10

Improving Multi Expression Programming: An Ascending Trail from Sea-Level Even-3-Parity Problem to Alpine Even-18-Parity Problem

Mihai Oltean

Department of Computer Science,
Faculty of Mathematics and Computer Science,
Babeş Bolyai University, Kogalniceanu 1, 3400 Cluj-Napoca, Romania,
`moltean@cs.ubbcluj.ro`, `www.cs.ubbcluj.ro/~moltean`

Multi Expression Programming is a Genetic Programming variant that uses a linear representation of individuals. A unique feature of Multi Expression Programming is its ability of storing multiple solutions of a problem in a single chromosome. In this chapter, we propose and use several techniques for improving the search performed by Multi Expression Programming. Some of the most important improvements are Automatically Defined Functions and Sub-Symbolic node representation. Several experiments with Multi Expression Programming are performed in this chapter. Numerical results show that Multi Expression Programming performs very well for the considered test problems.

10.1 Introduction

Multi Expression Programming (MEP)[1] [11, 12, 13] is a new and very efficient technique that may be used for solving difficult real-world problems. A unique feature of MEP is its ability of storing multiple solutions of a problem in a single chromosome. As shown in [11], this feature does not increase the complexity of the decoding process when compared to other Genetic Programming (GP) [7, 8] variants that store a single solution in a chromosome (such as Gene Expression Programming (GEP) [5], Genetic Algorithms for Deriving Software (GADS) [16], Grammatical Evolution (GE) [14], Cartesian Genetic Programming (CGP) [10]).

The MEP technique has been efficiently used for solving symbolic regression problems [11] and even-parity problems [13].

[1] MEP source code is available at www.mep.cs.ubbcluj.ro.

Parity problems arise in many practical applications related to the information technology, especially when data need to be safely transmitted over a network. According to [7] the Boolean even-parity functions are the most difficult Boolean functions to detect via a blind random search. Due to this reason, the ability of the evolutionary algorithms of performing an efficient search in the solutions space can be tested using this problem as a benchmark.

In [13], the MEP has been used for solving even-3 and even-4-parity problems. In this chapter we propose and use several techniques for improving the search performed by Multi Expression Programming. Some of these techniques are:

(i) Automatically Defined Functions (ADFs) [7].
(ii) Sub-Symbolic Node Representation [18].

Numerical experiments performed in this chapter include the use of MEP for solving the even-parity instances from even-3 up to even-18-parity.

MEP without ADFs was able to solve (using a reasonable population and within a reasonable timeframe) up to even-5-parity problem. When Automatically Defined Functions are employed a considerable improvement is obtained, allowing us to evolve a solution for up to even-8-parity problem. More improvements are done when a Sub-Symbolic node representation was employed.

Results of the numerical experiments are compared to those provided by Genetic Programming [7, 8, 18]. It can be easily seen that Multi Expression Programming outperforms Genetic Programming with more than one order of magnitude. Note that a perfect comparison between MEP and GP cannot be made due to the incompatibility of respective representations.

The chapter is organized as follows. In section 10.2 the Even-Parity problem is described. The Multi Expression Programming technique is briefly described in section 10.3. The metrics used to assess the performance of the MEP algorithm are described in section 10.4. Several numerical experiments with MEP for solving the even-3, even-4 and even-5-parity problems are performed in section 10.5. Automatically Defined Functions for MEP are introduced in section 10.6. Several numerical experiments with MEP and ADFs are performed in section 10.7. The sub-symbolic node representation and the smooth operators are introduced in section 10.8. Numerical experiments with MEP and sub-symbolic node representation are performed in section 10.9. Conclusions and the further work directions are suggested in section 10.10.

10.2 Problem Statement

Our aim is to find a Boolean function that satisfies a set of fitness cases. The particular function that we want to find is the Boolean even-parity function. This function has k Boolean arguments and it returns **T** (**True**) if an even number of its arguments are **T**. Otherwise the even-parity function returns **F**

(**False**) [7, 18]. According to [7] the Boolean even-parity functions appear to be the most difficult Boolean functions to detect via a blind random search.

In applying a Genetic Programming technique (particularly Multi Expression Programming) to the even-parity function of k arguments, the terminal set T consists of the k Boolean arguments d_0, d_1, d_2, ... d_{k-1}.

The function set F usually consists of four two-argument primitive Boolean functions (also called gates [9]): AND, OR, NAND, NOR [7, 8]. Using this set we can obtain a solution for small instances of the even-parity problem. Genetic Programming with Automatically Defined Functions has obtained a solution for up to even-11-parity problem using a reasonable population size. If we extend this set by including other Boolean functions (such as EQ and XOR) we can obtain solutions for larger instances. For instance, in [18] Genetic Programming using an extended set of function symbols has been used for solving up to even-22-parity problems. Note that in this case a parallel variant of GP was used on a network of computers structured in a client-server architecture.

The set of fitness cases for this problem consists of the 2^k combinations of the k Boolean arguments. The fitness of an MEP chromosome is the sum, over these 2^k fitness cases, of the Hamming distance (error) between the returned value by the MEP chromosome and the correct value of the Boolean function. Since the standardized fitness ranges between 0 and 2^k, a value closer to zero is better (the fitness is to be minimized).

10.3 Multi Expression Programming

In this section the *Multi Expression Programming* (MEP) [11] paradigm is briefly described.

10.3.1 Individual Representation

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

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

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

Example. We employ a representation where the numbers on the left positions stand for gene labels (or memory addresses). Labels do not belong to the chromosome, they are provided here only for explanation purposes.

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

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

10.3.2 Decoding MEP Chromosome and Fitness Assignment

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

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

For instance, genes 1, 2, 4 and 5 in the previous example encode simple expressions formed by a single terminal symbol. These expressions are $E_1 = a$, $E_2 = b$, $E_4 = c$ and $E_5 = d$. Gene 3 indicates the operation $+$ on the operands located at positions 1 and 2 of the chromosome. Therefore gene 3 encodes the expression $E_3 = a + b$. Gene 6 indicates the operation $+$ on the operands located at positions 4 and 5. Therefore gene 6 encodes the expression $E_6 = c + d$. Gene 7 indicates the operation $*$ on the operands located at position 3 and 6. Therefore gene 7 encodes the expression $E_7 = (a+b)*(c+d)$, wherein E_7 is the expression encoded by the whole chromosome.

There is neither practical nor theoretical evidence that one of these expressions is better than the others. Moreover Wolpert and McReady [20, 21] proved that we cannot use the search algorithm's behavior so far for a particular test function to predict its future behavior on that function. Thus we cannot choose one of the expressions (let us say expression E_7) to store the output of the chromosome. Even this expression proves to be useful for the first 10 generations we cannot guarantee that it will be the best option for all generations.

This is why each MEP chromosome is allowed to encode a number of expressions equal to the chromosome length. Each of these expressions is considered as being a potential solution of the problem.

The value of these expressions may be computed by reading the chromosome top down. Partial results are computed by Dynamic Programming [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. 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:

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

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

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

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

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

10.3.3 Genetic Operators

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

Crossover

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

Example. Let us consider the two parents C_1 and C_2 given in Table 10.1. The two offspring O_1 and O_2 are obtained by uniform recombination as shown in Table 10.1.

Mutation

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

Table 10.1. MEP uniform recombination.

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

Example. Consider the chromosome C given in Table 10.2. If the boldfaced symbols are selected for mutation, an offspring O is obtained as given in Table 10.2.

Table 10.2. MEP mutation.

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

10.3.4 MEP Algorithm

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

10.4 Assessing the Performance of the MEP Algorithm

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

(i) The relationship between the success rate and the number of genes in a MEP chromosome,
(ii) The relationship between the success rate and the size of the population used by the MEP algorithm,
(iii) The computational effort.

The success rate is computed using the equation (10.1).

$$Success\,rate = \frac{The\,number\,of\,successful\,runs}{The\,total\,number\,of\,runs}. \tag{10.1}$$

Another method used to assess the effectiveness of an algorithm, has been suggested by Koza [7]. The method consists of calculating the number of chromosomes, which would have to be processed to give a certain probability of success. To calculate this figure one must first calculate the cumulative probability of success $P(M, i)$, where M represents the population size, and i the generation number. The value $R(z)$ represents the number of independent runs required for a probability of success (given by z) at generation i. The quantity $I(M, z, i)$ represents the minimum number of chromosomes which must be processed to give a probability of success z, at generation i. The formulae are given by the equations (10.2), (10.3) and (10.4). *Ns(i)* represents the number of successful runs at generation i, and N_{total}, represents the total number of runs. Note that when $z = 1.0$ the formulae (10.3) and (10.4) are invalid (all runs successful). In the tables and graphs of this chapter z takes the value 0.99.

$$P(M,i) = \frac{Ns(i)}{N_{total}}. \tag{10.2}$$

$$R(z) = ceil\left\{\frac{\log(1-z)}{\log(1-P(M,i))}\right\}. \tag{10.3}$$

$$I(M,i,z) = M \cdot R(z) \cdot i. \tag{10.4}$$

Another important issue is related to the number of function evaluations performed by the considered techniques (MEP and GP in our case). Due to its special Multi-Expression ability MEP performs more function evaluations than GP (considering the same parameters for both algorithms). But, note that 1 function evaluation performed by MEP is not equivalent with 1 function evaluation performed by GP. MEP and GP have the same complexity for the process of decoding the individuals (that is $O(NG)$, where NG is the number of genes). MEP encodes NG solutions in a chromosome whereas GP encodes 1 solution in a chromosome. Thus, the complexity of performing 1 function evaluation is $O(1)$ for MEP and $O(NG)$ for GP. This is why we calculate the computational effort for both MEP and GP using the same formula 10.4 without taking into account the number of genes in a MEP chromosome.

10.5 Numerical Experiments

In this section we perform several experiments with standard MEP for solving several instances of the even-parity problem. General parameter settings for MEP are given in Table 10.3.

Table 10.3. General parameters of the MEP algorithm for solving even-parity problems.

Parameter	Value
Number of generations	51
Mutation probability	0.2
Crossover type	Uniform
Crossover probability	0.9
Selection	q-tournament (q = 10% of the Population size)
Function set	$F = \{\text{AND, OR, NAND, NOR}\}$

For reducing the chromosome length we keep all the terminals on the first positions of the MEP chromosomes. We also increased the selection pressure by using larger values (usually 10% of the population size) for the tournament sample.

Even-3-parity

The even-3-parity problem has three Boolean inputs and one Boolean output. The number of fitness cases is $2^3 = 8$. The relationship between the success rate and the number of genes in a chromosome and the population size is analyzed for this problem.

A population of 100 individuals has been used when the relationship between the success rate and the chromosome length has been analyzed. Chromosomes of 100 genes have been used for analyzing the relationship between the success rate and the population size. Other parameters of the MEP algorithm are given in Table 10.3. Results are depicted in Fig. 10.1.

Fig. 10.1 shows that MEP is able to solve very well this problem. A population of 240 individuals each having 100 genes (see Fig. 10.1 right side) or a population of 100 individuals with 200 genes (see Fig. 10.1 left side) is sufficient to yield a 100% probability of success GP used [7] a population of 4000 individuals in order to achieve a 100% probability of success for this problem.

The shortest evolved circuit implementing the even-3-parity problem has 6 gates. One of the evolved circuits is depicted in Fig. 10.2.The minimum computational effort required to solve this problem is 6840 and it has been obtained at generation 11 using a population of 40 individuals with 100 genes each.

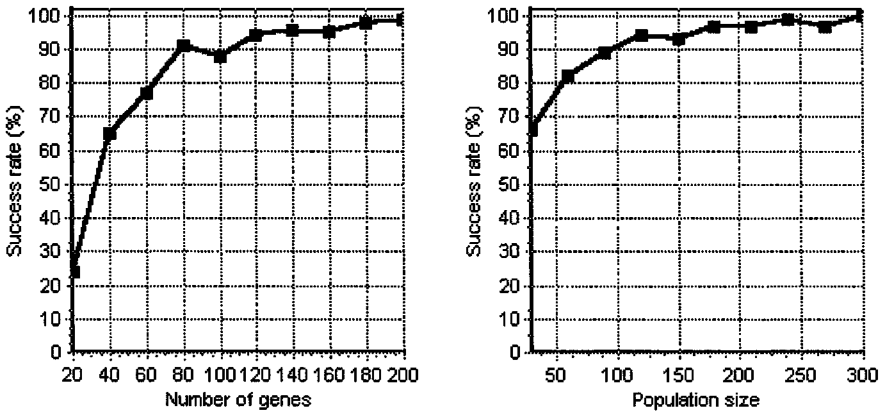


Fig. 10.1. The relationship between the success rate and the chromosome length (left side) and the population size (right side). Results are averaged over 100 runs.

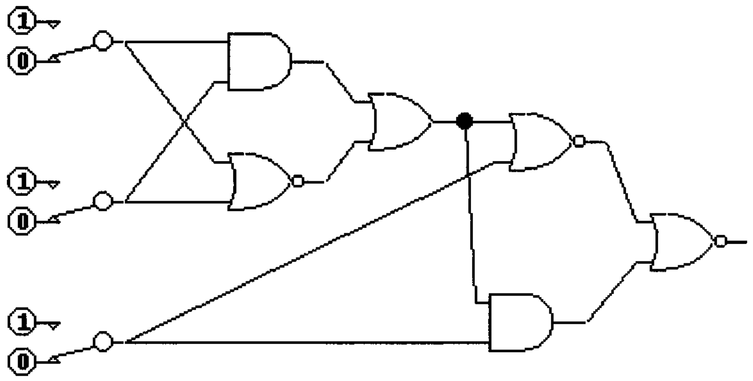
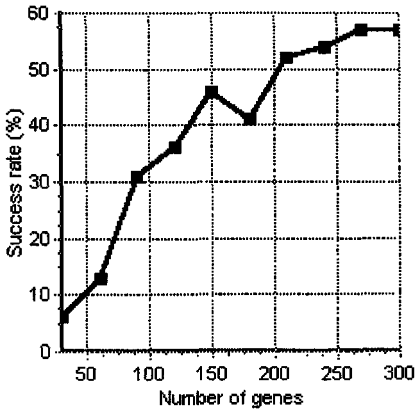


Fig. 10.2. A circuit for the even-3-parity problem.

Even-4-parity

In this experiment, the relationship between the number of genes in a chromosome and the success rate is analyzed for the even-4-parity problem. A population of 400 individuals has been used when the relationship between the success rate and the chromosome length has been analyzed. Chromosomes having 200 genes have been used for analyzing the relationship between the success rate and the population size. Other parameters of the MEP algorithm are given in Table 10.3. Results are depicted in Fig. 10.3.

Fig. 10.3 shows that MEP performs very well on the considered test problem. A population of 200 individuals each having 180 genes is sufficient for yielding a success rate of 42% (see Fig. 10.3 left side).



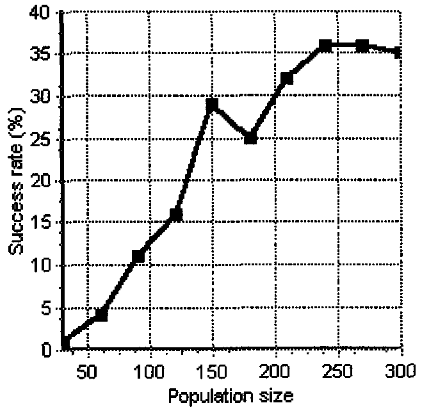


Fig. 10.3. The relationship between the success rate and the chromosome length (left side) and the population size (right side). Results are averaged over 100 runs.

Knowing that GP used a population of 4000 individuals to achieve a success rate of 42% we may infer that MEP needs a population smaller with one order of magnitude than the population needed by GP to solve the even-4-parity problem. The shortest evolved circuit implementing the even-4-parity problem has 9 gates. One of the evolved circuits is depicted in Fig. 10.4.

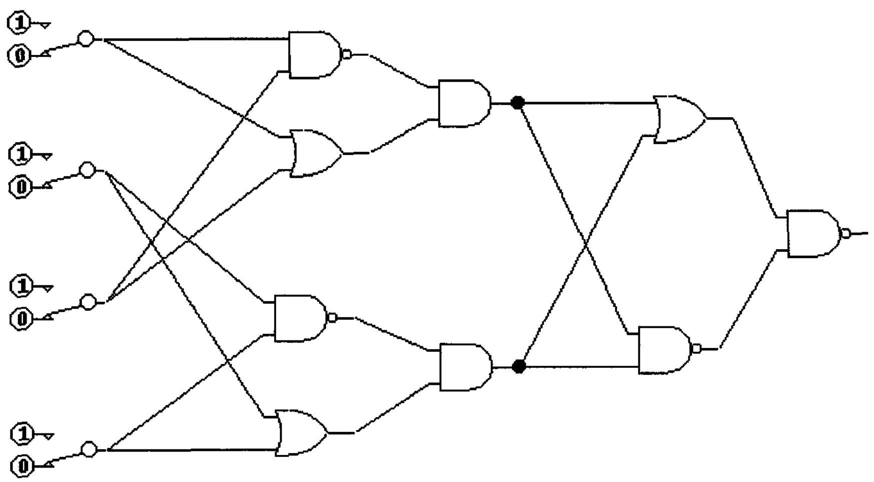


Fig. 10.4. A circuit for the even-4-parity problem.

The minimum computational effort required to solve this problem is 45,900 and it has been obtained at generation 9 using a population of 300 individuals with 200 genes each.

Even-5-parity

In this experiment, the behavior of the MEP algorithm for solving the even-5-parity problem is analyzed. For this problem MEP is run with a population of 4000 individuals having 600 genes each. In 5 runs (out of 30) MEP was able to find a perfect solution for this problem, yielding a success rate of 16.66%.

Note that for this problem GP - without Automatically Defined Functions (ADFs) - was not able to obtain a solution (within 20 runs) with a population of 4000 individuals [7]. When the population size was increased to 8000 individuals a solution was obtained by GP after 8 runs [7].

The curve representing the computational effort needed by MEP to solve the even-5-parity problem is depicted in Fig. 10.5.

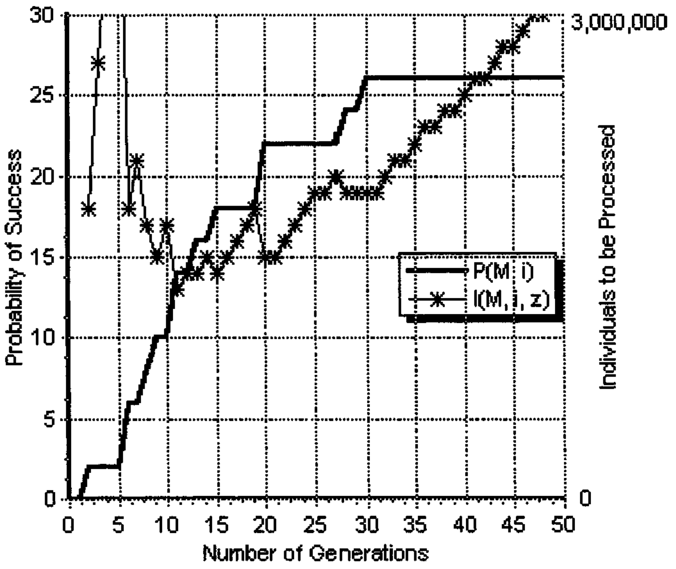


Fig. 10.5. The computational effort and the cumulative probability of success for the even-5-parity problem.

The minimum computational effort required to solve this problem is 1,364,000 and it was obtained at generation 11.

10.5.1 Summarized Results

The results obtained by GP and MEP are summarized in Table 10.4.

Table 10.4 shows that MEP outperforms standard GP with more than one order of magnitude for the even-3 and even-4-parity problems.

We may conclude that MEP significantly outperforms standard GP (without ADFs) for these particular cases of the even-parity problem.

Table 10.4. Computational effort required by GP and MEP for solving several even-parity instances. GP results are taken from [7].

Problem	GP	MEP
even-3-parity	80,000	6,840
even-4-parity	1,276,000	45,900
even-5-parity	6,528,000	1,364,000

10.6 Automatically Defined Functions in MEP

In this section we describe the way in which the Automatically Defined Functions [8] are implemented within the context of Multi Expression Programming.

The necessity of using reusable subroutines is a day-by-day demand of the software industry. Writing reusable subroutines proved to reduce:

(i) the size of the programs.
*(ii)*the number of errors in the source code.
*(iii)*the cost associated with the maintenance of the existing software.
*(iv)*the cost and the time spent for upgrading the existing software.

As noted by Koza [8] function definitions exploit the underlying regularities and symmetries of a problem by obviating the need to tediously rewrite lines of essentially similar code. Also, the process of defining and calling a function, in effect, decomposes the problem into a hierarchy of subproblems.

A function definition is especially efficient when it is repeatedly called with different instantiations of its arguments. GP with ADFs have shown significant improvements over the standard GP for most of the considered test problems [7, 8].

An ADF in MEP has the same structure as a MEP chromosome (i.e. a string of genes). The ADF is also evolved in the same way as a standard MEP chromosome. The function symbols used by an ADF are the same as those used by the standard MEP chromosomes. The terminal symbols used by an ADF are restricted to the function (ADF) parameters (formal parameters). For instance, if we define an ADF with two formal parameters p_0 and p_1 we may use only these two parameters as terminal symbols within the ADF structure, even if in the standard MEP chromosome (i.e. the main evolvable structure) we may use, let say, 20 terminal symbols only.

The set of function symbols of the main MEP structure is enriched with the Automatically Defined Functions considered in the system.

Example. Let us suppose that we want to evolve a problem using 2 ADFs, denoted ADF0 and ADF1 having 2 (p_0 and p_1) respectively 3 (p_0 and p_1 and p_2) arguments. Let us also suppose that the terminal set for the main MEP chromosome is $T = \{a, b\}$ and the function set $F = \{+, -, *, /\}$. The terminal and function symbols that may appear in ADFs and main MEP chromosome are given in Table 10.5.

Table 10.5. Parameters, terminal set and the function set for the ADFs and for the main MEP chromosome.

	Parameters	**Terminal set**	**Function set**
ADF0	p_0, p_1	T={p_0, p_1}	F={+,–,*,/}
ADF1	p_0, p_1, p_2	T={p_0, p_1, p_2}	F={+,–,*,/}
MEP chromosome	–	T={a, b}	F={+,–,*,/, ADF0, ADF1}

The ADF0 (p_0, p_1) could be defined as follows:

1. p_0
2. + 1, 1
3. p_1
4. / 3, 2
5. * 2, 4

The main MEP chromosome could be the following:

1. a
2. b
3. + 1, 2
4. ADF0 3, 1
5. a
6. ADF1 4, 5, 5
7. * 3, 6

The fitness of a MEP chromosome is computed as described in section 10.3.2. The quality of an ADF is computed in a similar manner. The ADF is read once and the partial results are stored in an array (by the means of Dynamic Programming [2]). The best expression encoded in the ADF is chosen to represent the ADF.

The genetic operators (crossover and mutation) used in conjunction with the standard MEP chromosomes may be used for the ADFs too. The probabilities for applying genetic operators are the same for MEP chromosomes and for the Automatically Defined Functions. The crossover operator may be applied only between structures of the same type (that is ADFs having the same parameters or main MEP chromosomes) in order to preserve the chromosome consistency.

10.7 Numerical Experiments with MEP and ADFs

In this section, several numerical experiments with Multi Expression Programming and Automatically Defined Functions are performed. The experiments performed in this section show that the ADF mechanism greatly improves the quality of the search, allowing us to perform a detailed analysis up to the even-8-parity problem. General parameters for Multi Expression Programming are given in Table 10.6.

Table 10.6. The general parameters of MEP with ADFs for solving even-parity problems.

Parameter	Value
Number of generations	51
Mutation probability	0.02
Crossover type	Uniform
Selection	q-tournament (q = 10% of the Population Size)
Function set	F = {AND, OR, NAND, NOR}

All terminals are kept on the first positions of the MEP chromosomes. The tournament size is set to 10% of the population size).

Even-4-parity

In this experiment the relationship between the success rate, the population size and the chromosome length for the even-4-parity problem is analyzed.

A population of 200 individuals is used when the relationship between the success rate and the chromosome length is analyzed. Chromosomes having 200 genes is used for analyzing the relationship between the success rate and the population size. Two Automatically Defined Functions taking two and three arguments are used in conjunction with Multi Expression Programming. The number of genes in ADFs was set to 50. Other parameters are given in Table 10.6. Results are depicted in Fig. 10.6.

The success rate of MEP is 100% when the population size is 200. By contrast, Genetic Programming uses a population of 4000 individuals to obtain the same success rate (100%) [7].

We also computed the effort needed to solve this problem. For this purpose we use a population of 60 MEP individuals having 200 genes each. The number of individuals that needs to be processed in order to obtain a solution with 99% probability is 7,440. This number was obtained at generation 43.

Even-5-parity

For this experiment we use a population with 400 individuals. Each individual has 200 genes. Three Automatically Defined Functions taking two, three and four arguments are used. The number of genes in each ADF is 50. Other MEP parameters are given in Table 10.6.

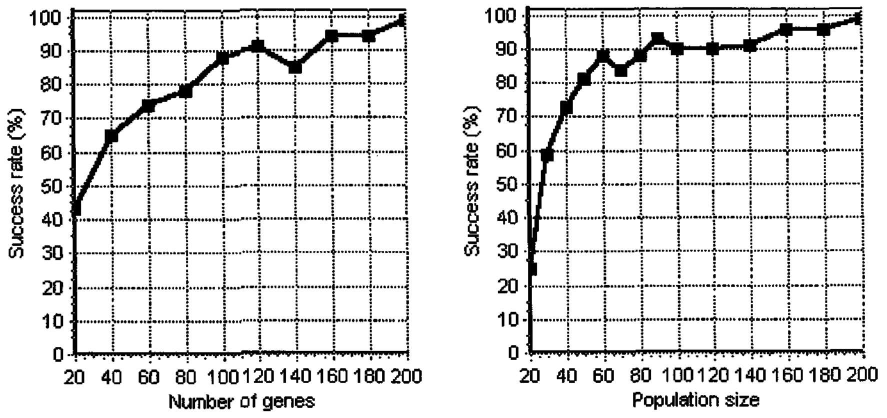


Fig. 10.6. The relationship between the success rate and the chromosome length (left side) and the population size (right side). Results are averaged over 100 runs.

The cumulative probability of success and the computational effort needed for solving this problem are depicted in Fig. 10.7.

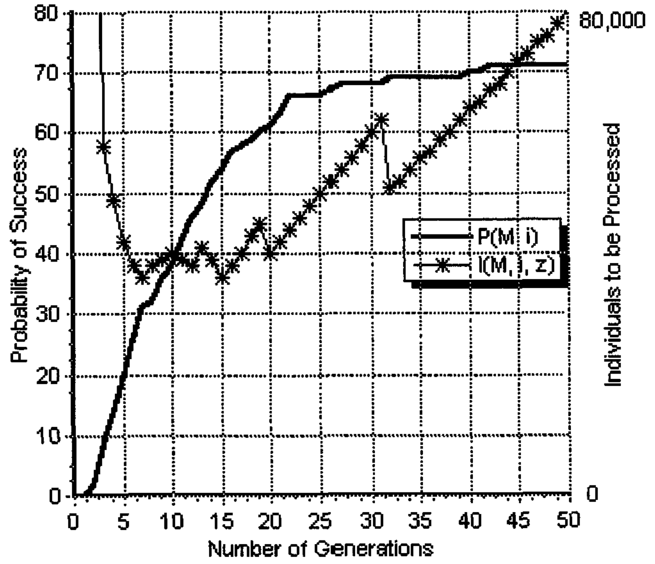


Fig. 10.7. The computational effort and the cumulative probability of success for the even-5-parity problem. Results are averaged over 100 runs.

The $I(M, i, z)$ curve reaches a minimum value at generation 15. Processing a number of 36,000 individuals is sufficient to yield a solution with 99% probability.

As a comparison, GP with ADFs requires 152,000 individuals to be processed in order to obtain a solution with 99% probability [8].

Even-6-parity

For this problem we use a population with 800 individuals. Each individual has 300 genes. Three ADFs taking two, three and four arguments are used. The number of genes in each ADF is 50. Other parameters of the MEP algorithm are given in Table 10.6. Results are presented in Fig. 10.8.

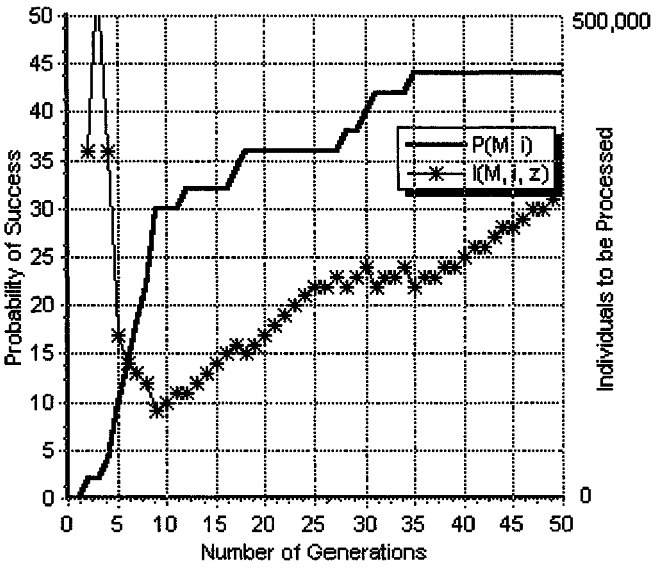


Fig. 10.8. The computational effort and the cumulative probability of success for the even-6-parity problem. Results are averaged over 50 runs.

The $I(M, i, z)$ curve reaches a minimum value at generation 9. Processing a number of 93,600 individuals is sufficient to yield a solution to with 99% probability.

Even-7-parity

For this experiment we use a population with 1000 individuals. Each individual has 400 genes. Three ADFs taking two, three and four arguments are

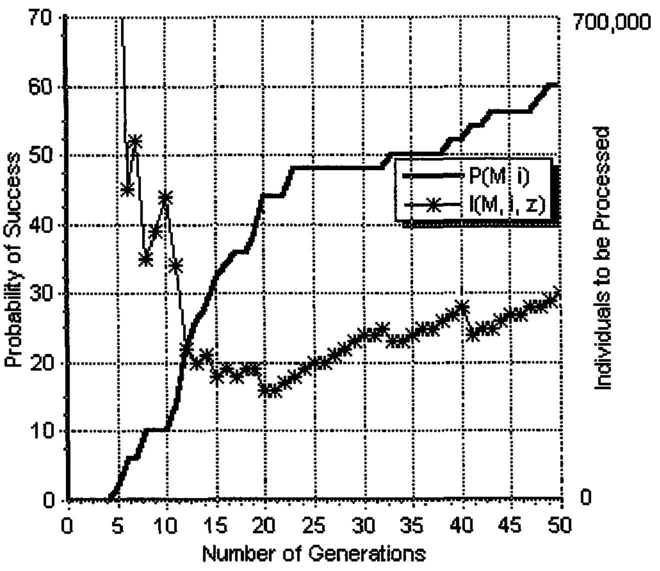


Fig. 10.9. The computational effort and the cumulative probability of success for the even-7-parity problem. Results are averaged over 50 runs.

used. The number of genes in each ADF is 100. Other parameters are given in Table 10.6. Results are given in Fig. 10.9.

Fig. 10.9 shows that the $I(M, i, z)$ curve reaches a minimum value at generation 20. Processing a number of 160,000 individuals is sufficient to yield a solution to with 99% probability. The cumulative probability of success is 60% at generation 50.

Even-8-parity

This case of the even-parity is the most difficult problem analyzed in this section. A population of 1000 individuals is used in this case. Each individual has 400 genes. Three ADFs taking two, three and four arguments are used. The number of genes in each ADF is 100. Other parameters are given in Table 10.6. Due to the increased computational time we performed only five runs which are not sufficient for computing a statistic (i.e. the success rate or the computational effort). A perfect solution (satisfying all fitness cases) was obtained in the fourth run.

10.7.1 Summarized Results

The results obtained by GP and MEP with Automatically Defined Functions are summarized in Table 10.7.

Table 10.7. Computational effort required by GP with ADFs and MEP with ADFs for solving several even-parity instances. GP results are taken from [8].

Problem	GP with ADFs	MEP with ADFs
even-4-parity	80,000	7,440
even-5-parity	464,000	36,000
even-6-parity	1,344,000	93,000
even-7-parity	1,440,000	160,000

Table 10.7 shows that MEP with ADFs outperforms GP with ADFs with more than one order of magnitude for the even-4, even-5, even-6, and even-7-parity problems.

10.8 Sub-Symbolic Node Representation

The Sub-Symbolic Node Representation [15, 18] in order to allow GP to perform small moves in the search space. It is widely known that a single point mutation, that can be applied to a MEP chromosome under the standard representation, may nevertheless result in a significant change in behavior of the MEP program. For instance, consider the gene AND 1 7, where the expressions encoded in positions 1 and 7 are Boolean expressions. If the operator AND is replaced with NAND, the return value of that subtree will be changed for all fitness cases. Instead of such a radical change we want a smoother mechanism that produced a more refined result (that is a mechanism that changes the results produced by only a subset of the training set).

A Boolean function of arity n can be represented as a truth table (bit-string) of length 2^n, specifying its return values on each of the 2^n input combinations. Thus, AND may be represented as 1000, OR as 1110, XOR as 0110. This representation is referred [15, 18] as *sub-symbolic* because function nodes are now seen as collection of entities rather than atomic units.

One feature of the Sub-Symbolic representation of Boolean function nodes is that, in contrast with the reduced function set normally used in Boolean classification tasks, it is unbiased, since it incorporates all 2^n nodes of arity n into its function set. Some of these may be superfluous (e.g. always-ON and always-OFF).

Our principal reason for including all Boolean functions of a given arity in our set is simplicity [18]. IF we want to reduce this set we have to put some constrains in the smooth operators (described in the next section). Note that the EQ and XOR functions are necessarily included in the arity 2 functions sets and that these will probably enhance the performance on the parity problems. On the other hand, the function set is much larger than normal leading to a significantly larger search space.

10.8.1 Smooth MEP Operators

In this section two new MEP operators are proposed. These operators are similar to the standard MEP operators but they can work with the sub-symbolic node representation.

Smooth Uniform Crossover

By crossover two parents are selected and are recombined. For instance, within the uniform recombination the offspring genes are taken randomly from one parent or another. The function parts, which are now binary strings of length 4, are recombined using the uniform crossover from the binary encoding [4].

Example. Let us consider the two parents C_1 and C_2 given in Table 10.8. The two offspring O_1 and O_2 are obtained by uniform recombination as shown in Table 10.8.

Table 10.8. MEP smooth uniform crossover.

Parents		Offspring	
C_1	C_2	O_1	O_2
1: ***b***	1: *a*	1: ***b***	1: *a*
2: **1110 1, 1**	2: *b*	2: *b*	2: **1110 1, 1**
3: **0100 2, 1**	3: 1011 1, 2	3: **0100 2, 1**	3: 1011 1, 2
4: ***a***	4: *c*	4: ***a***	4: *c*
5: **1001 3, 2**	5: *d*	5: *d*	5: **1001 3, 2**
6: ***a***	6: 1111 4, 5	6: 1111 4, 5	6: ***a***
7: **1101 1, 4**	7: 0011 3, 6	7: **1101 1, 4**	7: 0011 3, 6

Smooth Mutation

Each symbol (terminal, function reference and bit encoding the function symbol) in the chromosome may be target of mutation operator. Each binary position encoding the function symbol in a gene is affected by the smooth mutation operator with the same probability as all other symbols in a chromosome. To preserve chromosome consistency its first gene must encode a terminal symbol.

Example. Consider the chromosome C given in Table 10.9. If the boldfaced symbols are selected for mutation an offspring O is obtained as shown in Table 10.9.

Table 10.9. MEP smooth mutation.

C	*O*
1: *a*	1: *a*
2: **1000** 1, 1	2: 1101 1, 1
3: ***b***	3: 1110 2, 1
4: 1101 2, 2	4: 1101 2, 2
5: *b*	5: *b*
6: **1010 3**, 5	6: 1110 1, 5
7: *a*	7: *a*

10.9 Numerical Experiments with MEP and Sub-Symbolic Representation

The use of Sub-symbolic representation greatly improved the performance of MEP algorithm. Due to this reason we begin our experiments with the even-11-parity problem.

In [18] a parallel version of GP was used to solve the even-parity problem using a sub-symbolic representation. The parallel GP program was run on a client-server architecture with 50 processors. In [18] the authors performed a single run for all instances larger than the even-12-parity problem. More than that, a special technique called sub-machine code GP [17] was used in order to speed-up the GP program. The technique sub-machine code GP make use of processor's ability to perform some operations (such as AND) in parallel for all bits.

Due to the simplicity and efficiency of the MEP algorithm we performed multiple runs (at least 10) for each experiment. This allows us to compute the statistics described in section 10.4. Note that MEP was run on a single processor (at 850 MHz) architecture.

General parameter settings used by MEP in all the experiments performed in this section are given in Table 10.10.

Table 10.10. MEP parameters for solving even-parity problems using a sub-symbolic representation of operators.

Parameter	**Value**
Mutation probability	0.02
Crossover type	Uniform
Crossover probability	0.9
Selection	binary tournament
Function set	16 Boolean functions

Even-11-parity

The even-11-parity problem has 11 Boolean inputs and one Boolean output. The number of fitness cases is $2^{11} = 2048$.

The relationship between the success rate and the number of genes in a chromosome and the population size is analyzed for this problem.

A population of 50 individuals is used when the relationship between the success rate and the chromosome length is analyzed. Chromosomes with 300 genes are used for analyzing the relationship between the success rate and the population size. The number of generations was set to 100. Other parameters of the MEP algorithm are given in Table 10.11. Results are depicted in Fig. 10.10.

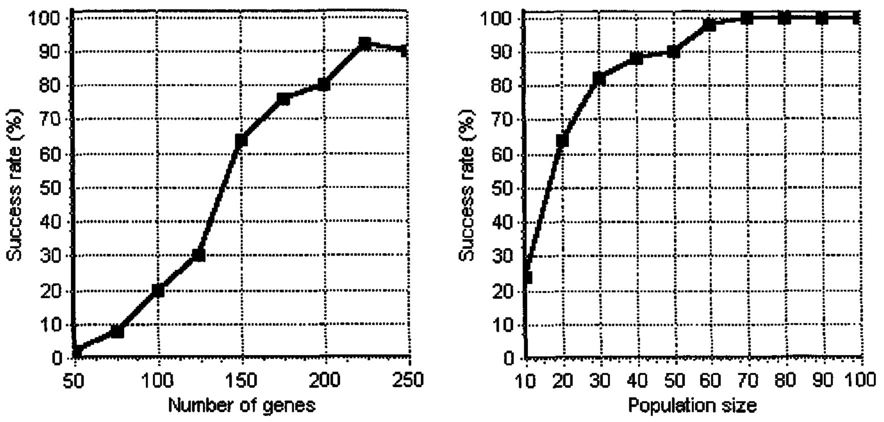


Fig. 10.10. The relationship between the success rate and the chromosome length (left side) and the population size (right side). Results are averaged over 50 runs.

Fig. 10.10 show that MEP is able to solve very well this problem. A population of 70 individuals having 300 genes each(see Fig. 10.10 right side) is sufficient to yield a 100% probability of success. The success rate increases as long as the number of genes in a MEP chromosome increases (see Fig. 10.10).

Even-12-parity

The number of fitness cases for the even-12-parity problem is 4096. For solving this problem with MEP we use a population of 25 individuals having 500 genes each. Other MEP parameters are given in Table 10.10. The program was run for 100 generations. Results over 100 independent runs are presented in Fig. 10.11.

The minimum number of individuals that needs to be processed in order to obtain a solution with a 99% probability of success is 7,420. This number is obtained at generation 99.

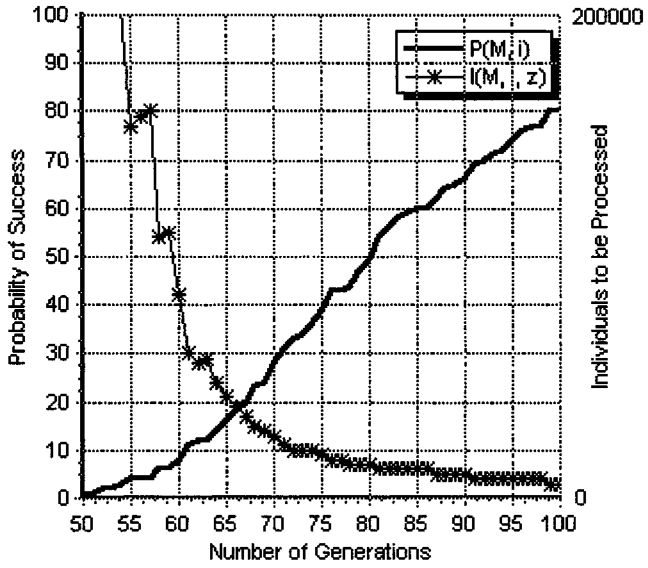


Fig. 10.11. The computational effort and the cumulative probability of success for the even-12-parity problem. Results are averaged over 100 runs.

By contrast, Genetic Programming with a population of 100 individuals requires 98,800 individuals to be processed in order to obtain a solution with 99% probability [18]. Thus, GP requires at least 13.6 times more individuals to be processed than MEP for solving this problem.

Even-13-parity

The number of fitness cases for this problem is 8192. We use the same MEP parameters as for the even-12-parity problem. The relationship between the number of generations and the cumulative probability of success is depicted in Fig. 10.12. The number of individuals to be processed in order to obtain a solution with 99% probability is computed for this problem, too.

The minimum number of individuals that needs to be processed in order to obtain a solution with a 99% probability of success is 2,325. This number is obtained at generation 93.

Even-14-parity

The number of fitness cases for the even-14-parity problem is 16384. For solving this problem with MEP we use a population of 40 individuals having 500 genes each. Other MEP parameters are given in Table 10.10. The program was run for 100 generations. Results over 100 independent runs are presented in Fig. 10.13.

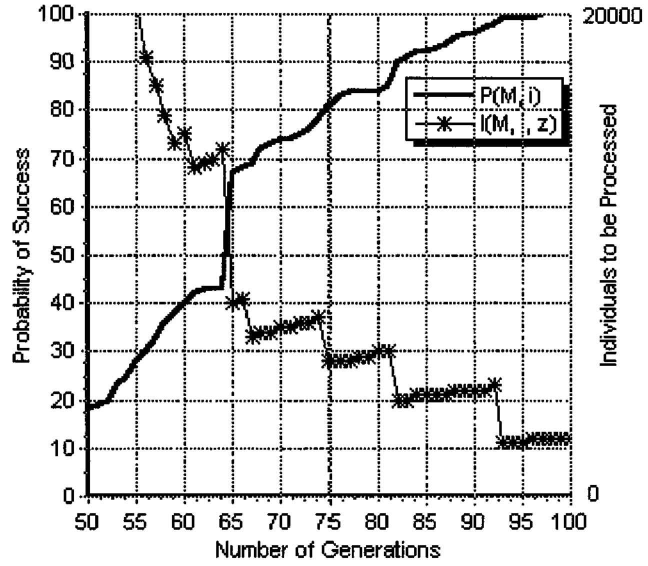


Fig. 10.12. The computational effort and the cumulative probability of success for the even-13-parity problem. Results are averaged over 100 runs.

The minimum number of individuals that needs to be processed in order to obtain a solution with a 99% probability of success is 7,210. This number is obtained at generation 89.

Even-15-parity

The number of fitness cases for the even-15-parity problem is 32768. For solving this problem with MEP we use a population of 100 individuals having 700 genes each. Other MEP parameters are given in Table 10.10. The program was run for 100 generations. Results over 100 independent runs are presented in Fig. 10.14.

The minimum number of individuals that needs to be processed in order to obtain a solution with a 99% probability of success is 29,700. This number is obtained at generation 99.

Even-16-parity

The number of fitness cases for the even-16-parity problem is 65536. For solving this problem with MEP we use a population of 100 individuals having 700 genes each. Other MEP parameters are given in Table 10.10. The program was run for 250 generations. Results over 100 independent runs are presented in Fig. 10.15.

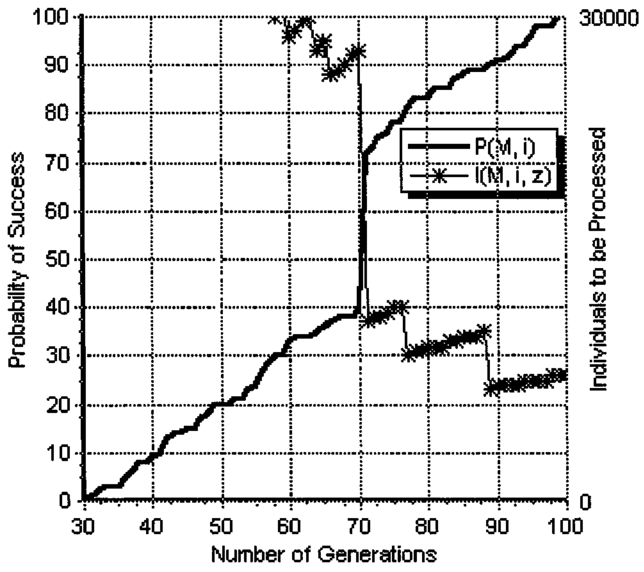


Fig. 10.13. The computational effort and the cumulative probability of success for the even-14-parity problem. Results are averaged over 100 runs.

The minimum number of individuals that needs to be processed in order to obtain a solution with a 99% probability of success is 28,000. This number is obtained at generation 140.

Even-17-parity

For this problem we performed 10 independent runs using the same parameters as those used for the problem even-16-parity. In all runs we obtained a perfect solution. The average number of generations required to obtain a solution is 131.

Even-18-parity

For this problem we performed 6 independent runs using the same parameters as those used for the problem even-16-parity. In 4 runs we obtained a perfect solution. The average number of generations required to obtain a solution is 168.

10.9.1 Summarized Results

The results obtained by MEP with Sub-Symbolic node representation are summarized in Table 10.11.

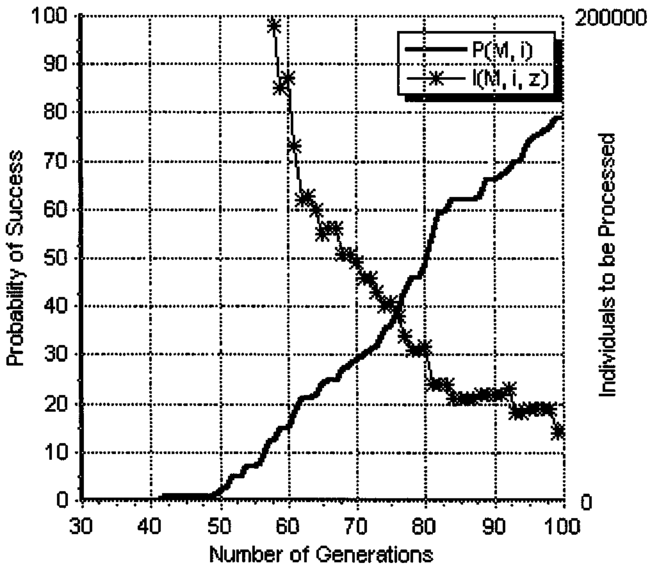


Fig. 10.14. The computational effort and the cumulative probability of success for the even-15-parity problem. Results are averaged over 100 runs.

Table 10.11. Computational effort required by GP and MEP with Sub-symbolic node representation for solving several even-parity instances. GP results are taken from [18].

Problem	GP with Sub-Symbolic node representation	MEP with Sub-Symbolic node representation
even-12-parity	98,800	7,420
even-13-parity	–	2,325
even-14-parity	–	7,210
even-15-parity	–	29,700
even-16-parity	–	28,000

Table 10.10 shows that MEP is able to solve the considered instances of the parity problem very well. The cells corresponding to GP are empty because GP was run only once for the considered examples.

10.10 Conclusions and Further Work

In this chapter, MEP technique has been used for solving even-parity problems. Two mechanisms for improving the MEP technique have been proposed and tested: Automatically Defined Functions and Sub-symbolic node representation.

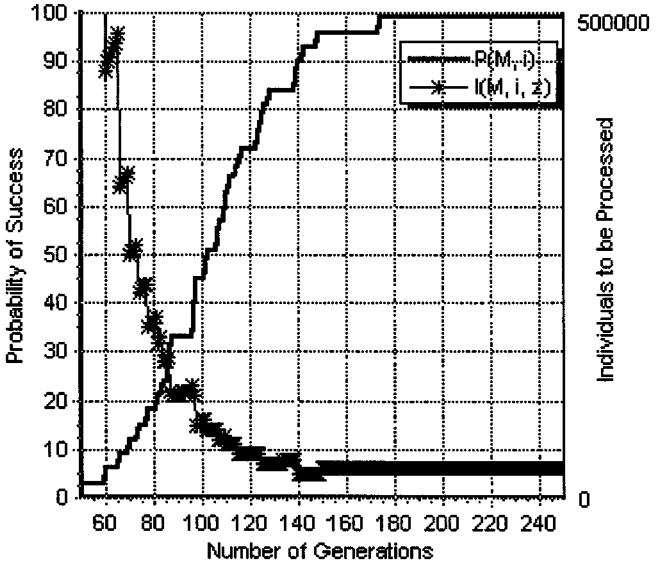


Fig. 10.15. The computational effort and the cumulative probability of success for the even-16-parity problem. Results are averaged over 100 runs.

Tables 10.4, 10.9 and 10.10 show that MEP outperforms GP when the success rate and the number of individuals to be processed is considered. As we said it before this statistics should be interpreted carefully since there are significant differences between GP and MEP representations and a perfect comparison between these two techniques cannot be made.

Further research will be focused on developing a Hierarchically Automatically Defined Functions [8] system within the context of Multi Expression Programming. In this system any function is allowed to call any other function already defined within the system.

Further efforts will be dedicated for implementing a parallel version of MEP (similar to that used in [18] for GP). Using this implementation we will be able to solve other large scale problems including higher versions of the even-parity problem.

Acknowledgments

The author is grateful to anonymous referees for their constructive comments and criticism of earlier versions of this chapter. The title of the chapter is adapted from [6].

References

1. A. Aho, R. Sethi, and J. Ullman, Compilers: Principles, Techniques, and Tools, Addison Wesley, 1986.
2. R. Bellman, Dynamic Programming, 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. D. Dumitrescu, B. Lazzerini, L. Jain, A. Dumitrescu, Evolutionary Computation, CRC Press, Boca Raton, FL, 2000.
5. C. Ferreira, Gene Expression Programming: a New Adaptive Algorithm for Solving Problems. Complex Systems, Vol. 13, Nr. 2, pp. 87-129, 2001.
6. A. S. Fraenkel, Scenic trails ascending from sea-level Nim to alpine chess, Games of No Chance, MSRI Workshop on Combinatorial Games, July, 1994, Berkeley, CA, MSRI Publications, R. J. Nowakowski (Editor), Vol. 29, Cambridge University Press, Cambridge, pp. 13-42, 1996.
7. J. R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, Cambridge, MA, 1992.
8. J. R. Koza, Genetic Programming II: Automatic Discovery of Reusable Programs, MIT Press, Cambridge, MA, 1994.
9. J. Miller, D. Job and V. Vassilev, Principles in the Evolutionary Design of Digital Circuits - Part I, Genetic Programming and Evolvable Machines, Vol. 1, pp. 7 - 35, Kluwer Academic Publishers, 2000.
10. J.F. Miller and P. Thomson, Cartesian Genetic Programming. The 3^{rd} International Conference on Genetic Programming (EuroGP2000), R. Poli, J.F. Miller, W. Banzhaf, W.B. Langdon, J.F. Miller, P. Nordin, T.C. Fogarty (Editors), LNCS 1802, Springer-Verlag, Berlin, pp. 15-17, 2000.
11. M. Oltean and D. Dumitrescu, Multi Expression Programming, technical report, UBB-01-2002, Babes-Bolyai University, Cluj-Napoca, Romania, available at www.mep.cs.ubbcluj.ro, 2002.
12. M. Oltean and C. Groşan, Evolving Evolutionary Algorithms using Multi Expression Programming, The 7^{th} European Conference on Artificial Life, Dortmund, W. Banzhaf (et. al), (Editors), LNCS 2801, pp. 651-658, Springer-Verlag, Berlin, 2003.
13. M. Oltean, Solving Even-parity problems with Multi Expression Programming, The 5^{th} International Workshop on Frontiers in Evolutionary Algorithm, K. Chen (et. al), (Editors) Research Park Triangle, North Carolina, pp. 315-318, 2003.
14. M. O'Neill and C. Ryan, Grammatical Evolution: A Steady State approach, The Second International Workshop on Frontiers in Evolutionary Algorithms, pp. 419-423, 1998.
15. J. Page, R. Poli and W. B. Langdon, Smooth Uniform Crossover with Smooth Point Mutation in Genetic Programming: A Preliminary Study. Genetic Programming, Proceedings of EuroGP'99, R. Poli, P. Nordin, W. B. Langdon and T. C. Fogarty, (Editors), LNCS 1598, pp. 39-49, Springer-Verlag, Berlin, 1999.
16. N.R. Patterson, Genetic Programming with Context-Sensitive Grammars, PhD thesis, University of St. Andrews, Scotland, 2003.

17. R. Poli and W. B. Langdon, Sub-machine Code Genetic Programming, Advances in Genetic Programming 3, L. Spector, W. B. Langdon, U-M O'Reilly and P. Angeline, (Editors), pp. 301-323, MIT Press, Cambridge, MA, 1999.
18. R. Poli and J. Page, Solving High-Order Boolean Parity Problems with Smooth Uniform Crossover, Sub-Machine Code GP and Demes, Journal of Genetic Programming and Evolvable Machines, Kluwer, pp. 1-21, 2000.
19. G. Syswerda, Uniform Crossover in Genetic Algorithms, in Proceedings of the 3^{rd} International Conference on Genetic Algorithms, J.D. Schaffer (Editor), Morgan Kaufmann Publishers, CA, 2-9, 1989.
20. D.H. Wolpert and W.G. McReady, No Free Lunch Theorems for Optimization, IEEE Transaction on Evolutionary Computation, Vol. 1, pp 67-82, 1997.
21. D.H. Wolpert and W.G. McReady, No Free Lunch Theorems for Search, Technical Report, SFI-TR-05-010, Santa Fe Institute, 1995.