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: moltean@nessie.cs.ubbcluj.ro

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. In this paper a new evolutionary paradigm, called Multi-Expression Programming (MEP), intended for solving computationally difficult problems is proposed. A new encoding method is designed. MEP individuals are linear entities that encode complex computer programs. In this paper MEP is used for solving some computationally difficult problems like symbolic regression, game strategy discovering, and for generating heuristics. Other exciting applications of MEP are suggested. Some of them are currently under development. MEP is compared with Gene Expression Programming (GEP) by using a well-known test problem. For the considered problems MEP performs better than GEP.

Keywords: Evolutionary Computation, Multi Expression Programming, Genetic Programming, linear representation, Tic-Tac-Toe, symbolic regression, game strategy, heuristics generation.

1. Introduction

Genetic Programming (GP) [1] is an evolutionary technique intended for solving computationally difficult problems. GP individuals are represented and manipulated as non-linear entities (usually trees).

Recently several linear variants of GP have been proposed. Some of them are: Grammatical Evolution (GE) [2], Linear GP[3] and Gene Expression Programming (GEP) [4]. Their aim is 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).

In this paper a new evolutionary paradigm called *Multi Expression Programming* (MEP) is proposed. MEP uses a new linear solution representation. Each MEP individual is a string encoding complex expressions (computer programs). A MEP individual may encode multiple solutions of the current problem. Usually the best solution is chosen for fitness calculation.

A MEP individual (a chromosome) encodes several potential phenotypic representations. This is a unique feature of MEP representation and it is called *strong implicit parallelism*.

Genes that are not phenotypically expressed may be considered as a special kind of introns (called them here *weak introns*). A weak intron may be phenotypically expressed later during the search process.

MEP has some evident advantages with respect to previously enumerated GP techniques. MEP supplies the shortest linear representation. Moreover, the MEP genome does not contain non-coding sequences.

In this paper MEP technique is used for solving different computationally difficult problems like symbolic regression, discovering game strategy, discovering heuristics for NP-complete problems. The overall conclusion is MEP performs very well for each considered problem.

Symbolic regression represents an easy but interesting application of MEP.

MEP also provides a simple way to discover game strategies. Consider a two-player game. The evaluation of a game configuration is realized using a computer program (a function) that is evolved by MEP algorithm.

In a training stage MEP evolves a computer program (a function) those output is the quality of each possible game configuration. This function describes the MEP strategy used for game playing purposes.

MEP strategy evaluates each game configuration that can be reached from the current state in one move. The best move is selected using this evaluation.

MEP algorithm was tested for Tic-Tac-Toe, a simple to describe but an enough complex game. MEP was able to discover quickly a computer program that plays unbeatable against a perfect player.

One of the most important applications of MEP is discovering heuristics for solving computationally difficult (Nondeterminist Polynomial – Complete (NP-Complete)) problems. Instead of solving a single instance of the problem the aim is here to discover a heuristic that solves the entire class of problems.

In this paper we indicate how MEP can be used to discover a heuristic procedure for solving TSP problem (or Hamiltonian cycle problem). The idea is similar with that used for discovering games strategy. In the proposed approach the Hamiltonian cycle starts with a randomly selected node. Each node reachable from the current node in one step is evaluated using the function (computer program) evolved by MEP algorithm. The best node is chosen and is added to the already founded path. The algorithm stops when the path contains all graph nodes.

During MEP trainin stage graphs having 3 to 50 nodes are considered. Evolved MEP function was tested for graphs having maxim 200 nodes. For each graph edge weights have real values between 0 and 1.

MEP technique is used to learn a function *f* that is directly used for building the optimum path. The corresponding learning process has a remarcable quality: the evolved (learned) heuristic works very well for data sets much larger than the training set.

MEP technique may be used for solving other difficult problems like handwritten character recognition, natural language processing, voice synthesis, automated code generation, forecasting, discovering recurrence relations for dynamic programming, etc.

In this paper MEP technique is compared with GEP using a well-known benchmark problem. The results suggest that MEP algorithm performs substantially better than GEP on the considered examples.

2. Linear GP representations

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

2.1. Grammatical Evolution

Grammatical Evolution (GE) ([2]) uses Backus - Naur Form (BNF) in order to express computer programs. BNF supplies a notation that allows a computer program to be expressed as a grammar.

A BNF grammar consists from terminal and non-terminal symbols. Grammar symbols may be re-written in other terminal and non-terminal symbols.

Each GE individual is a variable length binary string that contains in its *codons* (group of 8 bits) the information to select a production rule from a BNF grammar.

An example excerpt from a BNF grammar is given the following production rules:

S ::= *expr* (0)

| *if-stmt* (1)

| *loop* (2)

These productions rules state that the start symbol S can be replaced (re-written) either by one of the non-terminals *expr*, *if-stmt*, or by *loop*.

The grammar is used in a generative process to construct a program by applying production rules, selected by the genome, beginning from the start symbol of the grammar.

In order to select a GE production rule, the next codon value on the genome is generated and placed in the following formula:

Rule = *Codon_Value* **MOD** *Num_Rules*. (1)

If the next *Codon* integer value is 4, given that we have 3 rules to select from, as in the above example, we get

4 **MOD** 3 = 1.

S will therefore be replaced with the non-terminal *if-stmt*, corresponding to the second production rule.

Beginning from the left hand side of the genome codon integer values are generated and used to select rules from the BNF grammar, until one of the following situations arises:

i. A complete program is generated. This occurs when all the non-terminals in the expression being mapped, are transformed into elements from the terminal set of the BNF grammar.

ii. The end of the genome is reached, in which case the *wrapping* operator is invoked. This results in the return of the genome reading frame to the left hand side of the genome once again. The reading of codons will then continue unless an upper threshold representing the maximum number of wrapping events has occurred during this individual's mapping process. This threshold is currently set to ten events.

iii. In the event that a threshold on the number of wrapping events is exceeded and the individual is still incompletely mapped, the mapping process is halted, and the individual assigned the lowest possible fitness value.

GE uses a steady state replacement mechanism, such that, two parents produce two children the best of which replaces the worst individual in the current population if the child has a greater fitness. Standard variation operators (point mutation, and one point crossover are adopted). GE also employs a *duplication* operator that duplicates a random number of codons and inserts these into the penultimate codon position on the genome.

One of the most important difficulties of GE model is that invalid individuals may appear in system. They are generated by incomplete mapping of individuals.

2.2. Gene Expression Programming

Gene Expression Programming (GEP) [4] 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 function (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 evaluated by:

$$t = (n - 1)\, h + 1, \tag{2}$$

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 î:

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

In this case $n = 2$. If we choose $h = 10$, then 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 tree* encoded by the gene C is depicted in Figure 1. This tree represents the phenotypic transcription of a chromosome having C as its unique gene.

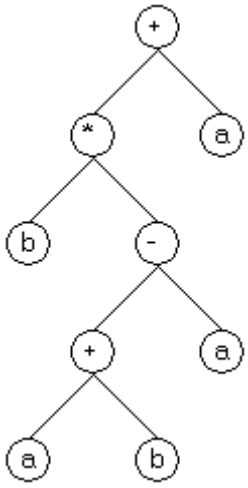

Figure 1. Expression tree of the chromosome with *C* as a unique gene.

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 [4]). 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.

The multigenic chromosome was introduced because it can be happen that the first symbol in a gene to be a terminal symbol and thus a single gene chromosome can not represent a complex expression. As an indirect consequence, if the first symbol of a gene is a terminal then the rest of the gene is unused.

Furthermore, the success rate of GEP increases with the number of genes in the chromosome [4]. 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 [4] 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.

Crossover and mutations change the number of instruction in chromosome. A limit of 256 instructions per chromosome is imposed in order to keep computer programs small and effective.

4. MEP technique

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

4.1. Standard MEP Algorithm

Standard MEP algorithm stars with a randomly chosen population of individuals. Each individual in current population is evaluated using a fitness function that depends by the problem on hand.

A fixed number of the best individuals enter in the next generation (elitism). The mating pool is filled using binary tournament selection. Individuals from mating pool are randomly paired and recombined. By recombination of two parents two offspring are obtained. The offspring are mutated and enter the next generation.

Standard MEP algorithm may be depicted as follows:

```
begin
   Generate Initial Population;
   t = 0;
   Evaluate_Individuals;
   while not Termination_Condition do
       Elitism;
       Selection;
```

```
      Recombination;
      Mutation;
      Evaluate_Individuals;
   endwhile
end
```

4.2. MEP representation

MEP genes are (represented by) substrings of variable length. Number of genes in a chromosome is constant and it represents *chromosome length*. Each gene encodes a terminal or a function symbol. A gene encoding a function includes pointers towards the function arguments. Function parameters 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 are obtained.

Example

Here we use a representation when numbers on the left positions stand for gene labels. Labels do not belong to the chromosome, but they are provided for explanation purposes only.

An example of chromosome is given below:

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

4.3. MEP Phenotypic transcription

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

MEP chromosomes are read downstream starting with the first position. A terminal symbol specifies a simple expression. A function symbol specifies a complex expression (formed by linking the operands specified by the argument positions with the current function symbol).

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

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

$$E_1 = a + b. \quad (3)$$

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

$$E_1 = c + d. \quad (4)$$

Gene 7 indicates the operation * on the operands located at position 3 and 6. Therefore 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 encodes more than one expression. These expressions may also be interpreted as computer programs.

Example 1

Consider the chromosome *C*:

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

Chromosome *C* can not encode a single expression which uses all of the genes. But *C* encodes two expressions:

$E_1 = a + b$, (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.

Each MEP chromosome encodes a number of expressions equal to the chromosome length (number of genes). Expression associated to each chromosome position is obtained by reading the chromosome upstream from the current position.

Example 2

Consider the chromosome:

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

This 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)

$E_7 = (a + b) * (c + d)$. (14)

4.4. 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.

Phenotypic transcription is used for fitness assignment purposes. Sometimes 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* the ability of MEP chromosome to encode several syntactically correct expressions in a chromosome.

4.4. MEP vs. GEP representation

MEP representation of a given function is more compact (shorter) than the corresponding GEP representation. Let us consider the expression.

$$E = a^4 + a^3 + a^2 + a. \quad (15)$$

The function set for describing E is

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

and the terminal set is

$T = \{a\}$.

GEP chromosome encoding the expression E is the linear structure:

+a++aa**a***aaaaaaaaaaaaaa*.

The head length is $h = 13$. This is the shortest possible head length. The total GEP chromosome length is thus 27.

In MEP model the expression E could be represented by the following chromosome:

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

This chromosome uses only 19 symbols.

Compactness of MEP representation is due to two reasons:

a) Code reuse.

It can be seen in the previous example that the term a^2 is represented once in the chromosome and the term a^4 is obtained by pointing twice to the position of a^2. In some situations improvements obtained by code reuse are significant. 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 expression $E = a^8$ is given below:

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

A MEP chromosome uses only $(3n + 1)$ symbols for encoding the expression a^{2^n}. One 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.

b) Within MEP each symbol in a chromosome is used for the phenotypic representation. In GEP, GE and Linear GP representations, a gene usually contains a non-coding sequence.

Some GP techniques, like Linear GP, remove non-coding sequences of chromosome during the search process. As already noted ([4], [3]) 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.

4.4. 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 [7]).

Some experiments for setting tournament size were performed. As can be seen in Experiment 7, binary tournament ($q = 2$) seems to work very good on considered test problems. Binary tournament will be used in all experiments considered in this paper.

4.5. 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.

4.5.1. Recombination

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

4.5.1.1. 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**

5.5.1.2. 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

4.5.1.3. 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

4.5.2. Mutation

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

4.5.2.1. 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. For other genes there is no restriction in symbols changing.

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).

4.5.2.2. 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.

4.5.3. 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 changed into a function symbol (gene 3). Positions of function parameters changed into another ones (gene 6).

4.6. 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.

4.7. MEP complexity

Complexity of MEP algorithm is:

$$O(N \cdot NG), \tag{16}$$

where N is the number of individual in the population and NG is the number of genes in chromosome.

The fitness of an individual may be computed in $O(NG)$ steps by dynamic programming. Thus MEP algorithm does not have a higher complexity than other GP - techniques that encode a single computer program in each chromosome.

5. MEP technique applied for solving symbolic regression problems

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

5.1. Standard MEP for symbolic regression

The aim of symbolic regression is to discover a function that satisfies a set of fitness cases. A set of fitness cases is depicted in Table 1.

x	$f(x)$
2.81	95.2425
6	1554
7.043	2866.5485
8	4680
10	11110
11.38	18386.0340
12	22620
14	41370
15	54240
20	168420

Table 1. A set of fitness cases for symbolic regression.

A function that satisfies the set of fitness cases given in Table 1 is:

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

We want to discover a function that satisfies the set of fitness cases with a given precision ε.

Let

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

be the set of functions and

$T = \{x\}$,

be a set of terminals.

Consider the problem precision is

$\varepsilon = 0.01$.

5.1.1. Experiment 1

In this experiment the success rate of the standard MEP algorithm is analyzed. A comparison with the results supplied by GEP technique is realized. The chromosome length is gradually increased. Algorithm parameters are given in Table 2.

In this paper we call *elitism size* the number of best individuals copied without modification in the next population.

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

Number of runs	100

Table 2. Algorithm parameters for 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.

The first position of MEP chromosomes is always a terminal symbol. Thus the total number of symbols in the chromosome is given by the formula:

$$Number\_of\_Symbols = 3 \cdot Number\_of\_Genes - 2. \quad (18)$$

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. Parameters of GEP algorithm were similar with those of MEP.

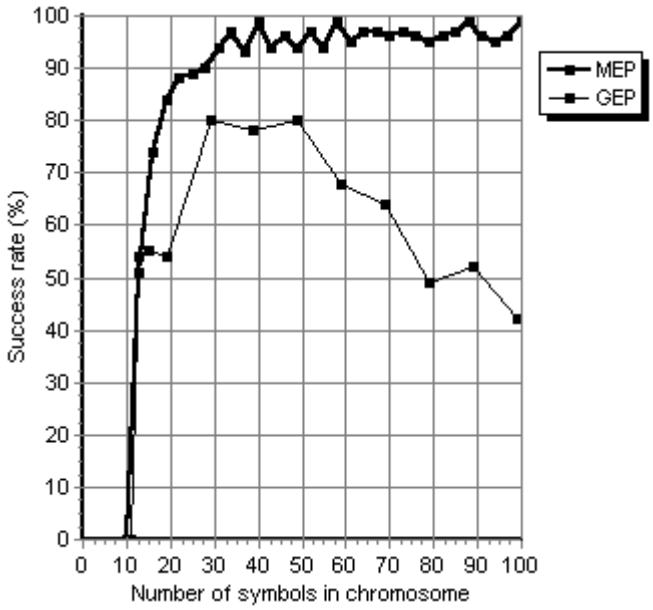

Figure 2. Success rate of MEP and GEP algorithms. One may note that the success rate of GEP increases up to 80% and then decreases. The success rate of the MEP algorithm increases with chromosome length and never decreases toward very low values.

The results of this experiment emphasis that, for considered symbolic regression problem, MEP performs better than GEP for each chromosome length.

5.1.2. Experiment 2

From Experiment 1 we may infer that for 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.

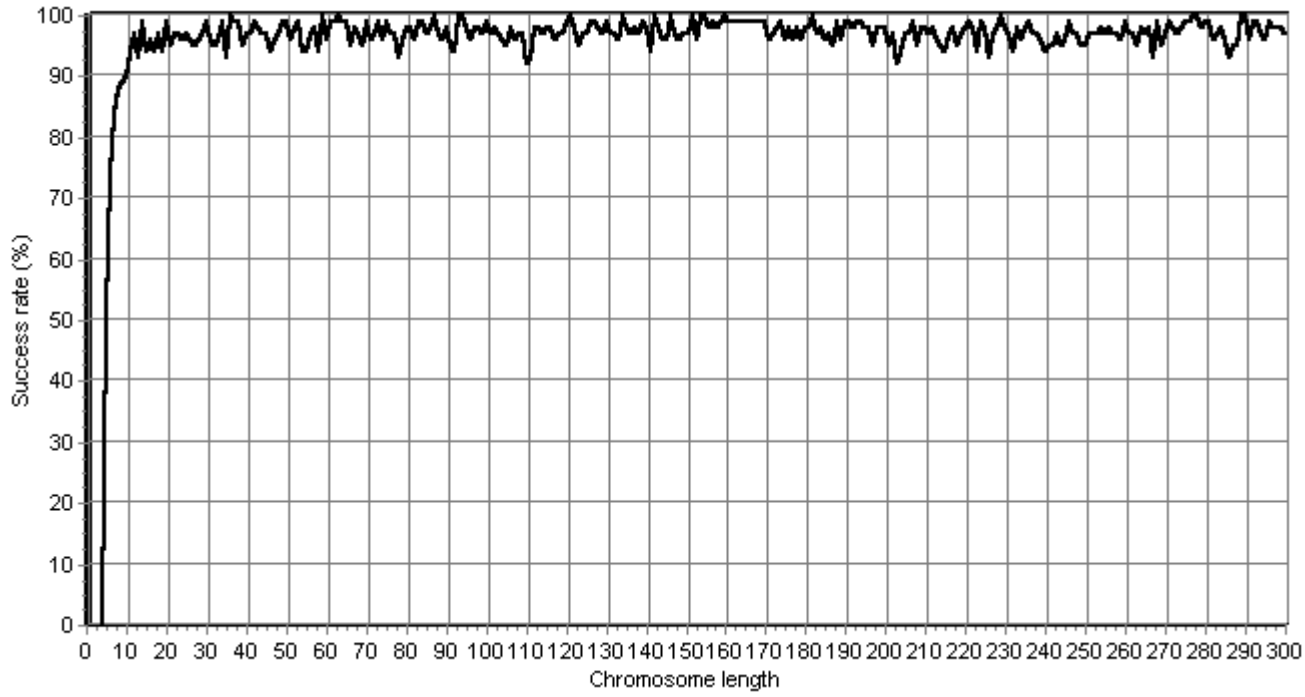

Figure 3. Success rate of MEP algorithm. One may note that after that the chromosome length becomes 10, the success rate never decrease under 90%.

5.1.3. Experiment 3

In this experiment the MEP success rate for different mutation probabilities is analyzed. Three mutation situations: one mutation / chromosome, two mutations / chromosome and three mutations / chromosome, are considered.

Algorithm parameters are given in Table 3

Population size	30
Number of generations	50
Crossover type	One-point-crossover
Selection	Binary tournament
Elitism size	1
Number of runs	100

Table 3. Algorithm parameters for Experiment 3.

Figure 4 illustrate how success rate of MEP algorithm depends on chromosome length when different numbers of mutations per chromosome are allowed.

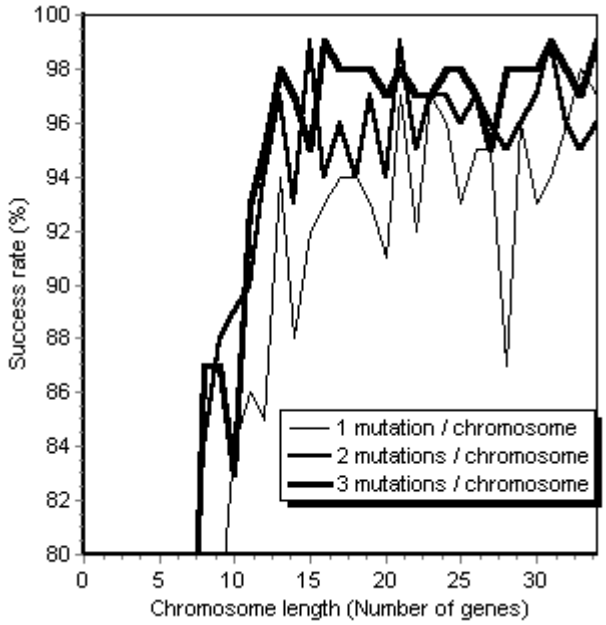

Figure 4. The success rate of MEP algorithm when different number of mutations per chromosome is used. The success rate seems to be the best when 3 mutations per chromosome are used.

5.1.4. Experiment 4

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 4.

Population size	30
Number of generations	50
Chromosome length	40 genes
Mutation	2 genes / chromosome
Crossover type	One-point-crossover
Selection	Binary tournament
Elitism size	1
Number of runs	100

Table 4. Algorithm parameters for Experiment 4.

Results for this Experiment are depicted in Figure 5.

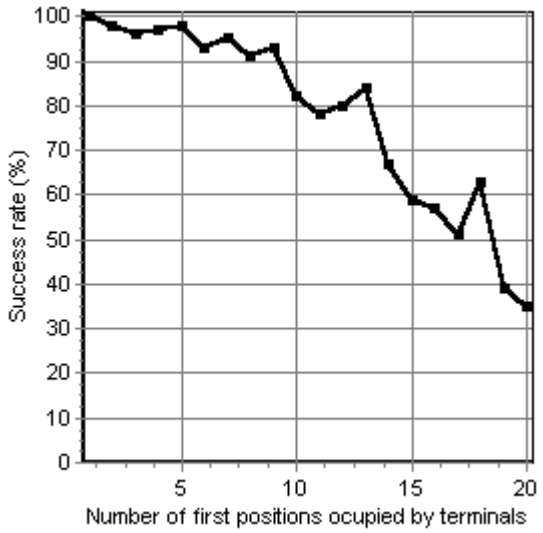

Figure 5. Success rate of MEP algorithm when the number of first positions occupied by terminal symbols varied. Success rate has a maximum value when only the first position in chromosome is compulsory filled by terminal symbols. After this value the success rate gradually decreases.

From Figure 5 we may infer that the best results are obtained when only the first position is compulsory a terminal symbol.

Imposing terminal symbols to other initial positions seems to generate performance decreasing. We may infer that this loss of performance is due to the reduction of positions encoding operators.

5.1.5. Experiment 5

In this experiment the relationship between the success rate and the population size is analyzed. Algorithm parameters for this experiment are given in Table 5.

Number of generations	50
Chromosome length	17 genes
Mutation	2 genes / chromosome
Crossover type	One-point-crossover
Selection	Binary tournament
Elitism size	1
Number of runs	100

Table 5. Algorithm parameters for Experiment 5.

Experiment results are given in Figure 6.

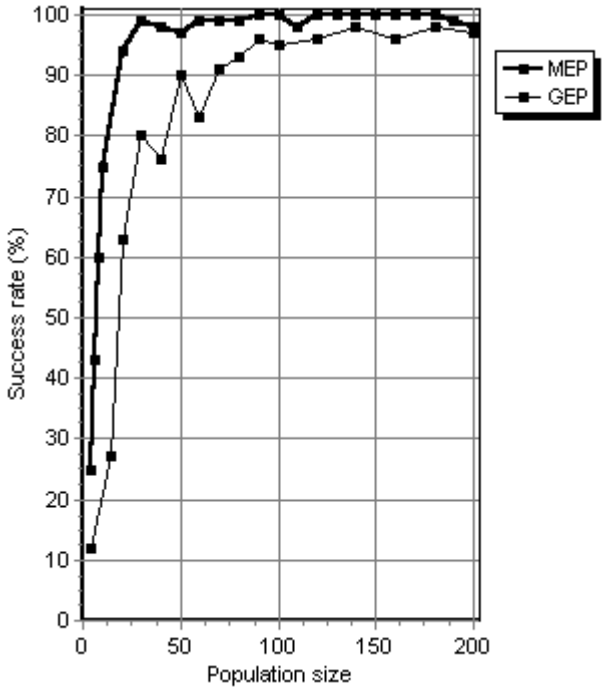

Figure 6. Success rate of MEP algorithm. Population size is varied between 2 and 200. Optimal population size is 30. The success rate for this value is 99%. This evaluation suggests that small populations can supply very good results.

For the considered problem (symbolic regression) and for the MEP algorithm parameters given in Table 5 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 similar 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 the MEP success rate is 98%.

5.1.4. Experiment 6

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

Algorithm parameters are given in Table 6.

Population size	30
Chromosome length	27
Mutation	2 genes / chromosome
Crossover type	One-point-crossover
Selection	Binary tournament
Elitism size	1
Number of runs	100

Table 6. Algorithm parameters for Experiment 6.

Experiment results are given in Figure 6.

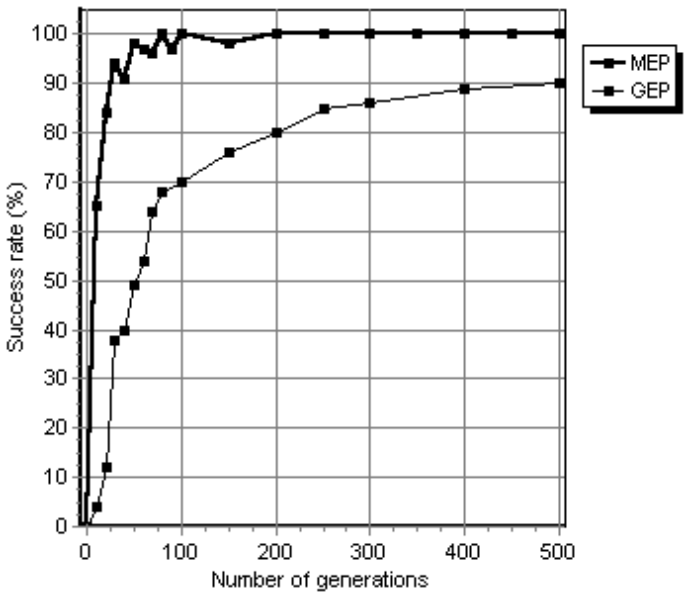

Figure 7. The relationship between the success rate of MEP algorithm and the number of generations used in the search process.

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

Moreover, according to [4], GEP algorithm reaches the success rate of 90% only when the number of generations reaches 500.

On the considered domain GEP success rate never reaches 100% while MEP success rate is stabilized to 100% since generation 200.

5.1.7. Experiment 7

In this experiment the relationship between the success rate of MEP algorithm and the number of individuals copied without modifications (clones) in the next generation by the effect of elitist selection is analyzed.

Algorithm parameters are given in Table 7.

Population size	50
Number of generations	50
Chromosome length	10 genes
Mutation	2 genes / chromosome
Crossover type	One-point-crossover
Selection	Binary tournament
Number of runs	100

Table 7. Algorithm parameters for Experiment 7.

In Figure 8 the relationship between the success rate and the elitism size is depicted.

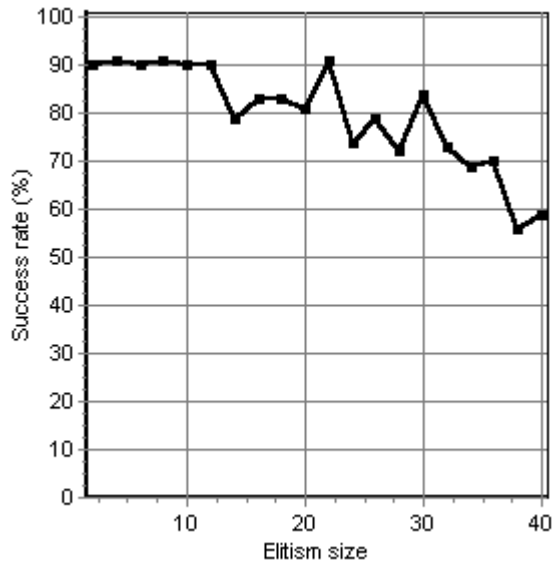

Figure 8. The success rate of MEP algorithm when different elitism sizes are used.

7.1.8. Experiment 8

Selection is an important operator of MEP algorithm. In previous experiments binary tournament selection was used. In this experiment a *q*-tournament procedure is considered: *q*-individuals are randomly chosen and the best of them is passed in the mating pool. Relationship between success rate and tournament size q, $q \geq 2$, is investigated.

Algorithm parameters are given in Table 8.

Population size	30
Number of generations	50
Chromosome length	10 genes
Mutation	2 genes / chromosome
Crossover type	One-point-crossover
Elitism size	1
Number of runs	100

Table 8. Algorithm parameters for Experiment 8.

In Figure 9 the relationship between MEP success rate and the tournament size (q) is depicted.

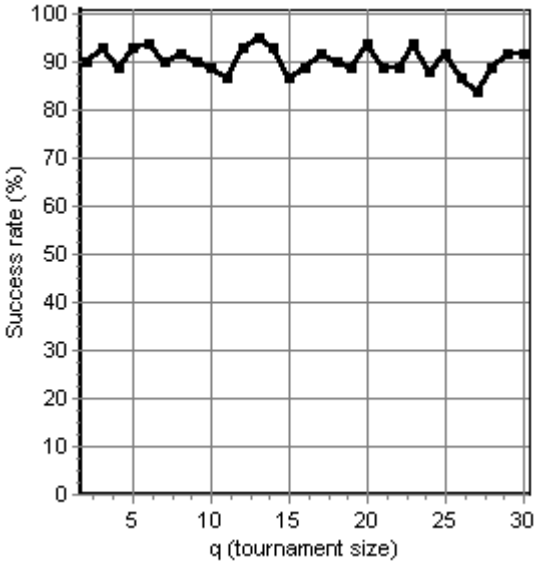

Figure 9. The relationship between success rate and q - tournament selection size. There is no evidence that a particular value for q is much better than others. For the speed purposes binary tournament will always be used.

Increasing tournament size does not generate performance improving. For this reason binary tournament is highly recommended and will be used in all experiments in this paper.

5.1.9. Experiment 9

In the last experiment the relationship between success rate and the crossover type is analyzed. Three kinds of crossover operators are considered: one-point recombination, two-point recombination and uniform recombination. Algorithm parameters used in this experiment are given in Table 9.

Population size	30
Number of generations	50
Chromosome length	20
Mutation	2 genes / chromosome
Selection	Binary tournament
Elitism size	1
Number of runs	100

Table 9. Algorithm parameters for Experiment 9.

The success rate of MEP algorithm is given in Table 10.

Recombination type	Success rate
One-point recombination	94%
Two-point recombination	96%
Uniform recombination	95%

Table 10. The success rates of MEP algorithm when different recombination operators are used. Two-point recombination seems to be the best for this problem, but all three algorithms have similar success rates.

Two-point recombination seems to be the best for this problem, but all three algorithms have similar success rates.

5.2. Weighted MEP for symbolic regression problem

For a given problem there are 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 problem of symbolic regression.

5.2.1. Numerical experiments for symbolic regression problem

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.

5.2.2. Experiment

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 11.

Population size	30
Number of generations	20
Chromosome length	20
Mutation	2 genes / chromosome
Selection	Binary tournament
Elitism size	1
Number of runs	100

Table 11. Algorithm parameters for weighted MEP.

Figure 10 depicts the relationship between the success rate and the chromosome length when different weights are assigned to the operators (function symbols).

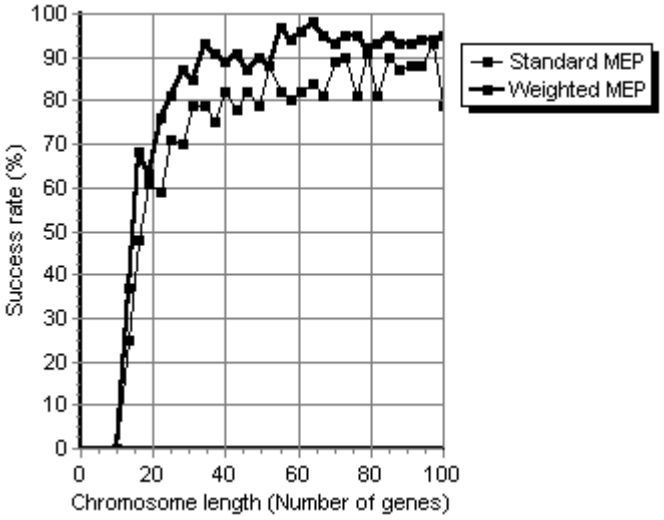

Figure 10. Success rate of MEP algorithm when operators are weighted. Operators + and * have double chance to appear in chromosome. The success rate is higher than that of standard MEP.

One may note that success rate is higher when operators are weighted. These improvements are correlated with a particular weight system based on observations on the problem.

6. Discovering game strategies with MEP

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

Koza [1] 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) [1].

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.

6.1. TTT game description

In Tic-Tac-Toe there are two players and a 3 x 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 first player moves first.

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 [5]. This algorithm will be reviewed in the next section.

6.2. Fogel'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 [5] the best obtained neural network is able to play to win or draw with a perfect play strategy.

6.3. 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. 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 11).

0	1	2
3	4	5
6	7	8

Figure 11. 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 12.

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

Table 12. 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 12 the fitness of the best individual in the population and average fitness during the search process are depicted.

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.

Proposed technique can also generate a function that forces the win whenever it is possible.

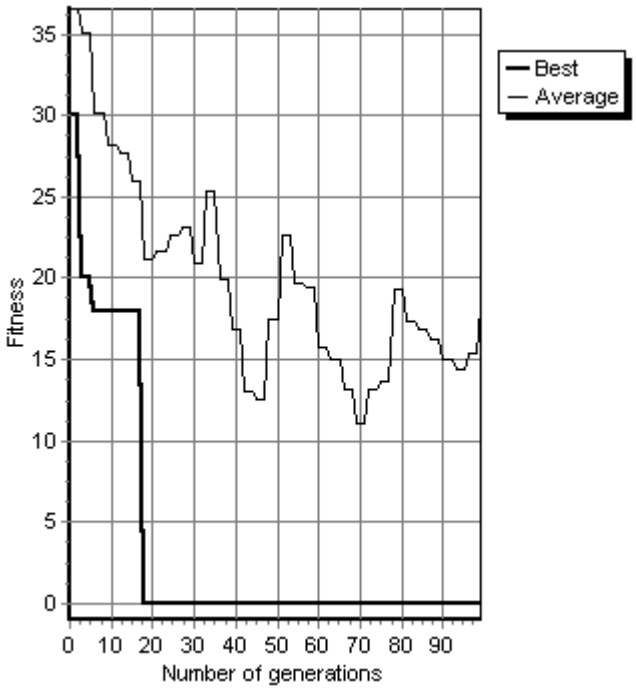

Figure 12. Fitness values of the best individual and average fitness in population during the search process. An individual representing a non-loosing strategy appears in the population at generation 17.

6.4. 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 is an algorithm faster than previously shown heuristic.

6.5. Applying MEP for generating complex game strategies

For complex games a different approach need to be used since the moves for the second player can not be simulated by an "all-possibilities" procedure.

One possibility is to use for training (evolving a game strategy) a heuristic procedure that plays 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 function using a coevolutionary algorithm. This approach seems to offer the most spectacular results. The population must develop intelligent behavior based only on internal competition.

7. Discovering heuristics using MEP

In this section we address the problem of discovering heuristics that can solve classes of computationally difficult search problems.

We propose a general discovering technique based on MEP approach. The idea is to find a function that quickly generates the solution of the problem. The function is evolved in a training stage using test cases from a certain class of problems. Evolved function is applied for solving problems in the given class.

In this section we focus on discovering of a heuristic for solving Traveling Salesman Person (TSP) problem or Hamiltonian cycle problem.

The TSP problem may be stated as follows:

Given a set $C = \{c_1, c_2,\ldots, c_n\}$ of cities, and a distance $d(c_i, c_j) \in Z^+$ for each pair of cities $c_i, c_j \in C$, $d(c_i, c_j) = d(c_j, c_i)$. The tour of all cities in C having total length minimum is needed.

Many heuristics have been proposed for this problem [6].

7.1. Evolving heuristics for TSP problem

The method described in what follows is intended for discovering of a heuristic algorithm for solving TSP problem rather than solving the problem for a particular instance.

Solving TSP problem is achieved in two stages. In the first stage MEP algorithm evolves a function that gives a way to choose graph vertices in order to obtain a minimal Hamiltonian cycle. The fitness of a function is evaluated by applying it on several randomly chosen graphs. Evolved function represents a heuristic procedure that solves meaningful instances of TSP problem.

Graph nodes are labeled with integer numbers in the set $\{0,\ldots,N\text{-}1\}$, where N is the nodes number of the graph.

We want to evolve a function f that iteratively solves the TSP problem. That means that using f nodes are added one by one to the already discovered path. Since we want to discover a cycle path we may always start with the node labeled zero.

Let us suppose that the path is filled with some nodes and the last visited node is y_1. We have to find the next node that will be added to the path. In this aim all unvisited nodes are considered. Let us denote by y_2 the next visited node.

A natural way for defining the set of terminals was to consider node labels as terminals, i.e.

$T = \{0,.., N\text{-}1\}$.

But this approach leads to some difficulties since our aim is to evolve (or learn) and test the functions for graphs having different number of nodes. For preserving generality we consider a special terminal set.

The considered set T of terminals (for evolving path function) consists of:

(i) $d\_y_1\_y_2$ - distance between the graph nodes y_1 and y_2,

(ii) $min\_g\_y_1$ ($min\_g\_y_2$) - the minimum distance from the nodes y_1 (y_2) to unvisited nodes,

(iii) $sum\_g\_y_1$ ($sum\_g\_y_2$) - the sum of the distances between nodes y_1 (y_2) and unvisited nodes,

(iv) $max\_g\_y_1$ ($max\_g\_y_2$) - the maximum distance from the nodes y_1 (y_2) to unvisited node.

The set T of terminals (function variables) is thus:

$T = \{d\_y_1\_y_2, min\_g\_y_1, min\_g\_y_2, max\_g\_y_1, max\_g\_y_2, sum\_g\_y_1, sum\_g\_y_2\}$.

Let us remark that members of T are not actual terminals (in the standard acception). For this reason we may call members of T as *instantied* (or *intermediated*) *nonterminals*. The use of instantiated non-terminals may offer flexibility and robustness to the representation and to the search process.

The set T of terminals (or instantied nonterminals) is chosen in such way to be independent of the number of graph nodes.

In order to evolve a MEP function for solving TSP problem we may consider the following set of functions symbols:

$F = \{+, -, /, *, cos, sin\}$.

The node y_2 that generates the lowest output of evolved function *f* is chosen to be the next node of the path.

Example

We may consider a path function *f* represented by the linear structure:

1: $d\ y_1, y_2$
2: $min\_g\ y_1$
3: + 1, 2
4: * 2, 3
5: $sum\_g\ y_2$

7.2. Fitness assignment

The fitness of function (computer program) *f* is the sum of the Hamiltonian cycles length of considered graphs. Thus the fitness is to be minimized.

7.3. Evolved function complexity

The complexity of the evolved heuristic is:

$$O(N^2)\cdot O(f), \tag{22}$$

where *N* is the number of graph nodes and $O(f)$ is the complexity of the evolved function *f*.

7.4. A numerical experiment

Let us denote by G_i the set of graphs having maximum i nodes.

MEP algorithm was trained using the set G_{50} (i.e. graphs having 3 to 50 nodes). Evolved MEP function was tested for graphs in G_{200} (i.e. graphs having maxim 200 nodes). For each graph edge weights were real values between 0 and 1.

A sample of 20 randomly chosen graphs was considered in the training stage (evolving MEP function).

Algorithm parameters are given in Table 13.

Population size	50
Number of generations	150
Chromosome length	20 genes
Mutation	3 mutations / chromosome
Selection	Binary tournament
Elitism size	1

Table 13. Algorithm parameters for Experiment 1.

The evolution of the best individual fitness and the average fitness in the population are depicted in Figure 13.

Best individual fitness stabilizes since generation 110. Average fitness has small oscillations.

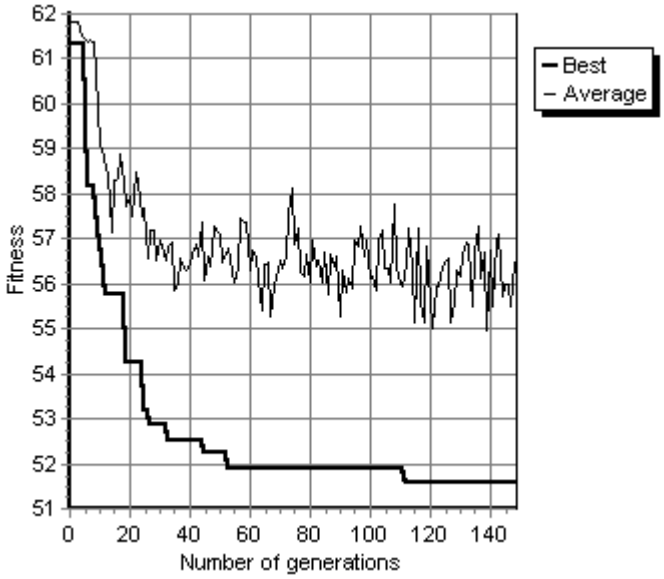

Figure 13. The evolution of fitness and the average fitness during the search process.

MEP technique is used to learn a function *f* that is directly used for building the optimum path. The corresponding learning process has a remarkable quality: the evolved (learned) heuristic works very well on data sets significantly larger than the training set. In our example the training set G_{50} is significantly smaller than the set G_{200} used for testing.

7.2. MEP vs. other TSP techniques

A path function evolved by the MEP algorithm in the previous experiment is:

$$f = 2 * d(y_1, y_2) + sum\_g(y2) * (d((y_1, y_2)$$

$$-(cos(min\_g(y_1)))) \quad (23)$$

The obtained formula was tested for 10000 randomly generated graphs from G_{200}.

For considered graphs the following quantities are also computed: the cycle length obtained by Nearest Neighbor (NN) algorithm [6], and the cycle length obtained by random walk technique.

In Table 14 the performance of MEP and NN algorithms are depicted.

Graphs	Number of graphs
Graphs for which MEP generates a cycle shorter that the cycle obtained with NN algorithm.	6965
Graphs for which MEP generates a cycle longer that the cycle obtained with NN algorithm.	3035

Table 14. TSP problem: MEP vs. NN algorithm.

In Table 15 the performances of MEP and random walk algorithms are depicted.

Graphs	Number of graphs
Graphs for which MEP generates a cycle shorter that the cycle obtained with random walk algorithm.	9919
Graphs for which MEP generates a cycle longer that the cycle obtained with random walk algorithm.	81

Table 15. TSP problem: MEP vs. random walk algorithm.

Experimental results show that computer program evolved by MEP algorithm performs better than NN algorithm.

7.3. Solution readability

For a particular TSP problem the path function f_1 evolved by MEP for TSP:

$$f_1 = d(y_1, y_2) - cos(d(y_1, y_2)). \quad (24)$$

This function is very similar to the function f_2 involved by the NN algorithm:

$$f_2 = d(y_1, y_2). \quad (25)$$

Expressions F_1 and F_2 are very readable but most of the evolved MEP expressions are not so readable.

Very complex MEP evolved expressions could be interpreted as the result of the hidden paths of evolution. However, even the most intricate expressions could be of very high practical interest.

7.4. Comments on MEP technique for discovering TSP heuristics

MEP technique was also tested for graphs having up to 1000 nodes. The performances of the obtained expressions were similar or better with the performances of NN algorithm.

An improvement can be done by allowing more function symbols to appear in chromosome.

Further improvement may be obtained by increasing the chromosome length. In this case the complexity of the evolved formula increases, but the performances of the obtained heuristic could be significantly better.

Since the problem computing Hamiltonian cycle is NP-Complete we cannot realistically expect that an absolutely correct evolved expression has lower complexity. Even if the expectation that evolution could overcome the exponential complexity problem is very exciting ([8]), we cannot realistically consider that this direction may lead to important theoretical or practical results.

Searching for heuristics that give near optimal solutions seems to be a more realistic approach. Using MEP techniques an easy way to evolve heuristics has been proposed.

Further efforts will be focalized in obtaining better heuristics for other NP-complete problems. Using highly parallel implementations of MEP technique and taking advantages of distributed programming the obtained heuristics could be improved.

8. Conclusions and further work

In this paper a new evolutionary paradigm is considered. Based on this paradigm a new evolutionary technique called MEP is proposed. 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 has several advantages over other evolutionary techniques. One of the most important features is that MEP individuals may encode more expressions (computer programs). The best expression is selected as phenotypic representation of the chromosome. Phenotypic representation is considered in the fitness assignment process.

MEP individuals do not contain non-coding sequences, because each gene is involved in at least one computer program (expression).

For the problem considered in this paper MEP algorithm performs better than similar evolutionary techniques (particularly GEP).

Further research will focus on applying MEP for solving some difficult problems like:

- discovering complex games strategy for highly difficult games like chess and GO,
- evolutionary theorem proving,
- discovering heuristics for other NP-complete problems.

References

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

[2] C. Ryan, J.J. Collins, M. O'Neill, *Grammatical Evolution: Evolving Programs for an Arbitrary Language*. Lecture Notes in Computer Science 1391, Proceedings of the First European Workshop on Genetic Programming, (Springer-Verlag, 1998, pp 83-95).

[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] C. Ferreira, *Gene Expression Programming: a New Adaptive Algorithm for Solving Problems*, (Complex Systems, **13**(2):87-129, 2001).

[5] K. Chellapilla, D. B. Fogel, *Evolution, Neural Networks, Games and Intelligence*, (Proc. IEEE. **87**(9), 1471-1496).

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

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

[8] R. Penrose *Shadows of the Mind*. (New York: Oxford University Press, 1994).