2006 IEEE Congress on Evolutionary Computation
Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada
July 16-21, 2006

Human Designed Vs. Genetically Programmed Differential Evolution Operators

N.G. Pavlidis, V.P. Plagianakos, D.K. Tasoulis, and M.N. Vrahatis

Abstract— **The hybridization and combination of different Evolutionary Algorithms to improve the quality of the solutions and to accelerate execution is a common research practice. In this paper, we utilize Genetic Programming to evolve novel Differential Evolution operators. The genetic evolution resulted in parameter free Differential Evolution operators. Our experimental results indicate that the performance of the genetically programmed operators is comparable and in some cases is considerably better than the already existing human designed ones.**

I. Introduction

Evolutionary Algorithms (EAs) are problem solving optimization methods that employ computational analogs of evolutionary processes. A variety of evolutionary algorithms have been proposed; the most commonly encountered being: Genetic Algorithms [1], [2], Evolution Strategies [3], [4], Genetic Programming [5], [6], Evolutionary Programming [7], [8], Ant Colony Optimization [9], Particle Swarm Optimization [10], and the Differential Evolution algorithm [11]. EAs rely on the concept of simulating the evolution of a set of individuals that constitute a population using a set of predefined operators. Commonly two types of operators are used: *selection* and *search* operators. The most widely used search operators are *mutation* and *recombination* (*crossover*).

The Differential Evolution (DE) algorithm utilizes the mutation and recombination operators as search mechanisms, and the selection operator to direct the search towards the most promising regions of the solution space [11]. DE is capable of optimizing difficult multimodal objective functions and has been applied to a large number of different optimization tasks. It has successfully solved many artificial benchmark problems [12], as well as hard real–world problems (see for example [13]–[16]). In [17], DE has been applied to train neural networks and in [18]–[20] we have proposed an efficient method to train neural networks having arbitrary, as well as, constrained integer weights. The DE algorithm has also been implemented on parallel and distributed computers [20], [21]. It employs uniform crossover operators that can take offspring parameters from a particular parent more often than they do from others. However, the choice of the appropriate mutation operator depends on the problem at hand and it is, in general, a nontrivial task that requires experimentation. In the literature numerous mutation operators have been proposed for DE [11], [22], bearing various effects on its exploration and exploitation capabilities, and the trade off between these two.

All the authors are with the Computational Intelligence Laboratory, Department of Mathematics, University of Patras, GR–26110 Patras, Greece (email: {npav,vpp,dtas,vrahatis}@math.upatras.gr)

The hybridization and combination of different EAs to improve the quality of the solutions obtained and to accelerate execution is a common practice. In this paper, we follow a different approach. We utilize Genetic Programming (GP) to evolve novel DE mutation operators. GP is a method for automatically creating working computer programs employing principles of Darwinian evolution, and having as input a high-level statement of the problem [5]. In other words, GP aims to address the problem of automatic programming, i.e. how to enable a computer to do useful things without providing it with step by step instructions. GP aspires to induce a population of computer programs that gradually improve as they evolve and experience the data on which they are evaluated. Notice that the term GP can also include systems that constitute, or contain, explicit references to programs (executable code), or to programming language expressions [23].

In this study, we present a comparison between already known human-designed mutation operators and new genetically programmed ones. In a similar manner, in [24] and [25] the evolution of optimal equations to control the particles of PSO using genetic programming was introduced. Using a restricted set of test problems to evaluate performance, it is likely to obtain operators that appear effective but are not on broader problem sets. It is interesting to note, however, that GP managed to evolve a special case of a particularly effective DE mutation operator. A genetically evolved operator also exhibited the most robust performance.

The rest of the paper is organized as follows. In Section II the DE algorithm is briefly described and in Section III the GP algorithm is outlined. Section IV describes the proposed approach and Section V is devoted to the presentation and the discussion of the experimental results. The paper ends with concluding remarks and some pointers for future work.

II. The Differential Evolution Algorithm

Differential Evolution [11] is a minimization method, capable of handling non-differentiable, nonlinear and multimodal objective functions. To this end DE has been designed as a stochastic parallel direct search method, which utilizes concepts borrowed from the broad class of EAs. The method typically requires few, easily chosen, control parameters. Experimental results have shown that DE has good convergence properties and outperforms other well known EAs [26].

DE is a population–based stochastic algorithm that exploits a population of potential solutions, *individuals*, to effectively probe the search space. The population of individuals is randomly initialized in the optimization domain with *Np*,

n–dimensional, vectors following a uniform probability distribution. Individuals evolve over successive iterations to explore the search space and locate the minima of the objective function. The population size, *Np*, is fixed throughout the execution. At each iteration, called *generation*, new vectors are derived by the combination of randomly chosen vectors from the current population. This operation is referred to as *mutation* and produces the *mutant individuals*. Each mutant individual is then mixed with another, predetermined, vector – the *target* vector – through an operation called *recombination*. This operation yields the so–called *trial* vector. The trial vector undergoes the *selection* operator, according to which it is accepted as a member of the population of the next generation only if it yields a reduction in the value of the objective function f relative to that of the target vector.

Next, we briefly describe the human designed search operators that were considered in this paper. The search operators efficiently shuffle information among the individuals, enabling the search for an optimum to focus on the most promising regions of the solution space. The first operator considered is mutation. Specifically, for each individual x_g^i, $i = 1, \ldots, Np$, where g denotes the current generation, a new individual v_g^i (mutant individual) is generated according to one of the following equations:

$$v_g^i = x_g^{\text{best}} + F(x_g^{r1} - x_g^{r2}), \tag{1}$$
$$v_g^i = x_g^{r1} + F(x_g^{r2} - x_g^{r3}), \tag{2}$$
$$v_g^i = x_g^i + F(x_g^{\text{best}} - x_g^i) + F(x_g^{r1} - x_g^{r2}), \tag{3}$$
$$v_g^i = x_g^{\text{best}} + F(x_g^{r1} - x_g^{r2}) + F(x_g^{r3} - x_g^{r4}), \tag{4}$$
$$v_g^i = x_g^{r1} + F(x_g^{r2} - x_g^{r3}) + F(x_g^{r4} - x_g^{r5}), \tag{5}$$

where x_g^{best} is the best member of the previous generation; $F > 0$ is a real parameter, called *mutation constant*, which controls the amplification of the difference between two individuals, and is used to prevent the stagnation of the search process; and $r_1, r_2, r_3, r_4, r_5 \in \{1, 2, \ldots, i-1, i+1, \ldots, Np\}$, are random integers mutually different and not equal to the running index i.

Trying to rationalize the above equations, we observe that Eq. (2) is similar to the crossover operator employed by some Genetic Algorithms; while Eq. (1) is derived from Eq. (2), by substituting the best member of the previous generation, x_g^{best}, for the random individual x_g^{r1}. Eqs. (3), (4) and (5) are modifications obtained by the combination of Eqs (1) and (2). Clearly, more such relations can be generated using the above ones as building blocks. For example, the recently proposed trigonometric mutation operator [22] performs a mutation according to the following equation, with probability τ_μ:

$$v_g^i = (x_g^{r1} + x_g^{r2} + x_g^{r3})/3 + (p_2 - p_1)(x_g^{r1} - x_g^{r2}) + \\ + (p_3 - p_2)(x_g^{r2} - x_g^{r3}) + (p_1 - p_3)(x_g^{r3} - x_g^{r1}), \tag{6}$$

while, with probability $(1 - \tau_\mu)$, the mutation is performed according to Eq. (2). Here, τ_μ is a user defined parameter, typically set around 0.1. The values of p_m, $m = \{1, 2, 3\}$ and p' are obtained through the following equations:

$$\begin{aligned} p_1 &= \left|f(x_g^{r1})\right| / p', \\ p_2 &= \left|f(x_g^{r2})\right| / p', \\ p_3 &= \left|f(x_g^{r3})\right| / p', \text{ and} \\ p' &= \left|f(x_g^{r1})\right| + \left|f(x_g^{r2})\right| + \left|f(x_g^{r3})\right|. \end{aligned}$$

In the remaining paper, we call DE_1 the DE algorithm that uses Eq. (1) as the mutation operator, DE_2 the algorithm that uses Eq. (2), and so on.

Having performed mutation, the recombination operator is applied to further increase the diversity of the population. To this end, the mutant individuals are combined with other predetermined individuals, called the target individuals. Specifically, for each component l ($l = 1, 2, \ldots, n$) of the mutant individual v_g^i, we randomly choose a real number r in the interval $[0, 1]$. Then, we compare this number with the *recombination constant*, Cr. If $r \leqslant Cr$, then we select, as the l–th component of the trial individual u_g^i, the l–th component of the mutant individual v_g^i. Otherwise, the l–th component of the target vector x_g^i becomes the l–th component of the trial vector. This operation yields the trial individual. Finally, the trial individual is accepted for the next generation only if it reduces the value of the objective function.

III. Genetic Programming

GP is an extension of Genetic Algorithms in which individuals are no longer fixed-length strings but rather computer programs expressed as *syntax trees*. GP individuals consist of function and terminal nodes. Terminal nodes store a value which they return as an output, while functions process their inputs to compute an output. The terminal set, T, is comprised of the inputs, the constants supplied, and the zero-argument functions. Thus, terminal nodes have an arity of zero. On the other hand, the function set, F, is composed of the statements and functions available to GP.

The primary GP search operators are *crossover* and *mutation*. In crossover, a randomly selected subtree from each of the two selected parents is exchanged between them to form two new individuals (offsprings). The idea is that useful building blocks for the solution of a problem are accumulated in the population and crossover permits the aggregation of good building blocks into even better solutions to the problem [6]. Crossover is the predominant search operator in GP [23]. Mutation operates on a single individual by altering a random subtree. Next, we briefly describe the GP initialization and the GP operators used in this paper.

A. The GP Initialization

The individuals in the GP population are initialized by recursively generating syntax trees composed of random function and terminal nodes. Two established GP initialization methods are the grow and the full method. Both methods require from the user to specify the maximum initial tree depth. According to the grow method, nodes are selected randomly from the function and the terminal sets. The grow method, therefore, produces trees of irregular shape, since once a terminal node is inserted the path ending with this node cannot be extended, even if the maximum initial depth has not been reached. On the other hand, in the full initialization method only function nodes are selected until the maximum initial depth is reached. Beyond that depth only terminal nodes are chosen to end the branches. This method results in a balanced tree, every branch of which reaches the maximum initial depth.

B. The GP Selection Algorithm

To derive the individuals that will comprise the population of the next generation, GP initially selects individuals from the current generation. The selection operators that have been proposed for Genetic Algorithms are also applicable to GP. In this study, we employed the most commonly encountered one, namely *roulette wheel selection*. Define the fitness of the ith individual as E_i, where E is the error function we wish to minimize. Then the probability of selecting individual i as a parent of an individual of the next generation is equal to $E_i^n / \sum_{j=1}^{N} E_j^n$; where $E_i^n = 1/(1+E_i)$.

C. The GP Crossover Operator

The crossover operator combines the genetic material of two parents chosen by the selection operator to yield two offsprings. In particular, a real number r is randomly chosen in the interval [0.1]. Crossover takes place only if $r \leqslant C$, where C is the predefined crossover constant. In this case, a random node in each parent is chosen and the subtrees rooted at these nodes are exchanged between the parents to yield the offsprings. If an offspring exceeds the maximum depth it is discarded and the corresponding parent individual takes its place in the population of the next generation. Thus, crossover produces offsprings by swapping a part of one parent with a part of the other. If crossover does not take place ($r > C$) the offsprings are exact copies of the parents.

D. The GP Mutation Operator

After the crossover operator has finished, each offspring produced undergoes mutation. The probability of mutation is a user defined parameter. The mutation operator in GP randomly selects a node of the tree. If the node is a function then it is replaced by another function. If the node is terminal, another randomly selected terminal is used instead (point mutation) [23]. The mutated individual is then placed back into the population.

IV. Genetically Programmed Differential Evolution Mutation Operators

As previously mentioned, the aim of this study is to discover new efficient DE mutation operators using GP. This is possible since, mutation operators are simply the composition of elementary functions such as addition, subtraction, and multiplication, operating on the vectors that represent individuals of the the DE population. To this end, the terminal set used for GP, included two numerical constants, the vector of the best so far DE individual, x_g^{best}, three vectors of different randomly selected DE individuals, and the fixed mutation constant F employed by the DE mutation operators. In detail, the terminal set used in this study was, $T = \{0.5, 1, F, x_g^{\text{best}}, x_g^{r1}, x_g^{r2}, x_g^{r3}\}$. The function set was $F = \{+, -, \odot, \oslash\}$, where $\odot$ and $\oslash$ are defined as follows:

$$\begin{aligned}\vec{x} \odot \vec{y} &= \vec{x}^{\top} \cdot \operatorname{diag}\{y_1, y_2, \ldots, y_n\} = \\ &= (x_1 y_1, x_2 y_2, \ldots, x_n y_n)^{\top}, \\ \vec{x} \oslash \vec{y} &= \vec{x}^{\top} \cdot \operatorname{diag}\{1/y_1, 1/y_2, \ldots, 1/y_n\} = \\ &= (x_1/y_1, x_2/y_2, \ldots, x_n/y_n)^{\top},\end{aligned}$$

where the vectors $\vec{x}, \vec{y} \in \mathbb{R}^n$, with $\vec{x}^{\top} = (x_1, x_2, \ldots, x_n)$ and $\vec{y}^{\top} = (y_1, y_2, \ldots, y_n)$. Note that the operator $\oslash$ utilizes a protected division; if the absolute value of the denominator is less 0.0001, then $\oslash$ returns 1.

The presentation of the problem and the fitness function typically define the space of candidate solutions for each particular problem. At present, more than one performance measure are applicable. One approach is to use the distance of the discovered minimizer from the global one to measure the operator's performance [24]. However, in many real life applications the location of the global minimizer is unknown. Conversely, the value of the global minimum could be known (for example when minimizing the sum of squares, a chemical or physical process, etc).

In this study, we defined a fitness function, suitable for general optimization tasks, which utilizes three well known benchmark optimization problems. More specifically, the performance of each operator was measured through the sum of the generations required to locate the global minimum on each benchmark function, plus the minimum function values that were discovered. It is known that the performance of the DE algorithm (like the performance of every other EA) can vary with the initial random individuals. To reduce the effect of the stochastic nature of DE, 10 independent evaluations were performed, and the final fitness was averaged. If the global minimum was not found after 100 generations DE terminated. Using this fitness function, we strain GP evolution towards obtaining DE operators capable of locating the global optimum, within a minimum number of generations. Next, we report the three benchmark optimization functions used along with their global minima and minimizers.

Train Problem 1: Shekel's Foxholes [27]

$$f_2(x) = \frac{1}{0.002 + \psi_1(x)}, \qquad x_j \in [-65.536, 65.536],$$

where,

$$\psi_1(x) = \sum_{i=0}^{24} \frac{1}{1 + i + \sum_{j=1}^{2} (x_j - a_{ij})^6}.$$

The parameters for this function are:

$$\begin{aligned} a_{i1} &= \{-32, -16, 0, 16, 32\}, \text{ where} \\ &\quad i = \{0, 1, 2, 3, 4\} \text{ and } a_{i1} = a_{i \bmod 5, 1} \\ a_{i2} &= \{-32, -16, 0, 16, 32\}, \text{where} \\ &\quad i = \{0, 5, 10, 15, 20\} \text{ and} \\ &\quad a_{i2} = a_{i+k,2},\ k = \{1, 2, 3, 4\}. \end{aligned}$$

The global minimum is $f_2(-32,-32) = 0.998004$.

Train Problem 2: Corana Parabola [28]

$$f_2(x) = \sum_{j=1}^{4} \begin{cases} \psi_2(x_j), & \text{if } |x_j - z_j| < 0.05, \\ \psi_3(x_j), & \text{otherwise,} \end{cases}$$

where $\psi_2(x_j) = 0.15\left(z_j - 0.05\text{sign}(z_j)\right)^2 d_j$, $\psi_3(x_j) = d_j x_j^2$, $z_j = \lfloor 5|x_j| + 0.49999 \rfloor \text{sign}(x_j) 0.2$ and $d_j = \{1, 1000, 10, 100\}$.

The Corana test function defines a paraboloid with axes parallel to the coordinate axes. The function is characterized by a multitude of local minima, increasing in depth as one moves closer to the origin. The global minimum of the function is $f_2(x) = 0$, for $x_j \in (-0.05, 0.05)$.

Train Problem 3: Levy No. 5 Function [29]

$$\begin{aligned} f_3(x) &= \sum_{i=1}^{5} \Big[i \ \cos\big((i-1)x_1 + i\big) \Big] \cdot \\ &\cdot \sum_{j=1}^{5} \Big[j \ \cos\big((j+1)x_2 + j\big) \Big] + \\ &+ (x_1 + 1.42513)^2 + (x_2 + 0.80032)^2, \end{aligned}$$

where $-10 \leqslant x_i \leqslant 10$, and $i = 1, 2$. There are about 760 local minima and one global minimum with function value $f_3(x) = -176.1375$, at $(1.3068, 1.4248)$. The large number of local minimizers makes it difficult for any method to locate the global one.

V. Presentation of Experiments

The computational experiments were performed utilizing a GP–DE interface developed in C++, using the GNU compiler collection (gcc) version 4.0.3 on a Debian Linux operating system. The C++ implementation relies on the interface for the creation of expressions described in [30].

We employed the full GP initialization method with a maximum initial tree depth of 3. Another critical GP parameter is the maximum allowed depth for the trees. The maximum depth parameter is the largest allowed depth between the root node and the outermost terminals. The maximum depth during the GP execution was 100. GP population size was 40, while the maximum number of generations was set to 1000. The mutation and crossover probabilities for GP were set to 0.6 and 0.1, respectively. The values for the parameters F and Cr employed by the DE algorithm (irrespective of the mutation operator), were set to 0.6 and 0.8, respectively.

We conducted 100 independent GP experiments. The five best performing DE mutation operators discovered are the following:

$$v_g^i = \left(x_g^{r1} + x_g^{r2}\right) \oslash \left(1 + (x_g^{r3} \oslash x_g^{\text{best}})\right), \tag{7}$$

$$v_g^i = x_g^{\text{best}} + 0.5(x_g^{r1} - x_g^{r2}), \tag{8}$$

$$v_g^i = \left(x_g^{r1} + x_g^{r3}\right) \oslash \Big(\left(x_g^{r1} \oslash x_g^{\text{best}}\right) + \left(x_g^{r2} \oslash x_g^{\text{best}}\right) \Big), \tag{9}$$

$$v_g^i = \left(x_g^{r3} + x_g^{\text{best}}\right) \oslash \Big(\left(x_g^{r3} \oslash x_g^{r1}\right) + \left(x_g^{r2} \oslash x_g^{\text{best}}\right) \Big), \tag{10}$$

$$\begin{aligned} v_g^i = &\Big(\left(x_g^{r1} \odot x_g^{r3}\right) \oslash \left(x_g^{r1} + x_g^{r3}\right) \Big) \odot \\ &\odot \Big(\left(x_g^{\text{best}} \oslash x_g^{r3}\right) + \left(x_g^{\text{best}} \oslash x_g^{r2}\right) \Big). \end{aligned} \tag{11}$$

Throughout the remaining paper, we call $\text{GPDE}_1, \text{GPDE}_2, \ldots, \text{GPDE}_5$ the DE algorithm that uses Eq. (7), Eq. (8), …, Eq. (11) as the mutation operator, respectively. It is evident that the proposed methodology allows us to routinely "invent" new specialized DE operators, which are optimal or near-optimal for a specific problem. Notice that although the mutation constant F was included in the terminal set, all the above mentioned GP derived DE mutation operators are parameter free. This is a considerable advantage since it alleviates the need for parameter tuning by the user.

The original DE algorithm exploits the information from the differences between pairs of individuals to guide its search in the solution domain [12], [31]. Although, in all the mutation operators discovered here, individuals interact in pairs, pairwise differences are not encountered in any GPDE operator but GPDE_2. Indeed, GPDE_2 is equivalent to DE_1 for the special case that $F = 0.5$. The experimental results reported below suggest that this particular setting is more effective than a typical value of F for the benchmark problems considered.

To measure the efficiency and effectiveness of the newly discovered GPDE operators, we tested them on the three previously mentioned optimization benchmark functions, as well as on two additional functions; namely the Griewangk's and the Rosenbrock's Saddle test functions. As a final validation test for the GP derived DE operators we apply them to minimize the Shifted-Rotated Weierstrass Function [32]. Below we report the optimization functions along with their global minima and minimizers.

Test Problem 1: Griewangk's Function [33]

$$\begin{aligned} f_4(x) &= \sum_{j=1}^{10} \frac{x_j^2}{4000} - \prod_{j=1}^{10} \cos\left(\frac{x_j}{\sqrt{j}}\right) + 1, \\ &\quad x_j \in [-400, 400]. \end{aligned}$$

This test function is riddled with local minima. The global minimum of the function is $f_4(0, \ldots, 0) = 0$.

Test Problem 2: Rosenbrock's Saddle [27]

$$f_5(x) = 100 \cdot (x_1^2 - x_2)^2 + (1 - x_1)^2, \ x_j \in [-2.048, 2.048].$$

This is a two–dimensional test function, which is known to be relatively difficult to minimize. The global minimum is $f_5(1, 1) = 0$.

The performance of the human-designed and the genetically programmed DE mutation operators is presented in

TABLE I
SIX HUMAN DESIGNED VS FIVE GENETICALLY PROGRAMMED DIFFERENTIAL EVOLUTION OPERATORS

	TRAINING PHASE						TESTING PHASE			
	PROBLEM 1		PROBLEM 2		PROBLEM 3		PROBLEM 1		PROBLEM 2	
	Gen.	(%)	Gen.	(%)	Gen.	(%)	Gen.	(%)	Gen.	(%)
DE_1	95.6	5	87.5	42	64.9	50	29.8	**100**	—	0
DE_2	83.8	**95**	—	0	76.8	**100**	56.8	**100**	—	0
DE_3	97.5	13	97.9	35	76.9	63	43.7	**100**	—	0
DE_4	—	0	39.8	79	89.2	16	52.3	74	96.8	12
DE_5	94.6	20	—	0	92.9	22	81.2	85	—	0
DE_6	81.2	64	—	0	72.1	**100**	63.1	96	—	0
$GPDE_1$	—	0	29.9	94	43.7	96	55.9	68	93.6	44
$GPDE_2$	84.1	23	98.8	3	**34.0**	88	**25.1**	**100**	—	0
$GPDE_3$	**78.5**	25	58.2	57	52.8	65	28.0	**100**	76.9	53
$GPDE_4$	—	0	**12.1**	**100**	82.6	32	38.8	**100**	**20.3**	**100**
$GPDE_5$	—	0	25.7	**100**	57.1	59	38.4	93	82.2	28

(— denotes that the algorithm failed to find the global minimum in all runs)

Table I. In particular, for each mutation operator and for each benchmark function, Table I reports the mean number of generations required to locate a global minimizer (Gen.), as well as, the percentage of times the algorithm was successful in locating a global minimizer (%). The reported results are averages over 100 independent experiments for each mutation operator. Note, that in the cases DE was unable to identify a global minimizer, the maximum allowed number of generations was added to the sum used to compute the mean number of generations required to locate a global minimizer. The entry "—" in the table suggests that the success rate of a mutation operator for the corresponding benchmark was zero. Finally, bold faced entries are used to indicate the mutation operator with the lowest mean number of generations to detect a global minimizer and the one with the highest success rate.

With respect to the mean number of generations required to detect a global minimizer, the best performing mutation operator is in all cases derived by Genetic Programming. The best performing mutation strategy in this respect is $GPDE_2$, which is a special case of DE_1. For two out the five optimization problems (Train Problem 3 and Test Problem 1), DE_1 requires the lowest mean number of generations to compute a global minimizer among the original DE operators. On the same two problems $GPDE_2$ is the overall best performing strategy in this respect, but it performs badly on Test Problem 2. $GPDE_4$ is by far the best performing strategy on Train Problem 2 and Test Problem 2, for which most operators performed badly. With respect to the percentage of times a minimizer was located, the two types of operators perform similarly well on Test Problem 1. Last but not least, it is important to note that the most robust operator with respect to both criteria is $GPDE_3$. It is the best performing operator with respect to mean number of generations on Train Problem 1, the second best on Test Problem 1, and the third best performing on Train Problem 3 and Test Problem 2. Furthermore, it is the only operator that achieved a positive percentage of locating a minimizer on all the test functions. Our experience is that $GPDE_3$ is stable and effective, and can be used to optimize an unknown function with good results.

A. Shifted-Rotated Weierstrass Function

Finally, we validated the GP derived DE operators on the Shifted-Rotated Weierstrass Function [32]. This is the test function F_{11} of the CEC 2005 benchmark for real functions. The Weierstrass function is a hard to minimize multimodal function that has been rotated and shifted. The Shifted-Rotated Weierstrass Function is defined as follows:

$$\begin{aligned} F_{11} &= \sum_{i=1}^{2}\left(\sum_{k=0}^{k_{max}}\left(a^k\cos(2\pi b^k(z_i+0.5))\right)\right) - \\ &- 2\cdot\sum_{k=0}^{k_{max}}\left(a^k\cos(2\pi b^k 0.5)\right) + bias_{11} \qquad (12) \end{aligned}$$

where $a = 0.5, b = 3, k_{max} = 20, z_i = (x_i - o_i) * M$, and $x_i \in [-0.5, 0.5], i = 1, 2$. For the rotation the linear transformation matrix M has been used and $x^* = [o_1, o_2]$ is the new (shifted) global optimum, with $F_{11}(x^*) = bias_{11} = 90$. Figure 1 illustrates the 3-d plot of this benchmark function, while Table II reports the experimental results obtained for this function.

VI. CONCLUSIONS

In this paper, we utilize Genetic Programming to evolve novel Differential Evolution operators. As the "no free lunch theorem" [34] implies, it is impossible to find a single DE operator that outperforms all the other in every test problem. Instead, we try to discover new DE operators better suited for general optimization problems, or classes of problems.

The experimental results indicate that the best performing DE mutation operator is in all cases GP derived. GP has been

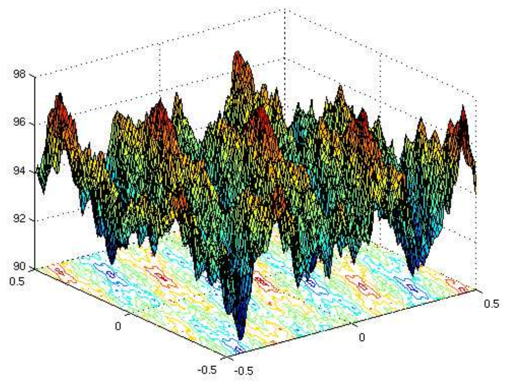

Fig. 1. 3–D Plot of the Shifted-Rotated Weierstrass Function

TABLE II
PERFORMANCE ON THE SHIFTED-ROTATED WEIERSTRASS FUNCTION

	Gen.	(%)
DE_1	81.88	78
DE_2	—	0
DE_3	—	0
DE_4	—	0
DE_5	—	0
DE_6	—	0
$GPDE_1$	93.27	42
$GPDE_2$	66.57	79
$GPDE_3$	64.66	98
$GPDE_4$	89.39	54
$GPDE_5$	73.05	71

(— denotes that the algorithm failed to find the global minimum in all runs)

able to automatically evolve a variety of new DE mutation operators that operate as well or considerably better, for the considered problems, than the already existing human-designed ones. It is interesting to note that all the new DE mutation operators are parameter free, in the sense that no mutation constant is needed.

In a future correspondence, we intend to investigate the parallel implementation of the proposed approach. Additionally, we will study the performance of the GP derived DE algorithms on difficult high–dimensional real–life problems encountered in bioinformatics, medical applications and neural network training.

ACKNOWLEDGMENT

The authors would like to thank the anonymous referees for their constructive comments and useful suggestions that helped to significantly improve this paper. We would also like to thank the European Social Fund, Operational Program for Educational and Vocational Training II (EPEAEK II), and particularly the Program "Pythagoras" for funding this work. We also acknowledge the financial support of the University of Patras Research Committee through a "Karatheodoris" research grant.

REFERENCES

[1] D. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison Wesley, 1989.

[2] J. Holland, *Adaptation in natural and artificial system*. University of Michigan Press, 1975.

[3] I. Rechenberg, "Evolution strategy," in *Computational Intelligence: Imitating Life*, J. Zurada, R. Marks II, and C. Robinson, Eds. Piscataway, NJ: IEEE Press, 1994.

[4] H.-P. Schwefel, *Evolution and Optimum Seeking*. New York: Wiley, 1995.

[5] J. R. Koza, "Hierarchical genetic algorithms operating on populations of computer programs," in *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 1989, pp. 768–774.

[6] ——, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.

[7] D. Fogel, *Evolutionary Computation: Towards a New Philosophy of Machine Intelligence*. Piscataway, NJ: IEEE Press, 1996.

[8] L. Fogel, A. Owens, and M. Walsh, *Artificial intelligence through simulated evolution*. Wiley, 1966.

[9] E. Bonabeau, M. Dorigo, and G. Théraulaz, *From Natural to Artificial Swarm Intelligence*. New York: Oxford University Press, 1999.

[10] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings IEEE International Conference on Neural Networks*, vol. IV. Piscataway, NJ: IEEE Service Center, 1995, pp. 1942–1948.

[11] R. Storn and K. Price, "Differential evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, pp. 341–359, 1997.

[12] ——, "Minimizing the real functions of the icec'96 contest by differential evolution," in *IEEE Conference on Evolutionary Computation*, 1996, pp. 842–844.

[13] M. DiSilvestro and J.-K. Suh, "A cross-validation of the biphasic poroviscoelastic model of articular cartilage in unconfined compression, indentation, and confined compression," *Journal of Biomechanics*, vol. 34, pp. 519–525, 2001.

[14] J. Hou, P. Siegel, and L. Milstein, "Performance analysis and code optimization of low density parity-check codes on rayleigh fading channels," *IEEE Journal on Selected Areas in Communications*, vol. 19, pp. 924–934, 2001.

[15] T. Richardson, M. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Transactions on Information Theory*, vol. 47, pp. 619–637, 2001.

[16] R. Storn, "Differential evolution design of an IIR-filter," in *IEEE International Conference on Evolutionary Computation ICEC'96*, 1996.

[17] J. Ilonen, J.-K. Kamarainen, and J. Lampinen, "Differential evolution training algorithm for feed forward neural networks," *Neural Processing Letters*, vol. 17, no. 1, pp. 93–105, 2003.

[18] V. P. Plagianakos and M. N. Vrahatis, "Training neural networks with 3–bit integer weights," in *Genetic and Evolutionary Computation Conference (GECCO'99)*, W. Banzhaf, J. Daida, A. Eiben, M. Garzon, V. Honavar, M. Jakiela, and R. Smith, Eds. Orlando, U.S.A.: Morgan Kaufmann, 1999, pp. 910–915.

[19] ——, "Neural network training with constrained integer weights," in *Congress of Evolutionary Computation (CEC'99)*, P. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalzala, Eds. Washington D.C., U.S.A.: IEEE Press, 1999, pp. 2007–2013.

[20] ——, "Parallel evolutionary training algorithms for 'hardware–friendly' neural networks," *Natural Computing*, vol. 1, pp. 307–322, 2002.

[21] D. K. Tasoulis, N. G. Pavlidis, V. P. Plagianakos, and M. N. Vrahatis, "Parallel differential evolution," in *IEEE Congress on Evolutionary Computation (CEC 2004)*, 2004.

[22] H. Y. Fan and J. Lampinen, "A trigonometric mutation operation to differential evolution," *Journal of Global Optimization*, vol. 27, pp. 105–129, 2003.

[23] B. Wolfgang, P. Nordin, R. Keller, and F. Francone, *Genetic programming: An Introduction: on the automatic evolution of computer programs and its applications*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998.

[24] R. Poli, W. Langdon, and O. Holland, "Extending particle swarm optimisation via genetic programming," in *Proceedings of the 8th European Conference on Genetic Programming*, ser. Lecture Notes in Computer Science, vol. 3447. Springer, 2005.

[25] R. Poli, C. Di Chio, and W. Langdon, "Exploring extended particle swarms: a genetic programming approach," in *GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation*, vol. 1. Washington DC, USA: ACM Press, 2005, pp. 169–176.

[26] J. Vesterstrom and R. Thomsen, "A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on

numerical benchmark problems," in *IEEE Congress on Evolutionary Computation (CEC 2004)*, vol. 2, 2004, pp. 1980–1987.

[27] K. D. Jong, "An analysis of the behaviour of a class of genetic adaptive systems," Ph.D. dissertation, University of Michigan, 1975.

[28] A. Corana, M. Marchesi, C. Martini, and S. Ridella, "Minimizing multimodal functions of continuous variables with the "simulated annealing" algorithm," *ACM Transactions Mathematical Software*, vol. 13, no. 3, pp. 262–280, 1987.

[29] A. Levy, A. Montalvo, S. Gomez, and A. Galderon, *Topics in Global Optimization*. Springer-Verlag, New York, 1981.

[30] A. Koenig and B. Moo, *Ruminations on C++: A Decade of Programming Insight and Experience*. Addison-Wesley, 1996.

[31] R. Storn and K. Price, "Differential evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, pp. 341–359, 1997. [Online]. Available: http://www.icsi.berkeley.edu/˜storn/code.html

[32] P.N. Suganthan, N. Hansen, J.J. Liang, K. Deb, Y.-P. Chen, A. Auger and S. Tiwari, "Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real–Parameter Optimization", Technical Report, Nanyang Technological University, Singapore, 2005.

[33] A. Griewank, "Generalized descent for global optimization," *Journal of optimization theory and applications*, vol. 34, no. 1, pp. 11–39, 1981.

[34] D. Wolpert and W. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.