Journal of Experimental & Theoretical Artificial Intelligence
Vol. 00, No. 00, Month-Month 200x, 1–19

Using Traceless Genetic Programming for Solving Multiobjective Optimization Problems

Mihai Oltean and Crina Groşan
Department of Computer Science
Faculty of Mathematics and Computer Science
Babeş-Bolyai University, Kogălniceanu 1
Cluj-Napoca, 3400, Romania.
{moltean,cgrosan}@cs.ubbcluj.ro

(*Received 00 Month 200x; In final form 00 Month 200x*)

Traceless Genetic Programming (TGP) is a Genetic Programming (GP) variant that is used in the cases where the focus is rather the output of the program than the program itself. The main difference between TGP and other GP techniques is that TGP does not explicitly store the evolved computer programs. Two genetic operators are used in conjunction with TGP: crossover and insertion. In this paper we shall focus on how to apply TGP for solving multiobjective optimization problems which are quite unusual for GP. Each TGP individual stores the output of a computer program (tree) representing a point in the search space. Numerical experiments show that TGP is able to solve very fast and very well the considered test problems.

1 Introduction

Koza (11; 12) suggested that Genetic Programming (GP) may be used for solving equations. In that approach (11) each GP tree represented a potential solution of the problem (equation). The internal nodes of the tree contained mathematical operators (+,–,*,/) and leaves contained real constants (randomly generated) (11). Even if this idea is very interesting and appealing, little work has been dedicated to this issue.

In this paper, we explore the possibility of encoding solutions of multiobjective problems as trees rather than as simple real values. The value of the (expression encoded into a) tree will represent the solution of the problem. Since we need the value of the tree rather than the entire tree, we will use a very fast and efficient GP variant called Traceless Genetic Programming.

Traceless Genetic Programming (TGP) [1] (16; 17) is a GP (11) variant as it evolves a population of computer programs. The main difference between TGP and GP is that TGP does not explicitly store the evolved computer programs. TGP is useful when the trace (the way in which the results are obtained) between input and output is not important. For instance, TGP is useful for solving problems where a numerical output is needed (i.e. solving equations, function optimization). In this way the space used by traditional techniques for storing the entire computer programs (or mathematical expressions in the simple case of symbolic regression) is saved.

TGP-based algorithm is applied for solving five difficult multiobjective test problems: ZDT1 - ZDT4, ZDT6 (2; 22; 23) whose definition domain consists of real-valued variables. We employ two variants of TGP. The first variant does not include an archive for storing the nondominated solutions already found. This very fast variant is able to detect in just one second a very good approximation of the Pareto front, but the diversity of the solutions is sometimes poor. A second variant maintains an archive for a better diversity of solutions. In this variant, the distribution along the Pareto front has been significantly improved.

The TGP-based algorithm is compared to Strength Pareto Evolutionary Approach (SPEA) (22; 23) and Pareto Archive Evolutionary Algorithm (PAES) (9; 10). Results show that TGP is significantly better than the other considered algorithms for most of the test functions.

[1] The source code for TGP is available at www.tgp.cs.ubbcluj.ro.

The paper is organized as follows: In section 2 the Traceless Genetic Programming technique is described in the context of GP problems. A possible field of applications for TGP is described in section 2.9. Section 3 describes TGP in the context of multiobjective evolutionary algorithms. The test problems are briefly presented in section 4. In section 6 several numerical experiments for solving the multiobjective problems are performed. An improved TGP algorithm with archive is introduced in section 7. Running time of the TGP algorithm is analyzed in section 8. TGP is compared to SPEA and PAES in section 9. Conclusions and further work directions are indicated in section 11.

2 Traceless Genetic Programming

In this section the TGP technique (16) is described in the context of Genetic Programming techniques and problems. Later, in section 3, TGP is presented in the context of numerical problems such as solving equations, function optimization and multiobjective function optimization.

2.1 *Prerequisite*

The quality of a GP individual is usually computed by using a set of fitness cases (11; 12). For instance, the aim of symbolic regression is to find a mathematical expression that satisfies a set of m fitness cases.

We consider a problem with n inputs: x_1, x_2, $\ldots x_n$ and one output f. The inputs are also called terminals (11). The function symbols that we use for constructing a mathematical expression are $F = \{+, -, *, /, sin\}$.

Each fitness case is given as a $(n+1)$ dimensional array of real values:

$$v_1^k, v_2^k, v_3^k, ..., v_n^k, f_k$$

where v_j^k is the value of the j^{th} attribute (which is x_j) in the k^{th} fitness case and f_k is the output for the k^{th} fitness case.

More fitness cases (denoted by m) are usually given and the task is to find the expression that best satisfies all these fitness cases. This is usually done by minimizing the difference between what we have obtained and what we should obtain:

$$Fitness = \left\| \begin{pmatrix} o_j^1 \\ o_j^2 \\ o_j^3 \\ ... \\ o_j^m \end{pmatrix} - \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ ... \\ f_m \end{pmatrix} \right\|.$$

By linearizing the formula above we obtain:

$$Q = \sum_{k=1}^{m} |f_k - o_k|,$$

where f_k is the target value for the k^{th} fitness case and o_k is the actual (obtained) value for the k^{th} fitness case.

2.2 *Individual representation*

Each TGP individual represents a mathematical expression evolved so far, but the TGP individual does not explicitly store this expression. Each TGP individual stores only the value already obtained for each

fitness case. Thus a TGP individual is:

$$\begin{pmatrix} o_k^1 \\ o_k^2 \\ o_k^3 \\ \dots \\ o_k^m \end{pmatrix}$$

where o_k is the current value for the k^{th} fitness case. Each position in this array (a value o_k) is a gene. As said before, behind these values lies a mathematical expression whose evaluation has generated these values. However, we do not store this expression. We only store the values o_k.

Remark

(i) The structure of a TGP individual can be easily improved so that it could store the evolved computer program (mathematical expression). Storing the evolved expression can provide an easier way to analyze the results of the numerical experiments. However, in this paper, we shall not store the trees associated with the TGP individuals.

(ii) TGP cannot be viewed as a GP technique with linear representation (15) mainly because TGP does not explicitely store the entire computer program (the tree).

2.3 *Initial population*

The initial population contains individuals whose values have been generated by simple expressions (made up of a single terminal). For instance, if an individual in the initial population represents the expression:

$$E = x_j,$$

then the corresponding TGP individual is represented as:

$$\begin{pmatrix} v_j^1 \\ v_j^2 \\ v_j^3 \\ \dots \\ v_j^m \end{pmatrix}$$

where v_j^k has been previously explained.

The quality of this individual is computed by using the equation previously described:

$$Q = \sum_{i=1}^{m} \left| v_1^k - f_k \right|.$$

2.4 *Genetic Operators*

The genetic operators which are used in conjunction with TGP are described in this section. TGP uses two genetic operators: crossover and insertion. These operators are specially designed for the TGP technique.

2.4.1 ***Crossover.*** The crossover is the only variation operator that creates new individuals. Several individuals (the parents) and a function symbol are selected for crossover. The offspring is obtained by

applying the selected operator to each of the genes of the parents.

Speaking in terms of expressions and trees, an example of TGP crossover is depicted in Figure 1.

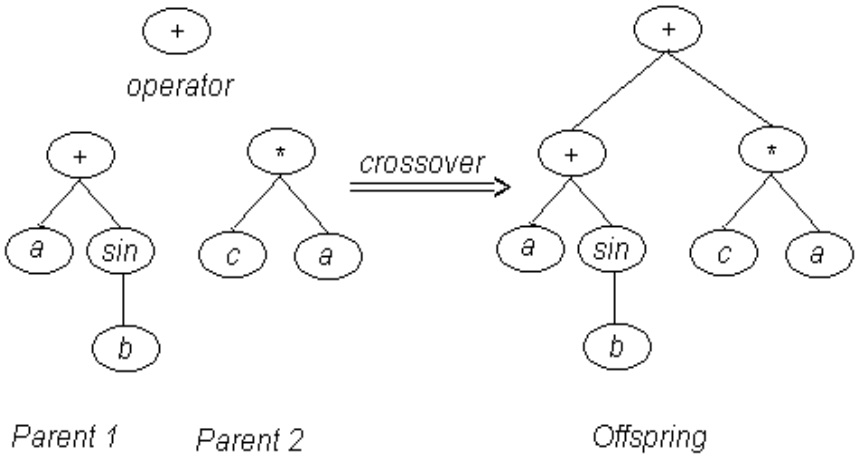

Figure 1. An example of TGP crossover.

From Figure 1 we can see that the parents are subtrees of the offspring.

The number of parents selected for crossover depends on the number of arguments required by the selected function symbol. Two parents have to be selected for crossover if the function symbol is a binary operator. A single parent has to be selected if the function symbol is a unary operator.

Example 1

Let us suppose that the operator + is selected. In this case two parents are selected and the offspring O is obtained as follows:

$$\begin{pmatrix} p_j^1 \\ p_j^2 \\ p_j^3 \\ \dots \\ p_j^m \end{pmatrix} \quad operator + \quad \begin{pmatrix} q_j^1 \\ q_j^2 \\ q_j^3 \\ \dots \\ q_j^m \end{pmatrix} \overset{Crossover}{\longrightarrow} \begin{pmatrix} p_j^1 + q_j^1 \\ p_j^2 + q_j^2 \\ p_j^3 + q_j^3 \\ \dots \\ p_j^m + q_j^m \end{pmatrix}$$

Example 2

Let us suppose that the operator *sin* is selected. In this case one parent is selected and the offspring O is obtained as follows:

$$\begin{pmatrix} p_j^1 \\ p_j^2 \\ p_j^3 \\ \dots \\ p_j^m \end{pmatrix} \quad operator \ \sin \overset{Crossover}{\longrightarrow} \begin{pmatrix} \sin(p_j^1) \\ \sin(p_j^2) \\ \sin(p_j^3) \\ \dots \\ \sin(p_j^m) \end{pmatrix}$$

Remark Standard GP crossover (11) cannot be simulated within TGP mainly because TGP does not store the entire tree and thus no subtree can be extracted.

2.4.2 ***Insertion.*** This operator inserts a simple expression (made up of a single terminal) in the population. This operator is useful when the population contains individuals representing very complex expressions that cannot improve the search. By inserting simple expressions we give a chance to the evolutionary process to choose another direction for evolution. Insertion is very similar to a restart operation.

2.5 *Constants within TGP system*

Some numerical constants might appear within the solution when solving symbolic regression problems. Constants are handled as any other variable. There are many ways to insert constants within TGP. One can use the same value for all the fitness cases, or one can generate different random values for each fitness case. An example of a constant chromosome is given below (values have been randomly generated):

$$\begin{pmatrix} 0.7 \\ 2.1 \\ 0.001 \\ ... \\ 3.4 \end{pmatrix}$$

The use of constants is important in solving real-world classification and symbolic regression problems (19; 24) where the evolved expression is more than a simple combination of inputs (1).

The use of constants becomes even more important in the case of function optimization (see the sections below) since, in this case, we don't have any input variables (as those supplied in the case of symbolic regression or classification). In this case all the solutions are made up of (combinations of) random numbers.

2.6 *TGP Algorithm*

TGP uses a special generational algorithm by using two populations, which is given below:

The TGP algorithm starts by creating a random population of individuals. The evolutionary process is run for a fixed number of generations. At the beginning of a generation the best individual is automatically copied into the next population. Then the following steps are repeated until the new population is filled: we may either choose to perform crossover between two existing individuals or to insert a randomly generated individual. Thus, with a probability p_{insert} we generate an offspring made up of a single terminal (see the Insertion operator). With a probability 1-p_{insert} select two parents using a standard selection procedure. The parents are recombined in order to obtain an offspring. The offspring enters the population of the next generation.

The standard TGP algorithm is depicted in Figure 2.

2.7 *Complexity of the TGP Decoding Process*

A very important aspect of the GP techniques is the time complexity of the procedure used for computing the fitness of the newly created individuals.

The complexity of that procedure for the standard GP is:

$$O(m * g),$$

where m is the number of fitness cases and g is average number of nodes in the GP tree.

By contrast, the TGP complexity is only

$$O(m)$$

because the quality of a TGP individual can be computed by traversing it only once. The length of a TGP individual is m.

STANDARD TGP ALGORITHM

```
S1.  Randomly create the initial population P(0)
S2.  for t = 1 to NumberOfGenerations do
S3.    P'(t) = ϕ
S4.    Copy the best individual from P(t) to P'(t)
S5.    while P'(t) is not filled do
S6.      with a fixed insertion probability p_insert do
             Create an offspring offspr made up of a single terminal
S7.      with a crossover probability 1 - p_insert do
S8.        Select an operator.
S9.        Select a number of parents equal to the
               number of arguments of the selected operator.
S10.       Crossover the selected parents.
             An offspring offspr is obtained
S11.     Add the offspring o to P'(t)
S12.   endwhile
S13.   P(t+1) = P'(t);
S14. endfor
```

Figure 2. Traceless Genetic Programming Algorithm.

Due to this reason we may allow TGP programs to run g times more generations in order to obtain the same complexity as the standard GP.

2.8 *Limitations of the TGP*

Despite its great speed, TGP has some drawbacks which limit its area of applicability.

In a standard GP system, the evolved tree could be used for analyzing new test data. However, in the TGP system the things are a little bit different. TGP purpose is to avoid storing of the evolved trees in order to achieve speed. This means that the evolved solutions cannot be applied for new and unseed test data. An immediate consequence is that both the training and the test must be known at the moment of training.

2.9 *When TGP is useful?*

As stated in section 2.8, TGP cannot be applied for solving symbolic regression or classification problems as we know them. Both the training and the test must be known at the moment of training. And, this is a hard constrain which reduce the space of applicability. After training, TGP cannot be used for analyzing new and unseen data.

But, TGP can be used in other, more subtle, applications:

For instance when we apply a standard Genetic Programming technique we need to know what is the optimal set of functions and terminals. If we run GP with an improper set of functions or terminals we will need a lot of generations and a big population.

We can use TGP for tuning the set of functions/terminals. Before running the standard GP we could perform many runs with TGP in order to find a good set of functions/terminals. The TGP will not give us a tree or a mathematical expression, but it will give us, very quickly, an indication on whether the selected set of functions and terminals can lead to a good solution of that problem if we apply standard GP.

Note that TGP is very fast. It solved (16) the even-5-parity problem in 3.2 seconds with no asm or other tricks. This time is infinitely smaller than the time required by a standard GP system to solve this problem. Thus, TGP can be run multiple times in order to see if a set of functions/terminals is good enough for obtaining a good solution with standard GP.

TGP can act as a kind of very fast preprocessing. TGP will not output a tree, but it will tell you very fast if the selected parameters for your algorithm are good or you have to change them before running the entire standard GP system.

We can significantly improve the speed of TGP if we use:

- the Submachine Code GP (18). The authors offered that SubMachine Code improved the speed by 1.8 orders of magnitude. We expect to improve the speed of TGP by a similar order of magnitude.
- implementation in machine language as in the case of Discipulus (14). The authors offered that Discipulus is sixty to two hundreds faster than standard Genetic Programming.

Other potentially interesting applications for TGP are currently under consideration: solving single and multi-objective optimization problems and solving equations.

3 TGP for multiobjective problems

The following modifications have been applied to the Traceless Genetic Programming algorithm in order to deal with specific multiobjective problems:

(i) In the case of the considered multiobjective problems (2) we do not have a training set as in the case of GP. The TGP chromosome will still be an array of values, but each value will represent a variable of the search space. Thus, the chromosome length will be equal to the number of variables of the considered multiobjective test problem. In our test problems (2) each variable will usually hold a value in a real interval (e.g. [0,1], or [-5,5]). This modification is more a formal than a structural one.

(ii) The initial population will consist of vectors of constants (see section 2.5). Each value in these arrays will be randomly generated over the definition domain of the problem being solved. The subsequent populations will be combinations of these constants or will be replaced, by insertion (see section 2.4.2), with some other vectors of constants.

(iii) The fitness of a TGP chromosome for multiobjective problems is computed by applying it to the considered test problems. For each objective we will have a fitness function. This is different from the standard TGP where we had a single fitness which was computed as a sum over all fitness cases.

(iv) All the nondominated solutions from the current generation will be copied into the next generation. This is again different from the standard TGP where only the best solution in the current population was copied into the next generation.

3.1 *TGP algorithm for multiobjective optimization problems*

The algorithm used for solving multiobjective optimization problems is very similar to those used in the context of GP problems (see section 2.6). The main difference is that all the nondominated solutions from the current population are automatically copied into the next generation. The algorithm may be described as follows:

The initial population is randomly generated and the entire evolutionary process is run for a fixed number of generations. At each generation the nondominated solutions from the current population are automatically copied into the next population. Then, the following steps are repeated until the new population is filled: we may either choose to perform crossover between two existing individuals or to insert a randomly generated individual. Thus, with a probability p_{insert} generate an offspring made up of a single terminal (see the Insertion operator). With a probability 1-p_{insert} select parents using a standard selection procedure. The parents are recombined (see section 2.4.1) in order to obtain an offspring. The offspring enters the new population. At the end of the generation, the new population is copied into the old one.

4 Test problems

The five test functions (2; 23) used in this paper for numerical experiments are described in this section.

Each test function is built by using three functions f_1, g, h. Biobjective function T considered here is:

$T(x) = (f_1(x), f_2(x))$.

The optimization problem is:

$$\begin{cases} Minimize\, T(x),\ where\, f_2(x) = g(x_2, \ldots, x_m)h(f_1(x_1),\, g(x_2, \ldots x_m)), \\ x = (x_1, \ldots, x_m) \end{cases}$$

4.1 *Test function ZDT1*

Test function ZDT_1 is defined by using the following functions:

$f_1(x_1) = x_1$,
$g(x_2, \ldots .x_m) = 1 + 9 \cdot \sum_{i=2}^{m} x_i/(m-1)$,
$h(f_1, g) = 1 - \sqrt{f_1/g}$,
where $m = 30$ and $x_i \in$ [0,1] i = 1,2,. . . ,m.

Pareto optimal front for the problem ZDT_1 is convex and is characterized by the equation $g(x) = 1$.

4.2 *Test function ZDT2*

Test function ZDT_2 is defined by considering the following functions:

$f_1(x_1) = x_1$
$g(x_2, \ldots .x_m) = 1 + 9 \cdot \sum_{i=2}^{m} x_i/(m-1)$
$h(f_1, g) = 1 - (f_1/g)^2$
where $m = 30$ and $x_i \quad \in$ [0,1], i = 1,2,. . . ,m.

Pareto optimal front is characterized by the equation $g(x)$=1.
ZDT_2 is the nonconvex counterpart to ZDT_1.

4.3 *Test function ZDT3*

Pareto optimal set corresponding to the test function ZDT_3 presents a discrete feature. Pareto optimal front consists of several noncontiguous convex parts. The involved functions are:

$f_1(x_1) = x_1$
$g(x_2, \ldots .x_m) = 1 + 9 \cdot \sum_{i=2}^{m} x_i/(m-1)$
$h(f_1, g) = 1 - \sqrt{f_1/g} - (f_1/g)\sin(10\pi f_1)$
where $m = 30$ and $x_i \quad \in$ [0,1], i = 1,2,. . . m.

Pareto optimal front is characterized by the equation $g(x) = 1$.

4.4 *Test function ZDT4*

The introduction of the function *sin* in the expression of function h causes discontinuity in the Pareto optimal front. However, there is no discontinuity in the parameter space.

The test function ZDT_4 contains 21^9 local Pareto optimal fronts and, therefore, it tests the EA ability to deal with multimodality. The involved functions are defined by:

$f_1(x_1) = x_1$
$g(x_2, \ldots .x_m) = 1 + 10(m-1) + \sum_{i=2}^{m} (x_i^2 - 10\cos(4\pi x_i))$
$h(f_1, g) = 1 - \sqrt{f_1/g}$
where $m = 10$, $x_1 \quad \in$ [0,1] and x_2,. . . ,$x_m \quad \in$ [-5,5].

Global Pareto optimal front is characterized by the equation $g(x) = 1$.
The best local Pareto optimal front is described by the equation $g(x) = 1.25$.
Note that not all local Pareto optimal sets are distinguishable in the objective space.

4.5 *Test function ZDT6*

The test function ZDT_6 includes two difficulties caused by the nonuniformity of the search space. First of all, the Pareto optimal solutions are nonuniformly distributed along the global Pareto optimal front (the front is biased for solutions for which $f_1(x)$ is a neat one). Secondly, the density of the solutions is lowest near the Pareto optimal front and highest away from the front.

This test function is defined by using:

$f_1(x_1) = 1 - \exp(-4x_1)\sin^6(6\pi x_1)$
$g(x_2, \ldots .x_m) = 1 + 9 \cdot (\sum_{i=2}^{m} x_i/(m-1))^{0.25}$
$h(f_1, g) = 1 - (f_1/g)^2$
where $m = 10$, $x_i \quad \in [0,1]$, $i = 1,2,\ldots m$.

The Pareto optimal front is characterized by the equation $g(x) = 1$, and is nonconvex.

5 Metrics of performance

Both metrics measure the convergence to the Pareto front and can be applied only if the Pareto front is known.

5.1 *Convergence metric*

Assume that the Pareto front is known. Let us denote by PA a set of Pareto optimal solutions. For each individual i from the final population FP distance (Euclidian distance or other suitable distance) d_{ij} to the all points j of P is computed.

The minimum distance:

$$mindist_i = \min_{j \in P} d_{ij}$$

is kept for each individual.

The average of these distances

$$CM = \frac{\sum_{i \in FP} \mathrm{mindist}_i}{|FP|}$$

represents the measure of convergence (the distance) to the Pareto front.

Remark
Lower values of the convergence metric represent good convergence.

5.2 *Diversity metric*

For each individual from the final population FP we consider the point from the set of Pareto optimal points P situated at minimal distance.

We called each such point from PA a *marked* point. The total number of different marked points from PA over the size of PA represents the diversity metric (DM).

Remark

(i) Higher values for DM indicate a better diversity.
(ii) We have considered 200 equidistant points on the Pareto front in order to compute the diversity metric. The value of diversity metric has been normalized by dividing it by 200.

6 Numerical experiments

Several numerical experiments using TGP are performed in this section by using the previously described multiobjective test problems. The general parameters of the TGP algorithm are given in Table 1. Note that the population size and the number of generations have been chosen as described in (9; 22) in order to provide a fair comparison of the methods.

Table 1. General parameters of the TGP algorithm for solving parity problems.

Parameter	**Value**
Population Size	100
Number of Generations	250
Insertion probability	0.05
Selection	Binary Tournament
Function set	$+$, $-$, $*$, sin, exp
Terminal set	Problem variables
Number of runs	30

Because we are dealing with real-valued definition domains (e.g. [0,1]) we have to find a way of protecting against overflowing these. For instance if we employ a standard addition of two numbers greater than 0.5 we would get a result greater than 1 (domain upper bound). Thus, each operator has been redefined in order to output result in the standard interval [0,1]. The redefinitions are given in Table 2.

Table 2. The redefinitions of the operators so that the output should always be between 0 and 1 if the inputs are between 0 and 1.

Operator	Redefinition
+	(x + y)/2
-	|x-y|
*	**None** (If you multiply two numbers between 0 and 1 you will always get a number between 0 and 1.
sin	sin(x)/sin(1)
exp	exp(x)/exp(1)

Thirty independent runs have been performed. The results are depicted in Figure 3.

The convergence metric, computed at every 10 generations, is depicted in Figure 4. Note that only nondominated solutions have been taken into account for metrics computing.

The diversity metric, computed at every 10 generations, is depicted in Figure 5.

Figures 3, 4 and 5 show that TGP is able to provide a very good convergence towards the Pareto front for all the considered test problems (five). Unfortunately, the diversity of the solutions is sometimes very poor (see test functions ZDT2 and ZDT4). We have depicted the results obtained in a single run in order to provide a better view of the solution diversity. The same poor diversity has been observed in all other runs. However, the convergence of the solutions is very good, which makes us trust the power of the TGP method. This is why we have made a further step for improving this method, by adding a diversity preserving mechanism.

Numerical values of the convergence and diversity metrics for the last generation are also given in section 9.

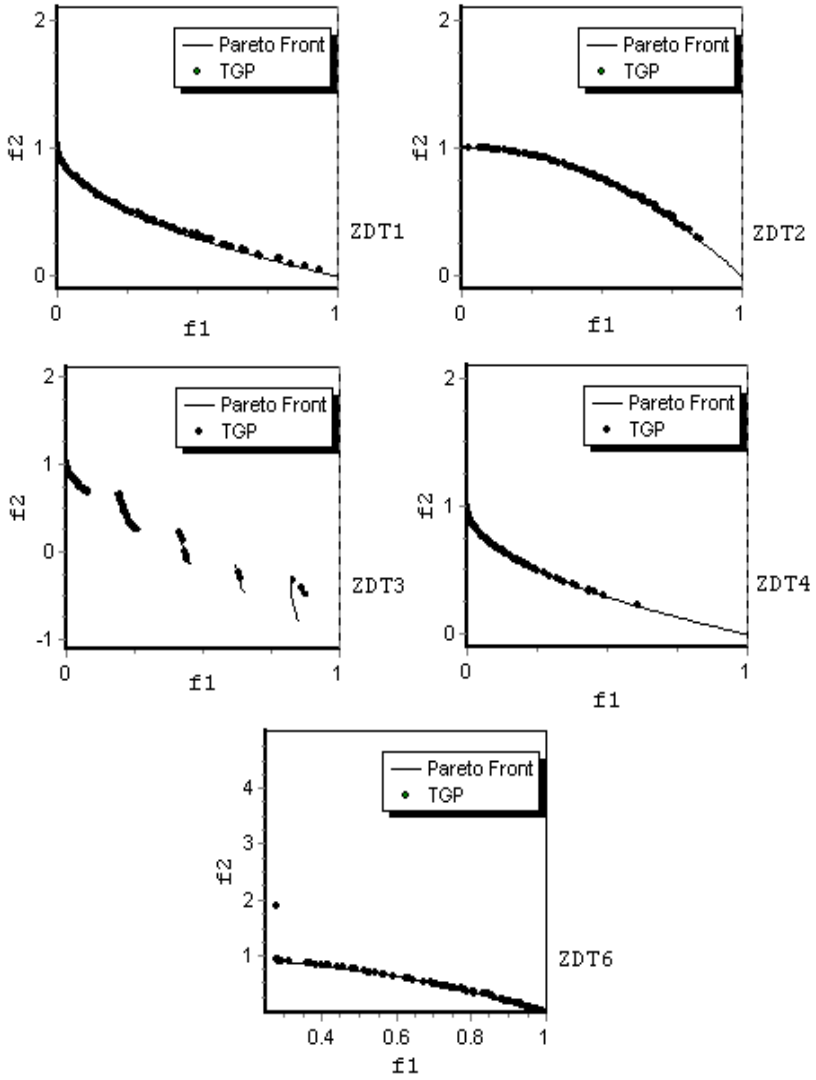

Figure 3. The results obtained by TGP without an archive for the test functions ZDT1-ZDT4, ZDT6. We have depicted the results obtained in one randomly chosen run (out of 30) in order to emphasize the poor distribution along the Pareto front.

7 Improving diversity by using an archive

The diversity of the obtained solutions is sometimes low (see Figure 3). The solutions tend to accumulate on the left side of the Ox axis. This shortcoming could be due to the use of some of the operators given in Table 2. For instance, the result of the multiplication operator will always be lower than the value of its operands, thus generating a tendency toward the left side of the Ox axis. We have tried to remove this operator from the function set, but we have obtained even worse results. It seems that the set of operators used in conjunction with TGP could sometimes affect the quality of the obtained solutions.

In order to improve the diversity of solutions we have added to our algorithm, an archive which stores the nondominated solutions found so far. The archive is updated after each generation so that it contains the nondominated solutions from the new population and, of course, those from the old archive. If the number of the nondominated solutions exceeds the size of the archive the following steps are repeated until we obtain the required number of solutions in archive: the closest two solutions are computed and one of them is removed. This mechanism helps us maintain a reasonable diversity among the solutions in archive.

The selection for crossover is uniformly performed in the archive and in the current population. Individuals are randomly chosen with no emphasis on the nondominated ones. The nondominated individuals from the current population are not copied into the next population any longer.

The algorithm is depicted in Figure 6.

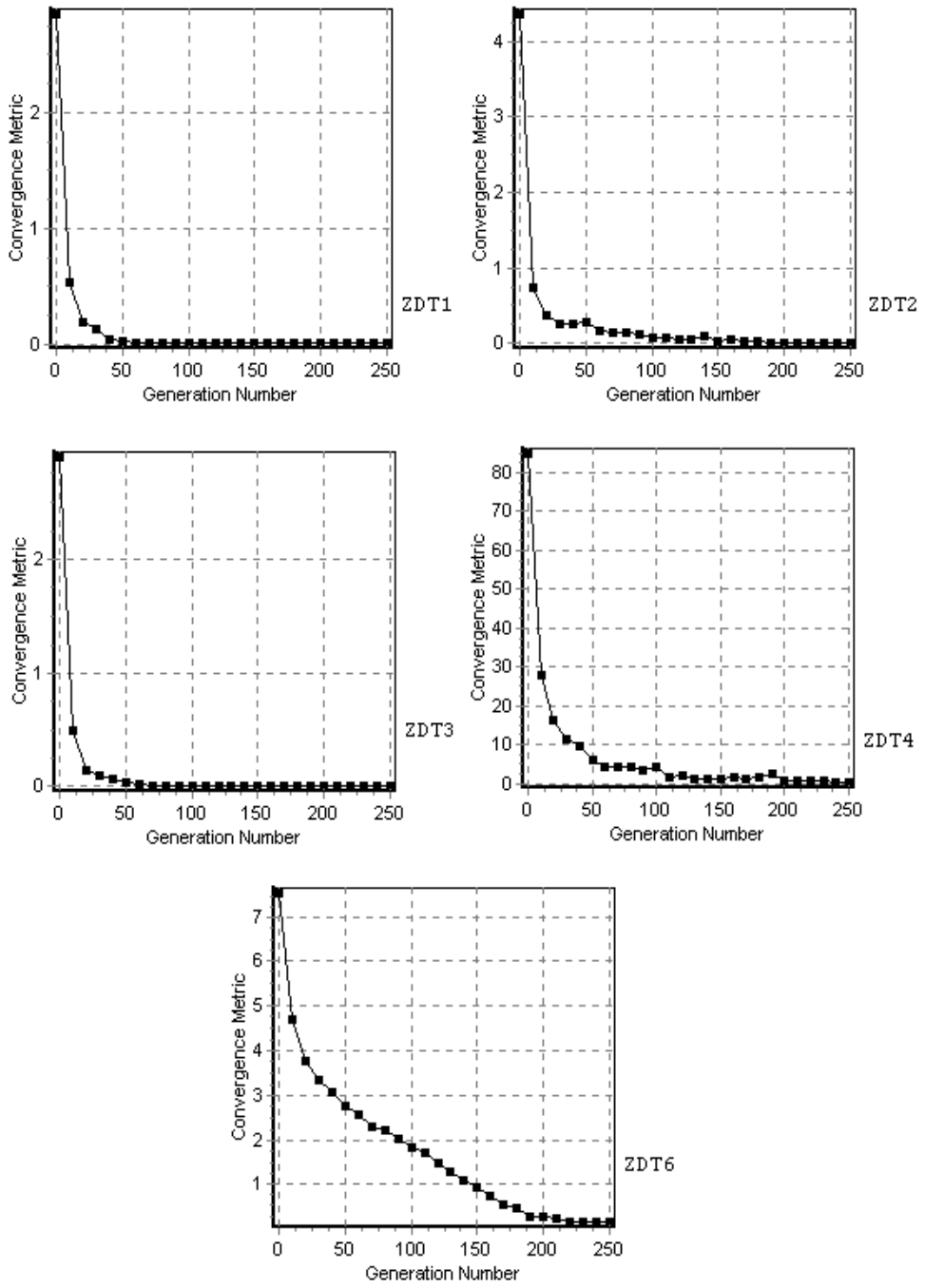

Figure 4. Convergence metric computed at every 10 generations. The results are averaged over 30 independent runs.

Thirty independent runs have been performed again for the test functions ZDT1-ZDT4, ZDT6. The results are depicted in Figure 7.

Figure 7 shows that the algorithm achieved both a very good convergence and a very good diversity. The use of the archive has proven to be essential for preserving the diversity among the solutions.

We have computed again the value of the convergence and diversity metrics in order to provide a better view of the newly introduced algorithm. The convergence metric, computed at every 10 generations, is depicted in Figure 8.

The diversity metric, computed at every 10 generations, is depicted in Figure 9.

Figure 8 shows that the TGP-based algorithm quickly converges to the Pareto front. Less than 100 generations are required in order to obtain a very good convergence for the test functions ZDT1-ZDT3. A slower convergence is obtained for the test function ZDT6, thus proving the difficulty of this problem (4).

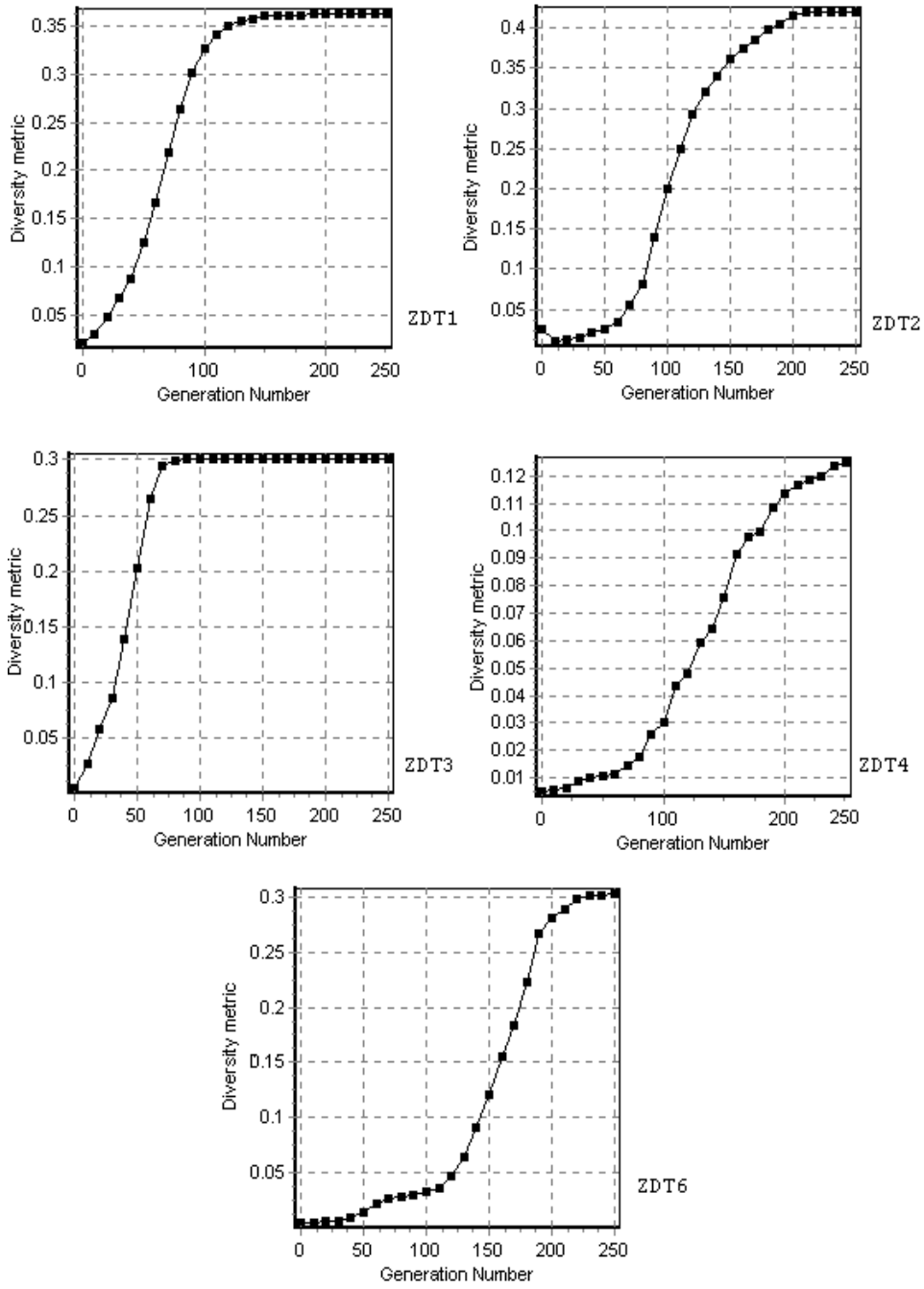

Figure 5. Diversity metric computed at every 10 generations. The results are averaged over 30 independent runs.

Numerical values of the convergence and diversity metrics for the last generation are also given in section 9.

8 Running time

Table 3 is meant to show the effectiveness and simplicity of the TGP algorithm by giving the time needed for solving these problems using a PIII Celeron computer at 850 MHz.

Table 3 shows that TGP without archive is very fast. An average of 0.4 seconds is needed in order to obtain a solution for the considered test problems. However, the time needed to obtain a solution increases with at least one order of magnitude when an archive is used.

TGP WITH ARCHIVE

```
S1.  Randomly create the initial population P(0)
S2.  Store the nondominated solutions in Archive
S3.  for t = 1 to NumberOfGenerations do
S4.    P'(t) = ϕ
S5.    while P'(t) is not filled do
S6.      with a fixed insertion probability p_insert do
S7.         Create an offspring offspr made up of constants only
S8.      with a crossover probability 1 - p_insert do
S9.        Select an operator.
S10.       Select a number of parents from the Archive ∪ P(t) equal to the
             number of arguments of the selected operator.
S11.       Crossover the selected parents. An offspring offspr is obtained
S12.     Add the offspring o to P'(t)
S13.   endwhile
S14.   P(t+1) = P'(t);
S15.   Keep the nondominated solutions, found so far, in Archive
S16. endfor
```

Figure 6. Traceless Genetic Programming with archive.

Table 3. The average time for obtaining a solution for the multi-objective optimization problems by using TGP. The results are averaged over 30 independent runs.

Problem	Time (seconds) without archive	Time (seconds) with archive
ZDT1	0.4	19.1
ZDT2	0.4	11.4
ZDT3	0.4	7.5
ZDT4	0.3	15.1
ZDT6	0.3	3.2

9 Comparison with other methods

We compare the results obtained by TGP with the results obtained by two other well-known techniques: SPEA (22) and PAES (9). We have chosen these two methods because of their popularity in the field of MOEAs. Also, the results obtained by running these techniques (for the considered test problems) have been provided by the authors on the Internet (25; 26).

Results of the convergence and diversity metrics are given in Tables 4 and 5.

Table 4. The convergence metric. The results are averaged over 30 independent runs. Lower values are better.

Problem	TGP	TGP with archive	SPEA	PAES
ZDT1	0.010	0.004	0.039	0.135
ZDT2	0.006	0.005	0.069	0.044
ZDT3	0.005	0.004	0.018	0.060
ZDT4	0.331	0.055	4.278	12.41
ZDT6	0.161	0.194	0.484	0.149

Tables 4 and 5 show that the TGP-based algorithm is able to solve the considered test problems very well, outperforming both SPEA and PAES. The only exception is at the ZDT6 test function where PAES converges better than TGP. The convergence of TGP is better with one order of magnitude than the convergence of SPEA and PAES on the test function ZDT4.

The diversity of the solutions produced by TGP is better than the diversity of SPEA and PAES for all

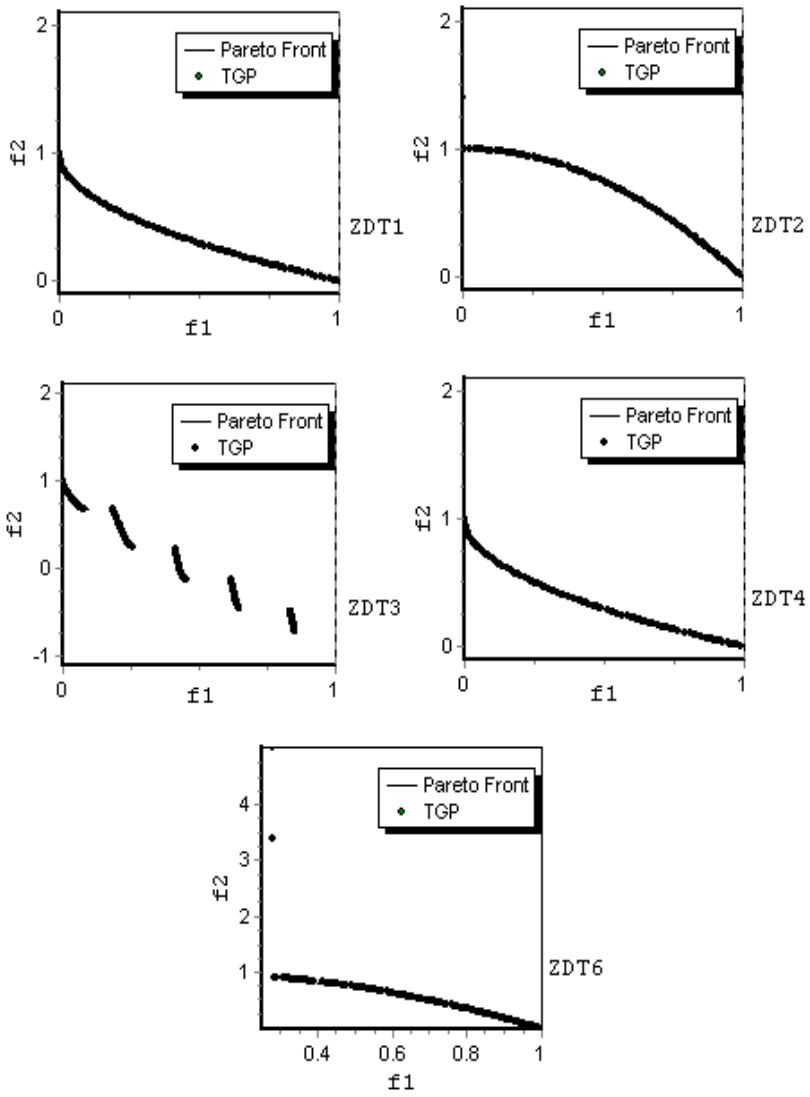

Figure 7. The results obtained by TGP with an archive for the test functions ZDT1-ZDT4, ZDT6. Only the results of a randomly chosen run have been depicted in order to provide a view of the solution diversity.

Table 5. The diversity metric. The results are averaged over 30 independent runs. Higher values are better.

Problem	TGP	TGP with archive	SPEA	PAES
ZDT1	0.34	0.465	0.299	0.213
ZDT2	0.419	0.478	0.196	0.213
ZDT3	0.300	0.399	0.159	0.151
ZDT4	0.124	0.312	0.005	0.005
ZDT6	0.302	0.253	0.030	0.153

test functions. In the case of test function ZDT4 the diversity of TGP is better with at least one order of magnitude than the diversity of PAES and SPEA.

The use of an archive has improved the diversity of TGP for the test functions ZDT1-ZDT4. TGP without archive has a better diversity than its counterpart with an archive for the test function ZDT6.

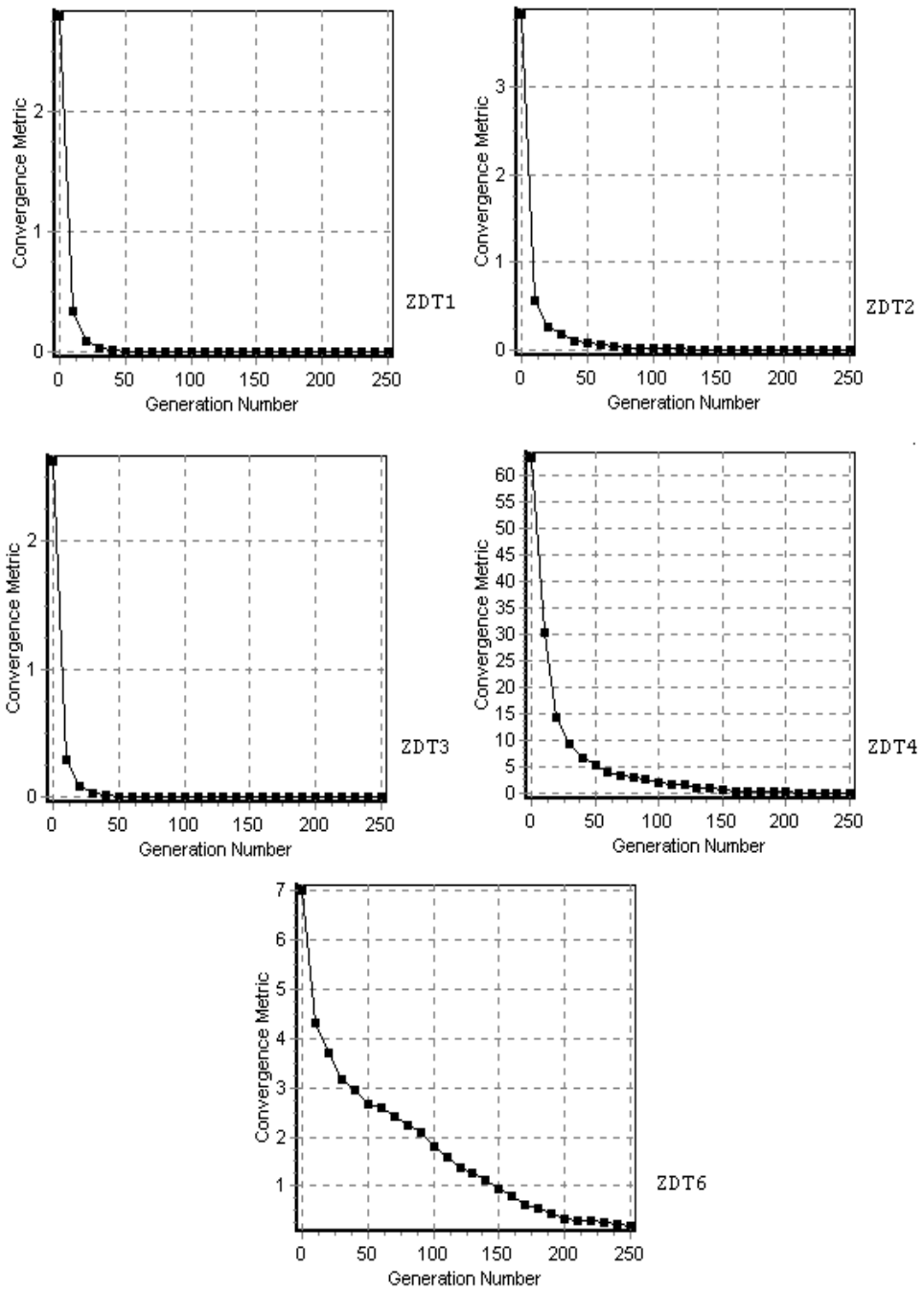

Figure 8. The convergence metric computed at every 10 generations for the TGP with archive. The results are averaged over 30 independent runs.

10 Further work

TGP with real-valued individuals has been applied (results not shown) for solving the multiobjective optimization problems (3; 7) having a scalable number of objectives. The preliminary results indicated that real-valued encoding and the associated genetic operators described in this paper are not poweful enough for solving these problems better than other techniques. More sophisticated genetic operations should be used.

We performed other experiments using binary encoding. In this case the results produced by our TGP algorithm have been significantly improved. They are competing the results produced by NSGA II (5; 7). More work should be done in this direction specially for tuning the parameters of the TGP. Also, binary representation (8) requires more investigation within the TGP framework.

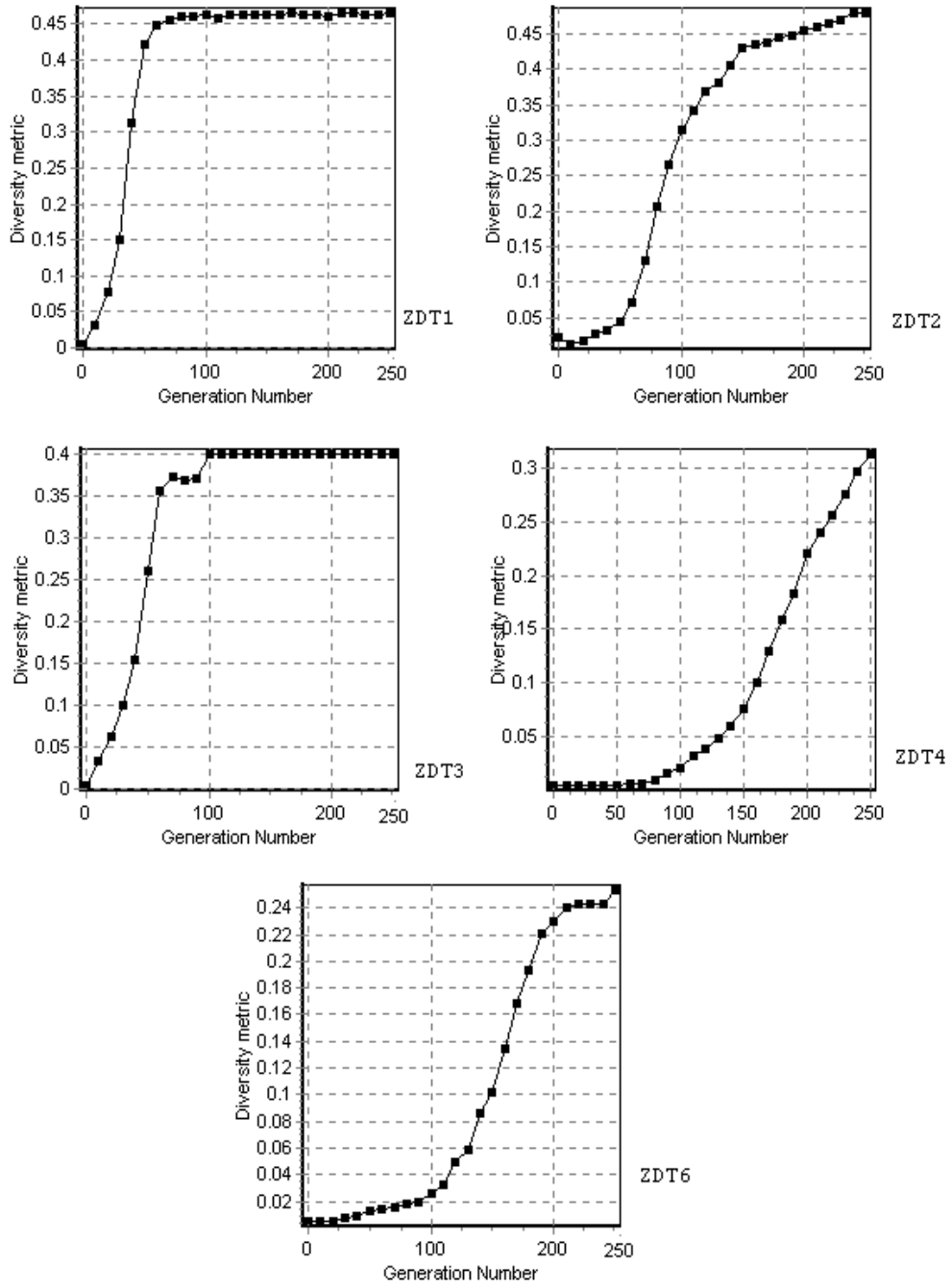

Figure 9. The diversity metric computed at every 10 generations for the TGP with archive. Results are averaged over 30 independent runs.

A more detailed analysis of the operators used in conjunction with TGP will be performed in the near future. In this paper we have used some basic operators $(+, -, *, /, sin, exp)$, but the choice of the function set may affect the quality of the obtained solutions. According to the No Free Lunch (NFL) theorems (20) we cannot expect to find an optimal set of functions which will perform in the best way for all the optimization problems.

TGP will also be used for solving single objective (21) and for solving equations.

11 Conclusions

Traceless Genetic Programming is a recent evolutionary technique which can be used for solving problems specific to GP and problems specific to GA. TGP uses a special individual representation, specific genetic operators and a specific evolutionary algorithm.

In this paper, two TGP variants have been used for solving difficult multiobjective problems. Numerical experiments have shown that TGP is able to evolve a solution for the problems within a very short period of time. TGP-based algorithms have outperformed SPEA and PAES on the considered test problems.

References

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

[2] Deb K., Multi-objective genetic algorithms: Problem difficulties and construction of test problems. Evolutionary Computation, 7(3), pp. 205-230, 1999.

[3] Deb K., Thiele L., Laumanns M. and Zitzler E., Scalable Test Problems for Evolutionary Multi-Objective Optimization, Proceedings of the Congress on Evolutionary Computation (CEC-2002), pp. 825-830, 2002.

[4] Deb K., Jain S., Running performance metrics for evolutionary multiobjective optimization, Technical Report 2002004, KanGAL, Indian Institute of Technology, Kanpur, India, 2002.

[5] Deb K., Agrawal S., Pratap A., and Meyarivan T., A fast elitist nondominated sorting genetic algorithm for multi-objective optimization: NSGA-II, In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J.Merelo, and H.-P. Schwefel (Eds.), Parallel Problem Solving from Nature - PPSN VI, Berlin, pp. 849-858, Springer-Verlag, 2000.

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

[7] Khare V., Yao X. and Deb K., Performance Scaling of Multi-objective Evolutionary Algorithms, In Carlos M. Fonseca, Peter J. Fleming, Eckart Zitzler, Kalyanmoy Deb and Lothar Thiele (editors): Proceedings of the 2^{nd} International Conference on Evolutionary Multi-Criterion Optimization (EMO03), Lecture Notes in Computer Science, pp. 376-390, Springer-Verlag, 2003.

[8] Holland, J. H., Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, 1975.

[9] Knowles J. D. and Corne D.W., The Pareto archived evolution strategy: A new baseline algorithm for Pareto multiobjective optimization. In Congress on Evolutionary Computation (CEC-99), Volume 1, Piscataway, NJ, pp. 98 - 105, 1999.

[10] Knowles J. D., Corne D.W.: Approximating the Nondominated Front using the Pareto Archived Evolution Strategies, Evolutionary Computation, Vol. 8, pp. 149-172, MIT Press, Cambridge, MA, 2000.

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

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

[13] Koza, J. R. et al., Genetic Programming III: Darwinian Invention and Problem Solving, Morgan Kaufmann, San Francisco, CA, 1999.

[14] Nordin P., A Compiling Genetic Programming System that Directly Manipulates the Machine Code, in Kenneth E. (editor), Advances in Genetic Programming, pp. 311–331, MIT Press, 1994.

[15] Oltean M., Groşan C., A Comparison of Several Linear Genetic Programming Techniques, Complex-Systems, Vol. 14, Nr. 4, pp. 282-311, 2003.

[16] Oltean M., Solving Even-Parity Problems using Traceless Genetic Programming, IEEE Congress on Evolutionary Computation, Portland, 19-23 June, edited by G. Greenwood (et. al), pp. 1813-1819, IEEE Press, 2004.

[17] Oltean M., Solving Classification Problems using Traceless Genetic Programming, World Computer

Congress, The Symposium on Profesional Practice in AI, 26-29 August, Toulouse, France, edited by E. Mercier-Laurent, J. Debenham, IFIP Press, pp. 403-412, 2004.

[18] Poli R. and Page J., Solving High-Order Boolean Parity Problems with Smooth Uniform Crossover, Sub-Machine Code GP and Demes", Journal of Genetic Programming and Evolvable Machines, Kluwer Academic Publishers, pp. 1-21, 2000.

[19] Prechelt L., PROBEN1: A Set of Neural Network Problems and Benchmarking Rules, Technical Report 21, University of Karlsruhe, 1994 (available from ftp://ftp.cs.cmu.edu/afs/cs/project/connect/bench/contrib/prechelt/ proben1.tar.gz.)

[20] Wolpert D.H., McReady W.G., No Free Lunch Theorems for Optimisation, IEEE Transaction on Evolutionary Computation, Vol. 1, pp. 67-82, 1997.

[21] Yao X., Liu Y., Lin G., Evolutionary Programming Made Faster, IEEE Transaction on Evolutionary Computation, Vol. 3(2), pp. 82-102, 1999.

[22] Zitzler E., Thiele L., Multiobjective Evolutionary Algorithms: A comparative case study and the Strength Pareto Approach, IEEE Transaction on Evolutionary Computation, 3(4), pp. 257-271, 1999.

[23] Zitzler E., Deb K., Thiele L., Comparison of multiobjective evolutionary algorithms: Empirical results. Evolutionary Computation, 8(2), Summer, pp 173-185, 2000.

[24] UCI Machine Learning Repository (available from www.ics.uci.edu/∼mlearn/MLRepository.html)

[25] Results of the running PAES for the test functions ZDT1-ZDT6. Available at www.reading.ac.uk/∼ssr97jdk/multi/PAES.html

[26] Results of the running SPEA for the test functions ZDT1-ZDT6. Available at http://www.tik.ee.ethz.ch/∼zitzler/testdata.html