Evolving Digital Circuits using Multi Expression Programming

Mihai Oltean and Crina Groşan
Department of Computer Science
Babeş-Bolyai University, Kogălniceanu 1
Cluj-Napoca, 3400, Romania
{moltean, cgrosan}@nessie.cs.ubbcluj.ro

Abstract. Multi Expression Programming (MEP) is a Genetic Programming (GP) variant that uses linear chromosomes for solution encoding. A unique MEP feature is its ability of encoding multiple solutions of a problem in a single chromosome. These solutions are handled in the same time complexity as other techniques that encode a single solution in a chromosome. In this paper MEP is used for evolving digital circuits. MEP is compared to Cartesian Genetic Programming (CGP) – a technique widely used for evolving digital circuits – by using several well-known problems in the field of electronic circuit design. Numerical experiments show that MEP outperforms CGP for the considered test problems.

1. Introduction

The problem of evolving digital circuits has been intensely analyzed in the recent past [4, 8, 10, 11, 12, 18]. A considerable effort has been spent on evolving very efficient (regarding the number of gates) digital circuits. J. Miller, one of the pioneers in the field of the evolvable digital circuits, used a special technique called Cartesian Genetic Programming (CGP) [9, 12] for evolving digital circuits. CGP architecture consists of a network of gates (placed in a grid structure) and a set of wires connecting them. For instance this structure has been used for evolving digital circuits for the multiplier problem [12]. The results [12] shown that CGP was able to evolve digital circuits better than those designed by human experts.

In this paper, we use Multi Expression Programming (MEP)[1] [13, 14, 15, 16] for evolving digital circuits. MEP is a Genetic Programming (GP) [6, 7] variant that uses linear chromosomes of fixed length. A unique feature of MEP is its ability of storing multiple solutions of a problem in a single chromosome. Note that this feature does not increase the complexity of the MEP decoding process when compared to other techniques storing a single solution in a chromosome. It has been documented [13, 16], that MEP performs significantly better than other competitor techniques (such as Genetic Programming, Cartesian Genetic Programming, Gene Expression Programming [6] and Grammatical Evolution [17]) for some well-known problems such as symbolic regression and even-parity [6].

In this paper we present the way in which MEP may be efficiently applied for evolving digital circuits. We show the way in which multiple digital circuits may be stored in a single MEP chromosome and the way in which the fitness of this chromosome may be computed by traversing the MEP chromosome only once.

Several numerical experiments are performed with MEP for evolving arithmetic circuits. The results show that MEP significantly outperforms CGP for the considered test problems.

The paper is organized as follows. In section 2, the problem of designing digital circuits is presented. Section 3 briefly describes the Cartesian Genetic Programming technique. The Multi Expression technique is presented in section 4. The way in which digital circuits are encoded in a MEP chromosome is presented in subsection 4.5. Several numerical experiments are performed in section 5.

2. Problem statement

The problem that we are trying to solve in this paper may be briefly stated as follows:

Find a digital circuit that implements a function given by its truth table.

The gates that are usually used in the design of digital circuits along with their description are given in Table 1.

#	Function	#	Function
0	0	10	$a \oplus b$
1	1	11	$a \oplus \overline{b}$
2	a	12	$a + b$
3	b	13	$a + \overline{b}$
4	$\overline{a}$	14	$\overline{a} + b$
5	$\overline{b}$	15	$\overline{a} + \overline{b}$
6	$a \cdot b$	16	$a \cdot \overline{c} + b \cdot c$
7	$a \cdot \overline{b}$	17	$a \cdot \overline{c} + \overline{b} \cdot c$
8	$\overline{a} \cdot b$	18	$\overline{a} \cdot \overline{c} + b \cdot c$
9	$\overline{a} \cdot \overline{b}$	19	$\overline{a} \cdot \overline{c} + \overline{b} \cdot c$

Table 1. Function set (gates) used in numerical experiments. Some functions are independent on the input (functions 0 and 1), other depend on only one of the input variables (functions 2-5), other functions depend on two input variables (functions 6-15) and the other functions depends on three input variables (functions 16-19). These functions are taken from [12].

Symbols used to represent some of the logical gates are given in Figure 1.

[1] MEP source code is available at https://github.com/mepx

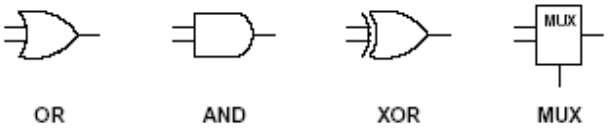

Figure 1. The symbols used to represent some of the logical gates in Table 1 (OR is function 12, AND is function 6, XOR is function 10 and MUX is function 16). In some pictures a small circle may appear on these symbols indicating the negation (inversion) of the respective results.

The MUX gate may be also represented using 2 ANDs and 1 OR. However some modern devices use the MUX gate as an atomic device in that all other gates are synthesized using this one [9].

Gates may also be represented using the symbols given in Table 2.

Gate	Representation
AND	.
OR	+
XOR	⊕
NOT	–

Table 2. Representation of some functions given in Table 1.

3. Cartesian Genetic Programming

Cartesian Genetic Programming (CGP) [9, 12] is a GP technique that encodes chromosomes in graph structures rather than standard GP trees. The motivation for this representation is that the graphs are more general than the tree structures, thus allowing the construction of more complex computer programs.

CGP is Cartesian in the sense that the graph nodes are represented in a Cartesian coordinate system. This representation was chosen due to the node connection mechanism, which is similar to GP mechanism. A CGP node contains a function symbol and pointers towards nodes representing function parameters. Each CGP node has an output that may be used as input for another node.

Each CGP program (graph) is defined by several parameters: number of rows (n_r), number of columns (n_c), number of inputs, number of outputs, and number of functions. The nodes interconnectivity is defined as being the number (l) of previous columns of cells that may have their outputs connected to a node in the current column (the primary inputs are treated as node outputs).

CGP chromosomes are encoded as strings by reading the graph columns top down and printing the input nodes and the function symbol for each node.

An example of CGP program is depicted in Figure 2.

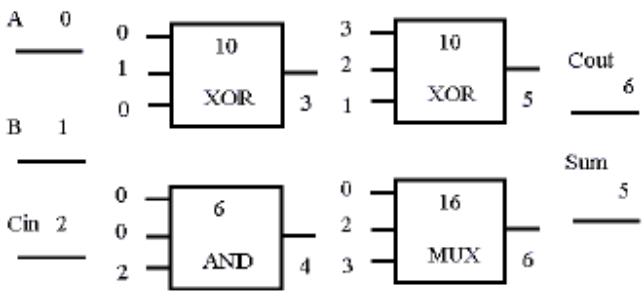

Figure 2. A CGP program for 1-bit adder problem.

In Figure 2, a gate array representation of a one-bit adder is given. *A*, *B*, and *Cin* are the binary inputs. The outputs *Sum* and *Cout* are the binary outputs. *Sum* represents the sum bit of the addition of *A*+*B*+*Cin*, and *Cout* the carry bit. The chromosome representation of the circuit in Figure 2 is the following (function symbols are given in bold):

0 1 0 **10** 0 0 2 **6** 3 2 1 **10** 0 2 3 **16** 6 5.

Evolutionary algorithm used in [12] to evolve digital circuits is a simple form of (1+λ)-ES [2], where λ was set to 4. This algorithm seems to perform very well in conjunction to CGP representation. However, a Genetic Algorithm (GA) [6, 19] may also be used as underlying mechanism for CGP.

4. Multi Expression Programming

In this section, *Multi Expression Programming* is briefly described.

4.1. MEP Algorithm

Standard MEP algorithm uses steady state [19] as its underlying mechanism. MEP algorithm starts by creating a random population of individuals. The following steps are repeated until a stop condition is reached. Two parents are selected using a selection procedure. The parents are recombined in order to obtain two offspring. The offspring are considered for mutation. The best offspring replaces the worst individual in the current population if the offspring is better than the worst individual.

The algorithm returns as its answer the best expression evolved along a fixed number of generations.

4.2. MEP Representation

MEP genes are represented by substrings of a variable length. The number of genes per chromosome is constant. This number defines the length of the chromosome. Each gene encodes a terminal or a function symbol. A gene encoding a function includes pointers towards the function arguments. Function arguments always have indices of lower values than the position of that function in the chromosome.

This representation is similar to the way in which ***C*** and ***Pascal*** compilers translate mathematical expressions into machine code [1].

The proposed representation ensures that no cycle arises while the chromosome is decoded (phenotypically transcripted). According to the proposed representation scheme the first symbol of the chromosome must be a terminal symbol. In this way only syntactically correct programs (MEP individuals) are obtained.

Example

We employ a representation where the numbers on the left positions stand for gene labels. Labels do not belong to the chromosome, they being provided only for explanation purposes.

For this example we use the set of functions $F = \{+, *\}$, and the set of terminals $T = \{a, b, c, d\}$. An example of chromosome using the sets F and T is given below:

1: *a*
2: *b*
3: + 1, 2
4: *c*
5: *d*
6: + 4, 5
7: * 3, 6

4.3. Decoding MEP Chromosomes and the Fitness Assignment Process

In this section it is described the way in which MEP individuals are translated into computer programs and the way in which the fitness of these programs is computed.

This translation is achieved by reading the chromosome top-down. A terminal symbol specifies a simple expression. A function symbol specifies a complex expression obtained by connecting the operands specified by the argument positions with the current function symbol.

For instance, genes 1, 2, 4 and 5 in the previous example encode simple expressions formed by a single terminal symbol. These expressions are:

$E_1 = a$,
$E_2 = b$,
$E_4 = c$,
$E_5 = d$,

Gene 3 indicates the operation + on the operands located at positions 1 and 2 of the chromosome. Therefore gene 3 encodes the expression:

$E_3 = a + b$.

Gene 6 indicates the operation + on the operands located at positions 4 and 5. Therefore gene 6 encodes the expression:

$E_6 = c + d$.

Gene 7 indicates the operation * on the operands located at position 3 and 6. Therefore gene 7 encodes the expression:

$E_7 = (a + b) * (c + d)$.

E_7 is the expression encoded by the whole chromosome.

There is neither practical nor theoretical evidence that one of these expressions is better than the others. Moreover, Wolpert and McReady [20, 21] proved that we cannot use the search algorithm's behavior so far for a particular test function to predict its future behavior on that function. This is why each MEP chromosome is allowed to encode a number of expressions equal to the chromosome length. Each of these expressions is considered as being a potential solution of the problem.

The value of these expressions may be computed by reading the chromosome top down. Partial results are computed by dynamic programming [3] and are stored in a conventional manner.

As MEP chromosome encodes more than one problem solution, it is interesting to see how the fitness is assigned.

Usually the chromosome fitness is defined as the fitness of the best expression encoded by that chromosome.

For instance, if we want to solve symbolic regression problems the fitness of each sub-expression E_i may be computed using the formula:

$$f(E_i) = \sum_{k=1}^{n} \left| o_{k,i} - w_k \right|, \qquad (1)$$

where $o_{k,i}$ is the obtained result by the expression E_i for the fitness case k and w_k is the targeted result for the fitness case k. In this case the fitness needs to be minimized.

The fitness of an individual is set to be equal to the lowest fitness of the expressions encoded in chromosome:

$$f(C) = \min_i f(E_i). \qquad (2)$$

When we have to deal with other problems we compute the fitness of each sub-expression encoded in the MEP chromosome and the fitness of the entire individual is given by the fitness of the best expression encoded in that chromosome.

4.4. Search Operators

Search operators used within MEP algorithm are crossover and mutation. Considered search operators preserve the chromosome structure. All offspring are syntactically correct expressions.

Crossover

By crossover two parents are selected and are recombined. For instance, within the uniform recombination the offspring genes are taken randomly from one parent or another.

Example

Let us consider the two parents C_1 and C_2 given below. The two offspring O_1 and O_2 are obtained by uniform recombination as follows:

Parents		*Offspring*	
C_1	C_2	O_1	O_2
1: ***b***	1: *a*	1: *a*	1: ***b***
2: ***** ***1, 1***	2: *b*	2: ***** ***1, 1***	2: *b*
3: **+ 2, 1**	3: + 1, 2	3: **+ 2, 1**	3: + 1, 2
4: ***a***	4: *c*	4: *c*	4: ***a***
5: ***** ***3, 2***	5: *d*	5: ***** ***3, 2***	5: *d*
6: ***a***	6: + 4, 5	6: + 4, 5	6: ***a***
7: **- 1, 4**	7: * 3, 6	7: **- 1, 4**	7: * 3, 6

Mutation

Each symbol (terminal, function of function pointer) in the chromosome may be target of mutation operator. By mutation some symbols in the chromosome are changed. To preserve the consistency of the chromosome its first gene must encode a terminal symbol.

Example

Consider the chromosome *C* given below. If the boldfaced symbols are selected for mutation an offspring *O* is obtained as follows:

C	*O*
1: *a*	1: *a*
2: * 1, 1	2: * 1, 1
3: ***b***	3: **+ 1, 2**
4: * 2, 2	4: * 2, 2
5: *b*	5: *b*
6: + **3**, 5	6: **+ 1**, 5
7: *a*	7: *a*

4.5. MEP for Evolving Digital Circuits

In this section we describe the way in which Multi Expression Programming may be efficiently used for evolving digital circuits.

Each circuit has one or more inputs (denoted by *NI*) and one or more outputs (denoted *NO*). In section 4.4 we present the way in which is the fitness of a chromosome with a single output is computed. When multiple outputs are required for a problem, we have to choose *NO* genes which will provide the desired output (it is obvious that the genes must be distinct unless the outputs are redundant).

In CGP, the genes providing the program's output are evolved just like all other genes. In MEP, the best genes in a chromosome are chosen to provide the program's outputs. When a single value is expected for output we simply choose the best gene (see section 4.3, formulas (1) and (2)). When multiple genes are required as outputs we have to select those genes which minimize the difference between the obtained result and the expected output.

We have to compute first the quality of a gene (sub-expression) for a given output:

$$f(E_i, q) = \sum_{k=1}^{n} \left| o_{k,i} - w_{k,q} \right|, \qquad (3)$$

where $o_{k,i}$ is the obtained result by the expression (gene) E_i for the fitness case k and $w_{k,q}$ is the targeted result for the fitness case k and for the output q. The values $f(E_i, q)$ are stored in a matrix (by using dynamic programming [3]) for latter use (see formula (4)).

Since the fitness needs to be minimized, the quality of a MEP chromosome is computed by using the formula:

$$f(C) = \min_{i_1, i_2, \ldots, i_{NO}} \sum_{q=1}^{q \le NO} f(E_{i_q}, q). \qquad (4)$$

In equation (4) we have to choose numbers i_1, i_2, …, i_{NO} in such way to minimize the program's output. For this we shall use a simple heuristic which does not increase the complexity of the MEP decoding process: for each output q ($1 \le q \le NO$) we choose the gene i that minimize the quantity $f(E_i, q)$. Thus, to an output is assigned the best gene (which has not been assigned before to another output). The selected gene will provide the value of the q^{th} output.

Remark:

i. Formulas (3) and (4) are the generalization of formulas (1) and (2) for the case of multiple outputs of a MEP chromosome.
ii. The complexity of the heuristic used for assigning outputs to genes is $O(NG \cdot NO)$ where *NG* is the number of genes and *NO* is the number of outputs.
iii. We may use another procedure for selecting the genes that will provide the problem's outputs. This procedure selects, at each step, the minimal value in the matrix $f(E_i, q)$ and assign the corresponding gene i to its paired output q. Again, the genes already used will be excluded from the search. This procedure will be repeated until all outputs have been assigned to a gene. However, we did not used this procedure because it has a higher complexity – $O(NO \cdot log_2(NO) \cdot NG)$ – than the previously described procedure which has the complexity $O(NO \cdot NG)$.

5. Numerical Experiments

In this section, several numerical experiments with MEP for evolving digital circuits are performed. For this purpose several well-known test problems [12] are used.

For reducing the chromosome length and for preventing input redundancy we keep all the terminals on the first positions of the MEP chromosomes.

For assessing the performance of the MEP algorithm three statistics are of high interest:

i. the relationship between the success rate and the number of genes in a MEP chromosome,

ii. the relationship between the success rate and the size of the population used by the MEP algorithm.

iii. the computation effort.

The success rate is computed using the equation (5).

$$Success\ rate = \frac{The\ number\ of\ successful\ runs}{The\ total\ number\ of\ runs}. \quad (5)$$

The method used to assess the effectiveness of an algorithm has been suggested by Koza [6]. It consists of calculating the number of chromosomes, which would have to be processed to give a certain probability of success. To calculate this figure one must first calculate the cumulative probability of success $P(M, i)$, where M represents the population size, and i the generation number. The value $R(z)$ represents the number of independent runs required for a probability of success (given by z) at generation i. The quantity $I(M, z, i)$ represents the minimum number of chromosomes which must be processed to give a probability of success z, at generation i. The formulae are given by the equation (6), (7) and (8). $Ns(i)$ represents the number of successful runs at generation i, and N_{total} , represents the total number of runs:

$$P(M,i) = \frac{Ns(i)}{N_{total}}. \quad (6)$$

$$R(z) = ceil\left\{\frac{\log(1-z)}{\log(1-P(M,i)}\right\}. \quad (7)$$

$$I(M,i,z) = M \cdot R(z) \cdot i. \quad (8)$$

Note that when $z = 1.0$ the formulae are invalid (all runs successful). In the tables and graphs of this paper z takes the value 0.99.

In the numerical experiments performed in this paper the number of symbols in a MEP chromosome is usually larger than the number of symbols in a CGP chromosome because in a MEP the problem's inputs are also treated as a normal gene and in a CGP the inputs are treated as being isolated from the main CGP chromosome. Thus, the number of genes in a MEP chromosome is equal to the number of genes in CGP chromosome + the number of problem's inputs.

5.1. Two Bit Multiplier

The two-bit multiplier [12] implements the binary multiplication of two two-bit numbers to produce a possible four-bit number. The training set for this problem consist of 16 fitness cases, each of them having 4 inputs and 4 outputs.

Several experiments for evolving a circuit that implements the two-bit multiplier are performed. In the first experiment we want to compare the computation effort spent by CGP and MEP for solving this problem. Gates 6, 7 and 10 (see Table 1) are used in this experiment.

The parameters of CGP are given in Table 3 and the parameters of MEP are given in Table 4.

Parameter	**Value**
Number of rows	1
Number of columns	10
Levels back	10
Mutation	3 symbols / chromosome
Evolutionary scheme	(1+4) ES

Table 3. Parameters of the CGP algorithm.

Parameter	**Value**
Code length	14 (10 gates + 4 inputs)
Crossover	Uniform
Crossover probability	0.9
Mutation	3 symbols / chromosome
Selection	Binary Tournament

Table 4. Parameters of the MEP algorithm.

One hundred runs of 150000 generations are performed for each population size. Results are given in Table 5.

Population size	**Cartesian Genetic Programming**	**Multi Expression Programming**	Δ
2	148808	53352	178.91
3	115224	111600	3.24
4	81608	54300	50.29
5	126015	59000	113.58
6	100824	68850	46.44
7	100821	39424	155.73
8	96032	44160	117.46
9	108036	70272	53.73
10	108090	28910	273.88
12	115248	25536	351.31
14	117698	26544	343.40
16	120080	21216	465.98
18	145854	17820	718.48
20	120100	21120	468.65
25	180075	23500	666.27
30	162180	19440	734.25
40	216360	16000	1252.25
50	225250	13250	1600.00

Table 5. Computation effort spent for evolving two-bit multipliers for different population sizes. CGP results are taken from [12]. The difference Δ (last column) is shown as a percentage considering the values of MEP as a baseline. Results are averaged over 100 runs.

From Table 5 it can be seen that MEP outperforms CGP for all considered population sizes. The differences range from 3.24% (for 3 individuals in the population) up to 1600% (for 50 individuals in the population). From this experiment we also may infer that large populations are better for MEP than for CGP. The computational effort decrease for MEP as the population size is increased.

We are also interested in computing the relationship between the success rate and the chromosome length and the population size.

The number of genes in each MEP chromosome is set to 20 genes when the relationship between the success rate and the population size is analyzed. When the relationship between the success rate and the population size is analyzed a population consisting of 20 MEP chromosomes is used. Gates 6, 7 and 10 are used in this experiment. Other MEP parameters are given in Table 4.

Results are depicted in Figure 3.

From Figure 3 it can be seen that MEP is able to find a correct digital circuit in many runs. A population consisting of 90 individuals with 20 genes yields a success rate of 100% (see Figure 3(b)) and a population with 20 individuals with 85 genes yields a success rate of 92% (see Figure 3(a)).

From Figure 3(a) we may infer that larger MEP chromosomes are better than the shorter ones. The minimum number of gates for this circuit is 7. This number has been achieved by Miller during his numerical experiments (see [12]). A MEP chromosome implementing Miller's digital circuit has 11 genes (the actual digital circuit + 4 input genes). From Figure 3(a) we can see that, for a MEP chromosome with 11 genes, only 6 correct solutions have been evolved. As the chromosome length increases the number of correct solutions evolved by also increases. If the chromosome has more than 21 genes the success rate never decreases below than 70%.

Even if the chromosome length is larger than the minimum required (11 genes) the evolved solutions usually have no more than 14 genes. This is due to the multi expression ability of MEP which acts like a provider of variable length chromosomes [13]. The length of the obtained circuits could be reduced by adding another feature to our MEP algorithm. This feature has been suggested by C. Coello in [4] and it consists of a multiobjective fitness function. The first objective is to minimize the differences between the expected output and the actual output (see formulas (3) and (4)). The second objective is to minimize the number of gates used by the digital circuit. Note that he first objective is more important than the second one. We also have to modify the algorithm. Instead of stopping the MEP algorithm when an optimal solution (regarding the first objective) is found we continue to run the program until a fixed number of generations have been elapsed. In this way we hope that also the number of gates (the second objective) will be minimized.

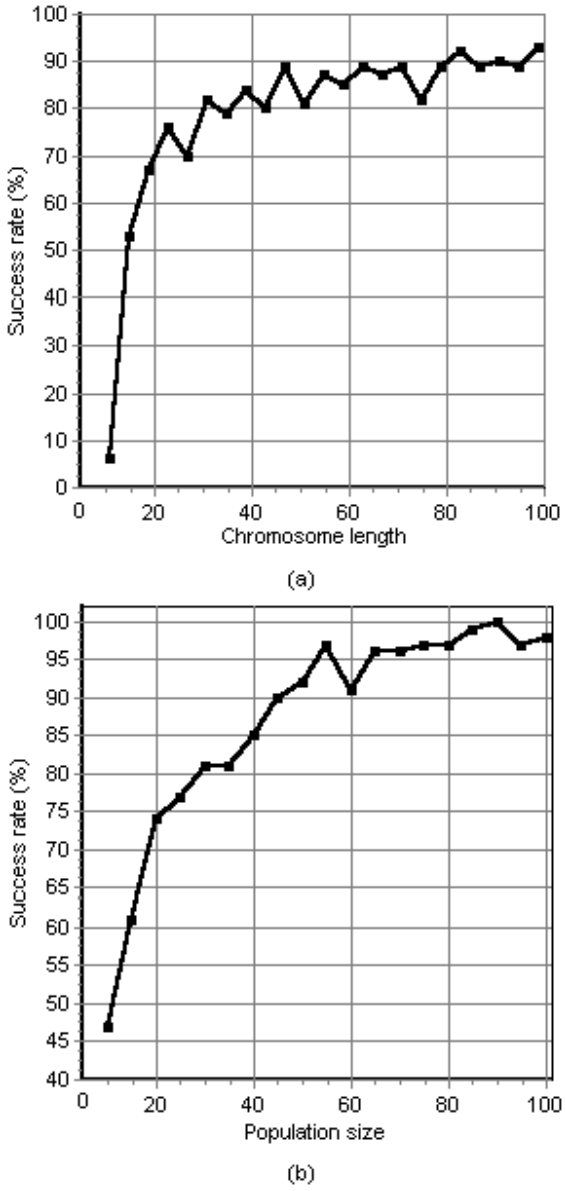

Figure 3. The relationship between the success rate of the MEP algorithm and (a) number of genes in a chromosome, (b) the number of individuals in population. Results are averaged over 100 runs.

5.2. Two Bit Adder with Carry

A more complex situation is the *Two Bit Adder with Carry* problem [12]. The circuit implementing this problem adds 5 bits (two numbers represented using 2 bits each and a carry bit) and gives a three-bit number representing the output.

The training set consists of 32 fitness cases with 5 inputs and 3 outputs.

The relationship between the success rate and the chromosome length and the population size is analyzed for this problem.

When the relationship between the success rate and the population size is analyzed the number of genes in each

MEP chromosome is set to 20 genes. When the relationship between the success rate and the population size is analyzed a population consisting of 20 MEP chromosomes is used. Gates 10 and 16 (see Table 1) are used in this experiment (as indicated in [12]). Other MEP parameters are given in Table 4. Results are depicted in Figure 4.

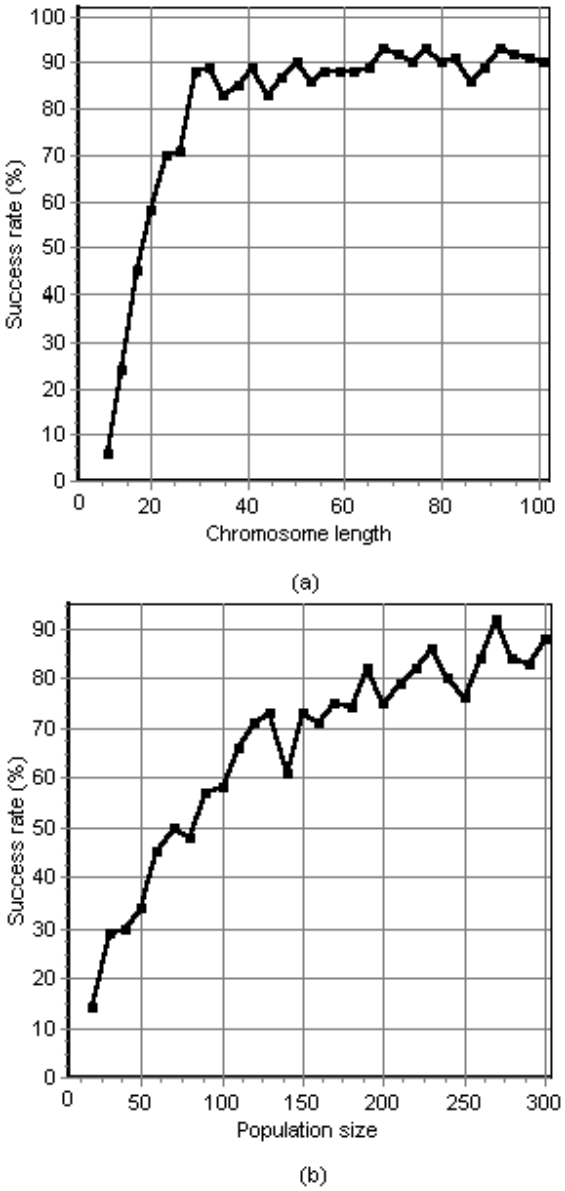

Figure 4. The relationship between the success rate of the MEP algorithm and (a) number of genes in a chromosome, (b) the number of individuals in population. Results are averaged over 100 runs.

From Figure 4 it can be seen that MEP is able solve this problem very well. When the number of genes in a MEP chromosome is larger than 30 in more than 80 cases (out of 100) MEP was able to find a perfect solution (see Figure 4(a)). After this value, the success rate does not increase significantly. A population with 270 individuals yields over 90 (out of 100) successful runs (see Figure 4(b)).

This problem is more difficult than the two-bit multiplier even if we used a smaller function set (functions 10 and 16) that the set used for the multiplier (function 6, 7 and 10).

6. Conclusions and further work

In this paper Multi Expression Programming has been used for evolving digital circuits. It has been shown the way in which multiple digital circuits may be encoded in the same chromosome and the way in which MEP chromosomes are read only once for computing their quality. There was no human input about *how* the circuits should be designed, just a measurement of the degree to which a given circuit achieves the desired response.

Several numerical experiments for evolving digital circuits have been performed. The circuits evolved during the numerical experiments are for the *Two-bit Multiplier* and the *Two-bit Adder with Carry* problems. These problems are well-known benchmark instances used for assessing the performance of the algorithms evolving digital circuits.

The results of the numerical experiments show that MEP outperforms CGP on some of the considered test problems. In some cases the MEP is better than CGP with more than one order of magnitude.

Even if MEP and CGP have some things in common there are some decisive aspects that make them different. Some of these aspects are listed in Table 6.

Multi Expression Programming	**Cartesian Genetic Programming**
Encodes multiple solutions of a problem in a single chromosome. The process of decoding a MEP chromosome has the same complexity as the CGP decoding process.	Encodes a single solution of a problem in a chromosome.
The problem's outputs are chosen as the best among the possible outputs.	The problem's outputs are subject to evolution.
Chromosomes are strings of genes. A unique parameter is needed for expressing the chromosome length and shape: the *chromosome length*.	Chromosomes are matrices which are then linearized. However, two parameters are needed for expressing a chromosome: the *number of rows* and the *number of columns*.
Problem's inputs are stored in chromosome.	The problem's inputs are not stored in chromosome.

Table 6. The differences between Multi Expression Programming and Cartesian Genetic Programming.

The differences presented in Table 6 show a significant advantage to the MEP algorithm over the CGP.

Further numerical experiments with Multi Expression Programming will be focused on evolving digital circuits for other interesting problems (such as three-bit and four-bit multipliers).

References

[1] A. Aho, R. Sethi, and J. Ullman, *Compilers: Principles, Techniques, and Tools*, Addison Wesley, 1986.

[2] T. Bäck, F. Hoffmeister and H.P. Schwefel, *A Survey of Evolutionary Strategies*, In Proceedings of the 4th International Conference on Genetic Algorithms, edited by R. Belew and L. Booker, Morgan Kaufmann, San Francisco, CA, pp. 2-9, 1991.

[3] R. Bellman, *Dynamic Programming*, Princeton, Princeton University Press, New Jersey, 1957.

[4] C. Coello, E. Alba, G. Luque and A. Aguire, *Comparing different Serial and Parallel Heuristics to Design Combinational Logic Circuits*, In Proceedings of 2003 NASA/DoD Conference on Evolvable Hardware, J. Lohn, R. Zebulum, J. Steincamp, D. Keymeulen, A. Stoica, M.I. Ferguson, pp 3-12, 2003.

[5] C. Ferreira, *Gene Expression Programming: a New Adaptive Algorithm for Solving Problems*, Complex Systems, Vol. 13, pp. 87-129, 2001.

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

[7] J. R. Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs*, MIT Press, Cambridge, MA, 1994.

[8] J. F. Miller and P. Thomson, *Aspects of Digital Evolution: Evolvability and Architecture*. In Proceedings of the Parallel Problem Solving from Nature V, A. E. Eiben, T. Bäck, M. Schoenauer, and H-P Schwefel (Editors), pp. 927–936, Springer, 1998.

[9] J.F. Miller and P. Thomson, *Cartesian Genetic Programming*. In Proceedings of the 3rd International Conference on Genetic Programming (EuroGP2000), R. Poli, J.F. Miller, W. Banzhaf, W.B. Langdon, J.F. Miller, P. Nordin, T.C. Fogarty (Eds.), LNCS 1802, Springer-Verlag, Berlin, pp. 15-17, 2000.

[10] J. F. Miller, P. Thomson, and T. Fogarty, *Designing Electronic Circuits using Evolutionary Algorithms. Arithmetic Circuits: A Case Study*. In Genetic Algorithms and Evolution Strategies in Engineering and Computer Science, D. Quagliarella, J. Periaux, C. Poloni and G. Winter (Editors), pp. 105–131, Chechester, UK-Wiley, 1997.

[11] J. F. Miller. *An Empirical Study of the Efficiency of Learning Boolean Functions using a Cartesian Genetic Programming Approach*. In Proceedings of the 1st Genetic and Evolutionary Computation Conference, W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, (Editors), Vol. 2, pp. 1135–1142, Morgan Kaufmann, San Francisco, CA, 1999.

[12] J. F. Miller, D. Job and V.K. Vassilev. *Principles in the Evolutionary Design of Digital Circuits - Part I*, Genetic Programming and Evolvable Machines, Vol. 1(1), pp. 7 – 35, Kluwer Academic Publishers, 2000.

[13] M. Oltean and C. Groşan, *A Comparison of Several Linear GP Techniques*, Complex Systems, 2004 (Accepted for publication).

[14] M. Oltean and C. Groşan, *Evolving Evolutionary Algorithms using Multi Expression Programming*, The 7th European Conference on Artificial Life, September 14-17, 2003, Dortmund, Edited by W. Banzhaf (et al), LNAI 2801, pp. 651-658, Springer Berlin, 2003

[15] M. Oltean, *Solving Even-Parity Problems with Multi Expression Programming*, Proceedings of the 5th International Workshop on Frontiers in Evolutionary Algorithms, The 7th Joint Conference on Information Sciences, September 26-30, 2003, Research Triangle Park, North Carolina, Edited by Ken Chen (et. al), pp. 315-318, 2003.

[16] M. Oltean and D. Dumitrescu, *Multi Expression Programming*, Journal of Genetic Programming and Evolvable Machines, Kluwer, second tour of review, 2002, (available at www.mep.cs.ubbcluj.ro).

[17] C. Ryan, J.J. Collins and M. O'Neill, *Grammatical Evolution: Evolving Programs for an Arbitrary Language*, In Proceedings of the First European Workshop on Genetic Programming, pp. 83-95, Springer-Verlag, Berlin, 1998.

[18] A. Stoica, R. Zebulum, X. Guo, D. Keymeulen, M. Ferguson and V. Duong, *Silicon Validation of Evolution Designed Circuits*, In Proceedings of the 2003 NASA/DoD Conference on Evolvable Hardware, pp. 21-26, 2003.

[19] G. Syswerda, *Uniform Crossover in Genetic Algorithms*, In Proceedings of the 3rd International Conference on Genetic Algorithms, J.D. Schaffer (Editor), Morgan Kaufmann Publishers, CA, 2-9, 1989.

[20] D.H. Wolpert and W.G. McReady, *No Free Lunch Theorems for Optimization*, IEEE Transaction on Evolutionary Computation, Vol. 1, pp 67-82, 1997.

[21] D.H. Wolpert and W.G. McReady, *No Free Lunch Theorems for Search*, Technical Report, SFI-TR-05-010, Santa Fe Institute, 1995.