Soft Comput (2011) 15:1689–1706
DOI 10.1007/s00500-010-0611-1

FOCUS

A framework for generating tunable test functions for multimodal optimization

Jani Rönkkönen · Xiaodong Li · Ville Kyrki · Jouni Lampinen

Published online: 11 July 2010

Abstract Multimodal function optimization, where the aim is to locate more than one solution, has attracted growing interest especially in the evolutionary computing research community. To evaluate experimentally the strengths and weaknesses of multimodal optimization algorithms, it is important to use test functions representing different characteristics and various levels of difficulty. The available selection of multimodal test problems is, however, rather limited and no general framework exists. This paper describes an attempt to construct a software framework which includes a variety of easily tunable test functions. The aim is to provide a general and easily expandable environment for testing different methods of multimodal optimization. Several function families with different characteristics are included. The framework implements new parameterizable function families for generating desired landscapes. Additionally the framework implements a selection of well known test functions from the literature, which can be rotated and stretched. The software module can easily be imported to any optimization algorithm implementation compatible with the C programming language. As an application example, 8 optimization approaches are compared by their ability to locate several global optima over a set of 16 functions with different properties generated by the proposed module. The effects of function regularity, dimensionality and number of local optima on the performance of different algorithms are studied.

Keywords Multimodal optimization · Test function generator · Global optimization · Differential Evolution · Niching · Crowding

J. Rönkkönen (✉) · V. Kyrki
Department of Information Technology, Lappeenranta University of Technology, P.O. Box 20, Lappeenranta 53851, Finland
e-mail: jani@ronkkonen.com

V. Kyrki
e-mail: ville.kyrki@lut.fi

X. Li
School of Computer Science and Information Technology, RMIT University, Melbourne, VIC 3001, Australia
e-mail: xiaodong@cs.rmit.edu.au

J. Lampinen
Department of Computer Science, University of Vaasa, P.O. Box 700, Vaasa 65101, Finland
e-mail: jouni.lampinen@uwasa.fi

1 Introduction

Real-world optimization problems often contain multiple global or local optima. Multimodal optimization aims to locate all global optima and sometimes also good local optima of a multimodal function. Evolutionary Algorithms (EAs) have become a popular choice as optimization techniques for many applications and are an interesting candidate for multimodal optimization due to their use of populations, which allows multiple solutions to be searched simultaneously. Examples of real-world multimodal problems having more than one global optima can be found for example from (Crutchley and Zwolinski 2002) or (Dong et al. 2006).

EAs in their original form are typically designed for locating a single global optimum. Many techniques for locating multiple solutions have been developed, commonly referred to as *niching* methods (Mahfoud 1995a). The two most well-known niching methods are probably *crowding* (De Jong 1975) and *fitness sharing* (Goldberg and Richardson 1987). In addition to crowding and fitness

sharing, and their variants, many other niching methods have been developed (Mahfoud 1995b; Beasley et al. 1993b; Harik 1995; Pétrowski 1996; Li et al. 2002). It is noticeable that most of these methods have generally been evaluated using only one- or two-dimensional multimodal test functions. Furthermore, these functions are often defined in a way which does not allow the function to be tuned in terms of the characteristics of multimodal landscapes. For example, for the Shubert function used in (Li et al. 2002), as the number of dimensions increases, the number of global optima grows exponentially ($D \cdot 3^D$, where D is the number of dimensions). There is no way to control the number of optima, nor how they are distributed. Additionally, the problem is separable and the global optima are positioned at regular intervals in the search space, both being easily exploitable features. In short, using solely the currently available selection of test functions is typically inadequate for proper analysis of the characteristics of different multimodal optimization algorithms. In the light of the no free lunch theorem (Wolpert and Macready 1995, 1997), which states that no optimization algorithm can outperform another over the set of all possible problems, it becomes increasingly important to differentiate the characteristics of the subset of problems in which each algorithm excels. This was also noted in (Whitley et al. 2006), which demonstrates the ability of different optimization algorithms to exploit different problem features. For the above reasons, a set of parameterizable functions is required whose characteristics can be changed independently to isolate the effects on performance of different optimization algorithms.

This paper describes an attempt to construct a software module offering a framework for evaluating the performance of multimodal optimization algorithms in locating multiple solutions. Desirable features of such a framework, which have been used as guidelines for designing the module, include the following:

1. The framework should be easy to use and tunable.
2. It should be possible to transform functions from separable to nonseparable by rotation.
3. There should be regular and irregular distributions of optima.
4. There should be a controllable number of global and local optima.
5. The functions should be scalable to different dimensions.
6. The framework should contain reproducible random functions.
7. The software should be easily expandable and freely available.
8. The framework should facilitate performance measures.

Several function generators able to generate multimodal functions have been previously presented in the literature: DF1 (Morrison and Jong 1999; Morrison 2004) and Moving Peaks (Branke 2002) focus on generating dynamic multimodal landscapes that change over time. Gaviano et al. (2003) generate differentiable multimodal functions with a single global optimum by distorting convex functions using polynomials. The constrained test cases generator (Michalewicz et al. 2000) generates function landscapes by dividing the search space into regions and constructing a unimodal function for each region; the main feature being the ability to define constraint functions for these regions. Macnish (2007) proposed a fractal landscape generator by simulating random meteor impacts. The Max Set of Gaussians (MSG) landscape generator (Gallagher and Yuan 2006) combines several independent peaks to form the function landscape. Liang et al. (2005) demonstrate a method of generating composition test functions in which the composition landscape is formed by combining several standard benchmark functions. Specifically related to genetic algorithms, Weise et al. (2008) introduces a model problem which allows the characteristics of fitness landscapes to be studied. The approach is related to binary-encoded genetic algorithms, in contrast to this paper, which presents encoding independent characteristics for continuous optimization. None of the methods above provide all of the features desired and thus a more versatile testing environment is required which is able to generate a variety of highly tunable, scalable, and controllable multimodal test functions.

In the experimental part of the paper, eight optimization approaches are compared based on their ability to locate global optima. Differential Evolution (DE) (Price et al. 2005) is used as a benchmark and as a base model for all evolutionary methods. The use of a single base model allows the differences between niching methods to become visible. Crowding DE (CRDE) (Thomsen 2004), Speciation-based DE (SDE) (Li 2005) and DE using local selection (DELL) (Rönkkönen et al. 2009) are used as representatives of different niching methods. Two multi-start gradient descent (GD) methods are included to offer a baseline for comparison. In addition, two hybrid approaches combining the gradient descent with local selection DE (DELG) (Rönkkönen et al. 2009) and crowding DE (DECG) are presented and studied.

This paper makes the following contributions: a test function framework for experimental evaluation of multimodal optimization is presented. Two new parameterizable test function families are presented and some existing multimodal test functions are combined inside the framework and their usability extended by allowing rotation and stretching. As an application example, eight optimization approaches are evaluated including two novel hybrid approaches. The results demonstrate the importance of understanding and being able to control the test function

characteristics in order to explain an algorithm's performance.

The remainder of this paper is organized as follows. Section 2 discusses important function features. The software framework is introduced in Sect. 3. Section 4 describes the optimization approaches used in the experimental part and Sect. 5 the used test setup. The results are presented and analyzed in Sect. 6. Section 7 concludes the paper and suggest directions for future research.

2 Important problem features

Global optimization algorithms aim to identify and exploit the features of a problem in the search for a solution. However, the no free lunch theorem (Wolpert and Macready 1995, 1997) states that no optimization algorithm can outperform another over the set of all possible problems. Thus, to compare algorithms we need to compare their ability to identify and exploit different problem features. This section lists such features in the context of multimodal optimization.

2.1 Dimensionality and the number of optima

As the dimensionality of the search space increases, its size grows exponentially. While it is possible to cover the search space extensively in low dimensional functions, this becomes increasingly difficult as the dimensionality increases. Typically for multimodal test functions used in literature the number of optima increases along with the number of dimensions although exceptions exist, like the Griewangk function (Whitley et al. 1996). The number of optima is one of the most important features of a function, especially in the context of multimodal optimization, as it affects also the goal of the optimization. Low dimensionality and a low number of optima favor methods that rely heavily on extensive coverage of the search space.

2.2 Relative area of attraction sizes

The size of the area of attraction (AOA) of an optimum directly indicates the probability for a sample to be placed there by a random placement. If we are able to get a point inside the AOA of each interesting optimum, running a local optimization algorithm from each of the sample points produces the solution. As the differences in relative AOA between optima increase, the optima with smaller AOA become harder to find.

2.3 Relative fitness of optima

The differences between the finesses of the optima are important for niching methods especially when the goal is also to locate local optima. Typically the chance of a niching method to maintain a local optimum is proportional to the optimum's fitness. For example, Mahfoud (1994) demonstrates that for methods using fitness sharing, the minimum population size required to maintain a local optimum of certain fitness is proportional to the relative fitness as well as the number of optima with a better fitness.

2.4 Separability

Separability is a synonym of decomposability. In separable functions the parameters are independent of each other, i.e. "there are no nonlinear interactions between variables" (Whitley et al. 1996, p. 246). A separable function can be optimized by optimizing each parameter individually. A nonseparable function is naturally the opposite of a separable function, meaning that parameters are not independent and optimizing parameters individually no longer works. A separable function can be made nonseparable by rotation. Salomon (1996) demonstrates that the complexity of finding optima of separable functions increases linearly along with the increase in dimensionality. For nonseparable functions the increase is typically exponential.

Epistasis (Beasley et al. 1993a) is a measure of the separability. It measures the number of the nonlinear interactions between the variables. A separable function has minimal epistasis and the amount of epistasis increases with the number of nonlinear interactions. Typically evolutionary algorithms using crossover operators are capable of exploiting separability (low epistasis).

2.5 Directional bias and regularity

Directional bias (Macnish 2007) in multimodal problems means that the optima are located in direct lines. If these lines correspond to the axis directions, a problem becomes separable. While rotation can be used to remove the separability, the directional bias will remain. While not as readily exploitable as separability, an algorithm that is able to adapt itself to the rotation can still benefit from the directional bias.

A function is regular if in addition to having a directional bias, all optima are of equal distance from each other. Thus for each dimension, only a single distance between the optimum points exists. Partially regular functions have several different distances along a single dimension between optima, at least in some of the dimensions. In irregular functions all the distances between each two optima are unique. The degree of regularity represents the proportion of identical distances along each dimension from the total number of such distances. The degree of regularity is maximal in regular functions which only contain single distance for each dimension and

minimal in irregular functions where the number of identical distances is zero.

2.6 Symmetry

A two-dimensional function is symmetric if $F(x_1, x_2) = F(x_2, x_1)$ (Whitley et al. 1996). Up to $D!$ such equivalences may exist for a D dimensional function. Symmetry can be exploited by exchanging the variables of a solution to locate symmetrically positioned solutions. Rotationally symmetric functions remain symmetric regardless of rotation. For example the Schaffer's F6 function (Macnish 2007; Thomsen 2004) is rotationally symmetric.

2.7 Continuity and differentiability

A continuous function does not contain points of discontinuity. Classical optimization methods often require the function to be differentiable, meaning that a meaningful value for a derivative can be calculated for any point. Thus in addition to being continuous, the function must not have discontinuities in slope.

2.8 Global structure

An important characteristic of a function is the overall global structure. For example, the Rosenbrock function (De Jong 1975) is characterized by a banana shaped valley which leads to the optima at [1, 1,..., 1]. Many traditional test functions, like Rastrigin (Törn and Zilinskas 1989) or Griewangk, have the global optimum at origin in the middle of a parabolic valley in the center of the search space. This allows a global optimization algorithm to estimate the direction to the global optimum by exploiting the global structure of the function. Additionally, methods based on averaging have typically a search bias towards the center, which allows the algorithm to exploit the centered global optimum. Another way of exploiting the location of an optimum is by copying parameter values from one dimension to another in cases where all the parameters are of equal value in the global optimum, as in the example functions above.

2.9 Optima on constraints

Optima located on constraints may pose a problem for some optimization methods like the grid based gradient descent (GRGD) implementation used in this paper. On the other hand, optima location on constraints is exploitable because the points on the constraints represent only part of the whole search space. Any algorithm concentrating its efforts on the constraints will thus have an advantage locating such optima.

3 The software framework

The proposed framework can be used to generate multimodal test functions for minimization. The functions are tunable with parameters, to allow generation of landscape characteristics that are specifically designed for evaluating multimodal optimization algorithms by their ability to locate multiple minima. At the moment, four families of functions exist but the software module is easily expandable and new families may be added. The current software version is 1.1 and it includes cosine, quadratic, hump and common families. These are next described in detail, followed by a discussion of issues related to the implementation of the software.

3.1 Cosine family

The cosine family allows functions with a high number of optima to be generated with low computational cost. The degree of regularity and separability may be controlled by rotating and stretching the functions. Two cosine curves are sampled together, of which one defines global minima and the other adds local minima. The basic internal structure is regular: all minima are of similar size and shape and located in rows of similar distance from each other (see Fig. 1a). The function family is defined by

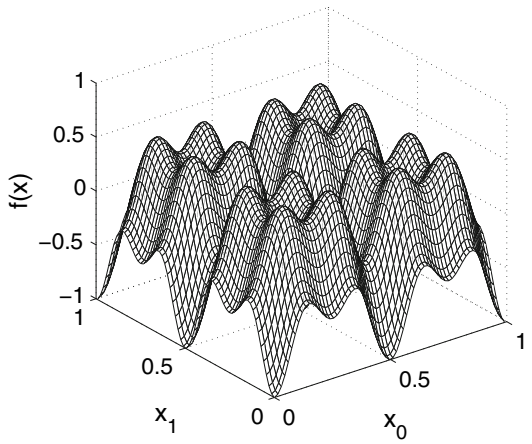
$$f_{\cos}(\vec{y}) = \frac{\sum_{i=1}^{D} -\cos((G_i - 1)2\pi y_i) - \alpha \times \cos((G_i - 1)2\pi L_i y_i)}{2D} \quad (1)$$

where $y \in [0, 1]^D$, D is the dimensionality, the parameters $\vec{G} = (G_1, G_2, \ldots, G_D)$ and $\vec{L} = (L_1, L_2, \ldots, L_D)$ are vectors of positive integers which define the number of global and local minima for each dimension, and $\alpha \in (0, 1]$ defines the amplitude of the sampling function (depth of the local minima).

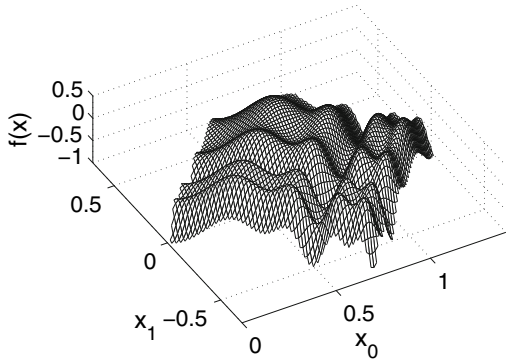
The framework allows the function to be rotated to a random angle and uses Bezier curves (Bezier 1968, 1986) to stretch each dimension independently to decrease the regularity (see Fig. 1b). To calculate the function value for input vector $\vec{x}, \vec{x}$ is first mapped to $\vec{y}$. The mapping proceeds in two steps, where the first step is to calculate $\vec{b}$, which is the rotated point corresponding to $\vec{x}$

$$\vec{b} = \mathbf{O}\vec{\mathbf{x}} \quad (2)$$

where the matrix $\mathbf{O} = [\vec{\mathbf{o}}_1, \ldots, \vec{\mathbf{o}}_D]$ is a randomly generated angle preserving orthogonal linear transformation as described in (Hansen and Ostermeier 2001). The domain of $\vec{x}$ (the search space) is the D-dimensional unit hypercube rotated with $\{\mathbf{O}^T$. Then $\vec{b}$ is mapped to $\vec{y}$ by applying a Bezier formula



(a) Original



(b) Stretched and rotated

Fig. 1 Example figures of two-dimensional cosine family functions using parameter values $\alpha = 0.8, \vec{G} = [3,3], \vec{L} = [2,2]$ (9 global and 16 local minima). Additionally $\vec{P}_1 = [0, 0.1, 0.2, 0.5, 1]$ and $\vec{P}_2 = [0, 0.5, 0.8, 0.9, 1]$ are used for **b**

$$y_i = \sum_{j=0}^{n_i} \binom{n_i}{j} P_{i,j}(1-b_i)^{n_i-j} b_i{}^j, \quad i = 1, \ldots, D \tag{3}$$

where n_i is the degree of the Bezier curve for dimension i and defines the number of the control points used. $\vec{P}_i$ are the control point vectors defined such that $P_{i,0}$ and P_{i,n_i} correspond to the lower and upper bound of y_i and the values between the bounds are strictly increasing.

The Bezier stretching will decrease the regularity of the function but generally not completely eliminate it because the function is regular along directions defined by **O**. The degree of regularity can be roughly measured by considering the minimum number of global minima points required to have a set which contains all possible differentials to jump from a neighboring minimum to the next in the axis directions. For completely regular functions, only $D + 1$ points are required (as long as there is more than one minimum along each dimension). For stretched functions the required number is $\sum_{i=1}^{D}(G_i - 1) + 1$ out of the total number of global minima $\prod_{i=1}^{D} G_i$. So the degree of regularity increases along with the number of dimensions. Bezier stretching also affects the shape and size of the minima, increasing the differences in their AOA.

The number of minima increases exponentially along with the number of dimensions. The number of local minima which are not global is $\prod_{i=1}^{D}[G_i + (G_i - 1)(L_i - 1)]$. For each dimension, two of the outermost minima will always be located on the constraints. This means that if any of the elements of $\vec{G}$ is less than 3, every minimum will be located on at least one constraint. In the unstretched case, each constraint on which the minimum sits, halves the area of attraction for that minimum compared to a minimum with one less constraint. The fraction of the AOA from the full possible area is thus $1/2^{D-l}$, where $l = 0, 1, ..., D$ describes the number of constraints the minimum sits on. $l = 0$ means a minimum located on no constraint (full possible AOA) and $l = D$ is a corner minimum with minimum AOA for dimension D. For methods that rely heavily on the initial points, locating the minima on corners becomes harder with increasing dimensionality, unless the information that the minima are located on the constraints is exploited.

Parameter α affects the depth of the local minima. Increasing the value makes the minima deeper, also increasing their area of attraction and thus slightly increasing the difficulty of the problem. Examples of two-dimensional cosine family functions are presented in Fig. 1.

3.2 Quadratic family

The quadratic family is used to generate completely irregular landscapes and allows the number of minima to be defined independently of the number of dimensions (see Fig. 2). The user may select any number of global and local minima. The function is created by combining several minima generated independently. Each minimum is described as a D dimensional general quadratic form, where a symmetric matrix **C** defines the shape. The functions in a quadratic family need not be stretched or rotated because no additional benefit would be gained since they are already irregular functions. However, axis-aligned (hyper) ellipsoidal minima may be randomly rotated by rotating the matrix **C** as follows:

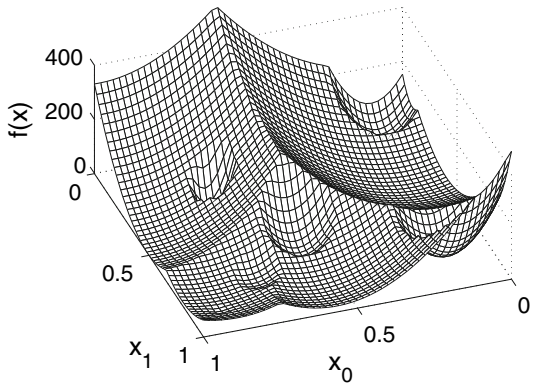
$$\mathbf{B} = \mathbf{OCO^T} \tag{4}$$

The functions are calculated by

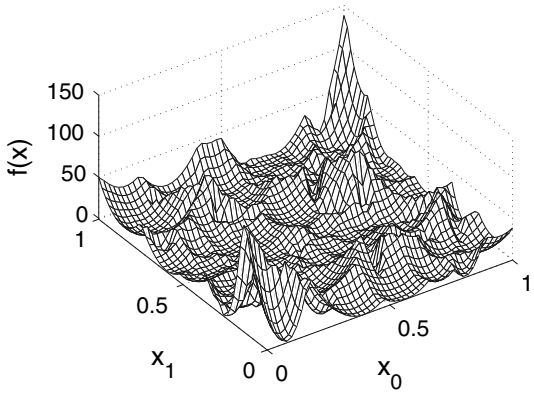
$$f_{\text{quad}}(\vec{x}) = \min_{i=1,2,\ldots,q} \left((\vec{x} - \vec{p}_i)^T \mathbf{B_i^{-1}} (\vec{\mathbf{x}} - \vec{\mathbf{p}}_\mathbf{i}) + \mathbf{v_i} \right) \tag{5}$$

where $\vec{x} \in [0,1]^D, \vec{p}_i$ defines the location, and v_i the fitness value of a minimum point for the i'th minimum. q is the number of minima.

The placement of minima is chosen randomly, although the minimum Euclidean distance between global minima



(a) Spherical, no local minima



(b) Rotated ellipsoidal with local minima

Fig. 2 Examples of two-dimensional quadratic family functions. The minimum Euclidean distance between global minima is set to 0.01, and the shape range for all minima to [0.003,0.03]. **a** Has 10 global spherical minima, while **b** has 10 global and 100 local rotated ellipsoidal minima such that the local minima points have fitness values in the range $[-0.95, -0.15]$. The globally minimal value is always -1

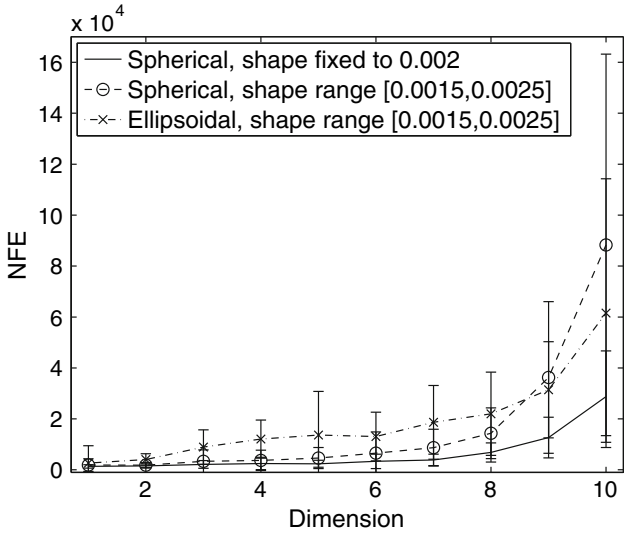
may be defined. The module makes sure that no minimum is completely engulfed by another, deeper minimum. The user may also define the shape of minima, which may be spherical, ellipsoidal or randomly rotated ellipsoidal. This selection is used for all generated minima. The shape range for global and local minima may be defined independently and the shapes are generated by using uniform random numbers from this shape range for each dimension in creating matrix **C**. The user may also define the range for the fitness values of local minima points. Figure 2 presents examples of functions from the quadratic family.

Since the locations of minima are random and the AOAs of minima can be inside each other, the sizes of the areas of attraction for different minima will vary greatly. Using fewer or shallower local minima will naturally leave more room for the global minima. Forcing a longer minimum Euclidean distance between the global minima will also leave more area for each minimum. As the dimensionality increases, the differences in the shape parameters will have an exponentially increasing effect on the AOAs. Small differences in shape parameter values can lead to large differences in the relative sizes of the AOAs in high dimensions.

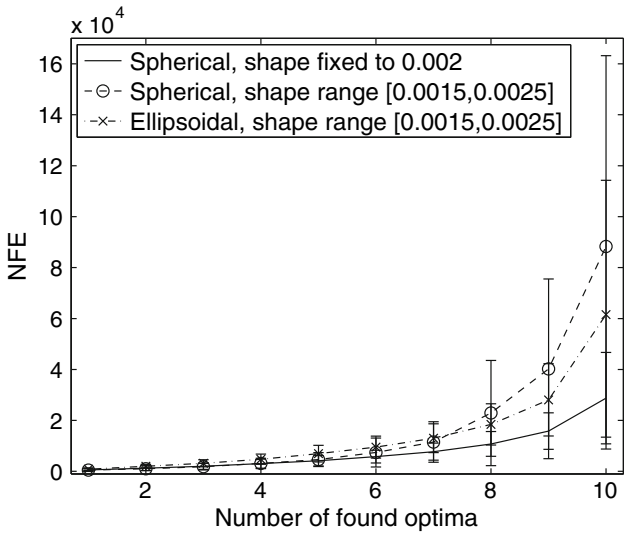
To demonstrate this, a random start gradient descent (RSGD) algorithm was run on three different sets of quadratic functions, with $D = 1, 2, ..., 10$. The average performance of RSGD is a decent estimate of the relative sizes of AOAs because to locate a minimum, the random starting point must be located in the AOA of that particular minimum. If the sizes differ significantly, the larger ones will draw more points, slowing down the process of locating the minima with a smaller AOA. Figure 3 displays the results of the runs. The first set of functions has spherical minima with identical sizes. The second also has spherical minima and the third ellipsoidal minima, both allowing the shape range to change by $\pm 50\%$. As can be seen from the figure, the required number of function evaluations (NFE) as well as the standard deviation increase notably more slowly along with the increase in dimensionality for the first set compared to the other two. This is to be expected because the identical shape eliminates the differences in AOA caused by shape. The ellipsoidal shape is the most difficult to solve when the dimensionality is low, but varying spherical shape becomes more difficult when $D > 8$. This is logical because for a spherical shape only one random value is generated, which is then used in all dimensions. If the value is small, it will affect all dimensions. For an ellipsoidal shape, each dimension will get a different random value and the AOA will on average vary less as more values are generated in higher dimensional cases. The slower performance on the low dimensional ellipsoidal set can be explained by the fact that the gradient descent tends to oscillate on nonspherical shapes and thus needs more line searches to find the minimum point compared to spherical shapes, where the gradient points directly to the minimum point.

3.2.1 Comparison to other methods

The MSG proposed by Gallagher and Yuan (2006) is an interesting test function generator. An approach similar to the quadratic family is used to produce peaks independently and the dominant peak is used to define the function value at point $\vec{x}$. MSG uses a Gaussian density function to define the peaks. For the quadratic family a general quadratic form was selected (which is similar to the exponent part of the Gaussian density function, when the constant is removed) to describe the minima. As a result, the landscapes generated by MSG have much steeper optima shapes as well as large almost flat regions compared to the ones generated by the quadratic family. In theory, both



(a) Function evaluations required to find all minima with accuracy 0.0001



(b) Function evaluations required to find the i'th minimum in the 10-dimensional case

Fig. 3 Performance of RSGD on different quadratic functions with 10 global and no local minima. The minimum Euclidean distance between global minima is set to 0.1. The function evaluations are averages from 100 independent runs and figures include standard deviations. For each run, a different random seed is used in generating the function

approaches allow landscapes with no completely flat areas to be generated. However, a potential problem with MSG is numerical accuracy: away from an optimum, it is possible that the search space contains areas which seem flat because of limited numeric accuracy. When using the quadratic form, this is not an issue.

The composition landscape in the functions proposed by Liang et al. (2005) is formed by combining several basic benchmark functions. When using solely unimodal basic functions, the produced landscape resembles the one generated by the quadratic family. The number and location of both global and local optima may be controlled independently of the number of dimensions. Additionally the basic functions may be rotated, although the only unimodal basic function currently included is the symmetric sphere, on which rotation has no effect. The proposed model allows each basic function to be scaled independently, but each dimension is scaled equally and the shape of the function remains unchanged. Including ellipsoidal unimodal basic functions and adding a dimensional scaling would allow the composition functions to mimic the rotated ellipsoidal optima shapes of the quadratic family. However, the most interesting feature of the composition principle is that it allows the landscape to be constructed of different basic components, which may already be multimodal. While this means that the number of optima would no longer be independent of the number of dimensions and the number and locations of local optima would be harder to define, it would allow the definition of changing landscapes with a lot of optima, which require less computational effort compared to the quadratic family. Thus adding a new family adopting the composition principle might be an interesting option in future development of the framework.

3.3 Hump family

The hump family implements the generic hump functions family proposed by Singh and Deb (2006), see Fig. 4. Like the quadratic family, the hump functions allow irregular landscapes to be generated and the number of minima to be defined independently of the dimensionality. The placement of minima is chosen randomly. Each minimum is defined by

$$f_h(\vec{y}) = \begin{cases} h_i\left[1 - \left(\frac{d(\vec{y},i)}{r_i}\right)^{\alpha_i}\right], & \text{if } d(\vec{y},i) \le r_i \\ 0, & \text{otherwise} \end{cases} \quad (6)$$

where $\vec{y} \in [0,1]^D$, h_i is the value of the ith minimum. $d(\vec{y},i)$ is the Euclidean distance between $\vec{y}$ and the center of the ith minimum. $r_i \in [0.001, \infty)$ defines the basin radius and $\alpha_i \in [0.001, 1]$ the shape of the ith minimum slope.

Each minimum in hump functions has a fixed radius value and all values outside the radius value are set to a constant 0. Thus, the function surface is flat between the minima. This is problematic if small radii are used because we end up with a needle in a haystack problem, where the majority of the search space is flat, including only some very thin holes. Especially for methods relying on gradient information, the flat surface makes gradients unusable. However, the hump family offers a better control over the AOAs of minima compared to the quadratic family because the AOAs cannot intersect. Furthermore, when reasonably large radii for the minima are used, the hump family is suitable for testing the

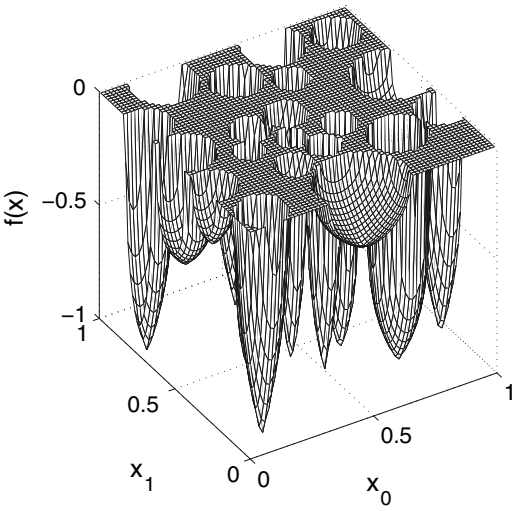


Fig. 4 Example of a two-dimensional hump family function. The function has 10 global and local minima such that the local minima points have fitness values in the range [−0.5, −0.15]. The globally minimal value is always −1. The radii range for all minima is set to [0.05,0.2] and the shape parameter α_i is in the range [0.2, 0.5]

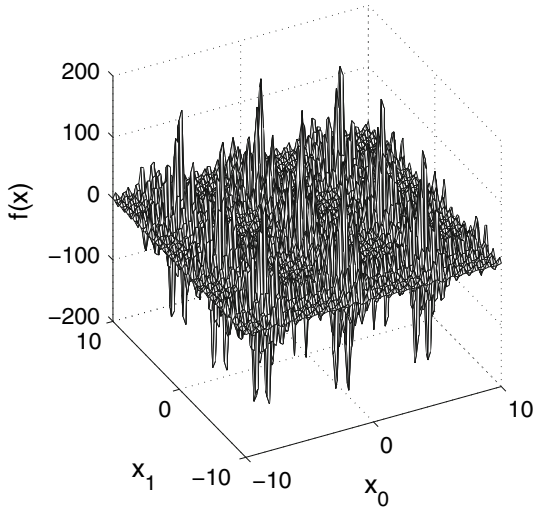


Fig. 5 Shubert function

ability of an algorithm to handle flat areas on a function surface. The minima are always spherical in shape but the hump functions can be rotated and stretched similarly to cosine family functions to change the shape.

3.4 Common family

The common family collects some well known multimodal test problems from the literature and implements them inside the module framework. The module allows similar rotation and Bezier stretching as used in the cosine family for all implemented functions in the common family. At the moment, eight different functions having multiple global minima are implemented.

The two-dimensional Shubert function (Li et al. 2002)

$$f_{\text{shu}}(\vec{y}) = \sum_{i=1}^{5} i\cos((i+1)y_1 + i) \sum_{i=1}^{5} i\cos((i+1)y_2 + i) \tag{7}$$

where $\vec{y} \in [-10, 10]^2$, has 18 global and 742 local minima. The global minima are situated in nine groups of two closely situated minima. The groups form a shape of 3 by 3 square as shown in Fig. 5. The groups and also the minima inside groups are regularly spaced.

The Vincent function (Shir and Bäck 2006)

$$f_{\text{vin}}(\vec{y}) = -\frac{1}{D}\sum_{i=1}^{D} \sin 10\log(y_i) \tag{8}$$

where $\vec{y} \in [0.25, 10]^D$, has 6^D global minima. The function is partially regular, like a Bezier stretched function of the cosine family. The differences between minima are not random but increase along the value of y. Figure 6 displays the function.

The modified two-dimensional Rastrigin function

$$f_{\text{ras}}(\vec{y}) = 20 + \sum_{i=1}^{2}\left(y_i^2 + 10\cos(2\pi y_i)\right) \tag{9}$$

where $\vec{y} \in [-5.12, 5.12]^2$, is a version of the Rastrigin function modified to contain more than one global minimum. It has four regularly spaced global minima and 96 local minima. Figure 7 displays the function.

Other functions included are the Branin (Michalewicz 1996), the Himmelblau (Beasley et al. 1993b), the Six-hump camel back (Michalewicz 1996; Ursem 1999) and Deb's first and third functions (Beasley et al. 1993b).

Additionally the family implements another set of eight functions with a single global minimum and a set of local minima to test the ability of algorithms to locate and keep good local minima in addition to the global ones. The functions are taken from (Thomsen 2004; Ursem 1999) and include the Bohachevsky, the Shekel's foxholes, the Ursem F1, F3, F4 functions and the Ursem waves function. The set also contains the Ripple (Thomsen 2004; Ghosh et al. 2000)

$$f_{\text{rip}}(\vec{y}) = \sum_{i=1}^{2} -e^{-2\ln 2\left(\frac{y_i-0.1}{0.8}\right)^2}\left(\sin^6(5\pi y_i) + 0.1\cos^2(500\pi y_i)\right) \tag{10}$$

and Ripple25 functions

$$f_{r25}(\vec{y}) = \sum_{i=1}^{2} -e^{-2\ln 2\left(\frac{y_i-0.1}{0.8}\right)^2}\left(\sin^6(5\pi y_i)\right) \tag{11}$$

where $\vec{y} \in [0, 1]^2$. The Ripple has 1 global and 252,004 local minima. The global form of the function consists of

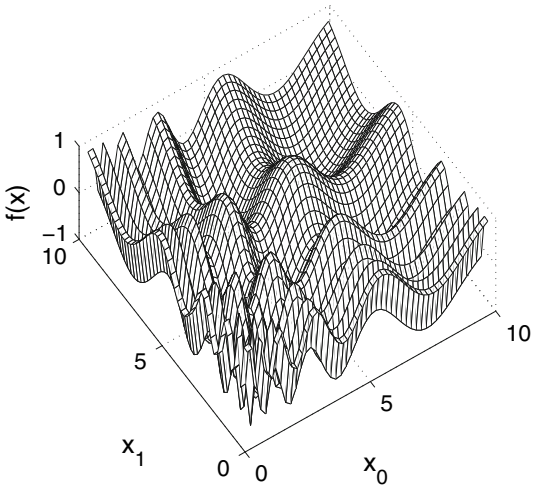


Fig. 6 Two-dimensional Vincent function

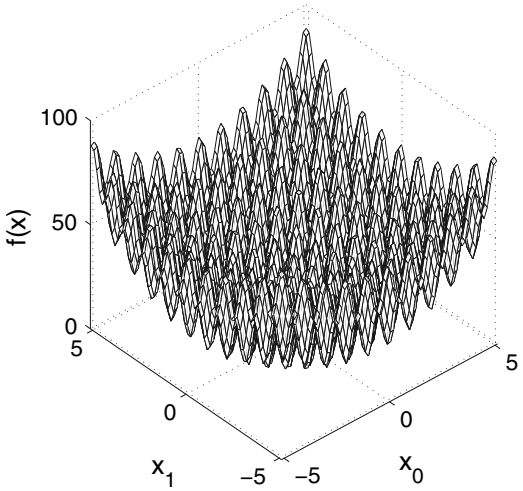


Fig. 7 Modified Rastrigin function

25 holes, which form a 5 by 5 regular grid as shown by Fig. 8. Additionally the whole function surface is full of small ripples caused by the high frequency cosine function, which creates a large number of small local minima. Among the minima, the lowest values of the 25 holes are used to decide an algorithm's success in locating minima. The global minimum is located at the corner of the grid. The Ripple25 contains the global form of the Ripple function without the ripple (Fig. 8). Ripple and Ripple25 can be used as a pair to compare an algorithm's ability to handle local noise.

3.5 Problem features and the proposed framework

Important features of multimodal functions were listed in Sect. 2. This section summarizes the relation of different function families to each of the features.

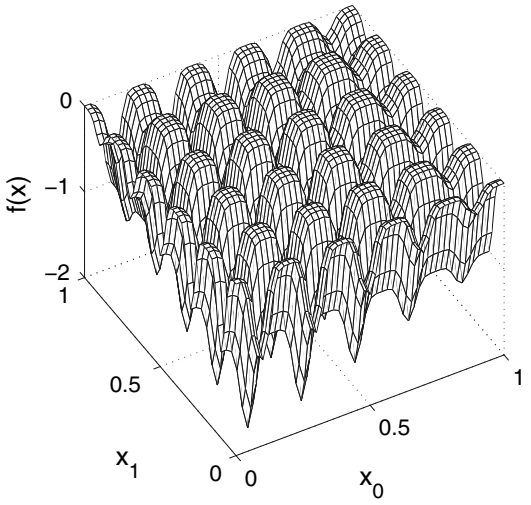


Fig. 8 Ripple25 function

3.5.1 Dimensionality and the number of optima

The cosine, quadratic and hump families allow the dimensionality to be scaled freely. Some common family functions have a fixed dimensionality. The quadratic and hump families allow the number of local and global optima to be set precisely and independently of the number of dimensions. For the cosine and common family functions, the number of optima increases exponentially along with the number of dimensions.

3.5.2 Relative area of attraction sizes

The hump family allows the relative AOAs to be directly controlled. The quadratic family allows similar but less precise control due to the fact that the AOAs of different optima may intersect. In the cosine and common families the relative AOA sizes can be altered by stretching the functions.

3.5.3 Relative fitness of optima

Cosine family functions have groups of local optima with similar fitness whose depth can be controlled. Quadratic and hump families allow the definition of a range in which the fitness of each local optimum is randomly generated. For the common family each individual function defines the fitnesses and they can not be modified.

3.5.4 Separability

Quadratic and hump families have high epistasis because they are always nonseparable. The epistasis of cosine and common family functions can be controlled by rotating the functions.

3.5.5 Directional bias and regularity

Quadratic and hump family functions are irregular and do not embody directional biases. Cosine family functions always have directional bias. Their degree of regularity can be controlled between regular and partially regular by using stretching and controlling the number and positions of the optima. Common family functions can also be stretched, but some of them are already irregular or partially regular by definition.

3.5.6 Symmetry

Quadratic and hump families are not symmetric. Cosine family functions for which G_i and L_i are defined equal for all dimensions are completely symmetric in their basic form. This means that they are symmetric in regard to all the $D!$ possible symmetry equivalences. Stretching and defining varying G_i or L_i of optima for different dimensions can be used to control the amount of symmetry: removing symmetry from n dimensions leaves $(D - n)!$ symmetry equivalences in regard to which the function is symmetric. Some of the common family functions are also symmetric. Using rotation removes the symmetry. The proposed framework does not currently contain rotationally symmetric functions.

3.5.7 Continuity and differentiability

All the functions within the framework are continuous and allow a numerical approximation of the first order derivative to be generated at any point, even if the derivative does not exist in the analytical sense. The framework is not designed to be used with methods which require analytical first or higher order derivatives. Hump family functions and some of the functions in the common family contain areas of flat fitness, which pose a special challenge for methods using derivatives to direct the search.

3.5.8 Global structure

Although the goal of multimodal optimization is to locate multiple optima, averaging and copying can be used to some extent in exploiting the structure in certain functions. Cosine family functions in their basic form will often have one global optimum in the middle of the search space and several optima which have equal values for all or most parameters. Rotation and stretching can be used to remove the equal parameter values from most of the optima, but one global optimum will always be located at the origin. Quadratic and hump families generate the optima locations randomly and thus planned exploitation of global structure is not possible. None of the families currently allow the design of specific types of global structure, like specifically planted valleys or compressions of optima.

3.5.9 Optima on constraints

As the optima locations in quadratic and hump families are randomly selected, they will rarely be positioned on constraints. However, depending on the selection of the $\vec{G}$ and $\vec{L}$ parameters of the cosine family, a significant portion or even all of the global optima may be located on the constraints. For methods using solely the module's internal constraint handling, the search space seems to continue past the constraints and the constraints become irrelevant. However, the module provides also the exact constraint information if required by an algorithm and thus allows it to be exploited.

3.6 Implementation and features

The software is written in the C programming language obeying the ANSI standard. From the outset the idea has been to develop a general framework for evaluating multimodal optimization algorithms. Thus the structure of the software has been designed to allow easy addition of new function families to the module. Figure 9 presents the general structure of the module as an UML class diagram. While ANSI C does not directly support object-oriented concepts, they can be closely imitated. Thus the term class will be used when referring to parts of the program.

Inheritance is imitated by defining a set of function pointers in the generator class which represent the functions provided by the public interface. Each class representing the different function families must provide an implementation for each of the interface functions and an initialization function which connects the function pointers to actual implementations. Adding a new function family thus requires adding a definition of the new initialization function to the generator class, which would not be necessary for languages supporting inheritance through dynamic binding. However, this is seen as a minor inconvenience since in all other respects implementation of the new family can be done completely within its own compile unit. All function calls from the public interface are first

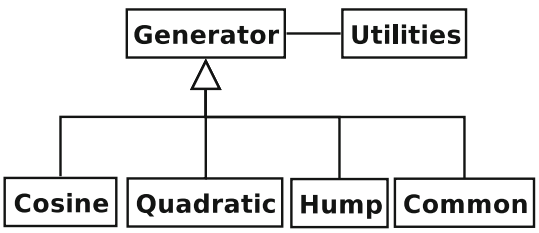


Fig. 9 UML class diagram of the software structure

handled by the generator class, which is responsible for handling tasks collective to all families, and the call is then redirected to the correct family as necessary. Thus each new family needs to contain only implementations directly related to the family. The utilities class provides implementations of functionalities used by multiple families, like rotation, stretching and constraint handling, which are readily available to new families.

The module is used by simply including the header file containing the public interface to optimization software. The user can then directly use all features provided by the module. Function configuration is performed using text files. Each function instance is explicitly specified by an initialization file, except for the seed to the internal random number generator for functions that use randomization. The user may provide the seed through the call of initialization function. The same function is always generated with the same seed. Thus, it is easy to define exact test sets by providing the initialization files and the information of used seeds. These features make the module easy to use.

The module framework includes an internal constraint handling method to keep the solutions within a given range. The constraints are handled by mirroring the violating value back from the violated boundary by the amount of violation. This makes the function space look continuous for the optimization approach during the run because any minimum which is located on the boundary looks symmetrical, although in reality the value is calculated in a mirrored point inside the boundaries. If required, the internal constraint handling can be ignored and the linear constraint functions can be acquired in analytical form.

To help in evaluating the quality of a solution provided by an algorithm, the module offers a method for deciding how many different globally minimal solutions a given population contains with a required accuracy and their exact locations. A similar method is provided for the quadratic, hump and part of the common family functions for deciding the number of found locally minimal solutions. Other useful features included are the possibility of initializing a population with uniform random values in a proper range for the used function, an internal counter for function evaluations, and the ability to acquire the number of minima a function contains.

The software package is freely available from: http://www.ronkkonen.com/generator. The package includes the source codes, a simple plotter program for visualizing the 2D functions generated, written in Matlab, and detailed documentation.

4 Optimization approaches

This section describes the eight optimization approaches used in the experimental part of the study. The methods are compared using the proposed test function framework by measuring their ability to locate global optima.

4.1 Gradient descent

The most fundamental of the approaches are the two multistart local searchers using a gradient descent algorithm. The strategies simply generate new starting points for the GD until the maximum number of function evaluations has been reached. The random start gradient descent uses random starting points while the grid based gradient descent uses a predetermined grid to generate the starting points. The grid is generated to maximize the coverage. First a single point is generated at the middle of the search space in regard to all dimensions. Then at each subsequent generation, new points are generated between existing points and the box constraints such that all distances are halved. Thus the number of points generated at the end of generation g is $(\sum_{i=0}^{g} 2^i)^D$. Since the growth is exponential, each new generation will contain more points than all previous generations combined. Especially with higher dimensions the number of generations completed will be low and the search typically stops short of completing a generation. Using the starting points in a predetermined order would create a bias in the search. For this reason, the points inside each generation are used in a random order.

The GD descends downhill by performing line searches in a numerically estimated gradient direction until all optima have been found or the maximum number of function evaluations have been reached. The GD uses bracketing (Press et al. 1992, p. 400) and Brent's method (Press et al. 1992, p. 404) (parabolic interpolation and golden section search). The gradient vector is approximated with a central difference estimate, using the largest absolute value of a box constraint for any of the dimensions of the problem scaled with 10^{-8} as the distances from the central point. The initial bracketing length of 10^{-3} is used for all cases.

4.2 Differential Evolution

The Differential Evolution algorithm was originally introduced by Storn and Price (1995) and several slightly different strategies exist. For this paper, only the DE/rand/1/bin scheme (Storn and Price 1997) is considered. The abbreviation DEGS stands for DE using global selection, which in a broader sense includes most traditional DE strategies. For convenience, DEGS is used as a synonym for DE/rand/1/bin for the rest of this paper.

DEGS starts from a random initial population. In each generation g, it goes through each D dimensional target vector $\vec{x}_{i,g}$ of the population and creates a corresponding trial vector $\vec{u}_{i,g}$ using mutation

$$v_{j,i,g} = x_{j,r_0,g} + F(x_{j,r_1,g} - x_{j,r_2,g}) \quad (12)$$

and crossover

$$u_{j,i,g} = \begin{cases} v_{j,i,g} & \text{if rand}[0,1] \leq CR \vee j = j_{\text{rand}} \\ x_{j,i,g} & \text{otherwise} \end{cases} \quad (13)$$

operations. The difference between two randomly chosen population vectors $(\vec{x}_{r_1,g} - \vec{x}_{r_2,g})$ defines the magnitude and direction of the mutation. This makes the mutation operation self adaptive, because the average mutation step length decreases as the population converges. To prevent crossover from duplicating the objective vector, $\vec{u}_{i,g}$ always inherits the parameter with randomly chosen index j_{rand} from $\vec{v}_{i,g}$. Indices r_1, r_2, and r_0 are mutually different and drawn from the set of population indices. *rand*[0, 1] is a random number drawn anew for each round from the uniform distribution in the range [0,1].

The control parameters for DE are the crossover rate *C R*, the mutation factor F and the population size *N P*. $F \in (0, 1 +]$ is a scaling factor for the mutation step length and affects the population's convergence speed. Additionally, using a value of $F = 1/\mu$ (where μ is an arbitrary integer) allows DE to efficiently exploit function regularities, as demonstrated by Rönkkönen and Lampinen (2007b); Rönkkönen et al. (2009). The effect is strongest with $F = 1$, as the exploitation requires at least $\mu + 1$ equally spaced optima in a row.

CR $\in$ [0, 1], controls the crossover by determining the average number of parameters the trial vector $\vec{u}_{i,g}$ inherits from the mutated vector $\vec{v}_{i,g}$. An important aspect to consider when using crossover is that the smaller the used value of CR, the less rotationally invariant the search becomes (Price et al. 2005). When the crossover is disabled (CR = 1), the search becomes completely rotationally invariant, i.e. the search is not biased in the axis directions.

At the end of each generation the selection operation

$$\vec{x}_{i,g+1} = \begin{cases} \vec{u}_{i,g} & \text{if } f(\vec{u}_{i,g}) \leq f(\vec{x}_{i,g}) \\ \vec{x}_{i,g} & \text{otherwise} \end{cases} \quad (14)$$

is performed comparing each trial vector $\vec{u}_{i,g}$ to the corresponding target vector $\vec{x}_{i,g}$. If the trial has equal or lower cost, it replaces the target vector. The term global selection refers to the fact that the trial vector $\vec{u}_{i,g}$ has no relation to the target vector $\vec{x}_{i,g}$. This allows the under-performing population members to be replaced by variants of the population's better solutions (Price and Rönkkönen 2006) creating a strong bias to converge towards a single optimum. This makes the algorithm generally ill-suited for multimodal optimization, but offers an interesting point of comparison for the other methods. DEGS is described in Algorithm 1.

Algorithm 1 DE/rand/1/bin (DEGS)

```
Initialize population, g = 1
while termination criterion not met do
  for i = 1; i ≤ NP; i = i + 1 do
    Randomly pick r0, r1, r2 ∈ {1, 2, ..., NP}, r0 ≠ r1 ≠ r2 ≠ i)
    Randomly pick j_rand ∈ {1, 2, ..., D}
    for j = 1; j ≤ D; j = j + 1 do
      Perform mutation using equation 12
      Perform crossover using equation 13
    end for
  end for
  for i = 1; i ≤ NP; i = i + 1 do
    Perform selection using equation 14
  end for
  g = g + 1
end while
```

4.3 Crowding DE

Crowding based DE (Thomsen 2004) substitutes the global selection of DEGS by crowding, where each trial solution $\vec{u}_{i,g}$ competes against the closest population member measured by Euclidean distance. The main problem of the algorithm is the dual nature of parameter F, which at the same time controls the speed of the local search and the effectiveness of the global search (Rönkkönen et al. 2009). The selection is always a compromise between these two: large values mean very slow convergence speed while small values allow faster convergence but cause the algorithm to neglect global information, which is often essential for success especially in difficult problems. Additionally, the crowding procedure increases the computational complexity, for each trial generated, NP Euclidean distance calculations are required. The advantage of CRDE is the lack of additional control parameters, which are required by most other niching approaches and for which good values are typically problem dependent.

4.4 Speciation-based DE

Speciation-based DE proposed by Li (2005) classifies the population into species according to their similarity measured by Euclidean distance. Each species and its corresponding species seed (the dominant individual) form a separate subpopulation that run DEGS locally. The main drawback of SDE is the requirement of a species radius R parameter in order to define the size of a species. The performance of the algorithm is highly dependent on the value of this parameter and the value is problem dependent.

4.5 DE using local selection

The idea of local selection is simply to compare a trial against its own parent instead of a random population member ($r_0 = i$ in Eq. 12). This has the effect of decreasing the selection bias towards a single solution and allows multimodal optimization. An algorithm based on local selection, DELL, was presented in (Rönkkönen and Lampinen 2007a) and improved in (Rönkkönen et al. 2009). The idea is to clearly divide the mutation into two separate operations to get rid of the cumbersome dual nature of parameter F. Local mutation uses small F for increased convergence speed, while global mutation uses an unscaled differential ($F = 1$) to achieve a global scale. For the local mutation $F = 1.3/\sqrt{D}$ is used to decrease the number of parameters. The selection is based on results in (Price and Rönkkönen 2006), which suggest it as an optimal value for convex unimodal problems. A method for adding gaussian noise to the global mutation to create an increased pool of potential trials was developed, but it is not used in this paper. Both mutations, local and global, are used in parallel and parameter PX $\in$ [0, 1] controls the probability that the global mutation is selected. The algorithm is not sensitive to the value of PX as long as the extremes are avoided.

4.6 Hybrid methods

The idea of separating local and global searches was taken further (Rönkkönen et al. 2009) by replacing the local mutation of DELL with a more efficient local search method. DELG hybridizes the local selection DE with the gradient descent algorithm. Each time the local phase is selected, the algorithm performs a single line search in the direction of the approximated gradient, not a complete local search. Thus, the global and local search advance in parallel, their relative emphasis decided by the PX parameter.

Finally the DECG algorithm shares the basic structure of DELG, but uses crowding instead of local selection, similarly as CRDE. The idea is to achieve a fair comparison between crowding and local selection without the convergence speed issues of CRDE.

5 Test setup

The purpose of the test setup is to demonstrate the usability of the proposed functions by studying the effects of changing the degree of regularity, dimensionality and number of local optima on the performance of different algorithms. Configuration files and example figures for each function along with a comprehensive set of plotted result curves are available at: http://www.it.lut.fi/ip/evo. In all cases, the constraint handling is done using mirroring (the value is mirrored from the violated boundary back inside the boundaries) provided by the module. To make the functions nonseparable, and thus eliminate the need to consider values smaller than CR = 1 for DE- based methods, all functions from cosine and common families are randomly rotated. Bezier curves with random control points of a tenth degree are used in all stretched cases. For all tests, 100 independent runs are performed such that each time a different random seed from 0, 1, ..., 99 is used to initialize the module. Thus each run is done with a different function. The same set is always used for each algorithm. This is done to minimize the ability to exploit a specific feature of a single function instance and to concentrate on the behavior of the algorithms with certain types of functions. To determine the performance of an algorithm, we mainly consider the average peak ratios (Thomsen 2004):

$$\text{peak ratio} = \frac{\text{number of optima found}}{\text{total number of optima}} \tag{15}$$

for global optima, with accuracy 10^{-4} (difference in fitness value) at a maximum of D*250, 000 function evaluations.

5.1 The functions

The test setup consists of four regular, five partially regular and seven irregular functions. Shubert and three functions from the cosine family are regular. To obtain partially regular functions, all of the regular cases are stretched. Additionally the Vincent function is included. In the cosine functions, the basic shape of the global optima is a 5 by 5 square ($\vec{G} = [5, 5]$) and $\alpha = 0.8$ is used. Among the resulting 25 global optima, 16 are located at the constraints and 9 are inside the search space. The functions are differentiated by the number of local optima, for which values $\vec{L} = [1, 1], \vec{L} = [10, 10]$ and $\vec{L} = [30, 30]$ are used, producing 0, 1,656 and 14,616 local optima.

The irregular functions are generated using the quadratic family. For all cases, rotated ellipsoidal shapes are used for the optima such that the values used to generate C are randomly generated in the range [0.003, 0.03], the optimum value of local optima in the range [−0.95, −0.15] (global optima have value −1) and the minimum possible Euclidean distance between two global optima points is set to 0.01. Versions with 0 and 100 local optima are generated for two, five and ten dimensions, each having ten global optima. In addition, a two dimensional function with 1,000 local optima is included. Table 1 summarizes the test function setup.

5.2 Parameter setup

For all DE based methods, crossover is always disabled by using CR = 1 and best performing population size among

Table 1 Function test set

Function	GO	LO	D	Family	Regularity
$f_{\text{cL1}}(\vec{x})$	25	0	2	Cosine	Regular
$f_{\text{cL10}}(\vec{x})$	25	1,656	2	Cosine	Regular
$f_{\text{cL30}}(\vec{x})$	25	14,616	2	Cosine	Regular
$f_{\text{shu}}(\vec{x})$	18	742	2	Common	Regular
$f_{\text{ScL1}}(\vec{x})$	25	0	2	Cosine	Partial
$f_{\text{ScL10}}(\vec{x})$	25	1,656	2	Cosine	Partial
$f_{\text{ScL30}}(\vec{x})$	25	14,616	2	Cosine	Partial
$f_{\text{Sshu}}(\vec{x})$	18	742	2	Common	Partial
$f_{\text{vin}}(\vec{x})$	36	0	2	Common	Partial
$f_{q2,0}(\vec{x})$	10	0	2	Quadratic	Irregular
$f_{q2,100}(\vec{x})$	10	100	2	Quadratic	Irregular
$f_{q2,1,000}(\vec{x})$	10	1,000	2	Quadratic	Irregular
$f_{q5,0}(\vec{x})$	10	0	5	Quadratic	Irregular
$f_{q5,100}(\vec{x})$	10	100	5	Quadratic	Irregular
$f_{q10,0}(\vec{x})$	10	0	10	Quadratic	Irregular
$f_{q10,100}(\vec{x})$	10	100	10	Quadratic	Irregular

GO and LO are the number of global and local optima, respectively

NP = [50, 100, 200, 300, 500, 1, 000, 2, 000] is searched for each problem. A fixed PX = 0.5 for DELL and PX = 0.9 for DELG and DECG are used in all reported results. For DEGS, CRDE and SDE, a best performing value for mutation step length among F = [0.1, 0.5, 0.9, 1] is searched for each function due to the difficulties of finding a well performing value for all problems. Additionally, a best value for R is similarly searched for SDE among R = [0.05, 0.5, 2] for common family functions, among R = [0.005, 0.05, 0.2] for other two-dimensional functions, among R = [0.05, 0.08, 0.2] for the five-dimensional cases and among R = [0.05, 0.11, 0.2] for the ten-dimensional functions. Part of the parameter combinations were left untested for some methods and functions based on existing results and assumptions of their poor chance of improving the results.

6 Analysis and results

Figure 10 presents a classification of the algorithms in regard to their ability to identify and exploit the regularity of the problem versus their convergence speed. The ability to exploit regularity can be seen as an indicator of an algorithm's ability to exploit global information. The convergence speed illustrates the speed in finding the first optimum in problems with no local optima and can be seen as an indicator of the effectiveness of the local search. Because the parameter setups affect the performance, the classification is qualitative, and should be seen as a guideline for the properties of the algorithms in typical cases.

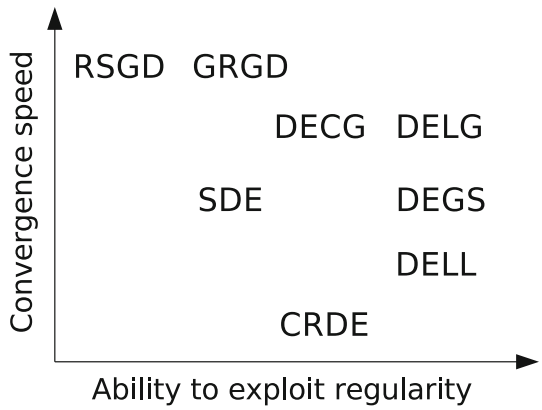


Fig. 10 Comparison of the used methods

The deterministic nature of GRGD makes it a difficult method to compare using the cosine family functions. The chosen method of generating the grid makes the algorithm inefficient in searching the constraints where the majority of the global optima are located. On the other hand, the algorithm will always instantly locate the first nine global optima of $f_{\text{cL}}1$, $f_{\text{cL}}10$ and $f_{\text{cL}}30$, because the first nine starting points will be generated directly to the optima. Generating starting points on the constraints at the beginning would allow the algorithm to instantly locate all global optima of the functions. However such a strategy would be problematic in a more general setup, because it would strongly bias the search to the constraints. 3^D points would be required to achieve an initial coverage of the constraints. While this works in low dimensional cases, in higher dimensions the algorithm would only generate points on the constraints. For this reason, the results for GRGD are more interesting for the quadratic family functions, where they demonstrate the effect of an even coverage of the search space.

As can be seen from the results in Table 2, DEGS is the top performer in all regular cases, but the performance crumbles as the degree of regularity decreases. This demonstrates well the advantages of a Framework with varied functions with tunable features as it allows the identification of features a specific algorithm excels in exploiting. Although DEGS is generally not suited to multimodal optimization, it is very efficient in special cases with regularly spaced optima.

Figure 11 presents a summary of the best performing methods with different types of two-dimensional functions. As expected, the methods able to efficiently exploit regularity perform well in the regular functions, but DELG gains the upper hand as the degree of regularity decreases. DECG becomes able to rival DELG in irregular functions as the number of local optima increases. The multistart methods are able to outperform DELG in functions with no

Table 2 Results

Function	Performance	DELG	DECG	CRDE	DELL	SDE	DEGS	RSGD	GRGD
f_{cL1}	D(IGR)LSEC	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	std	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
f_{cL10}	(DGL)(ECS)RI	**1.000**	**0.998**	**0.998**	**1.000**	0.997	**1.000**	0.960	*0.914*
	std	0.000	0.008	0.010	0.000	0.010	0.000	0.041	0.049
f_{cL30}	(DGL)(EC)SRI	**1.000**	**0.998**	**0.998**	**1.000**	0.990	**1.000**	*0.378*	*0.360*
	std	0.000	0.009	0.009	0.000	0.021	0.000	0.100	0.000
f_{shu}	D(LG)E(SC)IR	1.000	1.000	1.000	1.000	1.000	1.000	*0.994*	1.000
	std	0.000	0.000	0.000	0.000	0.000	0.000	0.017	0.000
$f_{\{ScL}1$	(RI)GS(LCE)D	**1.000**	0.996	**0.999**	**0.998**	**0.998**	*0.854*	**1.000**	**1.000**
	std	0.000	0.015	0.006	0.011	0.011	0.154	0.000	0.000
$f_{\{ScL}10$	GS(LE)RCID	**0.999**	0.956	0.809	0.964	0.985	*0.714*	0.858	*0.725*
	std	0.007	0.042	0.078	0.069	0.028	0.248	0.089	0.143
$f_{\{ScL}30$	GLECSDRI	**0.953**	0.815	0.684	0.871	0.654	0.616	0.291	*0.188*
	std	0.063	0.112	0.120	0.156	0.137	0.254	0.098	0.099
$f_{\{Sshu}$	G(LS)CEI(RD)	**1.000**	0.910	0.927	**1.000**	**0.999**	*0.874*	*0.882*	0.896
	std	0.000	0.100	0.153	0.000	0.008	0.151	0.115	0.122
$f_{\{vin}$	RGICLSED	0.959	0.696	0.901	0.804	0.770	*0.359*	**0.970**	0.956
	std	0.032	0.047	0.035	0.048	0.049	0.044	0.023	0.025
$f_{q2,0}$	(IR)G(SE)LCD	**1.000**	**0.997**	0.988	**0.997**	**1.000**	*0.574*	**1.000**	**1.000**
	std	0.000	0.017	0.036	0.017	0.000	0.147	0.000	0.000
$f_{q2,100}$	(GRIE)(LS)CD	**1.000**	0.992	0.944	0.987	0.985	*0.613*	**0.999**	**0.998**
	std	0.000	0.027	0.066	0.039	0.041	0.128	0.010	0.014
$f_{q2,1,000}$	(GE)(LI)RSCD	**0.990**	**0.985**	0.716	**0.983**	0.912	*0.635*	0.948	0.973
	std	0.030	0.036	0.268	0.038	0.091	0.144	0.069	0.053
$f_{q5,0}$	RIE(GSC)LD	0.957	**0.989**	0.954	0.777	0.958	*0.216*	**0.996**	**0.992**
	std	0.062	0.035	0.069	0.136	0.065	0.091	0.020	0.031
$f_{q5,100}$	RIEGLSCD	0.701	0.769	0.211	0.502	0.313	*0.120*	**0.905**	0.853
	std	0.145	0.138	0.129	0.161	0.135	0.085	0.104	0.116
$f_{q10,0}$	REIGCSLD	0.893	0.957	0.728	0.411	0.619	*0.117*	**0.981**	0.932
	std	0.101	0.061	0.222	0.175	0.106	0.038	0.042	0.075
$f_{q10,100}$	R(IE)GLCSD	0.550	0.643	0.205	0.252	0.117	*0.033*	**0.800**	0.667
	std	0.142	0.152	0.151	0.148	0.089	0.051	0.135	0.151

The method columns list the best achieved average peak ratios and the standard deviations (std) for each method. The best result for each function is marked in bold font as well as any other results which do not have a statistically significant difference to the best one with a significance level 0.05. Similarly the worst results are marked in italic font. The significance is determined using the two-tailed Wilcoxon signed-rank test (Wilcoxon 1945). The performance column lists the tested methods from best to worst performer in each problem such that C is CRDE, G means DELG, E stands for DECG, R is RSGD, D means DEGS, S stands for SDE, L is DELL and I is GRGD. In cases with no statistically significant difference in the peak ratios, the speed measured as the number of function evaluations is used to determine the performance order. The brackets are used to group methods when their relative performance order can not be confidently defined

local optima, but can not keep up with the other methods as the number of local optima increases.

Figure 12 displays a typical example of the convergence speeds of the different methods with regular functions which contain local optima. DEGS is the fastest, followed closely by DELG and DELL. Crowding somewhat slows down DECG in locating the last few optima. CRDE takes a lot of time to find the first optima due to the inefficient local search, but locates the rest quite fast by exploiting the regularity. The SDE and multistart methods, which primarily rely on local search, find the first optima faster but are slow to locate the subsequent optima due to their inability to exploit the regularity.

Figure 13 presents a summary of the best performing methods with the different irregular functions. While DELG and DECG demonstrate the top performance in the two-dimensional functions, RSGD is the top performer in all five- and ten-dimensional cases. Looking at the results in Table 2 also show DECG to outperform DELG. It seems that while the improved ability of crowding to keep

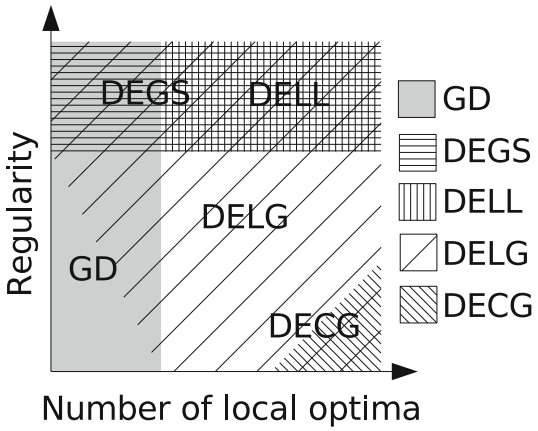


Fig. 11 Best performing methods with different two-dimensional functions

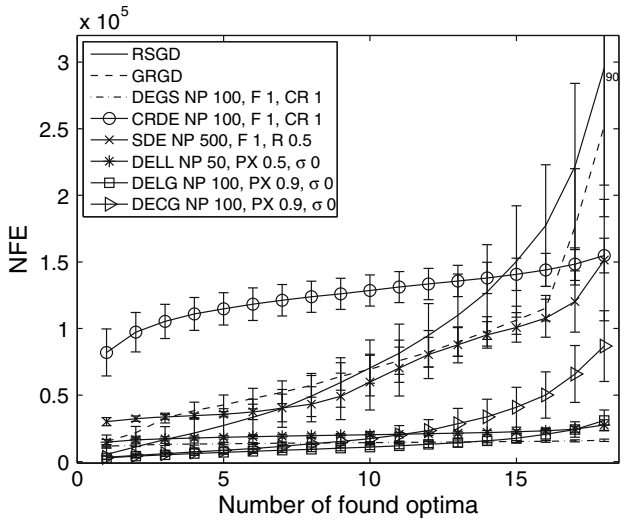


Fig. 12 Average number of function evaluations required to find the i:th optimum, including standard deviations for the f_{shu} function. RSGD was able to locate the 18th optimum in only 90% of the runs

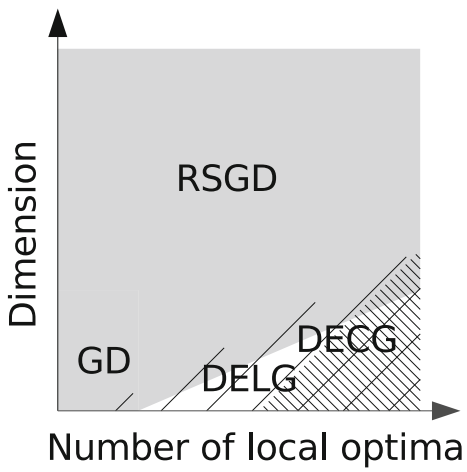


Fig. 13 Best performing methods with different irregular functions

population diversity disrupts the performance in low dimensional cases, it is able to offer an advantage as the dimensionality increases.

Comparing the performance of the multistart and population based methods is especially interesting. Lobo and Lima (2006) claim that a simple hill climber (like RSGD or GRGD) will very likely outperform EAs in problems with no global structure which the EA could exploit. Although their study only assumes a goal of finding a single optimum, the results in this study suggest that the same generally holds for multimodal optimization and includes the hybrids. The quadratic family does not offer a meaningful global structure nor does it offer other easily exploitable features. Furthermore as different random seeds are used, each quadratic function test set actually contains 100 different functions. This makes it hard to exploit any global information and the even coverage provided by GRGD offers a good comparison point for the other algorithms. However, the grid approach is not able to offer an advantage over randomly generated starting points. As can be seen by comparing the results in quadratic functions on Table 2, RSGD demonstrates superior performance over GRGD especially with the higher dimensional quadratic functions.

The DE part in the hybrids becomes an overhead in irregular functions, as it is no longer able to offer significant advantage to the search through exploiting the global information. Two-dimensional functions containing local optima, however, are an exception for this. As the two-dimensional search space is still rather compact, the niching DE part is able to keep the population well spread, preventing the hybrids from searching the same areas repeatedly. Similarly, GRGD is able to outperform RSGD in $f_{q2,1,000}$ through minimizing the redundant searches. As the volume of the search space increases along with the dimensionality, it becomes increasingly difficult to cover the search space with enough detail, and the overhead outweighs the gained benefits, giving an advantage to RSGD over the hybrids and GRGD. The pure DE- based methods are clearly the worst performers, as the improved global search capability simply can not compensate the loss of convergence speed when the global information is not easily exploitable.

7 Conclusions and future work

In this paper, a software framework for generating multimodal problems has been presented. The framework provides an easy way to construct parameterizable functions and offers an environment for testing multimodal optimization algorithms. To the authors' knowledge, no versatile environment designed especially for producing problems

for multimodal optimization exists and the proposed framework aims to remedy this situation. In addition to offering three families of parameterizable functions, the framework implements several well known test problems and provides an easy option to modify them by rotating and stretching.

The framework allows easy addition of new families. One such future family could be a variant of the current cosine family where the global optima would not be located on the constraints. This would allow easy comparison of algorithms in regard to their ability to locate optima on constraints and offer a different challenge for methods that rely on the constraints and initial population. Another area in which the framework should be developed is the ability to generate functions with desirable global structure. For the quadratic family we are planning to achieve this by adding an option giving the optima locations as parameters, instead of always generating them randomly. A new family based on the composition idea discussed in Sect. 3 could also be added to allow the generation of varying landscapes having a large number of optima without excessive computational cost. Additionally, more known test functions could be added to the common family and some of the existing functions currently limited to two-dimensions could be generalized to allow higher dimensional versions.

The usability of the proposed framework was demonstrated by generating a set of test functions for comparing eight optimization approaches in locating multiple optima. The results demonstrate the importance of understanding and being able to change the test function features in order to explain the performance of an algorithm. Additionally, the results show the advantage of separating the global and local search operations through hybridization. This is especially important in multimodal optimization, where the population must stay diverse throughout the optimization. The DE hybrids are able to offer a potential increase in performance through identifying and exploiting regularities the user is not aware of beforehand. The global search phase, however, is only able to offer an advantage in functions, which contain exploitable features. If the regularities are not available, the hybrids still offer superior performance compared to pure DE methods. Finally, the results suggest crowding to offer an advantage over local selection in irregular functions when the dimensionality of a function increases.

In future research, more comprehensive testing is required to fully understand the features of the approaches. Especially the effect of increasing dimensionality further should be studied. It seems that efficient global and local parts are both important for the success of a multimodal optimization algorithm. Nevertheless, a wide variety of different approaches for implementing the global and local part should be studied to determine the best combinations for different types of problems.

Acknowledgments The authors want to thank Harri Lattu and Jarmo Ilonen for their help with implementing the software, Jouni Sampo for mathematical consultation, and the anonymous reviewers for their useful comments.

References

Beasley D, Bull D, Martin R (1993a) An overview of genetic algorithms: part 2, research topics. Univ Comput 15:170–181

Beasley D, Bull D, Martin R (1993b) A sequential niche technique for multimodal function optimization. Evol Comput 1(2):101–125

Bezier P (1968) How renault uses numerical control for car body design and tooling. In: SAE Paper 680010, Society of Automotive Engineers Congress, Detroit, MI, USA.

Bezier P (1986) The mathematical basis of UNISURF CAD System. Butterworth-Heinemann, London. ISBN 978-0408221757

Branke J (2002) Evolutionary optimization in dynamic environments. Kluwer, Norwell

Crutchley DA, Zwolinski M (2002) Using evolutionary and hybrid algorithms for DC operating point analysis of nonlinear circuits. In: Proceedings of 2002 IEEE world congress on computational intelligence, pp 753–758, Honolulu, USA, 12–15 May 2002. ISBN 0-7803-7282-4

De Jong K (1975) An analysis of the behavior of a class of genetic adaptive systems. PhD thesis, University of Michigan

Dong Z, Lu M, Lu Z, Wong K (2006) A differential evolution based method for power system planning. In: Proceedings of 2006 IEEE world congress on computational intelligence, pp 2699–2706, Vancouver, Canada, 16–21 July 2006. ISBN 0-7803-9489-5

Gallagher M, Yuan B (2006) A general-purpose tunable landscape generator. IEEE Trans Evol Comput 10:590–603

Gaviano M, Kvasov D, Lera D, Sergeyev Y (2003) Algorithm 829: Software for generation of classes of test functions with known local and global minima for global optimization. ACM Trans Math Softw 29(4):469–480

Ghosh A, Tsutsui S, Tanaka H, Corne D (2000) Genetic algorithms with substitution and re-entry of individuals. Int J Knowl Based Intell Eng Syst 4(1):64–71

Goldberg D, Richardson J (1987) Genetic algorithms with sharing for multimodal function optimization. In: Grefenstette J (ed) Proceedins of the second international conference on genetic algorithms, pp 41–49

Hansen N, Ostermeier A (2001) Completely derandomized self adaptation in evolution strategies. Evol Comput 9(2):159–195

Harik G (1995) Finding multimodal solutions using restricted tournament selection. In: Eshelman L (eds) Proceedings of the sixth international conference on genetic algorithms. Morgan Kaufmann, San Francisco, pp 24–31

Li J, Balazs M, Parks G, Clarkson P (2002) A species conserving genetic algorithm for multimodal function optimization. Evol Comput 10(3):207–234 (ISSN 1063-6560)

Li X (2005) Efficient differential evolution using speciation for multimodal function optimization. In: Proceedings of the conference on genetic and evolutionary computation (GECCO 2005). Washington DC, USA, pp 873–880

Liang J, Suganthan P, Deb K (2005) Novel composition test functions for numerical global optimization. In: Proceedings of the 2005 IEEE congress on evolutionary computation, pp 68–75

Lobo F, Lima C (2006) On the utility of the multimodal problem generator for assessing the performance of evolutionary algorithms. In: Proceedings of the ACM genetic and evolutionary computation conference (GECCO 2006), ACM Press

Macnish C (2007) Towards unbiased benchmarking of evolutionary and hybrid algorithms for real-valued optimisation. Connect Sci 19(4):361–385

Mahfoud S (1994) Genetic drift in sharing methods. In: Profeedings of the First IEEE conference on evolutionary computation, pp 67–72

Mahfoud S (1995a) A comparison of parallel and sequential niching methods. In: Proceedings of 6th international conference on genetic algorithms, pp 136–143

Mahfoud S (1995b) Niching methods for genetic algorithms. PhD thesis, University of Illinois, Urbana, IL, USA

Michalewicz Z (1996) Genetic algorithms + data structures = evolution programs. Springer, Berlin

Michalewicz Z, Deb K, Schmidt M, Stidsen T (2000) Test-case generator for nonlinear continuous parameter optimization techniques. IEEE Trans Evol Comput 4:197–215

Morrison R (2004) Designing evolutionary algorithms for dynamic environments. Springer, Berlin

Morrison R, De Jong K (1999) A test problem generator for nonstationary evironments. In: Proceedings of the congress of evolutionary computation. IEEE Press, Piscataway, pp 1843–1850

Pétrowski A (1996) A clearing procedure as a niching method for genetic algorithms. In: Proceedings of the 3rd IEEE international conference on evolutionary computation, pp 798–803

Press W, Flannery B, Teukolsky S, Vetterling W (1992) Numerical recipes in C, 2nd edn. Cambridge University Press, Cambridge. ISBN 0-521-43108-5

Price K, Rönkkönen J (2006) Comparing the uni-modal scaling performance of global and local selection in mutation-only differential evolution algorithm. In: Proceedings of 2006 IEEE world congress on computational intelligence, Vancouver, Canada, pp 7387–7394, 16–21 July 2006. ISBN 0-7803-9489-5

Price K, Storn R, Lampinen J (2005) Differential evolution: a practical approach to global optimization. Springer, Berlin. ISBN 3-540-20950-6

Rönkkönen J, Lampinen J (2007a) An extended mutation concept for the local selection based differential evolution algorithm. In: Proceedings of genetic and evolutionary computation conference (GECCO 2007), London, England, pp 689–696. ISBN 978-1-59593-697-4

Rönkkönen J, Lampinen J (2007b) On determining multiple global optima by differential evolution. In: Evolutionary and deterministic methods for design, optimization and control, proceedings of eurogen 2007. Jyväskylä, Finland, pp 146–151. ISBN 978-84-96736-45-0

Rönkkönen J, Li X, Kyrki V (2009) The role of local and global search in solving problems with multiple global optima. Technical Report 110, Department of Information Technology, Lappeenranta University of Technology. ISBN 978-952-214-730-1

Salomon R (1996) Reevaluating genetic algorithms performance under coordinate rotation of benchmark functions. Biosystems 39:263–278

Shir O, Bäck T (2006) Niche radius adaptation in the cma-es niching algorithm. In: Parallel problem solving from nature (PPSN IX). Springer, Berlin, pp 142–151. ISBN 978-3-540-38990-3

Singh G, Deb K (2006) Comparison of multi-modal optimization algorithms based on evolutionary algorithms. In: Proceedings of the genetic and evolutionary computation conference. ACM Press, Seattle, WA, pp 1305–1312

Storn R, Price K (1995) Differential evolution—a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, International Computer Science Institute (ICSI)

Storn R, Price K (1997) Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. J Global Optim 11:341–359

Thomsen R (2004) Multimodal optimization using crowding-based differential evolution. In: Proceedings of the 2004 congress on evolutionary computation, vol 2. Portland, pp 1382–1389

Törn A, Zilinskas A (1989) Global optimization, Lecture Notes in Computer Science. Springer, Berlin. ISBN 9783540508717

Ursem R (1999) Multinational evolutionary algorithms. In: Proceedings of congress of evolutionary computation (CEC 1999), vol 3. IEEE Press

Weise T, Niemczyk S, Skubch H, Reichle R, Geihs K (2008) A tunable model for multi-objective, epistatic, rugged, and neutral fitness landscapes. In: Proceedings of the 10th annual conference on genetic and evolutionary computation. Atlanta, GA, USA, pp 795–802

Whitley D, Lunacek M, Sokolov A (2006) Comparing the niches of cma-es, chc and pattern search using diverse benchmarks. In: Parallel problem solving from nature (PPSN IX). Springer, pp 988–997. ISBN 978-3-540-38990-3

Whitley D, Mathias K, Rana S, Dzubera J (1996) Evaluating evolutionary algorithms. Artif Intell 85:245–276

Wilcoxon F (1945) Individual comparisons by ranking methods. Biometrics 1:80–83

Wolpert D, Macready W (1995) No free lunch theorems for search. Technical report SFI-TR-95-02-010, The Santa Fe Institute

Wolpert D, Macready W (1997) No free lunch theorems for optimization. IEEE Trans Evol Comput 1(1):67–82