Encoding Multiple Solutions in a Linear Genetic Programming Chromosome

Mihai Oltean[1], Crina Groşan[1], and Mihaela Oltean[2]

[1] Department of Computer Science,
Faculty of Mathematics and Computer Science,
Babeş-Bolyai University, Kogălniceanu 1
Cluj-Napoca, 3400, Romania.
{moltean, cgrosan}@cs.ubbcluj.ro
[2] David Prodan College, Cugir, 2566, Romania.
olteanmihaelaelena@yahoo.com

Abstract. Linear Genetic Programming (LGP) is a Genetic Programming variant that uses linear chromosomes for solution encoding. Each LGP chromosome is a sequence of ***C*** language instructions. Each instruction has a destination variable and several source variables. One of the variables is usually chosen to provide the output of the program. In this paper, we enrich the LGP technique by allowing it to encode multiple solutions for a problem in the same chromosome. Numerical experiments show that the proposed Multi-Solution LGP significantly outperforms the standard Single-Solution LGP on the considered test problems.

1 Introduction

Linear Genetic Programming (LGP) [1] is a Genetic Programming [2] variant that uses linear chromosomes for solution encoding. Each LGP chromosome is a sequence of ***C*** language instructions. Each instruction has a destination variable and several source variables. One of the variables is usually chosen to provide the output of the program.

In this paper an improved variant of Linear Genetic Programming is proposed. The obtained technique is called Multi-Solution Linear Genetic Programming (MS-LGP). In the proposed variant each chromosome stores multiple solutions of the problem being solved. All the solutions represented in a MS-LGP individual are decoded by traversing the chromosome only once. Partial results are stored by using Dynamic Programming. The best solution encoded in a MS-LGP chromosome will represent (will provide the fitness of) that individual.

Several numerical experiments with MS-LGP and with the standard Single-Solution Linear Genetic Programming (SS-LGP) are performed by using 4 test functions. For each test problem the relationships between the success rate and the population size and the code length are analyzed. Results show that MS-LGP significantly outperforms SS-LGP for all the considered test problems.

The paper is organized as follows. In section 2 Linear Genetic Programming is described. In sub-section 2.3 is described the way in which multiple solutions

are encoded in a LGP chromosome. Several numerical experiments are performed in section 3.

2 Linear Genetic Programming

Linear Genetic Programming (LGP) [1] uses a specific linear representation of computer programs. Instead of the tree-based GP expressions [2] of a functional programming language (like ***LISP***), programs of an imperative language (like ***C***) are evolved.

A 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 the LGP program is the following one:

void LGP(**double** r[8])
{
r[0] = r[5] + 73;
r[7] = r[3] – 59;
r[2] = r[5] + r[4];
r[6] = r[7] * 25;
r[1] = r[4] – 4;
r[7] = r[6] * 2;
}

2.1 Decoding LGP Individuals

A linear genetic program can be turned into a functional representation by successive replacements of variables starting with the last effective instruction [1].

Usually one of the variables (r[0]) is chosen as the output of the program. This choice is made at the beginning of the program and is not changed during the search process. In what follows we will denote this LGP variant as Single-Solution Linear Genetic Programming (SS-LGP).

2.2 Genetic Operators

The variation operators used in conjunction with Linear Genetic Programming are crossover and mutation. Standard LGP crossover works by exchanging continuous sequences of instructions between parents [1].

Two types of standard LGP mutations are usually used: micro mutation and macro mutation. By micro mutation an operand or an operator of an instruction is changed [1].

Macro mutation inserts or deletes a random instruction [1].

Since we are interested more in multi-solutions paradigm rather than in variable length chromosomes we will use fixed length chromosomes in all experiments performed in this paper. Genetic operators used in numerical experiments are uniform crossover and micro mutation.

LGP uniform crossover LGP uniform crossover works between instructions. The offspring's genes (instructions) are taken with a 50% probability from the parents.

Example

Let us consider the two parents C_1 and C_2 given in Table 1. The two offspring O_1 and O_2 are obtained by uniform recombination as shown in Table 1.

Table 1. LGP uniform recombination

Parents		Offspring	
C_1	C_2	O_1	O_2
$r[5] = r[3]$ * $r[2]$;	**r[2] = r[0] + r[3];**	$r[5] = r[3]$ * $r[2]$;	**r[2] = r[0] + r[3];**
$r[3] = r[1] + 6$;	**r[1] = r[2] * r[6];**	**r[1] = r[2] * r[6];**	$r[3] = r[1] + 6$;
$r[0] = r[4]$ * $r[7]$;	**r[4] = r[6] - 4;**	$r[0] = r[4]$ * $r[7]$;	**r[4] = r[6] - 4;**
$r[5] = r[4] - r[1]$;	**r[6] = r[5] / r[2];**	$r[5] = r[4] - r[1]$;	**r[6] = r[5] / r[2];**
$r[1] = r[6]$ * 7;	**r[2] = r[1] + 7;**	**r[2] = r[1] + 7;**	$r[1] = r[6]$ * 7;
$r[0] = r[0] + r[4]$;	**r[1] = r[2] + r[4];**	**r[1] = r[2] + r[4];**	$r[0] = r[0] + r[4]$;
$r[2] = r[3]$ / $r[4]$;	**r[0] = r[4] * 3;**	**r[0] = r[4] * 3;**	$r[2] = r[3]$ / $r[4]$;

LGP Mutation LGP mutation works inside of a LGP instruction. By mutation each operand (source or destination) or operator is affected with a fixed mutation probability.

Example

Consider an individual C which is affected by mutation. An offspring O is obtained as shown in Table 2 (modified variables are written in boldface):

Table 2. LGP mutation

C	O
$r[5] = r[3]$ * $r[2]$;	$r[5] = r[3]$ * $r[2]$;
$r[3] = r[1] + 6$;	$r[3] =$ **r[6]** + **r[0]**;
$r[0] = r[4]$ * $r[7]$;	$r[0] = r[4] + r[7]$;
$r[5] = r[4] - r[1]$;	**r[4]** $= r[4] - r[1]$;
$r[1] = r[6]$ * 7;	$r[1] = r[6]$ * **2**;
$r[0] = r[0] + r[4]$;	$r[0] = r[0] + r[4]$;
$r[2] = r[3]$ / $r[4]$;	**r[0]** $= r[3]$ / $r[4]$;

2.3 Multi Solutions Linear Genetic Programming

We enrich the LGP structure in two ways:

(i) We allow as each destination variable to represent the output of the program. In the standard LGP only one variable is chosen to provide the output.
(ii) We check for the program output after each instruction in chromosome. This is again different from the standard LGP where the output was checked after the execution of all instructions in a chromosome.

After each instruction, the value stored in the destination variable is considered as a potential solution of the problem. The best value stored in one of the destination variables is considered for fitness assignment purposes.

Example

Consider the chromosome C given below:
void LGP(**double** r[8])
{
$r[5] = r[3] * r[2];$
$r[3] = r[1] + 6;$
$r[0] = r[4] * r[7];$
$r[6] = r[4] - r[1];$
$r[1] = r[6] * 7;$
$r[2] = r[3] / r[4];$
}
Instead of encoding the output of the problem in a single variable (as in SS-LGP) we allow that each of the destination variables ($r[5]$, $r[3]$, $r[0]$, r[6], $r[1]$ or $r[2]$) to store the program output. The best output stored in these variables will provide the fitness of the chromosome.

For instance, if we want to solve symbolic regression problems, the fitness of each destination variable $r[i]$ may be computed using the formula:

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

where $o_{k,i}$ is the result obtained in variable $r[i]$ for the fitness case k, w_k is the targeted result for the fitness case k and n is the number of fitness cases. For this problem the fitness needs to be minimized.

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

$$f(C) = \min_i f(r[i]).$$

Thus, we have a Multi-Solution program at two levels: first level is given by the possibility that each variable to represent the output of the program and

the second level is given by the possibility of checking for the output at each instruction in the chromosome.

Our choice was mainly motivated by the No Free Lunch Theorems for Search [4]. There is neither practical nor theoretical evidence that one of the variables employed by the LGP is better than the others. More than that, Wolpert and McReady [4] 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.

The Multi-Solution ability has been tested within other evolutionary model such as Multi Expression Programming [3]. For these methods it has been shown [3] that encoding multiple solutions in a single chromosome leads to significant improvements.

3 Numerical Experiments

In this section several experiments with SS-LGP and MS-LGP are performed. For this purpose we use several well-known symbolic regression problems. The problems used for assessing the performance of the compared algorithms are:

$f_1(x) = x^4 + x^3 + x^2 + x$,
$f_2(x) = x^6 - 2x^4 + x^2$,
$f_3(x) = sin(x^4 + x^2)$,
$f_4(x) = sin(x^4) + sin(x^2)$.

For each function 20 fitness cases have been randomly generated with a uniform distribution over the [0, 1] interval.

The general parameters of the LGP algorithms are given in Table 3. The same settings are used for Multi Solution LGP and for Single-Solution LGP.

Table 3. The parameters of the LGP algorithm for symbolic regression problems

Parameter	**Value**
Number of generations	51
Crossover probability	0.9
Mutations	2 / chromosome
Function set	$F = \{+, -, *, /, sin\}$
Terminal set	Problem inputs + 4 supplementary registers
Selection	Binary Tournament
Algorithm	Steady State

For all problems the relationship between the success rate and the chromosome length and the population size is analyzed. The success rate is computed as the number of successful runs over the total number of runs.

3.1 Experiment 1

In this experiment the relationship between the success rate and the chromosome length is analyzed. The population size was set to 50 individuals. Other parameters of the LGP are given in Table 3. Results are depicted in Figure 1.

Figure 1 shows that Multi-Solution LGP significantly outperforms Single-Solution LGP for all the considered test problems and for all the considered parameter setting. More than that, large chromosomes are better for MS-LGP than short chromosomes. This is due to the multi-solution ability: increasing the chromosome length leads to more solutions encoded in the same individual. The easiest problem is f_1. MS-LGP success rate for this problem is over 90% when the number of instructions in a chromosome is larger than 12. The most difficult problem is f_4. For this problem and with the parameters given in Table 3, the success rate of the MS-LGP algorithm never increases over 47%. However, these results are very good compared to those obtained by SS-LGP (the success rate never increases over 5%).

3.2 Experiment 2

In this experiment the relationship between the success rate and the population size is analyzed. The number of instructions in a LGP chromosome was set to 12. Other parameters for the LGP are given in Table 3. Results are depicted in Figure 2.

Figure 2 shows that Multi-Solution LGP performs better than Single-Solution LGP. Problem f_1 is the easiest one and problem f_4 is the most difficult one.

4 Conclusions

In this paper an improved variant of the Linear Genetic Programming technique has been proposed. The improvement consists in encoding multiple solutions of a problem in a single chromosome. It has been show how to efficiently decode this chromosome by traversing it only once. Numerical experiments have shown that Multi-Solution LGP significantly outperforms Standard Single-Solution LGP for all the considered test problems.

References

1. Brameier M., and Banzhaf W.: A Comparison of Linear Genetic Programming and Neural Networks in Medical Data Mining, IEEE Transactions on Evolutionary Computation, Vol. 5, (2001) 17-26
2. Koza J. R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, Cambridge, MA, (1992)
3. Oltean M.: Solving Even-Parity Problems using Multi Expression Programming, in Proceedings of the the 7^{th} Joint Conference on Information Sciences, Edited by Ken Chen (et. al), (2003) 315-318
4. Wolpert D.H. and McReady W.G.: No Free Lunch Theorems for Search, Technical Report, SFI-TR-05-010, Santa Fe Institute, (1995)

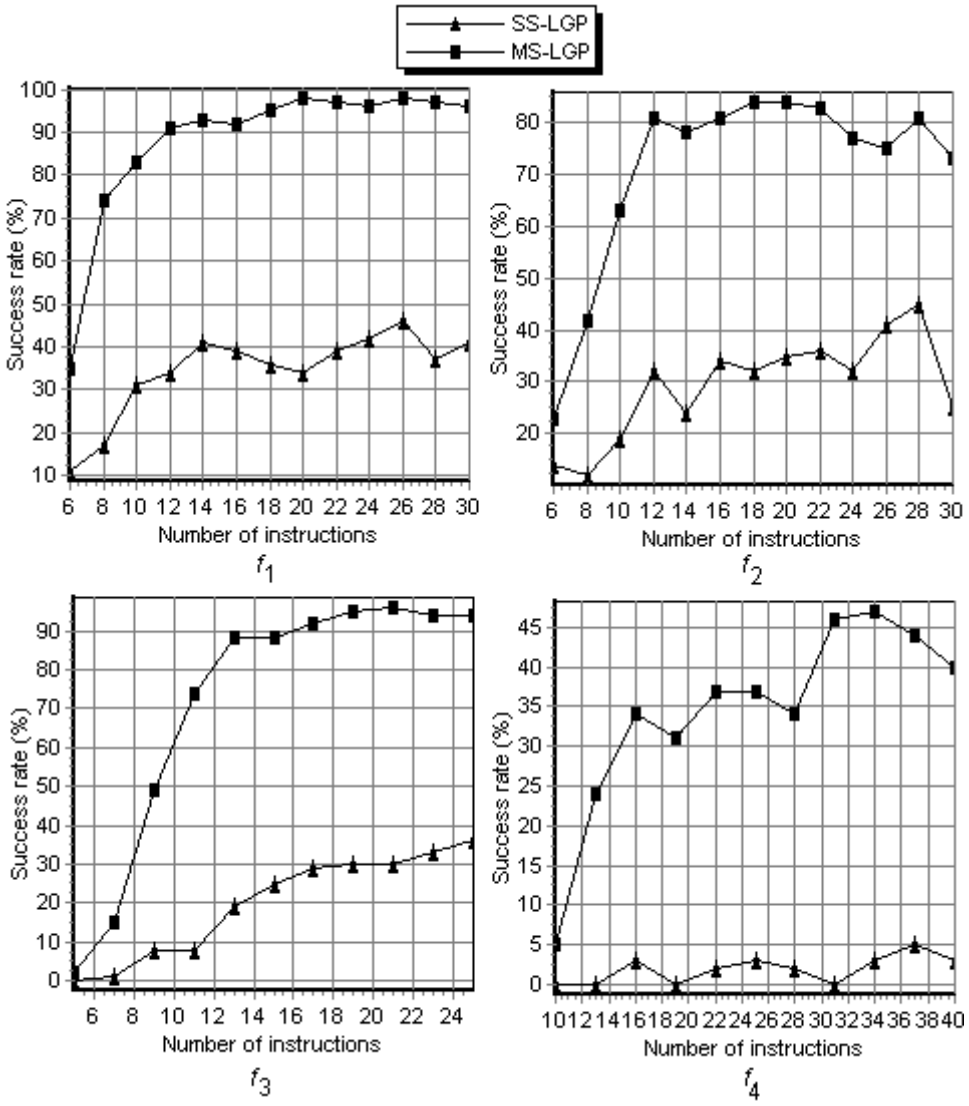

Fig. 1. The relationship between the success rate and the number of instructions in a chromosome. Results are averaged over 100 runs.

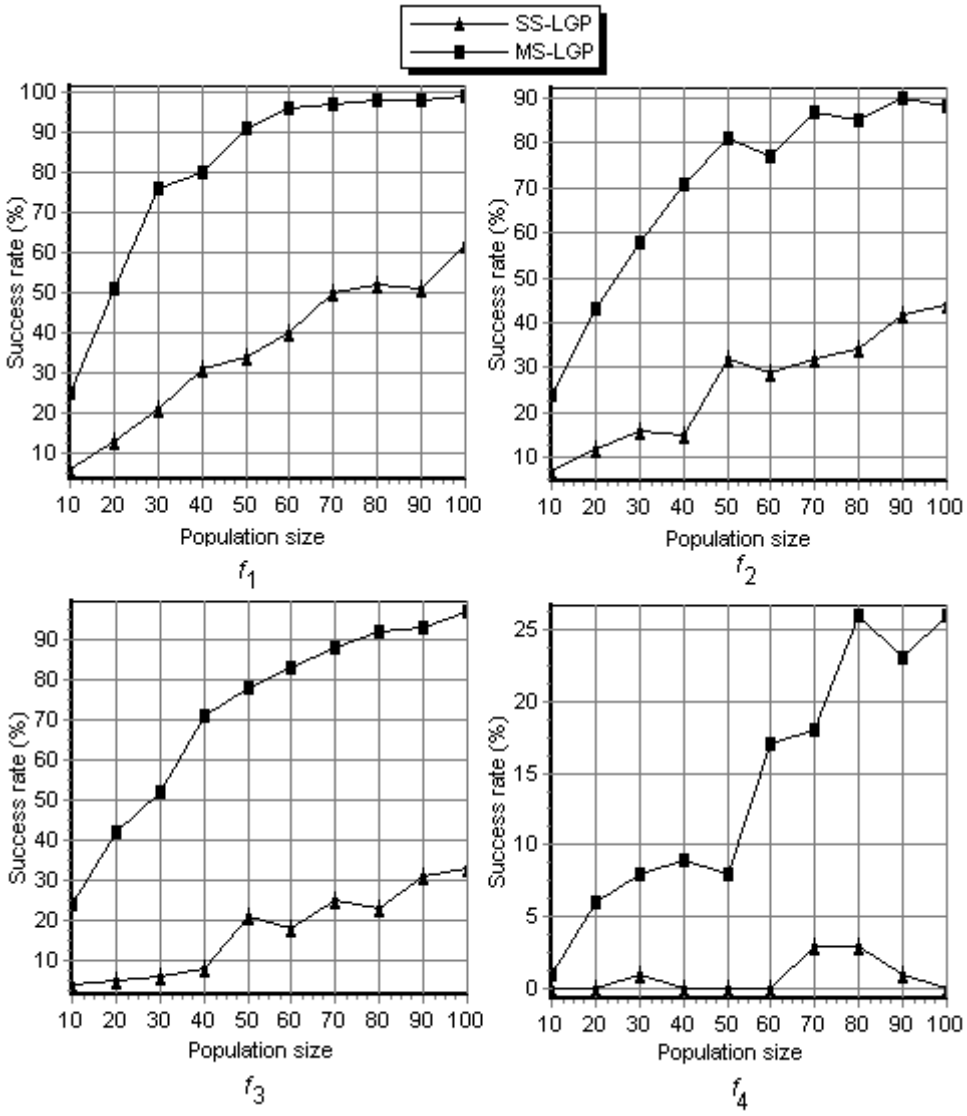

Fig. 2. The relationship between the population size and the success rate. Population size varies between 10 and 100. Results are averaged over 100 runs.