Int. J. Appl. Math. Comput. Sci., 2004, Vol. 14, No. 3, 375–384

EGIPSYS: AN ENHANCED GENE EXPRESSION PROGRAMMING APPROACH FOR SYMBOLIC REGRESSION PROBLEMS[†]

HEITOR S. LOPES*, WAGNER R. WEINERT*

* Centro Federal de Educação Tecnológica do Paraná / CPGEI
Av. 7 de setembro, 3165, 80230-901 Curitiba (PR), Brazil
e-mail: hslopes@cpgei.cefet.br, weinert@cpgei.cefetpr.br

This paper reports a system based on the recently proposed evolutionary paradigm of gene expression programming (GEP). This enhanced system, called EGIPSYS, has features specially suited to deal with symbolic regression problems. Amongst the new features implemented in EGIPSYS are: new selection methods, chromosomes of variable length, a new approach to manipulating constants, new genetic operators and an adaptable fitness function. All the proposed improvements were tested separately, and proved to be advantageous over the basic GEP. EGIPSYS was also applied to four difficult identification problems and its performance was compared with a traditional implementation of genetic programming (LilGP). Overall, EGIPSYS was able to obtain consistently better results than the system using genetic programming, finding less complex solutions with less computational effort. The success obtained suggests the adaptation and extension of the system to other classes of problems.

Keywords: evolutionary computation, symbolic regression, mathematical modeling, systems identification

1. Introduction

Evolutionary Computation (EC) constitutes an emerging area of research and it has been successfully applied to many problems ranging from computer science to engineering and biology. The central idea in EC is that solutions to a problem are represented as entities able to evolve throughout generations as a consequence of interactions with other candidate solutions and the application of genetic operators. The main factor in the evolution is selective pressure caused by the bias towards the best solutions. EC includes several paradigms which use concepts drawn from the natural evolution of living beings and genetics. Amongst these paradigms, the commonest are: Genetic Algorithms (GA) (Goldberg, 1989; Holland, 1995), Genetic Programming (GP) (Koza, 1992; 1994), Evolutionary Programming (EP) (Fogel *et al.*, 1966) and Evolution Strategies (ES) (Rechenberg, 1973; Schwefel, 1977). More recently, Ferreira (2001; 2003) proposed a new evolutionary technique as an extension of GP, named Gene Expression Programming (GEP). Since GEP is very recent, it has not yet gained widespread use, although its characteristics suggest a large application range, overlapping with those of GA and GP. This encourages the comparison of GEP with other evolutionary algorithms in particular classes of problems so as to analyse its performance.

This paper describes a flexible tool, named EGIPSYS (Enhanced Gene-expressIon Programming for SYmbolic regression problemS). This tool is based on GEP and was specifically developed for symbolic regression problems. EGIPSYS implements the basic GEP algorithm proposed in (Ferreira, 2001) and has several other improvements. Amongst the new features implemented in our system are: new selection methods, chromosomes of variable length, a new approach to manipulating constants, new genetic operators and an adaptable fitness function. In this paper we describe in detail the special features of EGIPSYS and evaluate the performance of such improvements with a test problem. An application of this tool to a number of problems is also reported, and results are compared with a traditional implementation of GP.

Symbolic regression is a class of problems that are characterized by a number of data points to which one wants to fit an equation. Contrary to linear, polynomial or other types of regression where the nature of the model is specified in advance, in symbolic regression one is given only instances of inputs-outputs (independent and dependent variables), and no information about the model. Thus, the goal consists in finding a mathematical expression involving the independent variable(s) that is able to minimize some measure of error between the values of

[†] This work was partly supported by a CAPES grant to W.R. Weinert, and a CNPQ grant to H.S. Lopes, process number 552022/02-0.

the dependent variable, computed with the expression and their actual values. In this context, finding both the functional form and the appropriate numeric coefficients of an expression at the same time is a real challenge for which no efficient mathematical procedure exists. Consequently, heuristic approaches, such as GP and GEP, have been devised to solve this problem (see, e.g., Ferreira, 2003; Hoai *et al.*, 2002; Salhi *et al.*, 1998; Shengwu *et al.*, 2003).

2. Fundamentals of Gene Expression Programming

Gene Expression Programming was proposed by Ferreira (2001) as an alternative to overcome the common drawbacks of GA and GP for real-world problems. The main difference between GEP, GA and GP resides in the way individuals of a population of solutions are represented. GEP follows the same Darwinian principle of the survival of the fittest and uses populations of candidate solutions to a given problem in order to evolve new ones. The evolving populations undergo selective pressure and their individuals are submitted to genetic operators.

In GEP, like in GA, an individual is represented by a genotype, constituted by one or more chromosomes. This work follows (Ferreira, 2001) in the sense that we use only one chromosome per individual. In GA, a chromosome is composed of one or more genes that represent the encoded variables of the problem. When decoded, they represent the phenotype. In GP, an individual is represented as a tree and, usually, there is no encoding, so that the genotype and the phenotype are equivalent (this is not true for particular implementations). In GEP, a chromosome is a linear and compact entity, easily manipulable with genetic operators (mutation, crossover, transposition, etc. — see Section 2.2). In living beings, genes encoded in the DNA strands of the chromosomes are expressed, meaning that they are translated into proteins with biological functions. In the same way, in GEP, expression trees (ETs) are the expression of a given chromosome. ETs constitute the phenotypic representation of the problem.

The first step of the GEP algorithm is the generation of the initial population of solutions. This can be accomplished by means of a random process or using some knowledge about the problem. Then, chromosomes are expressed as ETs, which are evaluated according to a fitness function that determines how good a solution is in the problem domain. Usually, the fitness function is evaluated by processing a number of instances of the target problem, known as fitness cases. If a solution of satisfactory quality is found, or a predetermined number of generations is reached, the evolution stops and the best-so-far solution is returned.

On the other hand, if the stop condition is not met, the best solution of the current generation is kept (this means elitism) and the rest is submitted to a selective process. Selection implements the survival-of-the-fittest rule, and the best individuals will have a better chance to generate descendants. This whole procedure is repeated for several generations. As generations proceed, it is expected that, on the average, the quality of the population is improved.

2.1. Chromosome Encoding

A chromosome is composed of genes, usually more than one (multigenic). Each gene is divided into a head and a tail. The size of the head (h) is defined by the user, but the size of the tail (t) is obtained as a function of h and a parameter n. This parameter is the largest arity found in the function set used in the run. The following equation relates the tail size with the other parameters:

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

Each gene encodes an expression tree. In the case of multigenic chromosomes, all ETs are connected together by their root node using a linking function. Every gene has a coding region known as an ORF (open reading frame) or a K-expression that, after being decoded, is expressed as an ET, representing a candidate solution for the problem. Symbolic regression problems are modelled using a set of functions and a set of terminals. The set of functions usually includes, for instance, basic arithmetic functions, trigonometric functions or any other mathematical or user-defined functions that the user believes can be useful for the construction of the model. The set of terminals is composed of constants and the independent variables of the problem. In the heads of genes, functions, terminals and constants are allowed, while in the tails, only terminals or constants. Figure 1 shows how a chromosome with two genes is encoded as a linear string and how it is expressed as an ET. Note that, in this example, both genes have coding (expressed) and non-coding regions, just like the coding and non-coding sequences of biological genes.

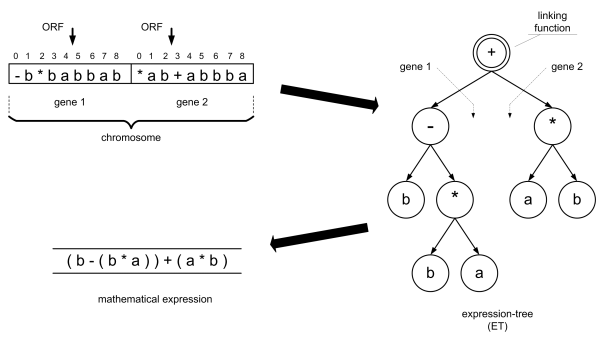

Fig. 1. Chromosome with two genes and its decoding in GEP.

2.2. Selection Method and Genetic Operators

GEP uses the well-known roulette-wheel method for selecting individuals. This method is sometimes used in both GA (Goldberg, 1989) and GP (Koza, 1992). In contrast to GA and GP, GEP has several genetic operators to reproduce individuals with modification.

GEP uses simple elitism (known as cloning) of the best individual of a generation, preserving it for the next one. Replication is an operation that aims to preserve several good individuals of the current generation for the next one. In fact, this is a do-nothing probabilistic operation that takes place during selection (using the roulette-wheel method), and replicated individuals will be subjected to the action of the genetic operators.

The mutation operator aims to introduce random modifications into a given chromosome. A particularity of this operator is that some integrity rules must be obeyed so as to avoid syntactically invalid individuals. In the head of a gene, both terminals and functions are permitted (except for the first position, where only functions are allowed). However, in the tail of a gene only terminals are allowed.

Similarly to GA, GEP uses one-point and two-point crossover. The second type is somewhat more interesting since it can turn on and off noncoding regions within the chromosome more frequently. In addition to that, another kind of crossover was implemented — gene recombination — that recombines entire genes. This operator randomly chooses genes in the same position in two parent chromosomes to form two new offsprings.

In GEP, there are two transposition operators: IS (insertion sequence) and RIS (root IS). An IS element is a variable-size sequence of elements extracted from a random starting point within the genome (even if the genome was composed of several chromosomes). Another position within the genome is chosen as the insertion point. This target site must be within the head part of a gene and cannot be the first element (gene root). The IS element is sequentially inserted in the target site, shifting all elements from this point onwards and a sequence with the same number of elements is deleted from the end of the head, so that the structural organization is maintained. This operator simulates the transposition found in the evolution of biological genomes. RIS is similar to the IS transposition, except that the insertion sequence must have a function as the first element and the target point must be also the first element of a gene (root).

3. Methodology

In this section we describe the improvements in the original GEP implemented in EGIPSYS.

3.1. Chromosome Structure and the Initial Population

As mentioned before, we propose a more flexible representation for individuals using chromosomes of variable length. These chromosomes can be formed by one or more genes of the same size. In the original GEP, finding the optimal size of the head of a gene is an open problem. Usually, bigger problems require a larger gene head (Ferreira, 2001). Since there is still no procedure for setting *a priori* the gene head size, frequently the user has to run the algorithm several times with different gene head sizes until finding a suitable dimension for a satisfactory solution. To circumvent this problem, in EGIPSYS the population of solutions can have chromosomes of various length.

When the initial population is created, care must be taken so as to have a large diversity of chromosomes. That is, the initial population needs to have as many different individuals as possible so as to better explore the search space in further generations. The original GEP generates the initial population at random. In EGIPSYS, by default, half of the population is uniformly created with chromosome sizes proportional to a user-defined parameter that specifies the gene head size range. The remaining elements of the initial population are randomly generated within the same range. This method for generating the initial population was inspired in the well-known ramped-half-and-half method for GP proposed by Koza (1992). Experiments reported in Section 4 demonstrate that the procedure proposed here for generating the initial population is beneficial to the evolutionary process.

3.2. Constants

A crucial property that functions and terminals sets must have in GP is sufficiency (Koza, 1992). This means that these sets must have all the elements needed to represent a satisfactory solution for the problem. However, sometimes one does not have a full insight into the problem to determine those sets beforehand. This is specially true when considering the use of constants in the terminal set. In particular, for symbolic regression problems, constants can be useful, allowing solutions to be fine-tuned.

In GEP, constants can be created either by the algorithm itself or using a list of ephemeral constants that makes part of the chromosome (Ferreira, 2003). In EGIPSYS, we propose a user-defined policy for constants, defined by two parameters: the probability of using constants and their initial range. During evolution the absolute value of the constants can extrapolate the initial range due to the mutation operator. EGIPSYS implements a local search operator (see Section 3.5) that uses a hill-climbing policy to fine-tune constants. Also, the system allows the use of pre-defined constants, like π, e or other user-defined values. This is particularly interesting when

the user knows, for example, that some physical constant will be present in the final expression.

3.3. Alternative Selection Methods

Originally, GEP uses the fitness roulette wheel method to select individuals to be replicated and then to undergo the action of genetic operators. For the application of the operators, replicated individuals are chosen at random. Besides this strategy, in EGIPSYS we implemented two other methods: always using the roulette wheel (without random selection) or always using the stochastic tournament. Both the strategies are common in GAs. The first one induces a strong selective pressure and usually makes convergence faster (most often to a local maximum). To circumvent this possibility, we also implemented a dynamic linear scaling, as proposed by Goldberg (1989) for GAs, to be used in conjunction with this method (see Section 3.6 for details). The default selection method in EGIPSYS is the stochastic tournament. This method uses a parameter that indicates the percentage of the population to be chosen at random for the tournament. These individuals will compete and the best ones will be selected to be replicated.

3.4. Regular Genetic Operators

EGIPSYS uses elitism in the same way as in the original GEP. Transposition operators were not changed in their essence, except that they were adapted to work with variable-length chromosomes. This adaptation was necessary to warrant the creation of synctactically valid individuals. Single point crossover was not implemented, only the two-point version was considered. Finally, gene recombination operates only over chromosomes of the same size so as to guarantee that all chromosomes keep their genes with the same head and tail sizes.

The mutation operator was the one that was most deeply changed, basically to cope with constants. When mutation is applied to a constant (with the default probability, see Table 1), two outcomes of this operation are possible: either a small perturbation is added to this constant or it is substituted by another element (a function, a terminal or a random constant). The probability for each of these outcomes is 50%. In the case when a random perturbation is to be added to the constant, it works as follows: if a random-generated number (between 0 and 1) is greater than or equal to 0.5, another random value no larger than 10% of the current value of the constant is added to it. Otherwise, the same value is subtracted from it. In the case when a constant is substituted by another element, the structural constraints of GEP must be respected, such that in the tail of genes only terminals and constants can appear.

3.5. Local Search Operator

The difficulty in finding appropriate values for the constants of an expression is a common problem emerging when using GP for symbolic regression problems. Usually, GP (and also GEP) is not able to fine-tune constants, which results in solutions of lower quality. In EGIPSYS we devised a local search operator, especially suited for fine-tuning the constants of a chromosome. Since this operator has a high computational cost, it is probabilistically applied depending on a user-defined parameter. This operator is intelligent in the sense that, after its application, the current modified solution is evaluated and, if an improved solution is obtained, it is kept. Otherwise, the operation is undone. The operator is applied in two steps as follows: first, the current fitness of a chromosome is saved and, starting from the left outermost chromosome towards the right outermost one, one seeks for a constant. Once found, the value of the constant is incremented by 10%. The solution is then re-evaluated and, if the fitness is higher than before, the constant will be increased again. This procedure is repeated until the fitness no longer increases, or a limit of 10 operations is reached. If, after the first increment, the fitness value decreases, the operation is undone and the constant is then decreased by 10%. The procedure is repeated as before while the fitness is improving or 10 operations are done. This finishes the first step. If the limit number of operations was reached in the first step (either incrementing or decrementing the constant), no further step is needed. Otherwise, the last two values of the constant are considered: k_1 (the last value, when the fitness has decreased) and k_2 (the last but one value, when the fitness is the highest of the step). It is not possible to guarantee that k_2 is the best value for the constant and a new local search procedure is started aiming to fine-tune that value. A new value for the constant is obtained using the average: $k_{new} = (k_1 + k_2)/2$. The chromosome is re-evaluated: if the fitness increases, we set $k_2 = k_{new}$, otherwise $k_1 = k_{new}$. The procedure is repeated 10 times, thus completing Step 2. Then the next constant of the chromosome is sought and the two-step local search procedure is repeated. It is worth emphasizing that the local search operator has a very high computational cost and its application must be careful.

3.6. Fitness Function

The fitness function evaluates how good a candidate solution is for the problem. In EGIPSYS, we normalized the fitness function between 0 and 1 such that 0 represents the worst possible value and 1, the best. This normalization helps users to understand the evolution of fitness throughout generations independently of the problem. For symbolic regression problems, it is customary to employ an error measure like the sum of absolute or quadratic errors.

We improved these two measures including two parameters, ref_val and $mult$,

$$fitness_{(i,t)} = \frac{ref\_val}{ref\_val + mult \sum_{j=1}^{Ne} |S(i,j) - C(j)|}, \quad (2)$$

$$fitness_{(i,t)} = \frac{ref\_val}{ref\_val + mult \sum_{j=1}^{Ne} [S(i,j) - C(j)]^2}, \quad (3)$$

where:

ref_val: user-defined reference value,
$fitness_{(i,t)}$: fitness of individual i in generation t,
$mult$: user-defined multiplying factor,
$S(i,j)$: value returned by expression i for fitness case j,
$C(j)$: actual value of fitness case j,
Ne: number of fitness cases.

Both $mult$ and ref_val play important roles in the fitness function since they can be used for scale compression and uncompression. Depending on the value of the fitness function for the individuals of a generation, it can be difficult to establish an efficient selective pressure and, therefore, evolution can stagnate. On the other hand, if the discrepancies among fitness values are large, the high selective pressure leads to premature convergence. The two parameters of the fitness functions in (2) and (3) can be set by the user to adjust the normalized fitness to the magnitude of the error measure (see Fig. 2). Typical values for $mult$ are 10, 1 or 0.1, and for ref_val they are 1, 10 or 100. Besides this static adjustment of the fitness values, there is also a dynamic adjustment given by a linear scaling, as suggested by Goldberg (1989) for GAs. When this scaling is on, fitness values are adjusted by a linear equation such that the average fitness is kept constant and the maximum fitness is adjusted to the doubled average fitness. This fitness adjustment is used only for selection purposes and is computed in every generation.

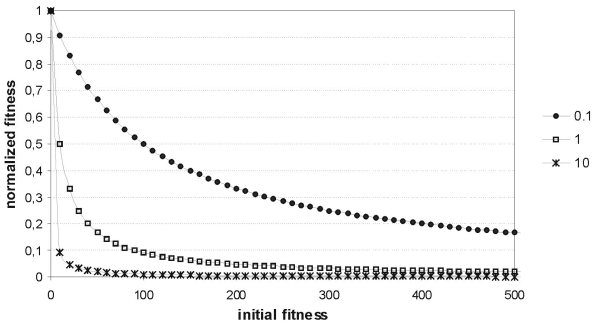

Fig. 2. Fitness normalization using $ref\_val = 10$ for different values of $mult$.

3.7. Default Parameters

Based on the original GEP (Ferreira, 2001) and on a number of empirical experiments (not reported here), we defined standard values for the running parameters of EGIPSYS, such that it can reveal a good performance for various problems. Generality in symbolic regression problems was the focus instead of efficiency for a specific problem. It is clear that complex problems may request a specific configuration of parameters, as will be shown later. Table 1 defines all default parameters for EGIPSYS.

Table 1. Default parameters for EGIPSYS.

Parameter	Value
Population size	30
Number of generations	50
Linking function	sum
Function set	$\{+, -, *, /\}$
Number of genes	3
Gene head size	6
Probability of using constants	0.2
Selection method for replication	Stochastic tournament
Tournament size	10% of population size
Elitism operator	Cloning
Mutation probability	0.05
IS and RIS transpositions probabilities	0.1
Two-point crossover probability	0.3
Gene recombination probability	0.1
Accuracy	0.01
Fitness function	cf. Eqn. (2)
$mult$	0.1
ref_val	10
Use dynamic linear scaling	yes

4. Experiments and Results

In this section we present the results of experiments using EGIPSYS for selected symbolic regression problems. EGIPSYS was developed under the graphics interface of Microsoft Windows 2000 and all experiments reported in this paper were run on a PC-clone with an AMD Athlon-XP 2.4 MHz processor and 512 MBytes of main memory. These experiments aimed to evaluate the improvements featured in EGIPSYS, as well as to compare its performance with a popular GP system, namely LilGP (Zongker

et al., 1998). LilGP is based on the genetic programming system proposed by Koza (1992), and is useful for various problems, including symbolic regression. LilGP version 1.1 is freely available on the Internet[1] and, for the experiments reported here, we used the default parameters shown in Table 2.

Table 2. Default parameters for LilGP.

Parameter	Value
Population size	500
Number of generations	50
Method for generating the initial population	Ramped half-and-half
Initial tree depth	$[2..6]$
Maximum tree depth during run	17
Breeding phases	2 (crossover and reproduction)
Selection method for both phases	Roulette wheel
Crossover probability	0.9
Reproduction probability	0.1

The first problem (cf. Section 4.1) concerns the prediction of the number of sunspots, based on previous observations. This is a classical time-series prediction problem, a special type of symbolic regression. This problem is used to evaluate the improvements proposed over the basic GEP.

The next problem (cf. Section 4.2) is the identification of a quadratic function corrupted by additive noise. It consists of a simple toy problem for symbolic regression and, therefore, shall not represent a great challenge for both systems. The remaining three problems (Sections 4.3–4.5) represent increasing levels of difficulty and were drawn from a database of identification problems available on the Internet[2].

The results of the experiments are presented in tables for both systems, EGIPSYS and LilGP. We present the correlation coefficient (r) that quantifies the similarity between the given set of points of a problem and those produced by the equation found. This statistical measure ranges from $+1$ to -1. At the extremes, there are exact correlations between the observed and predicted values (directly proportional, i.e., $r = 1$, or inversely proportional, i.e., $r = -1$). The closer r to zero, the less correlation between observed and predicted values. We also present the number of generations necessary to find the best solution (gen_{best}) that will be used to estimate the computational effort, and the number of nodes (functions and terminals) of the best result found ($nodes_{best}$). Due to the stochastic nature of both systems, we run each experiment 10 times, with different random seeds and we report the average values and their standard deviation. Except for the sunspot problem, unless otherwise stated, all the experiments used the default parameters shown in Table 1 for EGIPSYS and the parameters shown in Table 2 for LilGP.

[1] *http://garage.cps.msu.edu/software/software-index.html*
[2] *http://www.esat.kuleuven.ac.be/~tokka/daisydata.html*

4.1. Sunspot Problem

In this section, in contrast to the following, we aimed at verifying what is the effect of the proposed improvements implemented in EGIPSYS, compared with the original GEP. Data used in this experiment are related to the number of sunspots observed yearly, from 1700 to 1988. This dataset was used for testing several machine-learning systems, including GEP (Ferreira, 2003; Weigend *et al.*, 1992). Originally, there were 289 consecutive observations, but we use only 100, as the same data were used by (Ferreira, 2003). For this time-series problem, it was assumed that the prediction of a given value depends on the previous 10 observations. Therefore, the problem has 10 inputs and one output.

We run EGIPSYS using parameters simulating the basic GEP (Ferreira, 2001) as the baseline for further comparisons. Next, using the same parameters, the effect of five features implemented in EGIPSYS was tested separately. Finally, all the proposed improvements were used together. These experiments were arranged in seven series in which the system was run 100 times each with different random seeds. The following experiments were done:

(A) Basic GEP;

(B) GEP with different chromosome lengths. The objetive is to verify the influence of a larger diversity in the initial population. Gene head lengths were set to the range $[6..12]$;

(C) GEP with tournament selection. The objective is to verify the influence of the selection method in the overall performance;

(D) GEP with linear scaling. This experiment aims to check whether or not linear scaling can alleviate the selective pressure caused by the roulette wheel selection method throughout generations;

(E) GEP with a different fitness function. The objective is to verify the utility of the fitness function defined in Eqn. (2), in comparison with the original method proposed in (Ferreira, 2001). Parameters ref_val and $mult$ were set to default values (see Table 1);

(F) GEP with constants and the special mutation operator. This experiment aims to evaluate the impact of using constants as building blocks for the algorithm. The

probability of using constants was set to 0.2 and the initial range to $[-10, 10]$;

(G) EGIPSYS with default parameters[3]. The objective is to verify the joint effect of (B+C+D+E+F).

In Table 3, f_{best} is the average fitness value of the best individual (using the fitness function originally proposed for GEP), AME is the average of the sums of the absolute mean errors (used in the fitness function), p_{time} is the average processing time (in seconds) for the complete run. The other measures were defined before. Notice that, for Experiments E and G, we used Eqn. (2) as the fitness function. However, in these cases, the original fitness of GEP was also computed for the best individual, but it was used only for comparison with the other experiments.

Table 3. Results of different experiments for 100 runs of the sunspot problem.

Exp.	f_{best}	AME	p_{time}	r	gen_{best}	$nodes_{best}$
A	7502.95	16.63	56.19	0.799	44.8	22.7
B	7604.12	15.51	48.21	0.837	42.4	19.5
C	7620.84	15.32	61.23	0.825	44.6	23.5
D	7586.90	15.70	56.43	0.822	43.6	21.2
E	7551.51	16.09	57.99	0.820	44.2	22.1
F	7705.66	14.38	55.28	0.836	44.5	21.2
G	7756.88	13.81	50.50	0.845	46.9	19.8

In Table 3 it can be seen that, except for gen_{best} and $nodes_{best}$, the basic GEP performed worse than any other improvement, notably for the performance measures. On the other hand, Experiment G demonstrates that the improvements implemented in EGIPSYS are really advantageous.

4.2. Noisy Quadratic Function Problem

This is a synthetic problem of a simple polynomial regression where the output is corrupted by additive noise. For this problem, a total of 201 data points were generated by

$$y = 2x^2 - 3x + 4 + noise, \tag{4}$$

where $noise = (rnd/5) - 0.1$, and rnd is a randomly generated number in the range $[0, 1]$. The input vector $x(i)$ was obtained from $x(i + 101) = \sin(i/10)$, with $i = -100, \ldots, 100$.

The results presented in Table 4 show that both systems produced very good results. To illustrate this, the best solution found by EGIPSYS was $y = 2x^2 - 3x + 3.981$, rather close to Eqn. (4).

[3]Parameters shown in Table 1, except for the use of different gene head lengths, see Experiment B.

Table 4. Results of 10 runs for the noisy quadratic function problem.

Output	System	r	gen_{best}	$nodes_{best}$
y	EGIPSYS	0.987 ± 0.003	34.8 ± 10.9	27.4 ± 3.6
	LilGP	0.989 ± 0.000	39.2 ± 7.4	158.8 ± 84.9

4.3. Lake Erie Problem

The data for this problem are a result of a simulation related to the identification of the western basin of the lake Erie (USA/Canada) and were first reported in (Guidorzi *et al.*, 1980). This database has 4 series of 57 samples with 5 input and 2 output parameters. The four series are: the original data with no noise and the same data with 10%, 20% and 30% additive white noise. The input variables are: water temperature (x_1), water conductivity (x_2), water alkalinity (x_3), NO_3 concentration (x_4), and the total hardness of water (x_5). The output variables are: the amount of dissolved oxygen (y_1) and algae concentration (y_2). In this study we choose only the output (y_1) for testing EGIPSYS and LilGP.

The results for this problem are shown in Table 5. Note that, in all cases, EGIPSYS performed considerably better than LilGP, even though the population size used in LilGP exceeds that of EGIPSYS by a factor of 16.

Table 5. Results of 10 runs for the lake Erie problem.

Output	System	r	gen_{best}	$nodes_{best}$
y_1 – no noise	EGIPSYS	0.891 ± 0.038	45.5 ± 6.0	31.2 ± 17.9
	LilGP	0.731 ± 0.164	36.5 ± 13.0	155.8 ± 102.5
y_1 – 10% noise	EGIPSYS	0.890 ± 0.030	47.2 ± 2.4	25.2 ± 4.9
	LilGP	0.718 ± 0.125	38.6 ± 14.1	44.8 ± 62.3
y_1 – 20% noise	EGIPSYS	0.847 ± 0.037	48.3 ± 1.9	24.8 ± 3.5
	LilGP	0.666 ± 0.127	38.8 ± 10.5	104.8 ± 74.4
y_1 – 30% noise	EGIPSYS	0.746 ± 0.067	45.5 ± 5.3	25.8 ± 4.3
	LilGP	0.691 ± 0.129	32.6 ± 12.4	146.0 ± 74.9

4.4. pH Problem

This is a highly nonlinear problem of the process industry and it is related to the simulation of a pH neutralization process in a constant-volume stirred tank (McAvoy *et al.*, 1972). The problem has two input variables: the acid solution inflow (x_1) and the base solution inflow (x_2), and one output dependent variable: the pH of the solution in the tank (y). There are 2001 samples collected at regular intervals (10 sec), which are used as fitness cases in both systems.

As shown in Table 6, EGIPSYS performs again considerably better than LilGP, despite the tremendous difference in population sizes.

Table 6. Results of 10 runs for the pH problem.

Output	System	r	gen_{best}	$nodes_{best}$
y	EGIPSYS	0.630±0.339	41.6±6.2	24.4±3.7
	LilGP	0.184±0.171	7.8±9.8	17.4±19.1

Another experiment was performed considering the output of the system as dependent not only on the current inputs, but also on the previous ones. Therefore, a new experiment was performed using both the current sample (i-th) and the previous one ($(i-1)$-th). The notation used is: ${}^{i}x_1$ for the current acid solution inflow and ${}^{i-1}x_1$ for the previous sample, and ${}^{i}x_2$ for the current base solution inflow and ${}^{i-1}x_2$ for the previous sample. Consequently, the problem now is to find a mathematical relationship between the current value of pH (${}^{i}y$) as a function of ${}^{i}x_1$, ${}^{i}x_2$, ${}^{i-1}x_1$ and ${}^{i-1}x_2$. Two further runs of EGIPSYS were performed to test its specific features. In the first run, the range for the head of genes was set to $[6..15]$, the population size was increased to 100 and the number of generations was set to 250. For the second run, the same parameters were used and we included the local search operator, being applied with a probability of 0.1 only in the last 10 generations, just to fine-tune the constants. All the remaining default parameters listed in Table 1 were used in both runs.

The results of these two runs are shown in Table 7, where it can be seen that EGIPSYS was able to improve the previous result further, at the expense of more generations.

Table 7. Results for two additional runs of EGIPSYS for the pH problem with non-standard parameters.

Output	System run	r	gen_{best}	$nodes_{best}$
y	EGIPSYS 1	0.766	150	24
	EGIPSYS 2	0.800	243	42

4.5. Power Plant Problem

This problem uses data collected from a 120 MW thermoelectric power plant (Pont-sur-Sambre in France). They were used in (Guidorzi and Rossi, 1974) and, later, in (Moonen *et al.*, 1989). There are 5 input variables: gas flow (x_1), turbine valves opening (x_2), super heater spray flow (x_3), gas dumpers (x_4) and air flow (x_5), and 3 output variables: steam pressure (y_1), main steam temperature (y_2) and reheat steam temperature (y_3). A total of 200 samples are available as fitness cases.

Table 8 reports the results obtained for this problem. Each independent variable represents a different degree of difficulty for symbolic regression. For all three subproblems, EGIPSYS performed better than LilGP.

Table 8. Results of 10 runs for the power plant problem.

Output	System	r	gen_{best}	$nodes_{best}$
y_1	EGIPSYS	0.827±0.057	47.6±2.6	28.6±8.4
	LilGP	0.790±0.090	25.4±12.3	40.0±29.1
y_2	EGIPSYS	0.634±0.087	47.6±2.2	26.0±6.3
	LilGP	0.458±0.150	21.9±20.5	22.4±26.9
y_3	EGIPSYS	0.616±0.070	47.5±5.2	25.2±5.5
	LilGP	0.525±0.117	26.9±13.8	162.2±147.1

Again, an additional run of EGIPSYS was done with special parameters. Now, only output y_3 was used (the one with the worst average results in Table 8). The same parameters of the second additional run of the pH problem were used, except that the local search operator was also applied every 50 generations (with a probability of 0.1).

The results for this additional run are in Table 9. With the additional computational effort needed by the local search operator (and more generations), an improvement over the previous solution was observed.

Table 9. Results for an additional run of EGIPSYS for the power plant problem with non-standard parameters.

Output	System	r	gen_{best}	$nodes_{best}$
y_3	EGIPSYS	0.699	200	31

5. Discussion and Conclusions

In this paper we presented an enhanced gene expression programming system (EGIPSYS) specially suited for symbolic regression and we compared its performance against a traditional genetic programming system (LilGP) in several instances of identification problems. Besides, we experimentally showed that all improvements proposed in EGIPSYS over the basic GEP are advantageous.

For both EGIPSYS and LilGP, one can evaluate the average computational effort necessary for finding the best solution by means of a product $(gen_{best} \cdot nodes_{best} \cdot Ne)$. This product reflects the number of trials that an algorithm needed to find the best solution of the run. Since the runs were performed using the same number of fitness cases (Ne) for each problem, this parameter can be disregarded for comparison purposes between the methods.

Another performance metric to be analyzed is the complexity of the solution, related to the number of nodes of the tree representing the best solution. Solutions with a large number of nodes, besides being difficult to understand (especially in systems where the transfer function

has some physical meaning), can be overfitted to the input data (fitness cases). In this case, the extrapolation of the mathematical expression obtained beyond the range of input data should be performed with care. Therefore, less complex solutions tend to be more general. Neither EGIPSYS nor LilGP have any explicit mechanism to simplify the mathematical expressions manipulated (like the edition operator suggested by Koza (1992)). Consequently, all obtained solutions could be simplified, reducing the overall number of nodes. Nevertheless, it is possible that the form GEP represents individuals (coding and non-coding regions within the chromosome) that may lead to simpler solutions than those obtained by GP. The bloat effect is a well studied issue in GP, but not in GEP. Therefore, although this hypothesis seems to be fair, its proof is beyond the scope of this paper. This seems to be an open field for further research.

The experiments performed to evaluate the improvements proposed in EGIPSYS clearly show their advantages. In all these experiments, the performance measured by f_{best} and AME was equivalent and the use of a different fitness function (experiment E) led to a small improvement over the basic GEP. The small difference in the r values for all experiments suggests that this is not a good quality measure for a solution. The use of a stochastic tournament as the selection method is computationally more expensive than roulette wheel. This is observed in the high p_{time} value of Experiment C. However, this method does not impose a high selective pressure throughout generations, thus avoiding fast convergence to local minima. This can be inferred from the good performance regarding both f_{best}, AME and r. It is possible that for harder problems this selection method can be more useful. The use of linear scaling to control the selective pressure induced by roulette wheel (Experiment D) did not show significative improvements in performance, but, on the other hand, a small computational effort was needed. The use of chromosomes of different lenghts (Experiment B) decreases significantly the processing time (p_{time}). This is because the average length of chromosomes is smaller than in the case where the population was created with full-length chromosomes. This is reflected directly in the average size of the obtained solutions ($nodes_{best}$) and, consequently, in the computational effort. The explicit use of constants is definitely important for some symbolic regression problems. This was observed in the excellent performance and small computational effort obtained in Experiment F. Finally, Experiment G shows the joint benefits of all previous improvements, demonstrating an interesting trade-off between performance, processing time and computational effort.

As expected, the noisy quadratic function problem actually did not represent a real challenge for both systems. Both EGIPSYS and LilGP found good solutions, fitting the target transfer function, cf. Eqn. (4), almost perfectly. This can be inferred by the average correlation coefficients in Table 4, which are practically the same and close to 1. However, the computational effort needed by LilGP was 18 times higher than that of EGIPSYS (recall the population size of both systems), and the best solutions found were about 6 times larger than those found by EGIPSYS.

For the lake Erie problem, EGIPSYS found good results, whereas LilGP was unable to do so. As expected, the quality of solutions for both systems decreased as the noise increased (see Table 5). However, EGIPSYS consistently found better solutions than LilGP. Regarding the number of generations to find the best solutions, they remained almost the same, independently of the noise level of the problem. Also, EGIPSYS needed much less computational effort than LilGP (around 13 times less, independently of the noise level). For most cases, the solutions found by LilGP were about 5 times more complex than those found by EGIPSYS.

For the pH problem, neither EGIPSYS nor LilGP was capable of finding good solutions. The extremely low quality solutions found by LilGP suggest that this algorithm was not able to escape from local minima. Even so, overall, EGIPSYS revealed again a better performance than LilGP. The two independent runs of EGIPSYS with non-standard parameters showed its potential to deal with difficult nonlinear problems. The assumption that the previous inputs have influence on the current output is commonly made in nonlinear identification problems. With this approach the difficulty of the problem increased, as more independent variables were used. Therefore, a greater computational effort was necessary to find a better solution (see Table 7). When the local search operator was turned on, an even better result was found, reinforcing its usefulness in difficult problems.

For the power plant problem, only for output y_1 it was possible to find satisfactory solutions. Both EGIPSYS and LilGP failed to find good solutions for the other two outputs using the default parameters (see Table 8). However, for all cases the quality of solutions found by EGIPSYS was better than those found by LilGP. In the same way as in the previous problems, the average computational effort required by EGIPSYS was much smaller than that of LilGP (about 9 times). For output y_2 it was the only case of the experiments where the average complexity of solutions found by LilGP was smaller than the one found by EGIPSYS. An obvious explanation for this fact can be deduced from the joint analysis of the (low) correlation coefficient and the (small) number of generations LilGP needed to find the best solution: due to the particularities of the fitness landscape, it was not possible for this system to escape from local minima (like in the pH problem).

From the reported results, it can be observed that the standard deviations of r, gen_{best} and $nodes_{best}$ are proportionaly smaller for EGIPSYS than for LilGP (when compared with the respective averages). This smaller variance indicates that EGIPSYS (that is, GEP) is more consistent with results from run to run than LilGP (that is, GP). This analysis, together with the fact that GEP uses much fewer individuals than GP, strongly suggests that the former algorithm explores more efficiently the search space than the latter.

Overall, the gene expression programming system proposed produces consistently better results than the system using genetic programming. Also, it finds less complex solutions with less computational effort. The main contribution of this work consists in the improvements made in the basic gene expression programming algorithm first proposed in (Ferreira, 2001). We understand that most of these improvements can be useful for other types of problems that can be dealt by such an evolutionary computation technique. Hence, future work will focus on the adaptation and extension of EGIPSYS to other classes of problems. Aiming to encourage further research and experimentation with this new technique, EGIPSYS will be made freely available for academic use.

References

Ferreira C. (2001): *Gene Expression Programming: A new adaptive algorithm for solving problems*. — Complex Systems, Vol. 13, No. 2, pp. 87–129.

Ferreira C. (2003): *Function finding and a creation of numerical constants in gene expression programming*, In: Advances in Soft Computing, Engineering Design and Manufacturing (J.M. Benitez, O. Cordon, F. Hoffmann and R. Roy, Eds.). — Springer-Verlag: Berlin, pp. 257–266.

Fogel L.J., Owens A.J. and Walsh M.J. (1966): *Artificial Intelligence Through Simulated Evolution*. — New York: Wiley.

Goldberg D.E. (1989): *Genetic Algorithms in Search, Optimization and Machine Learning*. — Reading: Addison-Wesley.

Guidorzi R.P. and Rossi P. (1974): *Identification of a power plant from normal operating records*. — Automat. Contr. Theory Applic., Vol. 2, No. 1, pp. 63–67.

Guidorzi R.P., Losito M.P. and Muratori T. (1980): *On the last eigenvalue test in the structural identification of linear multivariable systems*. — Proc. 5th Europ. Meeting *Cybernetics and Systems Research*, Vienna, pp. 217–228.

Hoai N.X., McKay R.I., Essam D. and Chau R. (2002): *Solving the symbolic regression problem with tree-adjunct grammar guided genetic programming: The comparative results*. — Proc. *2002 Congress on Evolutionary Computation*, Honolulu, USA, Vol. 2, pp. 1326–1331.

Holland J.H. (1995): *Adaptation in Natural and Artificial Systems*. — Ann Arbor: The University of Michigan Press.

Koza J.R. (1992): *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. — Cambridge: MIT Press.

Koza J.R. (1994): *Genetic Programming II: Automatic Discovery of Reusable Programs*. — Cambridge: MIT Press, 1994.

McAvoy T.J., Hsu E. and Lowenthal S. (1972): *Dynamics of pH in controlled stirred tank reactor*. — Ind. Eng. Chem. Process Des. Develop., Vol. 11, No. 1, pp. 71–78.

Moonen M., De Moor B., Vandenberghe L. and Vandewalle J. (1989): *On- and off-line identification of linear state-space models*. — Int. J. Contr., Vol. 49, No. 2, pp. 219–0232.

Rechenberg I. (1973): *Evolutionsstrategie: Optimierung Technischer Systemen nach Prinzipien der Biologischen Evolution*. — Stuttgart: Frommann-Holzboog Verlag.

Salhi A., Glaser H. and DeRoure D. (1998): *Parallel implementation of a genetic-programming based tool for symbolic regression*. — Inf. Process. Lett., Vol. 66, pp. 299–307.

Schwefel H-P. (1977): *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*. — Basel: Birkhäuser.

Shengwu X., Weinu W. and Feng L. (2003): *A new genetic programming approach in symbolic regression*. — Proc. 15th IEEE Int. Conf. *Tools with Artificial Intelligence*, Sacramento, USA, pp. 161–165.

Weigend A.S., Huberman B.A. and Rumelhart D.E. (1992): *Predicting sunspots and exchange rates with connectionist networks*, In: Nonlinear Modeling and Forecasting (S. Eubank and M. Casdagli, Eds.). — Redwood City: Addison-Wesley, pp. 395–432.

Zongker D., Punch B. and Rand B. (1998): *Lilgp 1.1 User's Manual*. — Lansing: Michigan State University.