Test Function Generators for Assessing the Performance of PSO Algorithms in Multimodal Optimization

Julio Barrera and Carlos A. Coello Coello

CINVESTAV-IPN (Evolutionary Computation Group), Computer Science Department, México
julio.barrera@gmail.com, ccoello@cs.cinvestav.mx

Summary The particle swarm optimization (PSO) algorithm has gained increasing popularity in the last few years mainly because of its relative simplicity and its good overall performance, particularly in continuous optimization problems. As PSO is adopted in more types of application domains, it becomes more important to have well-established methodologies to assess its performance. For that purpose, several test problems have been proposed. In this chapter, we review several state-of-the-art test function generators that have been used for assessing the performance of PSO variants. As we will see, such test problems sometimes have regularities which can be easily exploited by PSO (or any other algorithm for that sake) resulting in an outstanding performance. In order to avoid such regularities, we describe here several basic design principles that should be followed when creating a test function generator for single-objective continuous optimization.

1 Introduction

Multimodal problems are those in which the search space has several local optima and possibly more than one global optimum. They constitute a type of optimization problem in which the particle swarm optimization (PSO) algorithm has been only scarcely applied [1]. Such multimodal problems are interesting not only because of the challenge that represents avoiding local optima or the localization of more than one global optimum at the same time, but because there exist several real-world problems presenting such features.

The most common multimodal test functions currently available in the specialized literature show regularities, such as symmetry with respect to one axis, uniform spacing among optima, exponential increase in the number of global optima with respect to the increase in the number of decision variables, among others. Such regularities can be exploited by an optimization algorithm such as PSO, decreasing their degree of difficulty [2]. To overcome these regularities and have a better testing environment to assess the perfor-

mance of an optimization algorithm, some test functions generators have been developed [3] as well as methodologies to create new test functions by using a composition procedure or through the application of linear transformations on common test functions [4]. Our particular interest are scalable test functions presenting several local optima, but only one global optimum.

In this chapter we present a brief introduction to particle swarm optimization, to linear transformations and to the composition of functions. We also provide guidelines to create a composition of functions in a simple way, using any sort of test functions at hand. Additionally, we review the state-of-the-art regarding test function generators, and conclude with some pointers towards promising directions to extend the test functions generators currently available in the specialized literature.

2 The Particle Swarm Optimization Algorithm

Kennedy and Eberhart introduced the PSO algorithm in the mid-1990s [5], and it quickly became popular as an optimizer mainly because of its ease of use and efficacy. In PSO, the position of a possible solution (a particle) is updated using the two rules shown in equations (2) and (1).

$$\mathbf{v}_{t+1} = \mathbf{v}_t + c_1 r_1(\mathbf{g} - \mathbf{x}_t) - c_2 r_2(\mathbf{p} - \mathbf{x}_t) \tag{1}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{v}_{t+1} \tag{2}$$

where $\mathbf{x}_t$ and $\mathbf{v}_t$ are the current position and the current velocity of the particle, respectively, $\mathbf{p}$ is the position in which the particle has obtained its best (so far) fitness value, $\mathbf{g}$ is the position with the best (so far) fitness value obtained by the entire swarm, c_1 and c_2 are called the learning constants, r_1 and r_2 are random numbers in the range $[0, 1]$ generated using an uniform distribution. The initial tests of the PSO algorithm were made using the Schaffer F6 function and training a neural network. Further improvements to the PSO algorithm were later introduced by Eberhart and Shi [6], by adding an inertia weight constant to the update rule of the velocity, as shown in equation (3)

$$\mathbf{v}_{t+1} = \omega \mathbf{v}_t + c_1 r_1(\mathbf{g} - \mathbf{x}_t) - c_2 r_2(\mathbf{p} - \mathbf{x}_t) \tag{3}$$

The constant ω acts as a damping parameter, regulating the transition between the exploration and exploitation phases of the algorithm. In this case, the Schaffer F6 function was also adopted for the validation of the PSO algorithm. In some further work, Shi and Eberhart [7] presented a version in which the inertia weight value was linearly decreased. In that work the PSO algorithm was assessed using four test functions: the Sphere, Rosenbrock,

Rastrigin, and Griewank. All of these test functions can be scaled to any number of variables.

An analysis of the PSO algorithm from the point of view of the dynamic systems was presented by Clerc and Kennedy [8]. This work introduced another modification to the velocity update rule, as expressed in equation (4).

$$\mathbf{v}_{t+1} = \chi\left[\mathbf{v}_t + c_1 r_1(\mathbf{g} - \mathbf{x}_t) - c_2 r_2(\mathbf{p} - \mathbf{x}_t)\right] \tag{4}$$

where the coefficient ω is computed according to equation (5).

$$\chi = \frac{2\kappa}{|2 - \phi - \sqrt{\phi^2 - 4\phi}|} \tag{5}$$

Here, κ is an arbitrary value in the range $[0, 1]$. It is common to use the value $\kappa = 1$. From equation (5) we can see that, in order to obtain a real value in the square root of the denominator, it is necessary that $\phi \leq 4$. In the case $\phi = 4$, with $\kappa = 1$, we obtain a value $\chi = 1$, and we get the original PSO update rules. In addition to the four test functions used by Shi and Eberhart [7], that can be scaled up to any number of variables, other four test functions were used, namely, De Jong F4, Schaffer F6, Foxholes (De Jong F5), and a Rosenbrock variant with only two variables. Of the last four test functions only the De Jong F4 test function can be used with any number of variables.

Other modifications to the PSO algorithm include the topology of the particles. Initially, the global best position $\mathbf{g}$ is computed by inspecting each of the particles in the swarm. In this case, all the particles can exchange information among them. Thus, the information of which particle has the best fitness value is transferred quickly and easily. In an attempt to slow down the information transfer (and favor diversity), the particles are aligned in a ring. Each particle has only two neighbors with whom they can share information. The position of a particle in the ring is not related to its position in the search space.

The first of these two models (in which all the particles are examined in order to determine the position of a particle) is called *global best* (or *gbest*). The second model (in which a ring topology is used) is called *local best* (or *lbest*). Eberhart and Kennedy introduced the *lbest* topology in their work [9]. Subsequent works from Kennedy [10] and Kennedy and Mendes [11] examined in more detail the effects of using different topologies in the PSO algorithm. The test functions used in this case include the Sphere, Rosenbrock, Ranstring, Griewank, and in [11], the Schaffer F6 test function is used as well. Along with the *gbest* and *lbest* topologies, the von Neumann topology [11] has also been popular, particularly when adopting sub-swarms [12, 13]. In the von Neumann topology, the particles are arranged in a grid and each particle has four neighbors. Figure 1 illustrates the *gbest*, *lbest*, and von Neumann topologies.

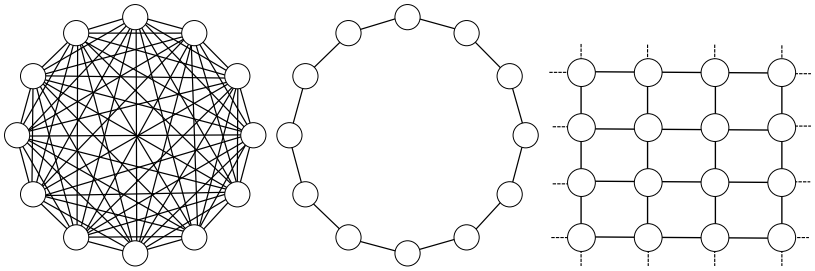

Fig. 1. Different topologies used with the PSO algorithm. From left to right: the *gbest* topology where all particles can share information among them, the *lbest* topology where each particle has only two neighbors and can only share information with them, and the von Neumann topology where the particles are arranged in a regular grid and each particle has four neighbors.

Although other, more robust and elaborate, PSO variants have been proposed (see for example [14, 15]), most of them rely on the use of the constriction factor and the inertia weight model, along with the *gbest*, *lbest* or von Neumann topologies.

3 Linear Transformations and Homogeneous Coordinates

Most of the basic test functions have regularities that can be exploited by optimization algorithms. Examples of such regularities are that the position of the optimum is in the origin, or that it has equal values for all of its coordinates. These regularities can be overcome by using linear transformations. A translation transformation can displace the location of the optimum. Indeed, the position of the optimum can be not only displaced, but its coordinates can also have different values. Other transformations such as rotation and scaling can also be used to break regularities in the test functions.

A well-known drawback of using linear transformations is that such transformations are applied separately. It is, for example, common to first translate a point using vector operations, then multiply it by a scalar in order to apply a scaling transformation, and finally, multiply the vector by a matrix in order to rotate its position. This is partly due to that a translation cannot be expressed as a matrix for a given dimension D. If all linear transformations could be expressed by a matrix, they could all be combined using matrix multiplication, and then, by using a single vector-matrix operation, we could apply all the linear transformations to a point. This can be accomplished by using homogeneous coordinates.

Homogeneous coordinates are commonly used in computer graphics [16]. Although they are used for perspective and projection transformations, they are also useful to express a translation transformation as a matrix. To use

homogeneous coordinates we only need to add an extra coordinate to the vector that represents the position. This extra coordinate is used only to help in the application of the linear transformations, and can be dropped after. For example, in a problem with two decision variables x and y, a point P of the search space is represented as $P = (x, y)$. The same point in homogeneous coordinates is expressed as $P' = (x, y, 1)$. A translation transformation that shifts an amount T_x the variable x, and an amount T_y the variable y is expressed as the matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{bmatrix} \tag{6}$$

To apply the translation transformation, we multiply the vector representing the position P' by the matrix representing the translation transformation as follows

$$\begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{bmatrix} = \begin{bmatrix} (x + T_x) & (y + T_y) & 1 \end{bmatrix} \tag{7}$$

We obtain a translated point Q' with coordinates $(x + T_x, y + T_y, 1)$. Removing the last coordinate of Q' we obtain the point $Q = (x + T_x, y + T_y)$ which is the point $P = (x, y)$ translated by an amount T_x and T_y. The translation transformation can be expressed as a matrix for any number of variables. A generalization of the translation transformation matrix is shown in equation (8).

$$T = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ T_1 & T_2 & T_3 & \cdots & T_D & 1 \end{bmatrix} \tag{8}$$

A scaling transformation can be also represented as a matrix. Equation (9) shows a matrix that represents a scaling transformation for two variables. The values S_x and S_y in the diagonal of the matrix, represent the scaling factors for the variables x and y, respectively. Equation (10) shows the scaling transformation matrix in homogeneous coordinates for two variables.

$$S = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \tag{9}$$

$$S = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{10}$$

The scaling transformation matrix can also be expressed for any number of variables. A scaling transformation matrix in D dimensions using homogeneous coordinates is represented in equation (11).

$$S = \begin{bmatrix} S_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & S_2 & 0 & \cdots & 0 & 0 \\ 0 & 0 & S_3 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & S_D & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \tag{11}$$

In the case of a rotation transformation, a matrix can also represent it. However, the matrix that represents a rotation transformation has additional properties, since the matrix must be orthogonal. A rotation transformation can also be difficult to build. For example in three dimensions, and depending of the rotation axis, a rotation transformation matrix can be expressed in any of the following ways:

$$R_z = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{12}$$

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \tag{13}$$

$$R_y = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \tag{14}$$

Rotation can be done around any axis, but the matrix, in each case, takes a different form. As we mentioned before, a matrix that represents a rotation transformation must be orthogonal. Salomon [2] describes a method to generate a matrix that represents the application of random rotation transformations for more that 2 variables. A brief description of this method is provided next:

1. A function that generates a rotation matrix is defined as follows: $Rot(A, i, j)$ returns an identity matrix A with a change in four elements, namely $a_{ii} = a_{jj} = rand_1$ and $a_{ij} = rand_2, a_{ji} = -rand_2$, where $rand_1$ and $rand_2$ are random numbers in the range $[-1, 1]$.
2. To create a square $n \times n$ orthogonal matrix, n rotation matrices are created $A_k = Rot(A, 2, k)$ for $k = 1, \ldots, n$ and are multiplied to create A_1.
3. A second $n \times n$ matrix A_2 is created using the product of $A_m = Rot(A, m, n)$ matrices for $m = 3, \ldots, n-1$.
4. The final matrix A_R is computed as the product $A_R = A_1 A_2$.

A rotation matrix in homogeneous coordinates follows the same form that the translation and scaling matrices. In order to represent a rotation transformation matrix we only need to add a row and a column and put a value of 1 in the lower-right element of the matrix as in equations (8) and (11).

Now, we can represent all the linear transformations as matrices, and we can combine any number of transformations of any type in a single matrix. This will be useful when we describe how to use linear transformations to generate new test functions in Section 6.

4 Function Composition

In this section, we present a brief review of the basic notions of function and function composition that we will use to generate test functions.

A function f is a rule that associates the elements of a set A called *domain* to the elements of a set B called *codomain*. This relation is commonly represented as $f : A \to B$, and it is usually said that f maps the elements $a \in A$ into elements $b \in B$. This is written as $a \mapsto b$ or $f(a) = b$. A function has the restriction that an element $a \in A$ is associated only with one element $b \in B$. Thus, an element $a \in A$ cannot be associated with two elements $b_1, b_2 \in B$, but an element $b \in B$ can be associated with two different elements $a_1, a_2 \in A$.

As an example, let's consider the function $f : R \to R$ that maps $x \in R$ to $y \in R$ using the rule $x \mapsto mx + b$ (which is usually written as $f(x) = mx + b$ or $y = mx + b$). This function maps a $x \in R$ to a $y \in R$ and it is easy to see that if we have two elements $x_1, x_2 \in R$ they map to different elements $y_1, y_2 \in R$. But, with the function $g : R \to R$, and the rule $x \mapsto x^2$, the elements $x, -x \in R$ are mapped to the same element $x^2 \in R$.

Function composition is the sequential application of two or more functions. That is, we apply the function f to a point x and then function g to $f(x)$. This is written as $g \circ f$. We use function composition so commonly that sometimes we do not realize it. Mathematical operations, such as an addition or a product are functions. When performing a multiplication along with an addition, a function composition is applied. The application of two or more transformations, as we did in the previous section, is also function composition. For example, the function $F(x) = (x+3)^2 + 5$ is actually the composition

of three functions: a function $f : R \rightarrow R$ such that $x \mapsto x + 3$, adds 3 to x, the function $g : R \rightarrow R$ with the rule $y \mapsto y^2$, which raises $(x + 3)$ to the second power. Then, the function $h : R \rightarrow R$, which does $z \mapsto z + 5$, adds 5 to $(x + 3)^2$. Thus, the final result of the composition of the three functions is $h \circ (g \circ f)(x) = F(x) = (x + 3)^2 + 5$.

Before using the transformations and function composition to generate test functions, we will provide a review of test functions that have been commonly adopted in the PSO literature on multimodal optimization.

5 Test Functions Commonly Adopted

This section describes the test functions that have been the most commonly adopted to assess performance of PSO-based algorithms. Details of each of them are also provided, such as the search range, the position of their known optima, and other relevant properties.

The Sphere test function is one of the most simple test functions available in the specialized literature. This test function can be scaled up to any number of variables. It belongs to a family of functions called *quadratic* functions and only has one optimum in the point $\mathbf{o} = (0, 0, \ldots, 0)$. The search range commonly used for the Sphere function is $[-100, 100]$ for each decision variable. Equation (15) shows the mathematical description of the Sphere function and Figure 2 shows its graphical representation with two variables.

$$f(\mathbf{x}) = \sum_{i=1}^{D} x_i^2 \tag{15}$$

The first Schwefel test function is also a quadratic function, and it is defined by equation (16). It also has only one optimum at the point $\mathbf{o} = (0, 0, \ldots, 0)$ and its search range is the same as that of the Sphere function (i.e., $[-100, 100]$ for each variable). The graphical representation of the first Schwefel function is shown in Figure 3.

$$f(\mathbf{x}) = \sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_j \right)^2 \tag{16}$$

The Rosenbrock test function is defined by equation (17) and its graphical representation with two variables is shown in Figure 4. Although this picture shows an extense flat region, it only has one optimum located at the point $\mathbf{o} = (1, 1, \ldots, 1)$. It is also a quadratic function, and its search range is $[-30, 30]$ for each variable.

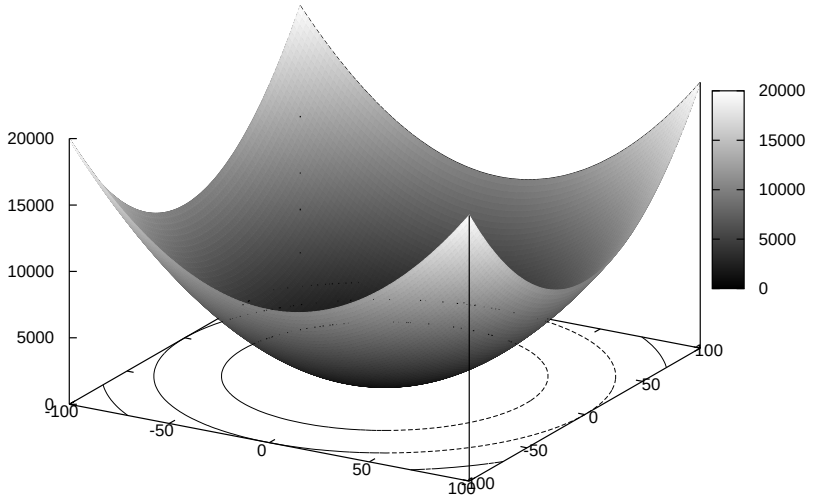

Fig. 2. Graphical representation of the Sphere test function in two dimensions.

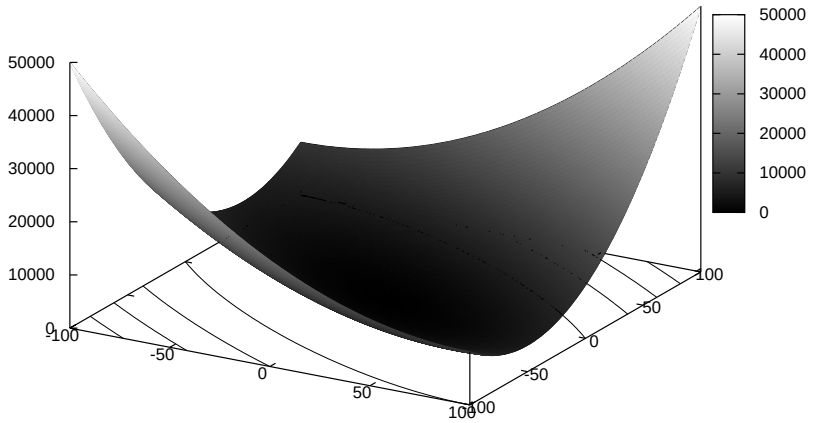

Fig. 3. Graphical representation of the first Schwefel test function in two dimensions.

$$f(\mathbf{x}) = \sum_{i=1}^{D-1} \left[100(x_{i+1} - x_i^2) + (x_i - 1)^2\right] \tag{17}$$

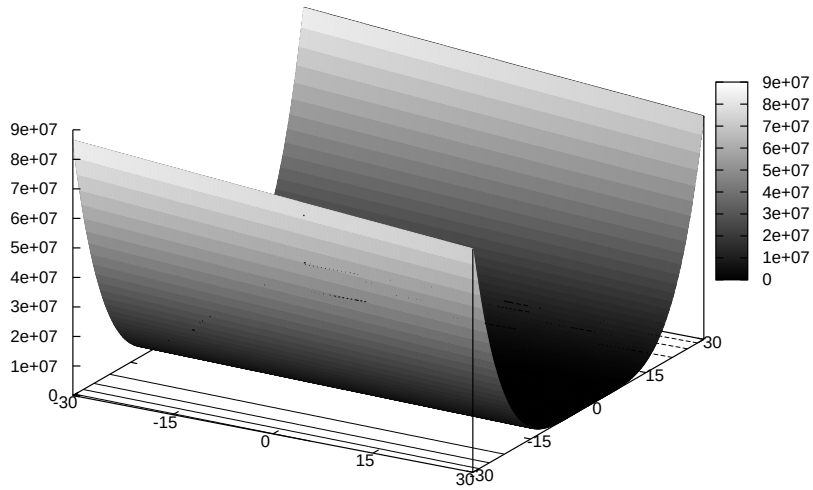

Fig. 4. Graphical representation of the Rosenbrock test function in two dimensions.

The second Schwefel test function includes a trigonometric function in the equation that defines it (see equation (18)). This provides the function with multiple local optima in the search range, which is, in this case, $[-500, 500]$ for each variable. However, it only has one optimum located at the point $\mathbf{o} = (420.96, 420.96, \ldots, 420.96)$. Its graphical representation, using two variables is shown in Figure 5.

$$f(\mathbf{x}) = -\sum_{i=1}^{D} x_i \sin(\sqrt{|x_i|}) \tag{18}$$

The generalized Rastrigin test function is also commonly adopted, and it is represented by equation (19). It includes a trigonometric function analogously to the second Schwefel function. The graphical representation of the Rastrigin function is shown in Figure 6. There, we can observe that it has several local optima arranged in a regular lattice, but it only has one global optimum located at the point $\mathbf{o} = (0, 0, \ldots, 0)$. The search range for the Rastrigin function is $[-5.12, 5.12]$ in each variable.

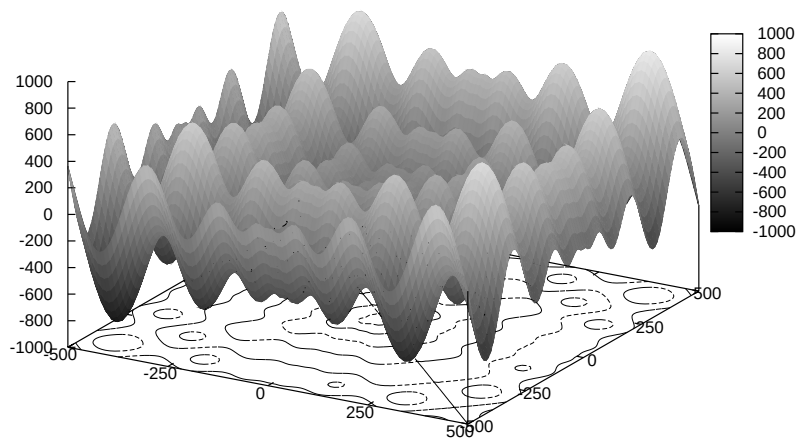

Fig. 5. Graphical representation of the second Schwefel test function in two dimensions.

$$f(\mathbf{x}) = 10 + \sum_{i=1}^{D} \left\{ x_i^2 - 10\cos\left(2\pi x_i\right) \right\} \tag{19}$$

The Ackley test function is defined by equation (20) and its graphical representation is shown in Figure 7. As we can see, the Ackley test function has several local optima that, for the search range $[-32, 32]$, look more like noise, although they are located at regular intervals. The Ackley function only has one global optimum located at the point $\mathbf{o} = (0, 0, \ldots, 0)$.

$$\begin{aligned} f(\mathbf{x}) = &-20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right) \\ &- \exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)\right) + 20 + e \end{aligned} \tag{20}$$

The Griewank test function is defined by equation (21). It also shows several local optima within the search region defined by $[-600, 600]$. Figure 8 shows its graphical representation for the case of two variables. It is similar to the Rastrigin function, but the number of local optima is larger in this case. It only has one global optimum located at the point $\mathbf{o} = (0, 0, \ldots, 0)$.

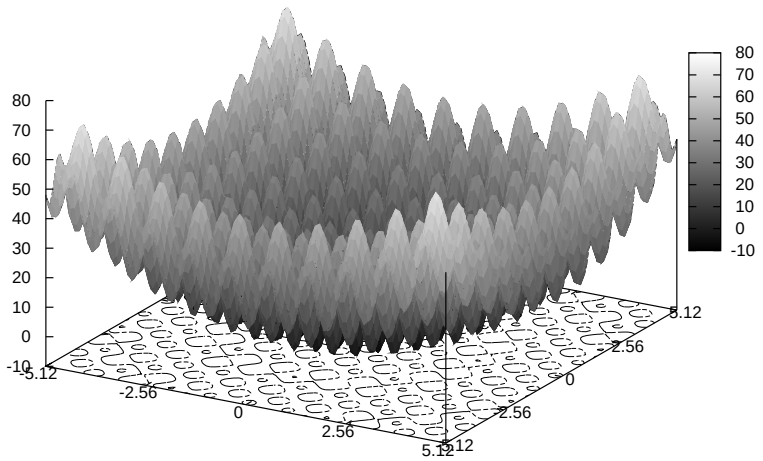

Fig. 6. Graphical representation of the Rastrigin test function in two dimensions.

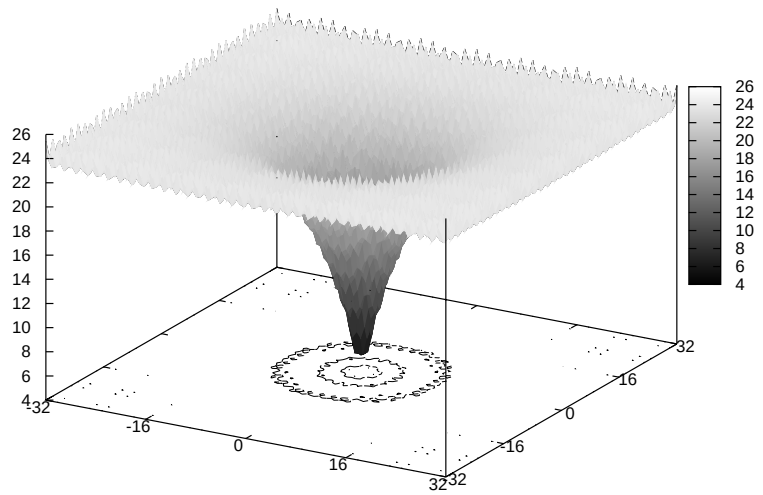

Fig. 7. Graphical representation of the Ackley test function in two dimensions.

$$f(\mathbf{x}) = \sum_{i=1}^{D} \frac{x_i^2}{4000} - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \tag{21}$$

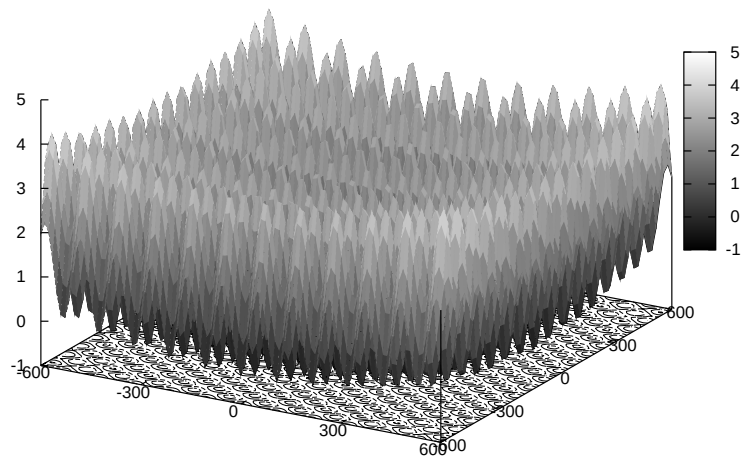

Fig. 8. Graphical representation of the Griewank test function in two dimensions.

The test function shown in this section correspond to those that are most commonly adopted in the specialized literature. They can be scaled up to any number of decision variables. There are also other commonly used test functions that were not included here because they are defined with only one or two decision variables, and are not scalable. We also did not include test functions such as Schubert's function, which has multiple global optima, since they are more suitable for methods dedicated to locate more than one optima in a single run. The interested reader is referred to the work of Bratton and Kennedy [17] for more test functions of this sort.

6 Generating Test Functions

This sections describes how to generate a new test function using a set of available test functions, by applying transformations and function composition.

6.1 Using linear transformations

We begin with the most simple function and a translation transformation. The Sphere test function is known to be a very simple function, since it only has one optimum centered in the origin. In order to change the position of the optimum of the Sphere function, we can apply a translation transformation. For example, if we use a translation transformation to move the location of the optimum of the Sphere function in two dimension from the origin to the point $(50, 50)$, we only need to multiply each point by a translation matrix before applying the Sphere function. That is, starting with a point $P = (x, y)$, we first add the dummy variable $w = 1$ and we get the point $P' = (x, y, 1)$. Then, we multiply this point by the matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -50 & -50 & 1 \end{bmatrix} \tag{22}$$

To obtain the translated point $Q' = (x - 50, y - 50, 1)$, we then drop the last coordinate of the point Q' in order to obtain the point $Q = (x-50, y-50)$. Finally, we apply the Sphere function to the point Q. Although this may look like a complicated operation to simply translate the optimum of the Sphere function, as we add more transformations, the procedure remains the same, and we only need to build a matrix representing all the transformations.

The effect of the translation applied to the Sphere function is shown in Figure 9. It is worth noting that the values used in the translation transformation are *negative*, and they translate the optimum to a *positive* position. This may be counterintuitive, but it has a reason: if we use positive values in the translation transformation $T_x = 50, T_y = 50$, the point where the optimum is located $\mathbf{o} = (0, 0)$ is translated to $\mathbf{o}' = (-50, -50)$. If we want the new optimum to be located at $P = (50, 50)$ after the translation, the values of T_x, T_y that we need to adopt for the translation transformation must be $T_x = -50, T_y = -50$. After applying the translation to the point $P = (50, 50)$, the outcome is the translated point $Q = (0, 0)$. So, the position of the optimum if we use as translation values $T_x = -50, T_y = -50$ will be located at the point $P = (50, 50)$. If we wish to translate the optimum to another point displaced by an amount x_1, y_1, we must use the values $T_x = -x_1, T_y = -y_1$ for the translation transformation.

It is worth mentioning that certain transformations do not have effect on the Sphere function. For example, a rotation does not have any effect on the complexity of this test function. In fact, rotation has no effect on any function with *radial symmetry*. To break the radial symmetry in the Sphere test function in two dimensions, we can apply a scaling transformation in only one of its variables. For example, to shrink its first coordinate, we can use the transformation matrix

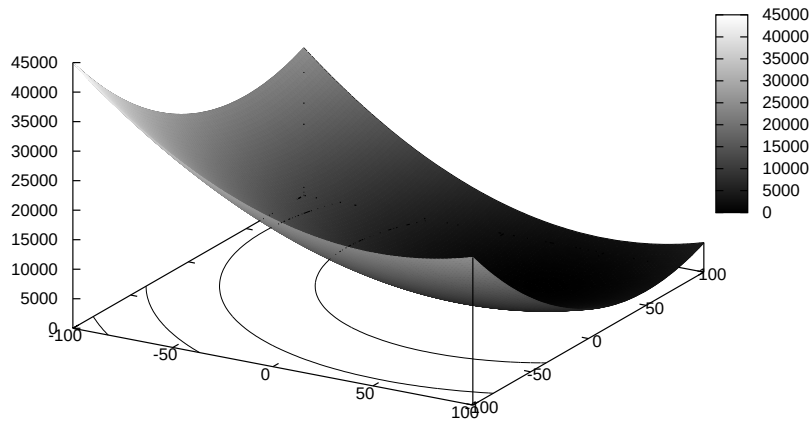

Fig. 9. The Sphere function with the optimum translated from the origin to the point $(50, 50)$.

$$S = \begin{bmatrix} 2.0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{23}$$

As in the case of the translation transformation in which we used the inverse value to translate the optimum, in this case, if we want to shrink by a 0.5 factor, the value that we must use in the transformation matrix is the inverse $1/0.5 = 2$. This also applies to the rotation transformation. The effect of the scaling transformation without a translation transformation is shown in Figure 10.

Without its radial symmetry, a rotation transformation will have an effect in the Sphere test function. A translation of 30 degrees can be done using the following transformation matrix

$$\begin{bmatrix} 0.866 & -0.5 & 0 \\ 0.5 & 0.866 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{24}$$

The graphical representation of the Sphere test function with both scaling and rotation is shown in Figure 11.

So far, we have only applied two transformations at the same time to the Sphere function, but we can apply as many as we wish, without forgetting that

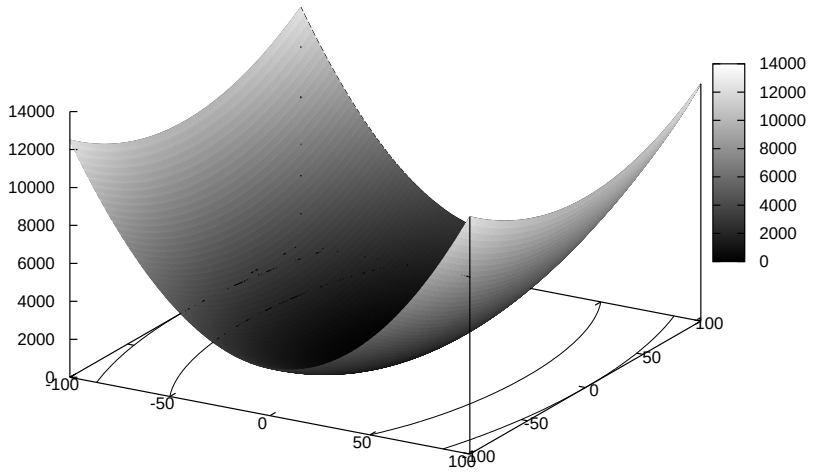

Fig. 10. The Sphere test function scaled.

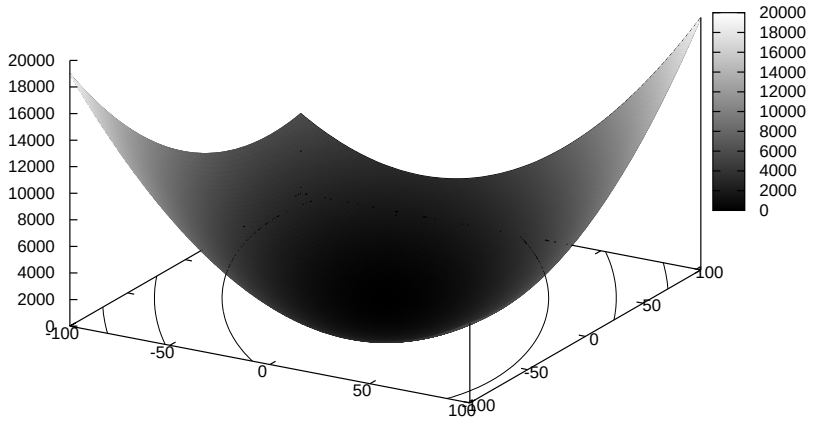

Fig. 11. The Sphere test function with both scaling and rotation transformations.

it is necessary to break the radial symmetry before applying a rotation. As an example, we compute the matrix R that represents three transformations: a translation followed by a scaling and, finally a rotation. Using the same matrices as before, our transformation matrix is computed as follows:

$$R = \begin{bmatrix} 0.866 & -0.5 & 0 \\ 0.5 & 0.866 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -50 & -50 & 1 \end{bmatrix} \tag{25}$$

$$= \begin{bmatrix} 0.866 & 0.5 & 0 \\ -0.5 & 0.866 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ -50 & -50 & 1 \end{bmatrix} \tag{26}$$

$$= \begin{bmatrix} 1.732 & -0.5 & 0 \\ 1.0 & 0.866 & 0 \\ -50 & -50 & 1 \end{bmatrix} \tag{27}$$

The graphical representation of the resulting Sphere test function after applying the transformations is shown in Figure 12

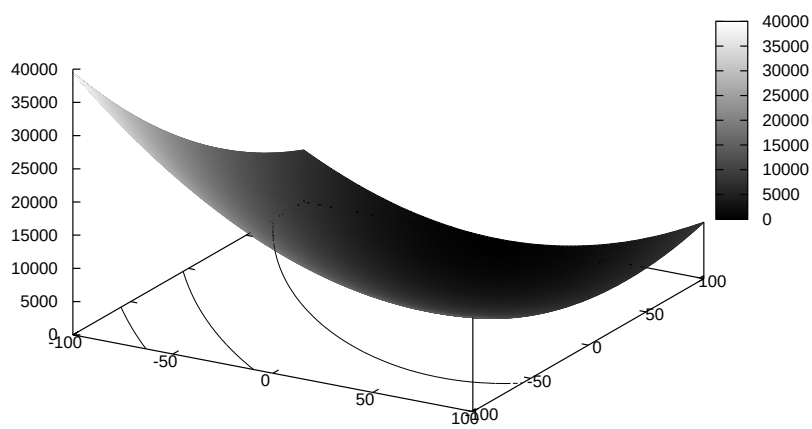

Fig. 12. The Sphere test function after applying three transformations.

The plots of the resulting Sphere test function with the same transformations applied in a different order are shown in Figure 13.

Using linear transformations, we can eliminate some of the drawbacks of a test function, such as having an optimum in a position with repeated values, as

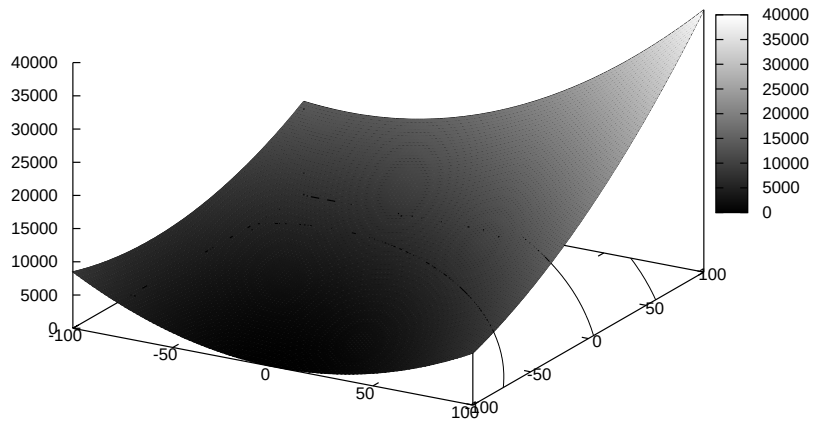

Fig. 13. The Sphere test function with three transformations applied in a different order.

well as having symmetry with respect to the axis, among others. However, we can also add more features to our test functions by using function composition.

6.2 Using function composition

In the previous subsection, we transformed the Sphere test function by changing the position of its optimum, by breaking its radial symmetry, and by adding a rotation with respect to its coordinate axis. However, the resulting Sphere test function still has only one optimum. The are two common options to do the composition of functions; one is to add the results of two or more functions. For example, if we have two Sphere functions f_1 and f_2, we generate the composite function F as follows

$$F(\mathbf{x}) = f_1(\mathbf{x}) + f_2(\mathbf{x}) \tag{28}$$

In general, if we have n functions, we can generate a composition function $F(x)$ by using the formula of equation (29).

$$F(\mathbf{x}) = \sum_{i=1}^{n} f_i(\mathbf{x}) \tag{29}$$

The second type of composition consists in computing the maximum or minimum of all our functions f_i. Again, in the case of two functions f_1 and f_2, the composite function F using the max function is:

$$F(\mathbf{x}) = \max\{f_1(\mathbf{x}), f_2(\mathbf{x})\} \tag{30}$$

and in general

$$F(\mathbf{x}) = \max_i\{f_i(\mathbf{x})\} \tag{31}$$

Both approaches have their own features and drawbacks that are explained using an example in which we generate two composite functions F_1 and F_2 using a sum and the min function, respectively, with two Sphere functions. If we use two Spheres without any transformation, no much complexity is added to the composite functions F_1 nor for F_2, so our Sphere function f_1 will be a standard Sphere function and f_2 will be a Sphere function with some of the transformations defined in the previous section.

We first use only translation on the f_2 function. The composite function F_1 must have two optima. However, an effect of the composition using addition is that the value of F_1 is different from zero in the search space. The f_1 function is only zero at the location of its optimum $P = (0, 0)$. The value of F_1 in the point P is $F_1(P) = f_1(P) + f_2(P) = f_2(P)$, since f_2 is translated $f_2(P) \neq 0$. The same happens at the point Q where the optimum of the f_2 function is located. This has the consequence that the position and value of the optima of the composite function F_1 are different from those of the individual functions f_1 and f_2. A plot of F_1 is shown in Figure 14.

Now, we examine F_2. In this case, we use the min function, and thus, our composite function F_2 is

$$F_2(\mathbf{x}) = \min\{f_1(\mathbf{x}), f_2(\mathbf{x})\} \tag{32}$$

Contrary to the composite function F_1, the optima of the composite function F_2 is located in the same place of the optimum of f_1 and f_2, and the value of the optima is the same. However, if we use the min function to choose the minimal value of the two Sphere functions, it is possible that in some regions, F_2 is not differentiable, and has abrupt changes in its landscape. Figure 15 shows the plot of F_2.

The composite function F_2 has two optima with value 0 in the position of the optima of the f_1 and f_2 functions, but what if we want only one global optimum? The value of the optima can also be changed with a transformation, but in this case the transformation needs to be done after the application of the test function. Until now, all the transformations have taken place before the application of the function. Since we are dealing with single-objective

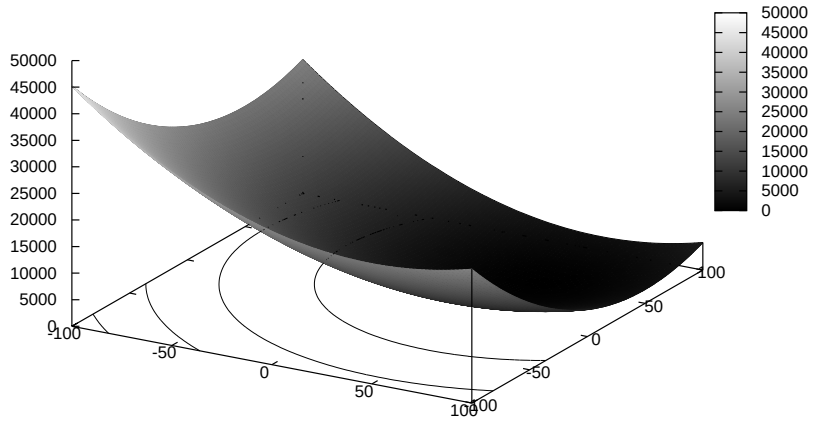

Fig. 14. Composite function F_1.

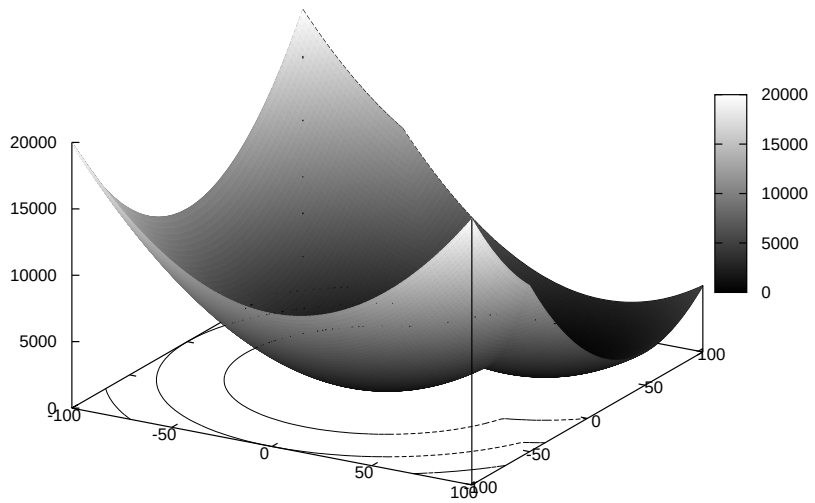

Fig. 15. Composite function F_2.

functions not all transformations can be used. For example, a rotation has no meaning in one dimension. Following our example, we translate the resulting value after applying the Sphere function f_1 by adding a constant value of 5000 and by defining

$$f_1'(\mathbf{x}) = f_1(\mathbf{x}) + 5000 \tag{33}$$

Our composition functions are now defined as

$$F_1(\mathbf{x}) = f_1'(\mathbf{x}) + f_2(\mathbf{x}) \tag{34}$$

$$F_2(\mathbf{x}) = \min\{f_1'(\mathbf{x}), f_2(\mathbf{x})\} \tag{35}$$

The plot of the two composite functions is shown in Figure 17. The translation transformation applied to f_2 after passing the point to the Sphere function can also be represented as a matrix in homogeneous coordinates. In this case, it is represented by a 2×2 matrix, and a scaling transformation can be represented in homogeneous coordinates as well.

Following the example we can describe a general procedure to generate a new test function using linear transformations and function composition as follows:

1. We begin with a point P in the search space.
2. A point Q in homogeneous coordinates is computed using P.
3. The point Q is multiplied by a matrix representing a sequence of linear transformations to obtain Q'.
4. Using Q', we compute P', which gives us the point P transformed in the search space.
5. The value of the test function f_i is obtained passing the point P' as the argument to the test function, and $f(P')$ is computed.
6. Before computing the composite function F, we can additionally apply other linear transformations to $f_i(P')$ and obtain a $f_i'(P')$.
7. Finally, we compute the composite function F by adding the $f_i(P')$ values or by computing the max of them.

Although we only used the Sphere test function in our examples, any other test function can be adopted for the composition previously described. Some considerations must be taken when we are using different test functions in the composition. One of them is the search range, For example, if we wish to make a composite function using the Sphere and Ranstrigin tests functions, we need to consider that the range for the Sphere test function is usually $[-100, 100]$ and in the case of the Rastrigin test function the range is $[-5.12, 5.12]$. If we use the range of $[-100, 100]$ in the Rastrigin test function, we obtain the plot shown in Figure 18. As we can observe, it looks like the Sphere function and does not show any of the original features of the Rastrigin function. Thus, it is necessary to apply a scaling transformation to it.

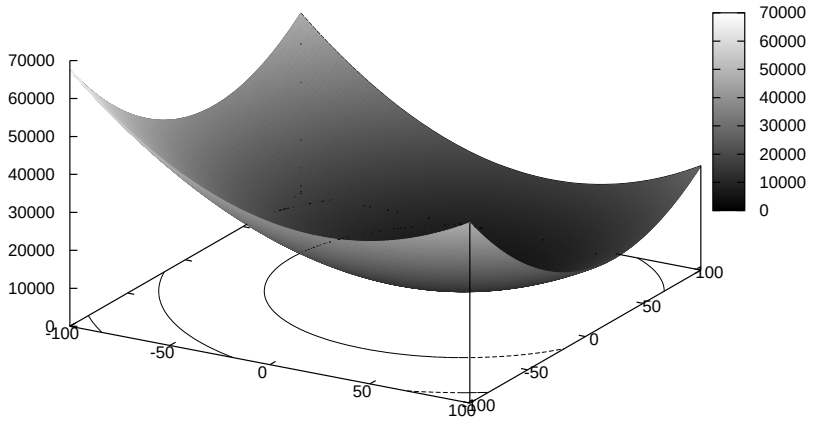

Fig. 16. Composite function F_1.

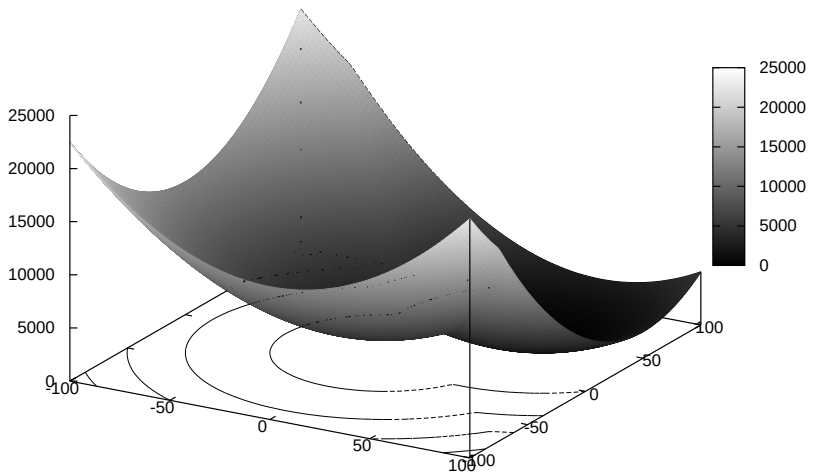

Fig. 17. Composite function F_2.

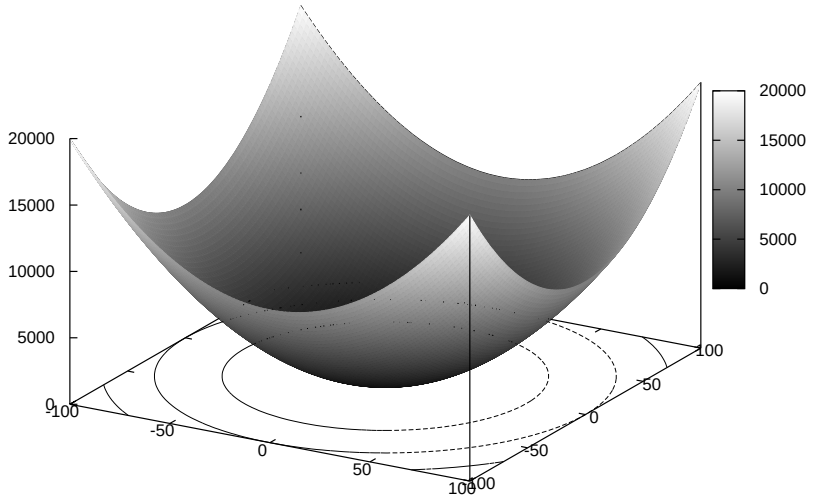

Fig. 18. The Rastrigin test function in the range $[-100, 100]$.

As a rule of thumb we first *normalize* by dividing by the length of the range of the function being scaled, and then we multiply by the length of the new range. In the case of the Rastrigin function, a scaling factor of $200/10.24 = 19.53$ is needed. In the transformation matrix we use the inverse of this value. If we scale the values of the search range, this does not mean that the values after applying the Rastrigin function are scaled. If we compare the plots of the Sphere and Rastrigin test functions (see Figures 2 and 6), the values that are computed using the Rastrigin test function are far smaller than those of the Sphere test function. It is then necessary to apply a scaling transformation after computing the values of the Rastrigin function. In general, the scaling factor depends on the maximum value of the functions involved. Such value may not be easy to compute. The plot of the composition of the Sphere and Ranstrigin test functions using scaling in the search range and after computing the Rastrigin value is shown in Figures 19 and 20.

Now we can generate new test functions from the common test functions using linear transformations and function composition. However, we should keep in mind the recommendations previously provided about the composition using the addition of two functions or the max or min functions, and be careful about the proper scaling of the search ranges and values.

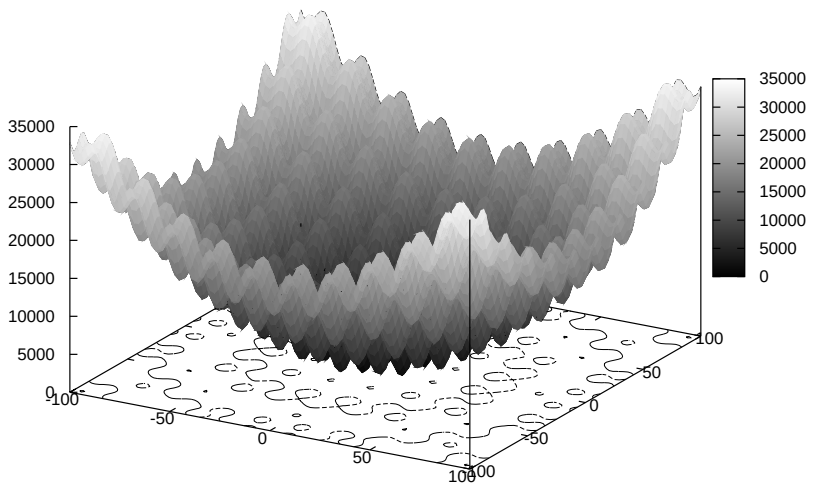

Fig. 19. The composition of the Rastrigin and the Sphere test functions using addition.

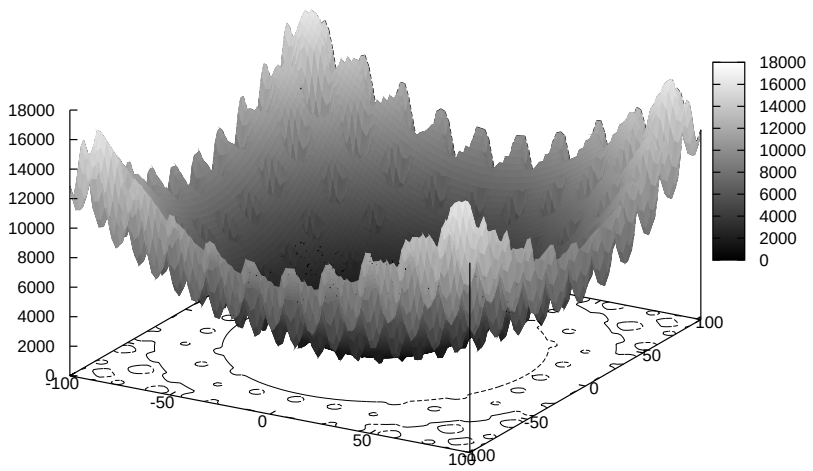

Fig. 20. The composition of the Rastrigin and the Sphere test functions using the min function.

7 Overview of Test Function Generators

In this section, we describe the state-of-the-art regarding test function generators involving the transformations and function composition procedures described in Sections 3 and 4.

We first examine the test function generator of Liang et al. [4]. In this work, the authors proposed a composition of functions using the formula of equation (36).

$$F(\mathbf{x}) = \sum_{i=1}^{n} \{w_i * [f_i'((\mathbf{x} - \mathbf{o}_i + \mathbf{o}_{old})/\lambda_i * M_i) + bias_i] + f_{bias}\} \tag{36}$$

If we examine equation (36), the argument of f_i' is a composition of linear transformations:

$$(\mathbf{x} - \mathbf{o}_i + \mathbf{o}_{old})/\lambda_i * M_i \tag{37}$$

The authors use a translation first to center the optimum of the function in the origin of the coordinate system with the $+\mathbf{o}_{old}$ factor. Then, we translate the optimum to a new position with the factor $-\mathbf{o}_i$. As we see in Section 3, it is not necessary to move first the optimum to the origin, since this displacement can be represented with a translation matrix T_i using homogeneous coordinates. Thus, equation (37) can be rewritten as follows

$$(\mathbf{x}T_i)/\lambda_i * M_i \tag{38}$$

Dividing by λ is also a linear transformation, and in this case, it is a *homogeneous* scaling transformation since they use the same scaling factor for each variable. This can also be represented by a scaling matrix S_i with a $1/\lambda_i$ scale factor in each element of the diagonal except for the last element (see Section 3). Again, equation (37) is rewritten as

$$\mathbf{x}T_iS_i * M_i \tag{39}$$

The $*$ operator in equation (37) represents a matrix product. Given M_i, which is a rotation transformation matrix, equation (37) can be expressed as a product of matrices representing linear transformations and the vector that represents a point in the search space, as expressed in equation (40).

$$\mathbf{x}T_iS_iM_i \tag{40}$$

Defining $R_i = T_iS_iM_i$, equation (36) can be written as

$$F(\mathbf{x}) = \sum_{i=1}^{n} \{w_i * [f_i'(\mathbf{x}R_i) + bias_i] + f_{bias}\} \tag{41}$$

Writing equation (36) in this form, it is easy to observe that after applying f_i' to the point $\mathbf{x}R_i$, a translation transformation is applied using the term $+bias_i$. Analogously, a scaling transformation is applied using the factor w_i. However, these transformations are done after computing f_i'. If we represent the translation and scaling transformations with the matrices T_i' and S_i', respectively, equation (36) has the form

$$F(\mathbf{x}) = \sum_{i=1}^{n} \{f_i'(\mathbf{x}R_i)T_i'S_i' + f_{bias}\} \tag{42}$$

and we can define $R_i' = T_i'S_i'$, and

$$F(\mathbf{x}) = \sum_{i=1}^{n} [f_i'(\mathbf{x}R_i)R_i' + f_{bias}] \tag{43}$$

If f_{bias} is constant, it can be added as a second translation transformation T_i'', defining $R_i' = T_i'S_i'T_i''$. Therefore, equation (36) is reduced to

$$F(\mathbf{x}) = \sum_{i=1}^{n} f_i'(\mathbf{x}R_i)R_i' \tag{44}$$

After computing the transformation matrices R_i and R_i', the computation of the composite function $F(\mathbf{x})$ is easily done using matrix multiplication. The values used for the scaling transformations w_i and λ_i are related to the maximum value of the functions f_i and to the relative size of the search space, respectively, as indicated in Section 6.

In the work of Singh and Deb [18] a test function is proposed, providing the desired positions of the optima and a radius that defines the region in which the optimum is located. The function is described by equation (45).

$$f(\mathbf{x}) = \begin{cases} h_k \left[1 - \frac{d(x,k)}{r_k}^{\alpha_k}\right], & \text{if } d_{ik} \leq r_k \\ 0, & \text{otherwise} \end{cases} \tag{45}$$

This is more like a max function applied on a radius basis. It is not a coincidence that the function is similar to the equation to compute fitness sharing in genetic algorithms [19]. As we saw in Section 6, the use of a max or a min function allows the definition of regions in which the composite

function is not differentiable. In this case, in the regions in which there is no intersection between the regions that contain the optima, the composite function f is constant with a value of 0. Another drawback is that depending on the value of α_k, the regions in which the optima lie can be very sharp.

In the work of Gallagher and Yuan [20], they use composition of exponential functions. The general equation for exponential functions is shown in equation (46).

$$g(x) = \left[\frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)\Sigma^{-1}(x-\mu)^T\right)\right]^{\frac{1}{n}} \tag{46}$$

where μ is a vector of means, that is, basically a translation of the center of the exponential. In this case, Σ is a square $n \times n$ covariance matrix, which corresponds to both scale and rotation transformations. They also suggest two types of composite functions:

$$F(x) = \sum_{i=1}^{m} w_i g_i(x) \tag{47}$$

and

$$G(x) = \max_i \{w_i g_i(x)\} \tag{48}$$

We discussed the features of each of these approaches in Section 6. It is easy to show that the baseline function in equation (46) is the exponential function

$$e^{x^2} \tag{49}$$

As they do not use homogeneous coordinates, the argument

$$(x-\mu)\Sigma^{-1}(x-\mu)^T \tag{50}$$

represents translation, rotation and scaling transformations. In homogeneous coordinates, the argument can be rewritten as

$$(xTRS)^2 \tag{51}$$

with T, R and S being translation, rotation and scaling transformations, respectively.

In a recent work of Rönkkönen et al. [3], a test function generator is proposed, in which three families of functions are defined: Cosine, Quadratic and Common families. They use rotation, translation and a non-homogeneous scaling using Bezier functions. They do not use homogeneous coordinates. The Cosine family is defined by equation (52).

$$f_{cos}(y) = \frac{\sum_{i=1}^{D} -\cos((G_i - 1)2\pi y_i) - \alpha \cos((G_i - 1)2\pi L_i y_i)}{2D} \tag{52}$$

The functions of the Quadratic family are computed using equation (53).

$$f_{quad}(x) = \min_i \left[(x - p_i)^T B_i^{-1}(x - p_i) - v_i\right] \tag{53}$$

As indicated before, the argument of these functions can be written as $(xTRS)^2$ which is the Sphere function. The Quadratic family coincides with the composition of the Sphere functions using the min function as described in Section 6. The Common family function consists of the Branin, Himmelblau, Shubert, Six-hump camelback, Vincent and the 1st & 3rd Deb's function which are functions in which the number of variables cannot be incremented. They do not use homogeneous coordinates either.

Finally, the work of Morrison and De Jong [21] is one of the few test functions generators designed to test algorithms for dynamic environments. They use function composition with the max function described by equation (54).

$$f(x, y) = \max_i \left[H_i - R_i\sqrt{(x - x_i)^2 + (y - y_i)^2}\right] \tag{54}$$

This is the Sphere function with the addition of a square root and the use of both a translation transformation H_i, and a scaling transformation R_i. They do not use rotation, since the scaling transformation is homogeneous and a rotation does not have any effect, as indicated in Section 6.

8 Guidelines to build a test function generator

So far we have discussed the basics of the test functions generators such as linear transformations and function composition, giving some advice on how to apply linear transformations to obtain a desired feature, and on the types of function compositions most commonly used. We have also reviewed some of the state-of-the-art test function generators available in the specialized literature. Using this knowledge, we can now offer some guidelines to create a test function generator.

The use of homogeneous coordinates is encouraged. Building a routine to transform a point in the search space into homogeneous coordinates is simple

and the transformation itself does not consume a big amount of computer time. However, by using it, we are allowed to write all our linear transformations as a single matrix, and we can apply all of them to a point in a single operation. Returning a transformed point from homogeneous coordinates to the search space also requires a simple operation.

Using linear transformations allows to modify the properties of a test function without altering it at all. Thus, it is not necessary to build new test functions, since those already available can be used. A suitable combination of linear transformations can break the regularities of a test function, and there is no limit on the number of linear transformation that can be used. However, it is important to keep in mind that some regularities cannot be easily changed with linear transformations, as when having radial symmetry, or a regular spacing of the optima as in the Rastrigin function.

We are not limited to linear transformations, since function composition can be used to create new test functions. Like when using linear transformations, any available test function can be used in function composition. In previous sections we have described the features and drawbacks of the two approaches most commonly used for function composition. The use of any of them should be decided according to the features that are desired for the test functions to be generated.

Finally, the use of test function families can simplify the process of creating new test functions. For example, quadratic functions are easy to implement and can be manipulated using linear transformations to break regularities. Their use in function composition is also simple, since the search range and the function values do not change much when linear transformations are applied.

9 Conclusions

It is necessary to have test functions with properties that are more challenging for any optimization algorithm, particularly, those of metaheuristic nature (such as particle swarm optimization). We believe that this would lead to the development of more robust and effective algorithms.

Most of the test functions currently available for validating single-objective particle swarm optimization algorithms have regularities such as symmetry, uniform location of the optima, etc. These features can be exploited by metaheuristics such as PSO, and may turn out to be not as difficult to solve as originally intended.

Some of these drawbacks of the currently available test functions can be avoided by adopting transformations, but, as seen in this chapter, a more in-depth knowledge of the effect of such transformations is required before applying them, in order to avoid unexpected side-effects. Another alternative is to use function composition, but again, some previous knowledge about this procedure is required as well, in order to obtain the intended effects.

In this chapter, we have also reviewed the most representative test function generators reported in the specialized literature. Our review has shown that such generators are relatively complicated to implement, and that, in some cases, the test functions generated do not exhibit features that are sufficiently challenging for a metaheuristic. It is evidently necessary to produce new test function generators which are easier to implement, to configure and to use, and that, at the same time, produce test functions with more challenging features. The main intention of this chapter has been, precisely, to motivate the design of such a test function generator, adopting the transformations and function composition procedures described here.

Following the examples presented in this chapter, it can be clearly seen that the use of homogeneous coordinates simplifies the application of linear transformations. However, homogeneous coordinates are not used in any of the test function generators that we found in the specialized literature.

10 Future work

The development of a truly simple and configurable test function generator is still an active topic of research. As we have seen in this chapter, the use of homogeneous coordinates can simplify the computations in a test function generator, and provides a more intuitive use of the linear transformations, hence the importance of incorporating homogeneous coordinates in test function generators.

We also believe that the generation of test functions must be focused on the features that we are interested on (e.g., non-uniform location of the local optima), rather than on the complexity of the test function itself (e.g., high nonlinearity in the objective function), since very scarce evidence exists regarding the actual features that turn out to be difficult for an algorithm such as PSO (or any other metaheuristic for that sake).

It is also desirable that the test function generators offer enough flexibility to allow a variety of combinations of features that we are interested in analyzing. Some of the most popular test functions in the current literature do not offer such flexibility. For example, in the Rastrigin function the optima are arranged in a regular lattice and it is impossible to break this property using linear transformations. The use of quadratic functions is therefore, more suitable, since it allows the addition of as many optima as needed, and each optima can be individually manipulated.

Acknowledgements

The first author acknowledges support from CONACyT through a postdoctoral fellowship at the Computer Science Department of CINVESTAV-IPN. The second author acknowledges support from CONACyT project no. 103570.

References

1. Engelbrecht, A.P.: Fundamentals of Computational Swarm Intelligence. John Wiley & Sons (2006)
2. Salomon, R.: Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions. a survey of some theoretical and practical aspects of genetic algorithms. Biosystems **39**(3) (1996) 263–278
3. Rönkkönen, J., Li, X., Kyrki, V., Lampinen, J.: A generator for multimodal test functions with multiple global optima. In: Proceedings of the 7th International Conference on Simulated Evolution and Learning (SEAL '08), Berlin, Heidelberg, Springer-Verlag (2008) 239–248
4. Liang, J., Suganthan, P., Deb, K.: Novel composition test functions for numerical global optimization. In: Proceedings of the 2005 IEEE Swarm Intelligence Symposium (SIS '05). (June 2005) 68–75
5. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of the IEEE International Conference on Neural Networks. Volume 4. (December 1995) 1942–1948
6. Shi, Y., Eberhart, R.: A modified particle swarm optimizer. In: Proceedings of the 1998 IEEE International Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence. (May 1998) 69–73
7. Shi, Y., Eberhart, R.: Empirical study of particle swarm optimization. In: Proceedings of the 1999 Congress on Evolutionary Computation (CEC '99). Volume 3. (1999) 1950
8. Clerc, M., Kennedy, J.: The particle swarm - explosion, stability, and convergence in a multidimensional complex space. IEEE Transactions on Evolutionary Computation **6**(1) (February 2002) 58–73
9. Eberhart, R., Kennedy, J.: A new optimizer using particle swarm theory. In: Proceedings of the Sixth International Symposium on Micro Machine and Human Science (MHS '95). (October 1995) 39–43
10. Kennedy, J.: Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance. In: Proceedings of the 1999 Congress on Evolutionary Computation (CEC '99). Volume 3. (1999) 1938
11. Kennedy, J., Mendes, R.: Population structure and particle swarm performance. In: Proceedings of the 2002 Congress on Evolutionary Computation (CEC '02), Washington, DC, USA, IEEE Computer Society (2002) 1671–1676
12. Passaro, A., Starita, A.: Particle swarm optimization for multimodal functions: a clustering approach. Journal Artificial Evolution and Applications **8**(2) (2008) 1–15
13. Bird, S., Li, X.: Enhancing the robustness of a speciation-based pso. In: IEEE Congress on Evolutionary Computation (CEC '06). (2006) 843–850
14. Liang, J., Qin, A., Suganthan, P., Baskar, S.: Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. IEEE Transactions on Evolutionary Computation **10**(3) (June 2006) 281–295
15. de Oca, M.M., Stützle, T., Birattari, M., Dorigo, M.: Frankenstein's PSO: A Composite Particle Swarm Optimization Algorithm. IEEE Transactions on Evolutionary Computation **13**(5) (October 2009) 1120–1132
16. Harrington, S.: Computer Graphics: A Programming Approach. Second edn. Mcgraw-Hill College Press (1987)

17. Bratton, D., Kennedy, J.: Defining a standard for particle swarm optimization. In: Proceedings of the 2007 IEEE Swarm Intelligence Symposium (SIS '07). (April 2007) 120–127
18. Singh, G., Deb, K.: Comparison of multi-modal optimization algorithms based on evolutionary algorithms. In: Proceedings of the 8th annual conference on Genetic and evolutionary computation (GECCO '06), New York, NY, USA, ACM (2006) 1305–1312
19. Deb, K., Goldberg, D.E.: An Investigation of Niche and Species Formation in Genetic Function Optimization. In Schaffer, J.D., ed.: Proceedings of the Third International Conference on Genetic Algorithms, San Mateo, California, George Mason University, Morgan Kaufmann Publishers (June 1989) 42–50
20. Gallagher, M., Yuan, B.: A general-purpose tunable landscape generator. IEEE Transactions on Evolutionary Computation **10**(5) (Octuber 2006) 590–603
21. Morrison, R., Jong, K.D.: A test problem generator for non-stationary environments. In: Proceedings of the 1999 Congress Evolutionary Computation (CEC 99). Volume 3. (1999) 2053