Multi Expression Programming

Mihai Oltean

Department of Computer Science,
Faculty of Mathematics and Computer Science,
Babeş-Bolyai University, Kogălniceanu 1,
Cluj-Napoca, 3400, Romania.
email: mihai.oltean@gmail.com

D. Dumitrescu

Department of Computer Science,
Faculty of Mathematics and Computer Science,
Babeş-Bolyai University, Kogălniceanu 1,
Cluj-Napoca, 3400, Romania.
email: ddumitr@nessie.cs.ubbcluj.ro

Abstract. Multi Expression Programming (MEP) is a new evolutionary paradigm intended for solving computationally difficult problems. MEP individuals are linear entities that encode complex computer programs. MEP chromosomes are represented in the same way as C or Pascal compilers translate mathematical expressions into machine code. MEP is used for solving some difficult problems like symbolic regression and game strategy discovering. MEP is compared with Gene Expression Programming (GEP) and Cartesian Genetic Programming (CGP) by using several well-known test problems. For the considered problems MEP outperforms GEP and CGP. For these examples MEP is two magnitude orders better than CGP.

Keywords: Multi Expression Programming, Genetic Programming, Gene Expression Programming, Tic-Tac-Toe, Symbolic Regression, Discovering Game Strategy.

1. Introduction

History of algorithms is the history of improvements. Better algorithms are developed every day to replace others old and less efficient algorithms. These improvements concern two main issues: speed and memory requirements. Whereas the computer performances have been continuously improved, the need for solving more and more complex problems made these improvements to be quite inefficient. Thus the theoreticians searched for new gates for eluding the computers powerless. The efforts have been directed in to two main directions: (a) developing new and powerful algorithms, (b) parallelization existing ones.

History of evolutionary algorithms may be identified with the history of algorithms: inefficient and slow implementations at the beginning and then more and more efficient algorithms and implementations.

Genetic Programming (GP) [11], [12], which was originally implemented in LISP is a typical example. This programming language permits elegant implementation of programs that change their code during execution. This is all that the evolutionary algorithms that evolve source code need. Though LISP and GP perfectly match, the 5^{th} generation languages (as LISP and PROLOG) do not perfectly match today computer processors. Thus, the execution of such programs is slow down by that bottleneck.

Advent of C++ implementations of GP [18], improved the speed of GP programs by one order of magnitude at least [17].

For dealing with some problems (as bloat) generated by tree representation of GP, several linear variants of GP have been proposed. Some of them are: Cartesian Genetic Programming (CGP) [14], Grammatical Evolution (GE) [13], Linear GP [3] and Gene Expression Programming (GEP) [6]. Their aim was to improve

the performance of GP and to offer a viable alternative of implementing GP in 3rd and 4th generation programming languages. All enumerated GP variants have a common feature: individuals are represented as linear entities (strings) that are decoded and expressed like non-linear entities (trees).

Proposed *Multi Expression Programming*[1] (MEP) technique follows the improvements trend of GP techniques. Some of MEP features are summarized in what follows:

a) MEP individuals are strings of genes encoding complex computer programs,

b) When MEP individuals encode expressions for symbolic regression problems, their representation is similar with the way in which compilers translate C or Pascal expressions into machine code [1]. This may lead to very efficient implementations into assembler languages. The ability of evolving machine code leads to very important speedups and it has been considered by others researchers too. For instance Nordin [15], which evolves programs represented in machine code form. Poli and Langdon proposed sub-machine-code GP [17], which exploits the processor ability to perform some Boolean operations in parallel.

c) MEP individuals store multiple solutions in a single chromosome. Usually the best solution is chosen for fitness assignment. This represents a unique feature of MEP and it is called strong implicit parallelism. This ability has also been investigated by others researchers ([3] [9]). It is remarkable that this feature does not increase MEP complexity when compared with others techniques like GEP and GE.

d) Evaluation of the expressions encoded into a MEP individual can be performed by a single parsing of the chromosome. Note that GEP chromosomes need at least two parsings for computing the fitness.

e) Offspring, obtained by crossover and mutation, are always syntactically correct MEP individuals (computer programs). Thus, no extra processing for repairing newly obtained individuals is needed.

MEP technique is used for solving different problems like symbolic regression and discovering game strategy. The overall conclusion is MEP performs very well for each considered problem.

MEP technique is compared with GEP and CGP on several well-known benchmark problems. The results suggest that MEP algorithm outperforms GEP and CGP on the considered examples.

2. Linear GP representations

In this section some GP techniques using linear representations are reviewed.

2.1. Cartesian Genetic Programming

Cartesian Genetic Programming (CGP) [14] 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 trees structures, thus allowing the construction of more complex computer programs [15].

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 with 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 other(s) node(s).

[1] MEP source code is available for download from www.mepx.org or https://github.com/mepx

An example of CGP program is depicted in Figure 1.

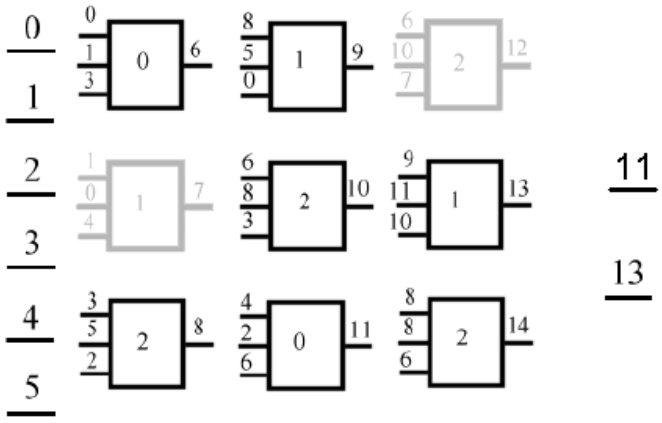

Figure 1. A CGP program with 5 inputs, 2 outputs and 3 functions (0, 1, 2 inside square nodes). The grey squares represent unconnected nodes

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 from top down and printing the input nodes and the function symbol for each node. The CGP chromosome depicted in Figure 1 is encoded as:

C = 0 1 3 0 1 0 4 1 3 5 2 2 8 5 0 1 6 8 3 2 4 2 6 0 6 10 7 2 9 11 10 1 8 8 6 2 11 13

Standard string genetic operators (crossover and mutation) are used with CGP. Crossover may be applied without any restrictions, but mutation operator requires that some conditions to be met. Node suppling the outputs are not fixed as they are also subject to crossover and mutation.

2.2. Gene Expression Programming

Gene Expression Programming (GEP) [6] uses linear chromosomes. A chromosome is composed of genes containing terminal and nonterminal symbols. Chromosomes are modified by mutation, transposition, root transposition, gene transposition, gene recombination, one-point and two-point recombination.

GEP genes are composed of a *head* and a *tail*. The head contains both functions (nonterminal) and terminals symbols. The tail contains only terminal symbols.

For each problem the head length (denoted h) is chosen by the user. The tail length (denoted t) is calculated using the formula:

$$t = (n - 1)\, h + 1, \qquad (1)$$

where n is the number of arguments of the function with more arguments.

Translation of a tree–program into a GEP gene is made by breadth-first parsing.

Consider a gene composed of symbols in the set S:

$S = \{*, /, +, -, a, b\}$.

In this case we have $n = 2$. If we choose $h = 10$ we get

$t = 11$,

and the length of the gene is 10 + 11 = 21. Such a gene is given below:

$C = +*ab-+aa/+ababbbababb$.

The expression encoded by the gene C is

$E = ((a + b) – a) * b + a$.

The expression E represents the phenotypic transcription of a chromosome having C as its unique gene.

Usually a GEP gene is not entirely used for phenotypic transcription. If the first symbol in the gene is a terminal symbol the expression tree consist of a single node. If all symbols in the head are nonterminals the expression tree uses all the symbols of the gene.

GEP genes may be linked by a function symbol is order to obtain a fully functional chromosome. In the current version of GEP the linking functions for algebraic expressions are addition and multiplication. A single type of function is used for linking multiple genes.

In some situation this seems to be enough (see [6]). But, generally, it is not a good idea to assume that the genes may be linked either by addition or by multiplication. If the functions {+, -, *, /} are used as linking operators then the complexity of the problem grows substantially (since the problem of determining how to mixed these operators with the genes is as hard as the initial problem).

When solving computationally difficult problems (like automated code generation) one may not assume that a unique kind of nonterminal symbol (like **for, while** or **if** instructions) is necessary for inter-connecting different program parts.

Furthermore, the success rate of GEP increases with the number of genes in the chromosome [5]. But after a certain value the success rate decreases if the number of genes in the chromosome increases. This is because one can not force a complex chromosome to encode a less complex expression.

Thus when you use GEP you must be careful with the number of genes that form the chromosome. The number of genes in the chromosome must be somehow related with the complexity of the expression that you want to discover.

According to [6] GEP performs better than standard GP for several particular problems.

2.3. Linear genetic programming

Linear Genetic Programming (LGP) [3] uses a specific linear representation of computer programs. Instead of tree-based GP expressions of a functional programming language (like ***LISP***) programs of an imperative language (like ***C***) are evolved.

An LGP individual is represented by a variable-length sequence of simple ***C*** language instructions. Instructions operate on one or two indexed variables (registers) r or on constants c from predefined sets. The result is assigned to a destination register, e.g. $r_i = r_j * c$.

An example of LGP program is the following one:

```
void ind(v)
{
```

```
  double v[8];
  …
  v[0] = v[5] + 73;
  v[7] = v[9] – 59;
  if (v[1] > 0)
  if (v[5] > 21)
    v[4] = v[2] *v[1];
  v[2] = v[5] + v[4];
  v[6] = v[9] * 25;
  v[6] = v[4] – 4;
  v[1] = sin(v[6]);
  if (v[0] > v[1])
    v[3] = v[5] * v[5];
  v[7] = v[6] * 2;
  v[5] = [7] + 115;
  if (v[1] <= v[6])
    v[1] = sin(v[7]);
}
```

A linear genetic program can be transformed into a functional representation by successive replacements of variables starting with the last effective instruction. Variation operators are crossover and mutation. By crossover continuous sequences of instructions are selected and exchanged between parents. Two types of mutations are used: micro mutation and macro mutation. By micro mutation an operand or an operator of an instruction is changed. Macro mutation inserts or deletes a random instruction.

3. MEP technique

In this section *Multi Expression Programming* (MEP) paradigm is described. MEP uses a new representation technique and a special phenotype transcription model. A MEP chromosome usually encodes several expressions.

3.1. Standard MEP Algorithm

Standard MEP algorithm starts with a randomly chosen population of individuals.

A fixed number of the best individuals enter in the next generation (elitism). We call *elitism size* the number of best individuals copied without modification in the next population. The mating pool is filled using binary tournament selection. Individuals from mating pool are randomly paired and recombined with a fixed crossover probability p_c. By recombination of two parents two offspring are obtained. The offspring are mutated and enter the next generation. Considered variation operators ensure the chromosome length is a constant of the search process. The algorithm returns as its answer the best expression evolved along a fixed number of generations.

Standard MEP algorithm is outlined below:

STANDARD MEP ALGORITHM

```
begin
   Generate Initial Population;
   Evaluate_Individuals;
   while not Termination_Condition do
       Elitism;
       Selection;
       Recombination;
       Mutation;
       Evaluate_Individuals;
   endwhile
end
```

3.2. MEP representation

MEP genes are (represented by) substrings of 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 itself in the chromosome.

Proposed representation ensures 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

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

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 F and T is given below:

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

The maximum number of symbols in MEP chromosome is given by the formula:

$$Number\_of\_Symbols = (n + 1) \cdot (Number\_of\_Genes - 1) + 1 \quad (2),$$

where n is the number of arguments of the function with the highest number of arguments. The maximum number of effective symbols is achieved when each gene (excepting the first one) encodes a function symbol with the highest number of arguments.

The minimum number of effective symbols is equal to the number of genes and it is achieved when all genes encode terminal symbols only.

3.3. Decoding MEP chromosome and fitness assignment process

Now we are ready to describe how MEP individuals are translated into computer programs. This translation represents the phenotypic transcription of the MEP chromosomes.

Phenotypic translation is obtained by parsing 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.

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

$E_1 = a + b.$ (3)

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

$E_2 = c + d.$ (4)

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

$E_3 = (a + b) * (c + d).$ (5)

E_3 is the expression encoded by the whole chromosome.

It is obvious that a MEP chromosome generally can not encode an expression that uses all the genes. Consider the chromosome:

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

It is obvious that this chromosome can not encode a single expression which uses all of its genes. But this chromosome encodes two complex expressions, namely:

$E_1 = a + b,$ (6)

$E_2 = c + d.$ (7)

The connection between expressions E_1 and E_2 it is not specified by chromosome so we do not know how to combine them in a unique expression.

This is why each MEP chromosome is allowed to encode a number of expressions equal to the chromosome length (number of genes). Previously described chromosome encodes the following expressions:

$E_1 = a,$ (8)

$E_2 = b,$ (9)

$E_3 = a + b,$ (10)

$E_4 = c,$ (11)

$E_5 = d,$ (12)

$E_6 = c + d.$ (13)

The value of these expressions may be computed by parsing the chromosome top down. Partial results are computed by dynamic programming [2] 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.

Let us consider that a procedure for computing the fitness $f(E_i)$ is available.

Chromosome fitness is defined by considering the fitness of each sub-expression encoded by the chromosome. Therefore we may define:

$$f(C) = \underset{E_i}{\oplus} f(E_i),$$

where E_i are the expressions encoded by the chromosome C, and $\oplus$ stands for a suitable operation. According to the problem $\oplus$ could be *min*, *max*, *mean*, etc.

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:

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

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 with the lowest fitness of the expressions encoded in chromosome:

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

When we have to deal with others 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.

3.4. MEP strong implicit parallelism

Generally a GP chromosome encodes a single expression (computer program). This is also the case for GEP and GE chromosomes. By contrast, a MEP chromosome encodes several expressions (it allows a multi-expression representation). Each of the encoded expressions may be chosen to represent the chromosome, i.e. for giving chromosome phenotypic transcription. Usually the best of the expressions the chromosome encodes supplies phenotypic transcription (represents the chromosome).

It is also possible that several expressions may be selected to represent a single chromosome. In this case we may say that the chromosome has a manifold phenotypic transcription. Multi-expression representation gives a supplementary power to the method.

Therefore MEP technique is based on a special kind of implicit parallelism. A chromosome usually encodes several well-defined expressions. We may call *strong implicit parallelism* (SIP) the ability of MEP chromosome to encode several syntactically correct expressions in a chromosome.

Some numerical experiments presented in Section 4.5 reveals the practical importance of SIP.

The ability of storing multiple solutions in a single chromosome has been suggested by others researchers, too (see for instance [10]). Several attempts have been made for implementing this ability in GP technique. For instance Handley [9] stored the entire population of GP trees in a single graph. In this way a lot of memory is saved. Also if partial solutions are efficiently stored we can get an important speed up. Linear GP [3] is also very suitable for storing multiple solutions in a single chromosome. In that case the multi expression ability is given by the possibility of choosing any variable as the program output.

3.5. Discussion about MEP representation

We can see that the effective expression length can increases exponentially with the length of the chromosome. This is happening because some sub-expressions may be used more than once to build a more complex (bigger) expression. Consider, for instance, we want to obtain a chromosome that encodes the expression a^{2^n}, and only the operators {+, -, *, /} are allowed. If we use GEP representation the chromosome has to contain at least $(2^{n+1} - 1)$ symbols since we need to store 2^n terminal symbols and $(2^n - 1)$ function operators. A GEP chromosome that encodes the expression $E = a^8$ is given below:

C = *******aaaaaaaa*.

A MEP chromosome uses only $(3n + 1)$ symbols for encoding the expression a^{2^n}. A MEP chromosome that encodes expression $E = a^8$ is given below:

1: *a*
2: * 1, 1
3: * 2, 2
4: * 3, 3

As a further comparison, when $n = 20$, a GEP chromosome has to have 2097151 symbols, while MEP needs only 61 symbols.

MEP representation is similar with GP and CGP, in the sense that each function symbol provides pointers towards its parameters. Whereas both GP and CGP have complicated representations (trees and graphs), MEP provides an easy and effective way to connect (sub) parts of a computer program. Moreover, the motivation for MEP was to provide an expression representation close to the way in which C or Pascal compilers interprets the mathematical expressions [1]. That code is also called three adreses code or intermediary code.

Some GP techniques, like Linear GP, remove non-coding sequences of chromosome during the search process. As already noted ([3], [6]) this strategy does not give the best results. The reason is sometimes a part of the useless genetic material have to be kept in the chromosome in order to maintain population diversity.

There is a notable difference of speed between MEP and GEP due to the number of chromosome parsing operations. MEP chromosomes are traversed one to compute the fitness, whereas GEP chromosomes need to be parsed twice for computing the fitness.

3.6. Selection

Standard MEP algorithm is a generational evolutionary procedure. Selection ensures the fittest individuals having a higher chance to be represented in the next generation. *q*-tournament selection is used. The best individual from *q* randomly chosen individuals enters the mating pool (see [5], [8]).

Some experiments for setting tournament size were performed (results not shown). Binary tournament ($q = 2$) seems to work very good on considered test problems. Thus binary tournament will be used in all experiments considered in this paper.

3.7. Search operators

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

3.7.1. Recombination

Three variants of recombination have been considered and tested within our MEP implementation: one-point recombination, two-point recombination and uniform recombination.

3.7.1.1. One-point recombination

One-point recombination operator in MEP representation is analogous to the corresponding binary representation operator. One crossover point is randomly chosen and the parent chromosomes exchange the sequences at the right side of the crossover point.

Example

Consider the chromosomes C_1 and C_2:

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

Choosing the crossover point after position three two offspring F_1 and F_2 are obtained as follows:

F_1	F_2
1: ***b***	1: *a*
2: *** 1, 1**	2: *b*
3: **+ 2, 1**	3: + 1, 2
4: *c*	4: ***a***
5: *d*	5: *** 3, 2**
6: + 4, 5	6: ***a***
7: * 3, 6	7: **- 1, 4**

3.7.1.2. Two-point recombination

Two crossover points are randomly chosen and the chromosomes exchange genetic material between the crossover points.

Example

Let us consider the chromosomes C_1 and C_2:

C_1	C_2
1: ***b***	1: *a*
2: *** 1, 1**	2: *b*
3: **+ 2, 1**	3: + 1, 2
4: ***a***	4: *c*
5: *** 3, 2**	5: *d*
6: ***a***	6: + 4, 5

7: **- 1, 4** 7: * 3, 6

Suppose that the crossover points were chosen after the position 2 and 5. In this case the obtained offspring F_1 and F_2 are:

F_1	F_2
1: ***b***	1: *a*
2: *** 1, 1**	2: *b*
3: + 1, 2	3: **+ 2, 1**
4: *c*	4: ***a***
5: *d*	5: *** 3, 2**
6: ***a***	6: + 4, 5
7: **- 1, 4**	7: * 3, 6

3.7.1.3. Uniform recombination

Within uniform recombination offspring genes are taken randomly from one parent or another.

Example

Consider the chromosomes C_1 and C_2:

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

Using uniform recombination two offspring F_1 and F_2 are obtained as follows:

F_1	F_2
1: *a*	1: ***b***
2: *** 1, 1**	2: *b*
3: **+ 2, 1**	3: + 1, 2
4: *c*	4: ***a***
5: *** 3, 2**	5: *d*
6: + 4, 5	6: ***a***
7: **- 1, 4**	7: * 3, 6

3.7.2. Mutation

Each gene in the chromosome may be the target of mutation operator. A mutation probability (p_m) is considered when applying mutation operator.

3.7.2.1. Standard mutation

By mutation some symbols in the chromosome are changed. To preserve the consistency of the chromosome its first gene must encode a terminal symbol.

If the current gene encodes a terminal symbol it may be changed into another terminal symbol or into a function symbol. In the last case the positions indicating the function arguments are also generated by mutation.

If the current gene encodes a function the gene may be mutated into a terminal symbol or into another function (function symbol and pointers towards arguments).

3.7.2.2. Smooth mutation

For changing a function symbol one may apply a *smooth mutation operator*. Smooth mutation changes each symbol in the gene (i.e. function symbol or function parameters position) with a fixed probability p_{sm}.

If the function in the gene has two parameters the value

$p_{sm} = 0.33$

is suggested. This is a mutation probability value equivalent with one position mutation per gene.

Smooth mutation is an additional search operator intended to perform a fine grained search in the solutions space.

3.7.3. Example

In this example we consider a mutation probability

$p_m = 0.28$

equivalent with two mutations per chromosome. Smooth mutation operator is not used. Thus each gene is freely changed into another gene.

Consider the chromosome:

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

If genes 3 and 6 are selected for mutation then an offspring could be the following one:

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

In this case a terminal symbol has been changed into a function symbol (gene 3). Positions of function parameters changed into another ones (gene 6).

3.8. Handling exceptions within MEP

Exceptions are special situations that interrupt the normal flow of expression evaluation (program execution). An example of exception is *division by zero* which is raised when the divisor is equal with zero.

Exceptions handling is a mechanism that performs special processing when an exception is thrown.

Usually GP techniques use a *protected exception* handling mechanism [13], [14]. For instance if a division by zero exception is encountered, a predefined value (for instance 1 or the numerator) it is returned.

GEP uses a different mechanism: if an individual contains an expression that generates an error that individual receive the lowest fitness possible.

MEP uses a new and specific mechanism for handling exceptions. When an exception is encountered (which is always generated by a gene containing a function symbol), the gene that generated the exception is mutated into a terminal symbol. Thus, no infertile individual appears in population.

3.9. Improving MEP algorithm: weighting symbols

For particular problems there is an easy way to improve performance of MEP algorithm by weighting the function as well as terminal symbols. Obtained evolutionary procedure is called *Weighted MEP* (WMEP) algorithm.

By weighting, some symbols may appear in chromosome more often than other symbols. However suggested improvement is based on observations on particular problems. Generally it can not be assumed that some symbols appear in chromosome more frequently than other symbols.

3.10. MEP complexity

Let N be the number of individual in population and NG the number of genes in chromosome.

The fitness of an individual (when solving symbolic regression problems) can be computed in $O(NG)$ steps by dynamic programming [2]. In fact MEP chromosome needs to be traversed once for computing the fitness. GEP chromosomes must be traversed twice for computing the fitness.

The complexity of MEP algorithm when solving symbolic regression problems is:

$$O(N \cdot NG). \quad (16)$$

Thus, MEP algorithm does not have a higher complexity than other GP - techniques that encode a single computer program in each chromosome.

4. MEP technique applied for solving symbolic regression problems

In this section standard and weighted MEP are used for solving symbolic regression problems.

4.1. Standard MEP for symbolic regression

The aim of symbolic regression is to discover a function that satisfies a set of fitness cases. For instance we want to discover the function:

$$f(x) = x^4 + x^3 + x^2 + x, \quad (17)$$

based on a set of 10 fitness cases randomly generated over the interval [0, 20] as used by GEP in [6].

For this problem we use the set of functions

$F = \{+, -, *, /\}$,

and the set of terminals

$T = \{x\}$.

4.1.1. Experiment 1

In this experiment the success rate of the standard MEP algorithm is analyzed. A comparison with the results supplied by GEP technique [6] is realized. The chromosome length is gradually increased. Algorithm parameters are given in Table 1.

Population size	30
Number of generations	50
Mutation	2 genes / chromosome
Crossover type	One-point-crossover
Crossover probability	0.7
Selection	Binary tournament
Elitism size	1

Table 1. Algorithm parameters for the Experiment 1.

Because GEP and MEP use different chromosome representations we can not make a direct comparison based on chromosome length. Instead we will provide a comparison based on the number of symbols in chromosome.

In this case $n = 2$ and the maximum number of symbols in chromosome is:

$Number\_of\_Symbols = 3 \cdot Number\_of\_Genes - 2$.

GEP chromosome consists of a single gene with the head length of h. Thus the total number of symbols in GEP chromosome is $(2h + 1)$.

Success rates of the MEP and GEP algorithms depending on number of symbols in the chromosome are depicted in Figure 2. GEP parameters are the same as MEP parameters.

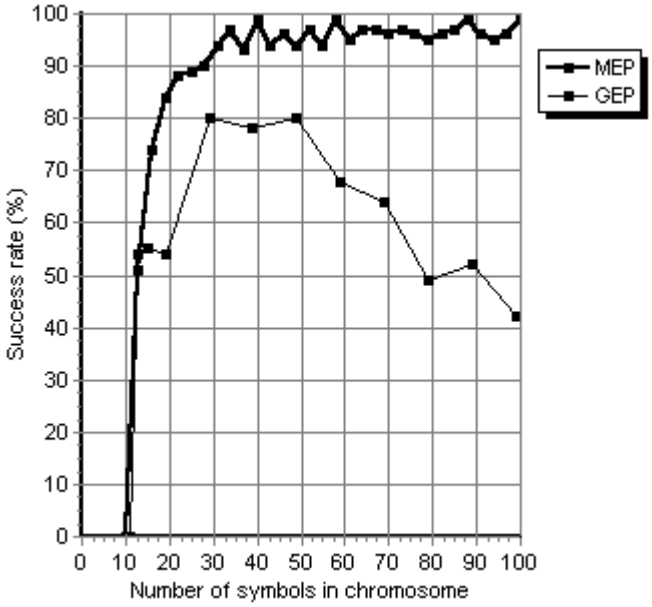

Figure 2. Success rate of MEP and GEP algorithms. The number of symbols in chromosome varies between 8 and 100. The results are summed over 100 runs.

One may note that the success rate of GEP increases up to 80% and then decreases. This indicates that very long GEP chromosomes cannot encode short expressions efficiently. The length of GEP chromosome must be somehow related with the length of the expression that must be discovered.

The success rate of the MEP algorithm increases with the chromosome length and never decreases toward very low values. When the search space (chromosome length) increases, an increased number of expressions are encoded by MEP chromosomes. Very large search spaces (very long chromosomes) are very beneficial for MEP technique because

4.1.2. Experiment 2

From Experiment 1 we may infer that for the considered problem the MEP success rate never decreases to very low values. To obtain an experimental evidence for this assertion longer chromosomes are considered. We extend chromosome length up to 300 genes (898 symbols).

The success rate of MEP is depicted in Figure 3. We can observe that the MEP success rate lies in the interval [92, 100] for the chromosome length greater than 12. One may note that after that the chromosome length becomes 10, the success rate never decrease fewer than 90%.

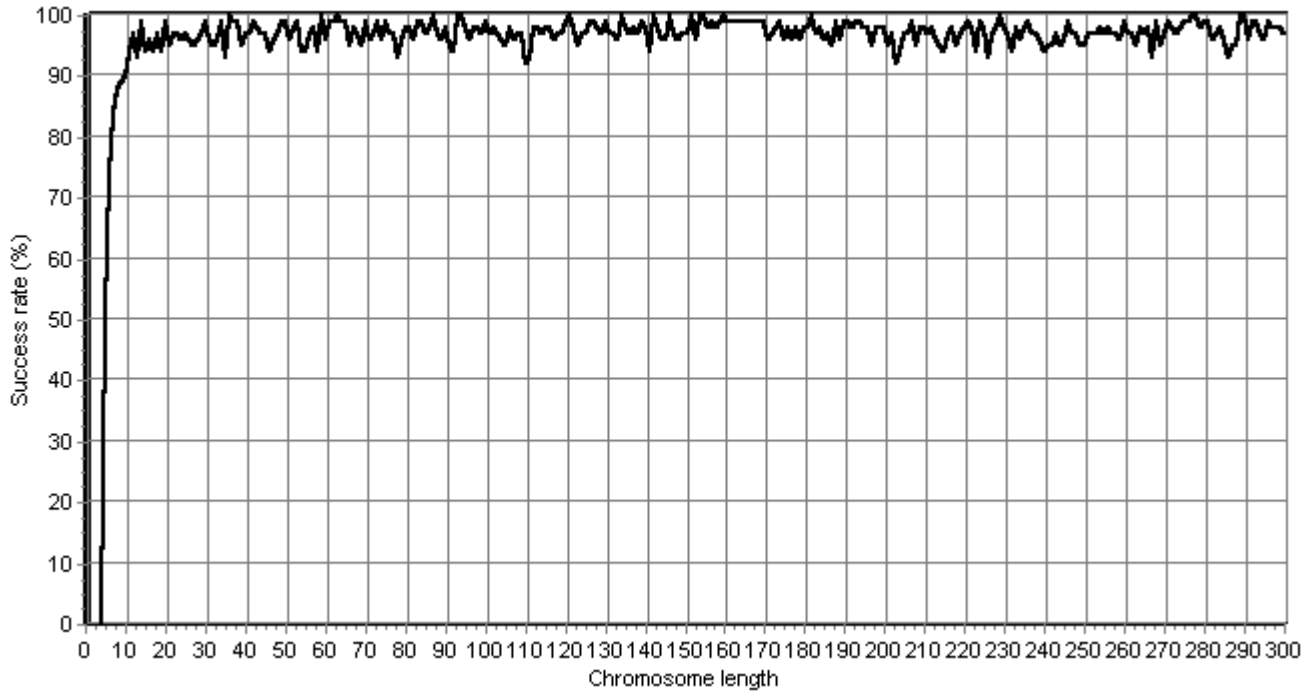

Figure 3. Success rate of MEP algorithm. The chromosome length varies between 5 and 100. The results are summed over 100 runs.

4.1.3. Experiment 3

The first position in each chromosome must be occupied by a terminal symbol in order to obtain syntactically correct computer programs. One may also fill positions 2, 3,… with terminal symbols.

In this experiment the relationship between success rate and the number of initial positions compulsory filled with terminal symbols is analyzed.

Algorithm parameters are given in Table 2.

Population size	30
Number of generations	50
Chromosome length	40 genes
Mutation	2 genes / chromosome
Crossover type	One-point-crossover
Crossover probability	0.7
Selection	Binary tournament
Elitism size	1

Table 2. Algorithm parameters for the Experiment 3.

The results of this experiment are depicted in Figure 4.

Figure 4. Success rate of MEP algorithm when the number of first positions occupied by terminal symbols varies. The number of first positions occupied by terminals varies between 1 and 20. The results are summed over 100 runs.

We may infer that the best results are obtained when only the first position is compulsory a terminal symbol. After this value the success rate gradually decreases.

Imposing terminal symbols to other initial positions seems to generate performance decreasing. This loss of performance is due to the reduction of chromosome positions encoding operators.

For some problems, where the solution is a simple expression consisting of a single terminal, the statement above may not hold. In those cases an increased number of positions containing terminal symbols may be very beneficial. However, if the solution contains many function symbols it is always a good idea to force only the first position in chromosome to hold a terminal symbol.

4.1.4. Experiment 4

In this experiment the relationship between the success rate and the population size is analyzed. Algorithm parameters for this experiment are given in Table 3.

Number of generations	50
Chromosome length	17 genes
Mutation	2 genes / chromosome
Crossover type	One-point-crossover
Crossover probability	0.7
Selection	Binary tournament
Elitism size	1

Table 3. Algorithm parameters for the Experiment 4.

The maximum number of symbols in MEP chromosome is equal with the number of symbols in GEP chromosome (49 symbols).

Experiment results are given in Figure 5.

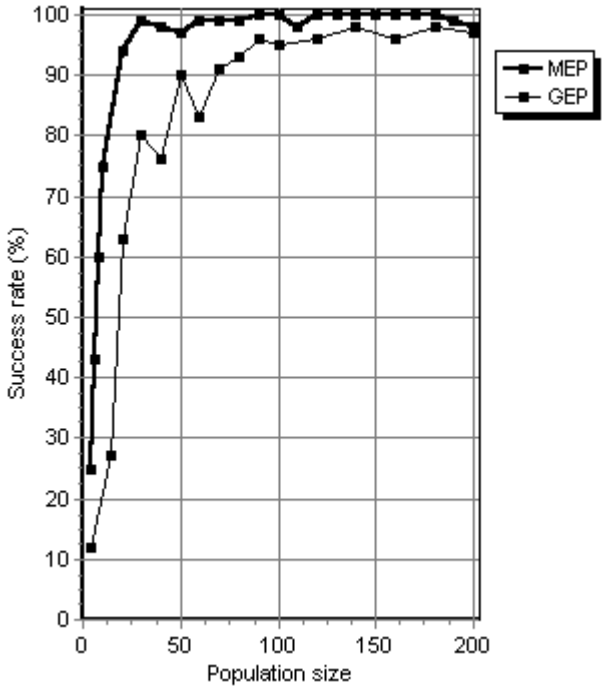

Figure 5. Success rate of MEP algorithm. Population size varies between 2 and 200. The results are summed over 100 runs.

For the considered problem (symbolic regression) and for the MEP algorithm parameters given in Table 3 optimal population size is 30. Corresponding success rate is 99%. This result suggests that small MEP populations may supply very good results.

MEP has a higher success rate than GEP for the same parameter settings. For a population of size 30 the GEP success rate reaches 80%, while the success rate of MEP is 99%. For a population with 50 individuals the GEP success rate reaches to 90%, while MEP success rate is 98%.

4.1.5. Experiment 5

In this experiment the relationship between the MEP and GEP success rate and the number of generations used in the search process is analyzed.

MEP algorithm parameters are given in Table 4.

Population size	30
Chromosome length	27 genes
Mutation	2 genes / chromosome
Crossover type	One-point-crossover
Crossover probability	0.7
Selection	Binary tournament
Elitism size	1

Table 4. Algorithm parameters for the Experiment 6.

The maximum number of symbols in MEP chromosome is equal with the number of symbols in GEP chromosome (79 symbols).

Experiment results are given in Figure 6.

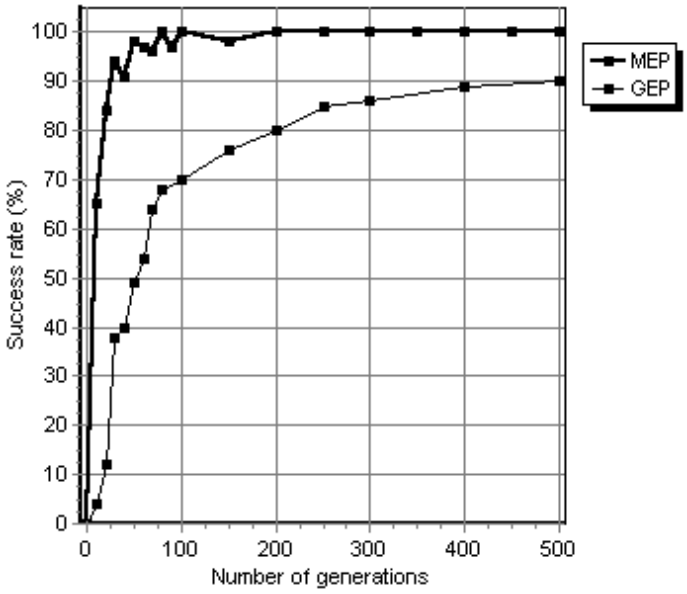

Figure 6. Relationship between the success rate of MEP algorithm and the number of generations used in the search process. The number of generations varies between 10 and 500. The results are summed over 100 runs.

The success rate reaches 100% when the number of generations reaches 80. GEP algorithm with similar parameter settings reaches the success rate of 69% when the number of generations reaches 70.

Moreover, according to [6], GEP algorithm reaches the success rate of 90% only when the number of generations reaches 500.

For the considered generation range GEP success rate never reaches 100%. MEP success rate is stabilized to 100% since generation 200.

4.2. Weighted MEP for symbolic regression problem

For a given problem could be some preferred symbols. These symbols are expected to appear more frequently in the solution expression.

To distinguish symbols we may associate a weight for each function symbol. A preference for a certain symbol may be forced by assigning it to a larger weight. We illustrate this method for the fourth polynomial problem considered at the beginning of the section 4.

We weight the symbols + and * such that these symbols have a double chance to appear in chromosome than other symbols. Results of weighted MEP technique are described in the following experiment.

4.2.2. Experiment with Weighted MEP

In this experiment we examine the success rate of weighted MEP algorithm when the number of genes in chromosome is variable. Results are compared with those obtained by the standard MEP.

Algorithm parameters are given in Table 5.

Population size	30
Number of generations	20
Chromosome length	20 genes
Mutation	2 genes / chromosome
Crossover type	One point crossover
Crossover probability	0.7
Selection	Binary tournament
Elitism size	1

Table 5. Algorithm parameters for the Weighted MEP.

Figure 7 depicts the relationship between the success rate and the chromosome length when function symbols are weighted. According to the considered weighing system, the operators + and * have double chance to appear in chromosome.

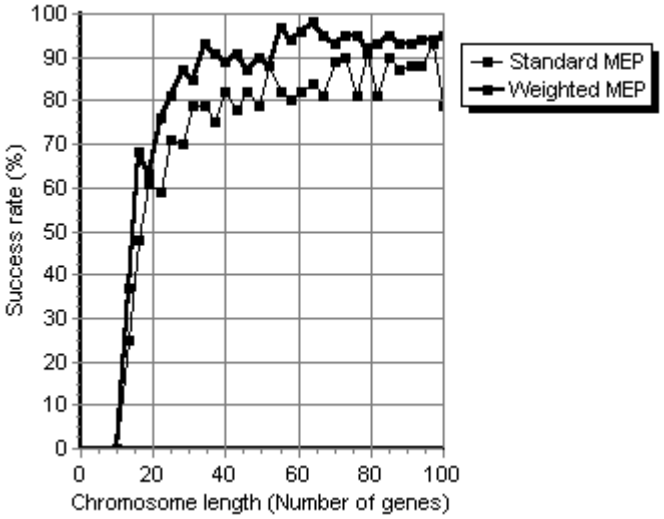

Figure 7. Success rate of MEP algorithm when operators are weighted. The chromosome length varies between 10 and 100. The results are summed over 100 runs.

One may note that success rate of Weighted MEP is higher than the success rate of Standard MEP. These improvements are correlated with a particular weight system based on observations on the problem.

4.3. MEP vs. CGP

In [14], CGP was used for symbolic regression of the sixth polynomial problem [12]:

$f(x) = x^6 - 2x^4 + x^2$.

In this section MEP technique is used to solve the same problem using parameters settings similar with those of CGP. To provide a fair comparison, all experiment conditions described in [14] are carefully reproduced for MEP technique.

CGP chromosomes were bounded by the following parameters: $n_r = 1$, $n_c = 10$, $l = 10$. MEP chromosomes are set to contain 12 genes (in addition MEP uses two supplementary genes for the terminal symbols {1.0, x}).

MEP parameters (similar with those used by CGP) are given in Table 6.

Chromosome length	12 genes
Mutation	2 genes / chromosome
Crossover type	One point crossover
Crossover probability	0.7
Selection	Binary tournament
Elitism size	1
Terminal set	{x, 1.0}
Function set	{+, −, *, /}
Number of runs	100

Table 6. Algorithm parameters for the MEP vs. CGP experiment.

CGP used a population of 500 individuals and a number of 8000 generations. We perform two experiments. In the first experiment we set the MEP population size to 500 individuals and we compute how many generations are needed to obtain the same success rate (61 %) as reported for CGP in [14].

When the MEP was run for 40 generations, the success rate was 60% (in 60 runs out of 100 MEP found the correct solution). Thus MEP requires 200 times less generations than CGP to solve the same problem (the sixth polynomial problem). This is an improvement with two orders of magnitude.

In the second experiment we keep fixed the number of generations (8000) and we use a small MEP population. We are interested to see which is the optimal population size required by MEP to solve this problem.

After several trials we found that MEP has a success rate of 60% when a population of 4 individuals is used. This means that MEP requires 125 times less individuals than CGP for solving the sixth polynomial problem. Again we deal with an improvement of two orders of magnitude.

4.5 Multi Expression Programming vs. Single Expression Programming

We are interested to see how the multi expression ability improves the performances of evolutionary algorithms. To achieve this, we consider a new algorithm that uses the MEP representation but only the last expression (the expression encoded by the last gene) is chosen to represent that chromosome. For instance, if we take the chromosome described in the section 3.1, the fitness of that chromosome is given by the fitness of the expression E_6.

The technique obtained in this way is called *Single Expression Programming* (SEP).

SEP is similar with GEP and CGP as it encodes a single solution in chromosome.

Several numerical experiments with MEP and SEP were performed to see if there are any differences between them. We analyze the relationships between the success rate of MEP and SEP with respect to the number of generations, the population size and the chromosome length.

In all experiments MEP and SEP use identical parameter settings. Some of these are given in Table 7.

Mutation	2 genes / chromosome
Selection	Binary tournament
Crossover type	1 point crossover
Crossover probability	0.9
Elitism size	1
Terminal set	{*x*}
Function set	{+, −, *, /, sin, cos}

Table 7. MEP and SEP algorithm parameters.

When the relationship between the success rate and the population size is analyzed, MEP and SEP are run over 100 generations using chromosomes of length 20.

When the relationship between the success rate and the number of generations is analyzed, MEP and SEP use populations of 50 individuals, each having 20 genes.

When the relationship between the success rate and the chromosome length is analyzed, MEP and SEP are run over 100 generations using populations of 50 individuals.

Both MEP and SEP algorithms were used to solve the sixth polynomial problem described in the section 4.4. Ten fitness cases (randomly generated over the interval [-1..1]) were used unchanged during the search process.

The results of these experiments are depicted in the Figure 9.

We can see that MEP outperforms SEP on the most cases (only one unimportant exception is recorded).

From Figure 9(b) we see that SEP has the same behavior as GEP: the success rate increases up to a value and then it has a decreasing tendency. The maximum success rate reached by GEP and SEP depends both on the training data and individuals representation.

Both SEP and GEP require a good match between the chromosome length and the length of the expression to be discovered.

SEP success rates, even when population size increases (Figure 9(c)), is very low (never jumps over 13%). Even a long population size (300 individuals) does not have beneficial effects over SEP success rate. This problem could be due to the need of a connection between the SEP chromosome length and the length of the target expression. It seems that this problem occurs very often with the systems that store a single expression in a chromosome (as is the case of GEP and SEP).

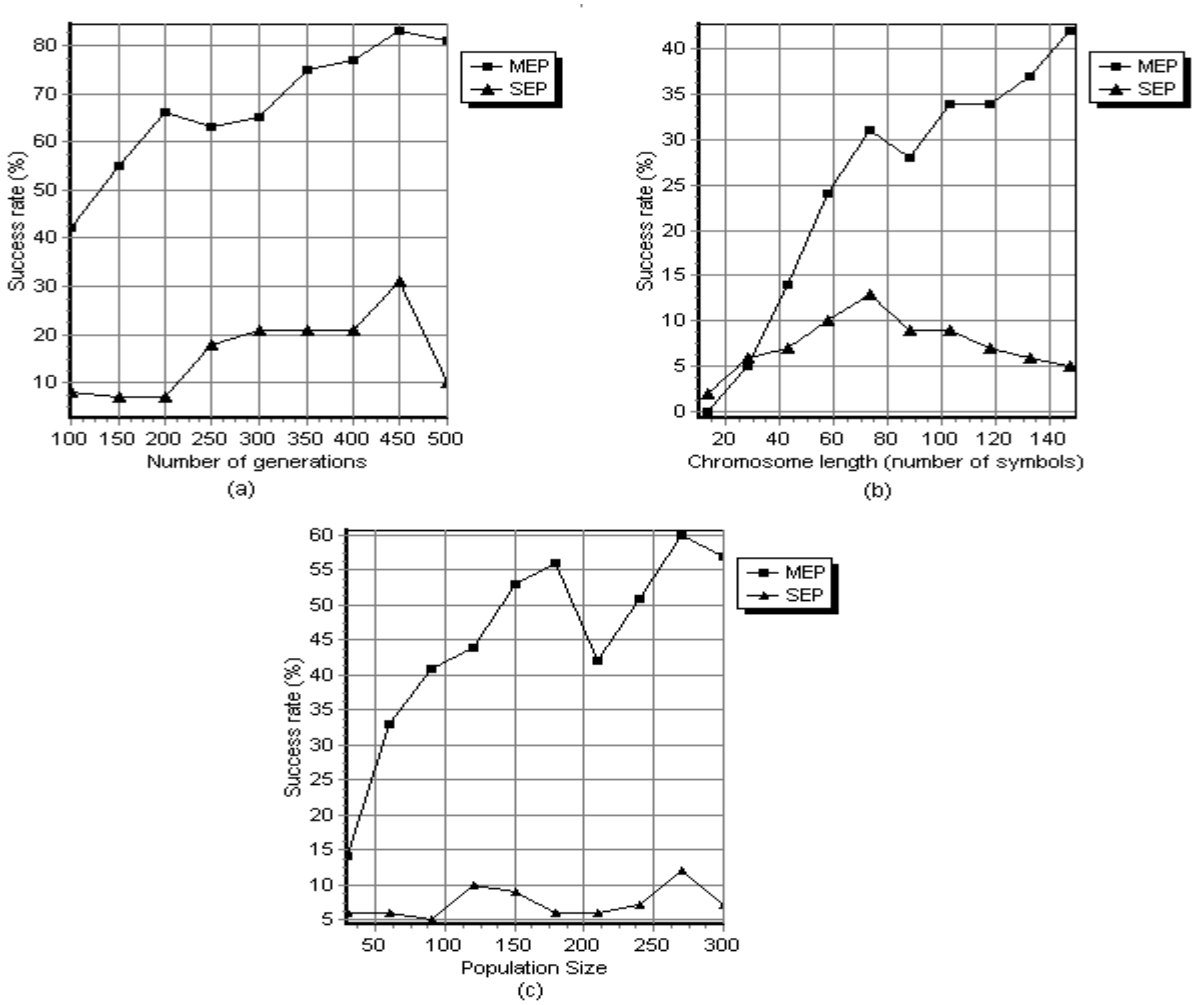

Figure 9. (a) Success rate of MEP and SEP when the number of generations varies between 100 and 500.

(b) Success rate of MEP and SEP when the chromosome length varies between 10 and 150.

(c) Success rate of MEP and SEP when the population size varies between 10 and 300.

The results are summed over 100 runs.

5. Discovering game strategies with MEP

In this section we investigate the application of MEP technique for discovering game strategies.

Koza [11] suggested that GP can be applied to discover game strategy. The game-playing strategy may be viewed as a computer program that takes the information about the game as its input and produces a move as output.

The available information may be an explicit history of previous moves or an implicit history of previous moves in the form of a current state of game (e.g. the position of each piece on the chess board) [11].

Tic-tac-toe (TTT, or naughts and crosses) is a game with simple rules, but complex enough to illustrate the ability of MEP to discover game strategy.

5.1. TTT game description

In Tic-Tac-Toe there are two players and a 3×3 grid. Initially the grid is empty. Each player moves in turn by placing a marker in an open square. By convention, the first player's marker is "X" and the second player's marker is "0".

The player that put three markers of his type ("X" for the first player and "0" for the second player) in a row is declared the winner.

The game is over when one of players wins or all squares are marked and no player wins. In the second case the game ends with draw (none of the players win). Enumerating the game tree shows that the second player can obtain at least a draw.

A well-known evolutionary algorithm that evolves game strategy has been proposed in [4]. This algorithm will be reviewed in the next section.

5.2. Chellapilla's approach of TTT

In [5] an evolutionary algorithm has been used in order to obtain a good strategy (that never loses) for the Tic-Tac-Toe game. A strategy is encoded in a neural network. A population of strategies encoded by neural networks is evolved.

Each network receives a board pattern as input and yields a move as output. The aim is to store in a neural network the function that gives the quality of a configuration. When a configuration is presented to the network, the network output (supplies) the next move.

Each neural network has an input layer with 9 nodes, an output layer with 9 nodes, and a hidden layer with a variable number of nodes.

Fogel's algorithm starts with a random population of 50 neural networks. For each network the number of nodes from the hidden layer is randomly chosen with a uniform distribution over integers 1...10. The initial weighted connection strengths and bias terms are randomly distributed according to a uniform distribution ranging over [-0.5, 0.5].

From each parent a single offspring is obtained by mutation. Mutation operator affects the hidden layer structure, weight connections and bias terms.

Each strategy encoded in a neural network was played 32 times against a heuristic rule base procedure.

The payoff function has several values corresponding to winning, loosing and draw.

The best individuals from a generation are retained to form the next generation.

The process is evolved for 800 generations. According to [4] the best obtained neural network is able to play to win or draw with a perfect play strategy.

5.3. MEP approach of TTT

In this section we illustrate the use of MEP to discover an unbeatable play strategy for Tic-Tac-Toe.

We are searching for a mathematical function *F* that gives the quality of each game configuration. Using this function the best configurations that can be reached in one move from the current configuration, is selected. Therefore function *F* supplies the move to be performed for each game configuration.

Function F evaluating each game configuration is represented as a MEP chromosome. The best expression encoded by a chromosome is chosen to be the game strategy of that chromosome.

Without any loose of generality we may allow MEP strategy to be the first player in each game.

All expressions in the chromosome are considered in the fitness assignment process. Each expression is evaluated using an "all-possibilities" procedure. This procedure executes all moves that are possible for the second player. The fitness of an expression E is the number of games that the strategy encoded by the expression E loses. Obviously the fitness has to be minimized.

Let us denote by

$P = (p_0, p_1 \ldots p_8)$

a game configuration.

Each p_k describes the states "X", "0" or an empty square. In our experiments the set {5, -5, 2} has been used for representing the symbols "X", "0" and the empty square.

The game board has been linearized by scanning board squares from up to down and from left to right. Thus the squares in the first line have indices 0, 1 and 2, etc. (see Figure 10).

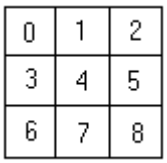

Figure 10. Game board linearized representation.

For this problem the set of nonterminal symbols is

$F = \{+, -, *, /\}$

and the set of terminal symbols is

$T = \{p_0, p_1, \ldots, p_8\}$.

Algorithm parameters are given in Table 8.

Population size	50
Chromosome length	50 genes
Mutation probability	0.05
Crossover type	Two-point-crossover
Selection	Binary tournament
Elitism size	1

Table 8. Algorithm parameters for TTT game.

The MEP algorithm is able to evolve a *perfect*, non-loosing, game strategy in 11 generations. This process requires less than 10 seconds when an Intel Pentium 3 processor at 1GHz is used.

In Figure 11 the fitness of the best individual in the best run and average fitness of the best individuals over all runs are depicted.

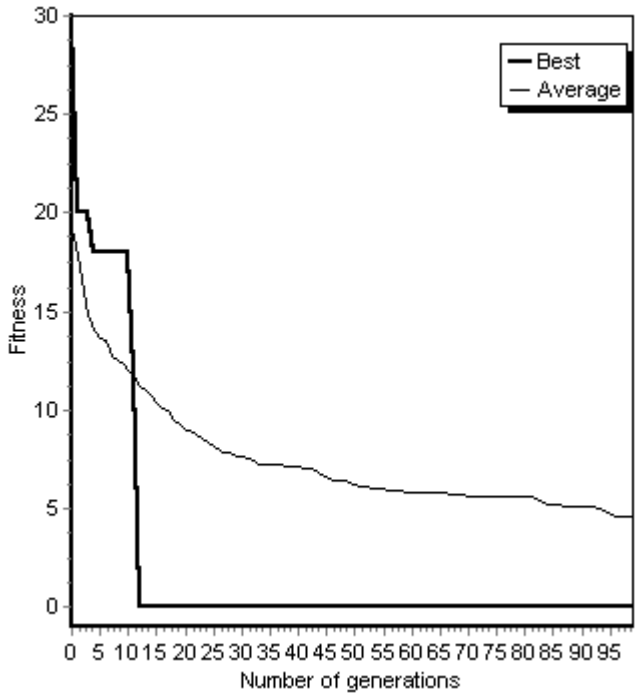

Figure 11. Fitness of the best individual in the best runs and the average fitness of the best individuals over all runs. The results are taken over 30 runs.

Form Figure 11 we may see that an individual representing a non-loosing strategy appears in the population at generation 14.

Some functions evolved by the MEP algorithm are given below:

$$F_1(P) = ((p_4-p_5-(p_6+p_5))*p_8+p_4*p_3)*(p_4-p_7), \quad (19)$$

$$F_2(P) = p_2-(p_8*p_7-p_4)-p_7-(p_2*p_5), \quad (20)$$

$$F_3(P) = (p_4*p_1+p_2)*p_7-(p_1-p_2+p_7*p_5)-(p_8-(p_3*p_5)). \quad (21)$$

These functions do not force the win when it is possible, but they never lose. This is a consequence of fitness assignment process. However, proposed technique can also generate a function that forces the win whenever it is possible.

It is not easy to compare this result with the result obtained by Chellapilla and Fogel [4] as the experiment conditions were not the same. In [4] evolved strategies play against a heuristic procedure, but here MEP formulas play against an all-moves procedure. Population size was the same (50 individuals). Individual sizes are difficult to be compared. All MEP individual have the same size: 148 symbols. Neural network's sizes used in [4] are variable since the hidden layer contains a variable number of nodes. If the number of nodes in the hidden layer is 9 then the size of the neural network (biases + connection weights) is 9 * 9 + 9 * 9 * 9 + 9 * 9 = 324.

MEP approach seems to be faster as MEP was able to discover a non-losing strategy in no more than 17 generations. As noted in [4] neural network approach requires 800 generations.

5.4. A good TTT heuristic vs. MEP

A good heuristic for Tic-Tac-Toe is described in what follows:

S1. *If a winning move is available make that move, else*
S2. *If a winning move is available for the opponent, move to block it, else*
S3. *If a move of the opponent that leads to two winning ways is available, block that move, else*
S4. *If the board center is available, move in the board center,*
S5. *If one ore more corners of the table are available, move in one of them, else*
S6. *Move randomly in an available square*.

This heuristic performs well on most of the game positions. But applying one of the formulas evolved by the MEP algorithm some benefits are obtained:

- easy implementation in programming languages,
- MEP evolved formula is an algorithm faster than previously shown heuristic.

5.5. Applying MEP for generating complex game strategies

Using the "all-possibilities" technique (a backtracking procedure that plays all the moves for the second player) allows us to compute the absolute quality of a game position.

For complex games a different fitness assignment technique is needed since the moves for the second player can not be simulated by an "all-possibilities" procedure (the number of moves that needs to be simulated is too large).

One fitness assignment possibility is to use a heuristic procedure that acts as the second player. But there are several difficulties related to this approach. If the heuristic is very good it is possible that none of evolved strategy could ever beat the heuristic. If the heuristic procedure plays as a novice then many evolved strategy could beat the heuristic from the earlier stages of the search process. In the last case fitness is not correctly assigned to population members and thus we can not perform a correct selection.

A good heuristic must play on several levels of complexity. At the beginning of the search process the heuristic procedure must play on easy level. As the search process advances the difficulty level of heuristic procedure must increases.

However for complex games such a procedure is difficult to implement.

Another possibility is to search for a game strategy using a coevolutionary algorithm [4]. This approach seems to offer the most spectacular results. In this case MEP population must develop intelligent behavior based only on internal competition.

7. Conclusions and further work

Multi Expression Programming is a new evolutionary technique that may be used for evolving computer programs. MEP technique uses a new solution representation and specific search operators.

It is documented that MEP technique can be used for solving various classes of difficult problems.

MEP correctly identified several important trends in nowadays Evolutionary Algorithms:

- encoding multiple solutions in single chromosome,
- individual encoding is similar to nowadays machine code (MEP representation is similar with the way in which C and Pascal compilers evaluate expressions),

- low complexity of fitness assignment process (MEP individuals are parsed only once for computing the fitness).

Several numerical experiments have been performed with MEP. For the considered problems MEP algorithm performs better than similar evolutionary techniques (particularly GEP and CGP).

Further efforts will be dedicated applying MEP for solving other real world difficult problems namely prediction of complex phenomena, classification, discovering heuristics for NP-Complete [7] problems, discovering strategies for complex games, etc.

References

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

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

[3] M. Brameier, W. Banzhaf, *A Comparison of Linear Genetic Programming and Neural Networks in Medical Data Mining*, IEEE Transactions on Evolutionary Computation, 5:17-26, 2001.

[4] K. Chellapilla, D. B. Fogel, *Evolution, Neural Networks, Games and Intelligence*, Proc. IEEE. Vol. 87, 1471-1496, 2000.

[5] D. Dumitrescu, B. Lazzerini, L. Jain, A. Dumitrescu, *Evolutionary Computation*. CRC Press, Boca Raton, FL, 2000.

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

[7] M.R. Garey, D.S. Johnson, *Computers and Intractability: A Guide to NP-completeness*, Freeman & Co, San Francisco, 1979.

[8] D.E., Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, MA, 1989.

[9] S. Handley, *On the Use of a Directed Graph to Represent a Population of Computer Programs*. In Proceeding of the IEEE World Congress on Computational Intelligence, 154-159, Orlando, Florida, 1994

[10] M.A. Lones, A.M. Tyrrell, *Biomimetic Representation in Genetic Programming* in proceedings of the Workshop on Computation in Gene Expression at the Genetic and Evolutionary Computation Conference 2001 (GECCO2001), July 2001.

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

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

[13] M. O'Neill, C. Ryan, *Grammatical Evolution*, IEEE Transaction on Evolutionary Computation, Vol 5, No 4, 2001.

[14] J. Miller, P. Thomson, *Cartesian Genetic Programming*, In R. Poli et al (editor) Third European Conference on Genetic Programming, Vol. 1802, Lecture notes in Computer Science, Springer, 2002.

[15] P. Nordin, *A Compiling Genetic Programming System that Directly Manipulates the Machine Code*, In K. E. Kinnear, Jr. (editor), Advances in Genetic Programming, 311-331, MIT Press, 1994.

[16] R. Poli, *Evolution of Graph-like Programs with Parallel Distributed Genetic Programming*. In T. Back (editor): Genetic Algorithms: Proceedings of the Seventh International Conference, 346-353, Morgan Kaufmann, San Francisco, CA, 1997.

[17] R. Poli, W. B. Langdon, *Sub-machine code Genetic Programming*, Technical report CSRP -98-18, School of Computer Science, University of Birmingham, 1998.

[18] A. Singleton, *Genetic Programming with C++*, BYTE, 171-176, 1991.