0 | 0 |
| 0.55 | 0 | 0 | 0 | 0 |
| 0.6 | 0 | 0 | 0 | 0 |
| 0.65 | 0 | 0 | 0 | 0 |
| 0.7 | 1 | 0 | 0 | 0 |
| 0.75 | 0 | 0 | 0 | 0 |
| 0.8 | 0 | 0 | 0 | 0 |
| 0.85 | 0 | 0 | 0 | 0 |
| 0.9 | 0 | 0 | 0 | 0 |
| 0.95 | 0 | 0 | 0 | 0 |

The results of the random search shows that most of the individuals have a very low fitness value. Surprisingly, one individual for point 0 has a high fitness value of 0.7, which may reflect the fact that point 0 evolution is the easiest problem of all. From the distribution, we can see that random search algorithm is so weak that it is almost impossible to find a good solution.

To investigate the possibility of evolving an 8 point transform, we tried to evolve point 1. Parameter setting is the same as for the four point DCT except the maximum program size is set to 200 and the terminal set contains 8 random numbers. Four trials were tested in our experiment, where each trial ran for five million generations. For point 1, the evolved best individual has the fitness value of 0.225. The time spent on each run is about 10 hours.



8 Point Result for point 1 over 4 trials

**Summary**

The results have shown that the DCT can be evolved by means of the genetic programming paradigm. The best result uses 15 multiplications and 32 additions to carry out a 4 point DCT. The purpose of this investigation was not to evolve a fast DCT but to explore the possibility of evolving a DCT by means of genetic programming. Evolution of a fast DCT and an 8–point DCT are suggested as future work.

**Conclusion**

Research into the DCT continues to be an important topic in transform coding and signal processing. All current algorithms are based on an exact ruled based approach. In this paper, we have successfully evolved an approximate evolutionary algorithm based DCT .Our results use four separate trees for four points. By examining the results, the evolved tree for point 1 and point 3 have higher fitness than that point 0 and 2. This may reflect the different parameter settings between point 0, point 2 and point 1, point 3. There are four random numbers in the terminal set of point 1 and point 3 while only one random number in the terminal set of point 0 and point 2. More random numbers in the terminal set may make for easier fine-tuning of the solution. The successful evolution of the four point DCT implies that an eight point DCT is also possible. This is the most useful size because it is used in many standards such as JPEG and MPEG.

The run time of the genetic programming system for the four point DCT varies. Point 0 and point 2 took one to two hours while point 1 and point 3 took about 6 hours..

The four point DCT used in this project is not used in practice. Future work may include an evolutionary algorithm for a fast 8 point DCT that is more useful.

The most important issue in implementation of the DCT is its efficiency. This is the reason why research is still being carried out on the DCT. Genetic programming provides a new approach for this field – not limited by rule based methods, but using an evolutionary algorithm. Theoretically, the most efficient 8 point DCT could be evolved using the evolutionary approach. To evolve a fast 8 point DCT, computation speed will be a key issue since the terminal set must contain many more elements. Compilation methods and Distributed Genetic Programming could be used to improve the evaluation speed significantly and increase the probability of success.

**References:**

[1]. Angeline, P. J. (1996). Genetic Programming's continued evolution. In *Advances in genetic programming* Vol. 2. Angeline, P. J. & Kinnear, K. (Ed). (pp89-110). MIT Press.

[2]. Angeline, P. J. & Pollack, J. B. (1993). Evolutionary model acquisition. In *Proceedings of the second annual conference on evolutionary programming*. La Jolla, California.

[3]. Bäck, T. (1994). Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. In Michaelewicz, Z. (Ed). Proceedings of the first IEEE world congress on evolutionary computation. IEEE world congress on computational intelligence. Vol. 1. (pp57-62). New York, NY. IEEE Press.

[4]. Bhaskaran, V. & Konstantinides, K. (1997). *Image and Video compression Standards. Algorithms and Architectures. Second Edition.* Kluwer Academic Publishers

[5]. Chen, W. H., Smith, C. H. & Fralick, S. C. (1977). A fast computational algorithm for the discrete cosine transform. IEEE. Trans On Comm. (pp1004-1009).

[6]. Clarke, R. J. (1985). *Transform Coding of Images.* Academic Press.

[7]. D'haeseleer, P. (1994). Context preserving crossover in genetic programming. In Michaelewicz, Z. (Ed). *Proceedings of the first IEEE World congress on evolutionary computation.* IEEE World congress on computation intelligence. Vol. 1. (pp256-261). New York, NY. IEEE Press.

[8]. Fukunaga, A. Stechert, A. & Mutz, D. (1998). A genome compiler for high performance genetic programming. In Koza, J. R., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Gogel, D. B., Garzon, M. H., Goldberg, D. E., Iba, H.& Riolo, R. (Ed). , *Genetic Programming 1998: Proceedings of the Third Annual Conference* , pages 86-94, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann.

[9]. Gathercole, C. & Ross, P. (1997). Tackling the boolean even N parity problem with genetic programming and limited-error fitness. In *Genetic programming*. Proceedings of the second annual conferences. (pp119-127). Stanford University. CA. USA. Morgan Kaufmann.

[10]. Holland, J. H. (1975). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* Ann Arbor: University of Michigan Press.

[11]. Hou, H. S. (1987). A fast recursive algorithm for computing the discrete cosine transform. IEEE. Tran. ASSP. (pp1455-1461).

[12]. Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press.

[13]. Langdon. W. B. (1998). *Genetic programming and data structures : genetic programming data structures = automatic programming.* Boston : Kluwer Academic Publishers.

[14]. Lee, B. G. (1984). A new algorithm to compute the discrete cosine transform. IEEE. Tran. ASSP. (pp1243-1245).

[15]. Loeffler, C., Lightenberg, A. & Moschytz, G. (1989). Practical fast 1-D DCT algorithms with 11 multiplications. Proc. IEEE ICASSP Vol. 2, (pp988-991).

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

[17]. Rao, K. R. & Yip, P. (1990). *Discrete Cosine Transform: Algorithms, Advantages, and Applications.* Academic Press