A General-Purpose Tunable Landscape Generator

Marcus Gallagher, *Member, IEEE*, and Bo Yuan, *Student Member, IEEE*

***Abstract*—The research literature on metaheuristic and evolutionary computation has proposed a large number of algorithms for the solution of challenging real-world optimization problems. It is often not possible to study theoretically the performance of these algorithms unless significant assumptions are made on either the algorithm itself or the problems to which it is applied, or both. As a consequence, metaheuristics are typically evaluated empirically using a set of test problems. Unfortunately, relatively little attention has been given to the development of methodologies and tools for the large-scale empirical evaluation and/or comparison of metaheuristics.**

In this paper, we propose a landscape (test-problem) generator that can be used to generate optimization problem instances for continuous, bound-constrained optimization problems. The landscape generator is parameterized by a small number of parameters, and the values of these parameters have a direct and intuitive interpretation in terms of the geometric features of the landscapes that they produce. An experimental space is defined over algorithms and problems, via a tuple of parameters for any specified algorithm and problem class (here determined by the landscape generator). An experiment is then clearly specified as a point in this space, in a way that is analogous to other areas of experimental algorithmics, and more generally in experimental design. Experimental results are presented, demonstrating the use of the landscape generator. In particular, we analyze some simple, continuous estimation of distribution algorithms, and gain new insights into the behavior of these algorithms using the landscape generator.

***Index Terms*—Continuous optimization, empirical algorithm analysis, estimation of distribution algorithm, test-problem generator.**

I. Introduction

ONE of the primary goals in the fields of metaheuristic algorithms (MHAs) and evolutionary computation is the development of powerful and practical problem-solving algorithms. Nature-inspired and other heuristic and modeling approaches to problem solving often result in algorithms that are relatively simple from an implementation point of view, as well as being applicable to a wide range of problems. As a consequence, hundreds of individual algorithms have been proposed in the literature. Even if we restrict the discussion to continuous, simple bound-constrained optimization problems, many algorithms (and variants of algorithms) exist and continue to appear regularly in conference proceedings and journal issues. To illustrate this point and give some examples of well-known algorithms in this class, we mention evolution strategies, evolutionary programming, real-coded genetic algorithms, continuous estimation of distribution algorithms (EDAs), particle swarm optimizers, differential evolution, continuous Tabu search and continuous memetic algorithms. In continuous EDAs alone (a relatively new class of methods), at least 15 algorithmic variants have been proposed.

While theoretical research in the analysis of MHAs is ongoing (see, e.g., [18] and [54]), it is typically not possible to study theoretically the performance of such an algorithm unless significant assumptions are made on either the algorithm itself, the problems to which it is applied, or both. In addition, researchers are often interested in the successful application of their algorithms when applied to real-world problem instances. For these reasons, newly developed algorithms are primarily evaluated empirically, using either artificial benchmark test problems or problems derived from real-world domains.

In comparison to the amount of research on algorithm development in MHAs, only a small amount of work has been focused on developing good methodology and practice in experimental analysis of such algorithms [3]. It has, however, been acknowledged that the standard of experimental research methodology has much room for improvement [12], [49].

In this paper, we aim to contribute to the improvement of experimental research in MHAs. The main contribution is to propose a test landscape generator (i.e., a randomized generator of instances of optimization problems). The landscape generator is based on a problem class that is defined by a parameterized generative model, based on a mixture of Gaussians (see Section III-C). We argue that this is useful for conducting experiments that provide more insight into the behavior and performance of MHAs. This is because experiments can be conducted in specific directions in the landscape/problem space, to allow for the exploration of trends in how performance changes when we move along these directions. Importantly, these directions correspond to geometrical landscape features that are appropriate for thinking about optimization, as well as already being commonly used in the literature to describe problems. This leads to the possibility of making more generalizable conclusions from the results, in terms of these landscape features.

The outline for this paper is as follows. In the following section, we provide a review of experimental algorithmics and experimental methodology and practice in the MHA and evolutionary computation literature. Section III introduces notation and terminology for describing experiments with MHAs, and discusses the idea of an "experimental space" for the description of experiments or sets of experiments over MHAs for continuous, bound-constrained optimization problems. The landscape generator proposed in this paper, based on a mixture of Gaussian functions, is described in Section IV. Experimental results are presented in Section V to illustrate the usage of the landscape generator. In particular, the landscape generator is used to gain new insight into the behavior of some simple continuous EDAs. Section VI provides a conclusion and suggestions for future work.

Manuscript received January 12, 2005; revised July 11, 2005. This work of B.Yuan was supported by an Australian Postgraduate Award.

The authors are with the School of Information Technology and Electrical Engineering, University of Queensland, Brisbane 4072, Australia (e-mail: marcusg@itee.uq.edu.au; boyuan@itee.uq.edu.au).

Digital Object Identifier 10.1109/TEVC.2005.863628

II. Current Practice in Experimental MHA Research

Experimental work in the analysis of algorithms is important to many areas of computer science [9], [25], [31]. Apart from the obvious practical reason (algorithms are engineered to solve real-world problems), the evaluation and development of algorithms is driven as much by insight gained from experimentation as it is by theoretical analysis. This is especially true in heuristic and metaheuristic algorithms, where theoretical analysis has proven difficult. Several researchers have noted that the standard of experimental work in publications on algorithmic research in general is a serious concern [13], [21], [25], and the development of the experimental methodologies is an ongoing and active area of research.

Previous research has discussed principles for conducting empirical research on algorithms. The paper by McGeoch [31] (with discussion) surveys a significant amount of literature and the principles discussed are applicable to many kinds of algorithms. In [32], McGeoch provides a recent survey of literature on experimental research practice that specifically addresses optimization algorithms. Rardin and Uzsoy [40] discuss experimental issues of specific relevance for heuristic and metaheuristic optimization algorithms. While their paper is focused on combinatorial and discrete problems, many of the ideas can be applied equally to continuous optimization. In the following, we adopt terminology from [31] and [40] regarding the description of experiments with algorithms.

Experimental results have always played a major role in the development of MHAs, partly because, by their nature, MHAs are typically easy to specify and implement, but notoriously difficult to analyze theoretically. While experimental results are easily generated, some authors have noted that the methodologies applied in the experimental evaluation and comparison of MHAs often have serious shortcomings. Whitley *et al.* [49] discuss problems with commonly used test functions and suggest several guidelines for the design of test suites for evolutionary algorithms (see Section III-C). Eiben and Jelasity [12] discuss the generalizability, reproducibility, and presentation of experimental results, as well as performance measures and defining experimental objectives.

The fields of evolutionary computation and metaheuristics continue to produce an extraordinary number of algorithms. Typically, an algorithm is introduced in a paper, with the main mechanisms (e.g., operators) and ideas discussed. The algorithmic parameters are also described, where sometimes the role of a parameter is intuitive (e.g., a learning rate or population size), while in other cases its meaning is less obvious. The range of feasible values for each parameter is also given, although this may in some cases be unbounded (e.g., population size).

To implement a newly developed algorithm class, appropriate values for the parameters need to be determined. It is normally not obvious a priori how to choose such values in order to obtain reasonable performance. The behavior of an algorithm is dependent on the combined values of all of its parameters, implying that the parameters values are not independent and that selecting their values is likely to be a nontrivial task. Typically, researchers mention that preliminary trials (not reported in the paper) are conducted on an ad hoc basis to determine a set of parameter values that seem to work reasonably well on some of the problems in the experimental suite. This test suite usually consists of a small number of specified problems (often less than ten or even five). The resulting experiments consist of a single set of parameter values for the algorithm, each of which will be run on each problem for some number of trials (often 10 or 20). An alternative scenario is when some attempt is made to examine the performance of the algorithm class as its parameter values are varied. Conducting experimental studies over more than a few algorithm parameters and/or problems typically leads to a very large number of experiments to cover all combinations (see Section III-D). Consequently, researchers may select a small number of experiments and conduct these in isolation, or vary the value of a single parameter at a time while fixing all other parameters.

Despite their shortcomings, test functions such as the DeJong suite (see, e.g., [49]) are still commonly used in the literature when evaluating new algorithms. This may be due in part to the fact that they are well known, easy to describe, and easy to implement. Alternatively, experimental studies may be primarily concerned with the solution of a specific real-world problem (or class of problems). In this case, it can be difficult to generalize about the performance of the algorithm(s) used, since they have likely been highly tuned to solving the particular problem(s) in question.

There are numerous possible performance measures for assessing the experimental performance of MHAs, most of which are equally applicable to other types of optimization or mathematical programming algorithms [23], [24]. Common approaches in experimental results on MHAs include the following [12].

- Running all trials for a prespecified number of generations (or objective function evaluations) and reporting on the distribution of fitness function values obtained (e.g., the average of the best fitness value found over a set of random restarts of a specified algorithm).
- Setting a termination criterion based on a desired level of fitness to be obtained and reporting on the distribution of the number of generations required to meet the criterion. A maximum number of generations is also usually specified, so that all experiments will be terminated after a certain amount of time.
- With reference to the previous point, the success of a trial can be defined in terms of whether the termination criteria was met (e.g., desired fitness value obtained) or not (maximum number of generations exceeded).
- For test problems (typically artificial) where the location and/or the objective function value of the global optimum $\mathbf{x}^*$ are known, it is possible to measure performance using a distance metric on the best search point found so far to the global optimum (e.g., $|\mathbf{x} - \mathbf{x}^*|$) or on the objective function values (e.g., $|f(\mathbf{x}) - f(\mathbf{x}^*)|$). This clearly permits a more informative assessment of the performance of an algorithm, where available.

These are measures involving what have been termed algorithm efficiency [23], [24] or accuracy. Other possibilities include reliability (i.e., how successful is an algorithm at solving a wide class or set of problems) or ease of use [23].

It is very important that the performance measurement used in a set of experiments be well matched to the aims of the experiments. For example, researchers are often more interested in the ability of an MHA to find the "basin of attraction" of the global optimum than the final rate of convergence to the exact optimal point. Renders and Flasse refer to this as "reliability" [42]. For simplicity, in this paper, we assume that performance is measured by a scalar, real-valued number.

III. Computational Experiments, EAs, and MHAs

In this section, we view the conducting of experiments involving MHAs as the exploration and evaluation of points in a space over algorithm instantiations and optimization problems, both of which we assume to be parameterized. While this is a common scenario in the evaluation of MHAs, this discussion does not cover all possible scenarios. Notably, the algorithmic parameters in question are assumed to be fixed constant values.

A. Definitions and Notation

We consider the following multivariate continuous optimization problem:

$$\max f(\mathbf{x}), f : \mathbb{R}^n \rightarrow \mathbb{R},\ \mathbf{x} \in \mathcal{S} \subseteq \mathbb{R}^n$$

where $\mathcal{S}$ defines our search space as a feasible region of $\mathbb{R}^n$. A (candidate) solution to the problem is given as an n-dimensional vector $\mathbf{x} = \{x_1, \ldots, x_n\}$. It is assumed that $\mathcal{S}$ is given as a simple symmetric boundary constraint such that for every component of a solution vector

$$|x_i| \leq c \,\forall\, \mathbf{x} \in \mathcal{S},\ 1 \leq i \leq n.$$

We are interested in algorithms that can be applied to solving an optimization problem of this form. Note, however, that we do not place any assumptions on the objective (or fitness) function f other than the ability to evaluate it at any point within $\mathcal{S}$ (e.g., f need not be differentiable or continuous). Furthermore, the problems under consideration may be multimodal and high-dimensional (large n). MHAs and evolutionary algorithms are often applied to such problems. Evolution strategies [45] and evolutionary programming [15] are traditionally formulated for continuous optimization; however, numerous real-valued genetic algorithms have also been developed [19] along with many other approaches such as particle swarm optimization [28], differential evolution [46], and continuous Tabu search [8].

B. Experiments on Parameterized Algorithms

When conducting computational experiments with algorithms, it is useful to consider the adjustable parameters of the algorithm as experimental "factors." The values of some or all of these factors can be varied to define a set of experiments (see, e.g., [2]). MHAs inevitably include one or more adjustable parameters that must be specified by the user before an algorithm can be applied to any problem. In this paper, we will make a distinction between an *algorithm class* $\mathcal{A}$ and an *algorithm specification*. A specified algorithm A_i can be expressed as a tuple

$$\langle A_i, p_1, p_2, \ldots \rangle$$

where $A_i \in \mathcal{A}$. Each different specification of the algorithm class will have one or more different values for each of its parameters p_j. For example, the simplest possible form of an evolution strategy (ES) consists only of a mutation operator, where each component or variable of an individual x_i is mutated by the addition of a single sample drawn from a Gaussian distribution $\mathcal{N}(0, \sigma)$ (see, e.g., [45]). To specify an algorithm of this class (i.e., a (μ, λ)-ES), settings are required for the size of the parent (μ) and offspring (λ) populations, as well as the value of the standard deviation of the mutation distribution σ. Thus, a specified algorithm includes all the parameterized information needed to run the metaheuristic on a given optimization problem.[1] Different algorithm classes will, of course, have different numbers of parameters that need to be specified. As mentioned earlier, in this context the literature has proposed a large number (at least hundreds) of algorithm classes.

Assume that a single optimization problem is to be solved. A performance measurement of an experimental trial is typically a scalar value such as the best objective function value found by an algorithm. The parameters of a specified algorithm are numerical, typically continuous or integer-valued, and may have defined bounds on the range of sensible values that they can take. We can imagine a configuration space of feasible algorithm parameter values, or a landscape over this space, with the height of the landscape at any point determined by the performance value obtained at that point. The performance of an algorithm A_i on a problem fundamentally depends on the values of its parameters. Researchers and practitioners are therefore interested in how the value of performance changes along a particular direction in this space (e.g., increasing the size of the population). Alternatively, we might hope that the performance of an algorithm is not highly sensitive to the parameter values (e.g., if the "recommended" parameter settings are perturbed slightly, this should not cause a large change in the algorithm performance).

For an experiment where multiple trials are conducted (e.g., random restarts), the picture is more complex, since these multiple trials define a random variable over performance at each point in the experimental space, rather than a single value. The empirical distribution of this random variable could be summarized most simply by a mean value, or if more information about the distribution of the variable is desirable, by error bars, box-whiskers plot, or a histogram.

Thinking about computational experiments with MHAs in the context of algorithm specifications is a simple but useful means of ensuring that the details of experiments are clearly and systematically stated.

C. Experiments on Parameterized Problems

Problem factors can also be considered in the design of experiments on algorithms. Consider a similar notation to the above. In general, the functional form of f is unknown. If we assume that problems can be distinguished from each other (so that we can label them), then we can form tuples

$$\langle f_i, c, n \rangle$$

[1]Other assumptions have to be made, notably how to initialize the population and how to handle the boundary constraint on the feasible region. These are typically not parameterized.

based on the problem, its feasible region, and the dimensionality of the solution space. Following the above terminology, let this refer to a *problem specification* belonging to a problem class $\mathcal{F}$. The most general example of a problem class in a real-valued space corresponds to the set of all possible mappings $f : \mathbb{R}^n \rightarrow \mathbb{R}$. Under the practical constraint that any computer implementation of continuous optimization is ultimately represented discretely (due to finite precision), the "no free lunch" theorems were originally formulated across this space of all possible functions [50] (but see also [22]). Other problem spaces can easily be conceived by imposing a bound on the maximum value of $|f|$ or imposing a smoothness or neighborhood constraint on f (see, e.g., [22]).

Given a specified algorithm A_i and problem f_j, we will refer to the execution of this algorithm on this problem as an *experiment*. A specified algorithm is usually subject to some initialization (often random) and the algorithm itself typically contains stochastic components. This means that the measured outcome of an experiment is often a random variable. Hence, a series of *trials* are often conducted for an experiment, where the initialization and stochastic nature of the algorithm are the only factors that contribute to make the outcome of one trial different from any other trial.

We describe experiments over a set of test problems (see Section III-A) in a similar way to experiments over algorithm parameters. Assume that a specified algorithm A_i is given. Dimensions of the problem subspace would correspond to problem-parameters, with the value of a point given by the performance[2] obtained by executing the algorithm on that specific problem.

One approach to the experimental analysis of algorithms is to produce problem instances that are generated through a parameterization of some class of problems [2], [40].

There have been a number of attempts to develop test problem generators as a tool for the evaluation of optimization algorithms. It is possible to make a distinction between what we will refer to as *problem* and *landscape* generators. Problem generators can be used to generate a set of instances of a specific class of (usually combinatorial) optimization problem [20], [39]. The control parameters of these problem generators are usually specific to the problem in question; for example, in Boolean satisfiability the number of variables and the number of disjunctive clauses [34].

Previous research has identified a number of properties that are considered to be desirable for optimization test problems when evaluating EAs [17], [49]. Good test functions may be:

- P1) difficult to solve using simple methods such as hill-climbing algorithms;
- P2) nonlinear, nonseparable, and nonsymmetric;
- P3) scalable in terms of problem dimensionality;
- P4) scalable in terms of time to evaluate the objective function;
- P5) tunable by a small number of user parameters;
- P6) able to be generated at random and are difficult to reverse engineer;
- P7) exhibit an array of landscape-like features.

[2]Or for multiple trials, a distribution of performance values.

Landscape generators do not focus on a specific class of problem, but rather aim to produce instances that have certain structural features. Stuckman [47] produces randomized continuous problems based on combinations of "square" sinc functions. The resulting landscapes have multiple optima and are not everywhere differentiable.

The NK landscape generator [10], [26] generates random landscapes over a binary search space, where N is the length of the binary string and K is the number of correlations that each bit has to other bits in a solution. The N and K parameters provide a coarse level of control over the complexity of the landscapes generated.

Whitley *et al.* [48], [49] propose a methodology for generating landscapes based on compositions of known analytical functions (e.g., the DeJong test function suite) such that the resultant functions have some of the properties listed above. In particular, the functions are nonlinear and nonseparable. Whitley *et al.* also show how to construct functions that are invariant to a binary/gray code representation, when using a binary EA to encode a floating-point solution to a continuous optimization problem. Analytical functions have also been used as the basis for generating test problem sets for multiobjective [11] and dynamic multiobjective problems [14].

The bit-string multimodal landscape generator [29] selects a number (P) of L-bit individuals (bit-strings) as peaks in the landscape. The fitness values of these peaks are set to 1.0. The fitness of a string is then determined by its Hamming distance to the nearest peak. This is a minimization problem with the lowest possible fitness value of zero. A continuous landscape generator was proposed [27] as an extension of this bit-string landscape generator. A landscape is generated by defining P peaks in an n-dimensional space, with each variable restricted to the interval [0,1] by applying a sigmoid function to the variable values. Again, the fitness value is determined by the (this time Euclidean) distance between an individual (point) and the nearest peak.

A landscape generator for dynamic optimization problems is proposed in [35]. A landscape is created using P cones, where the ith cone is parameterized by its position, height (H_i) and slope (R_i). The fitness value of each point on the two-dimensional (2-D) landscape is determined by the parameters of the cones

$$f(x,y) = \max_{i=1}^{P} \left\{ H_i - R_i \cdot \sqrt{(x - X_i)^2 + (y - Y_i)^2} \right\}$$

where (X_i, Y_i) denotes the center of the base of the ith cone. Finally, a continuous landscape generator is given in [33]. This landscape generator is concerned specifically with constrained optimization problems and has parameters to control features such as the ratio of the feasible region to the total search space, the connectedness of the feasible region, and the number of constraints. The idea used is to divide the landscape up into a number (k) of disjoint subspaces (n-dimensional hypercubes) and to define a unimodal function f_k for each cube

$$f_k(\mathbf{x}) = a_k \left(\prod_{i=1}^{n} \left(u_i^k - x_i \right) \left(x_i - l_i^k \right) \right)^{\frac{1}{n}}.$$

In the above, u_i^k and l_i^k are, respectively, the upper and lower bounds of the ith dimension of the kth subspace and a_k is a specified positive constant. The optimum of each subspace is located in its center. The landscape generator employs a set of constraints by further dividing each subspace into feasible and infeasible regions.

In Section IV, we discuss a landscape generator that is well suited to the experimental evaluation of continuous MHAs with simple boundary constraints.

D. Experimental Design and Statistical and Large-Scale Experimental Analysis of EAs

It is important to note that randomized test problem generators represent only one possible way of producing a suite of problems for use in large-scale experimental studies and that, generally speaking, each approach has its problems and limitations [40]. We refer the reader to the papers mentioned in this section.

As mentioned above, much of the research in MHAs has focused on the development of new algorithms and the application of these algorithms to solving practical problems. Consequently, methods of experimental design have not seen much application in the evaluation of MHAs. Recently, however, a number of papers have appeared, suggesting that this situation is beginning to change.

The obvious approach is to conduct a full factorial design (each specified algorithm is executed on each specified problem). Time constraints evidently result in a small selected set of algorithm/problem instantiations, or full factorial designs over a very small number of parameters, where these parameters are varied over only a few values. One well-known example of a more large-scale experimental study is by Schaffer *et al.* [44]. With some assumptions and with specific goals in mind (e.g., to find the best performing specified algorithm from a large set), it is also possible to use statistical techniques to reduce the number of experiments required in a design [53].

Bartz-Beielstein [3] surveys techniques of experimental design and illustrates the application of a number of these techniques to the evaluation of evolution strategies. These techniques are generally applicable to a wide range of MHAs: a subsequent paper [4] provides an evaluation of particle swarm optimization and the Nelder–Mead simplex algorithm. A more directed approach is taken by François and Lavergne [16] in the use of statistical modeling techniques to determine suitable parameter values for EAs via experimental studies. Assuming a functional relationship between the algorithm parameters and performance, a model is constructed using the data resulting from experimentation. Myers and Hancock [36] take a similar approach for parameter tuning in GAs, using experimental design and logistic regression.

E. Discussion

In the following section of this paper, we propose a new parameterized landscape generator based on a sum of Gaussian functions. While the previous work discussed above has proposed a number of test landscape and problem generators with specific features, none of them appear to satisfy all of the desirable properties [P1)–P7)] given above, while at the same time being applicable to continuous optimization problems/algorithms with simple boundary constraints. A preliminary version of this landscape generator can be found in [51].

IV. The Max-Set of Gaussians Landscape Generator

A. The Basic Model

The fundamental component of the landscape generator proposed here is based on an n-dimensional Gaussian function

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

where $\boldsymbol{\mu}$ is an n-dimensional vector of means and Σ is an $(n \times n)$ covariance matrix. Intuitively, a Gaussian function forms an n-dimensional "bump" or hill. A set of m Gaussians can be combined as a weighted sum in a function

$$F(\mathbf{x}) = \sum_{i=1}^{m} w_i g_i(\mathbf{x}) \tag{2}$$

as is commonly done in kernel density estimators and Gaussian mixture models (see, e.g., [5]). The Gaussian components g_i combine to form a more complex function. The influence of each component is determined by its associated weight value w_i. In density estimation, the weights of the mixture are constrained to sum to one to ensure that the volume under the estimator is always one. This is not important for our purposes so the weights can be used to control the amplitude of each component in the sum, allowing each hill to be scaled up or down independently.

Gaussian mixture models are known to be extremely flexible in terms of the shapes of densities that they are able to represent, if the number of components in the mixture can be large. In general, it is a difficult problem to locate the modes of a Gaussian mixture because of the potentially complex ways in which the components can interact [7]. However, for test optimization problems, it is desirable to know the values and locations of the optima of the objective function. To enforce this property, we propose to generate landscapes where the value of a point is given by the maximum value of any of the Gaussian components at that point

$$G(\mathbf{x}) = \max_i [w_i g_i(\mathbf{x})]. \tag{3}$$

An example 2-D landscape of this form is shown in Fig. 1. For such landscapes, the mean of each Gaussian corresponds to an optimum on the landscape, with one exception. If location of the mean $\boldsymbol{\mu}_i$ of any weighted component $w_i g_i$ is dominated by another component $w_j g_j$ (i.e., $w_j g_j(\boldsymbol{\mu}_i) \geq w_i g_i(\boldsymbol{\mu}_i)$, then the peak value of G will never be determined by g_i. Hence, the location of all optima in the landscape generated is known, with the global optimum being the one with the greatest value. Note also that the landscape produced by this max-set will no longer be everywhere continuous and differentiable; ridges are formed along intersections of Gaussian components (see Fig. 1). For our purposes, this is not a concern because MHAs do not typically assume or require such properties in the objective function. In addition, the probability of an algorithm evaluating points exactly (to machine precision) along these ridges is vanishingly small in a continuous space, so this feature of the landscape is

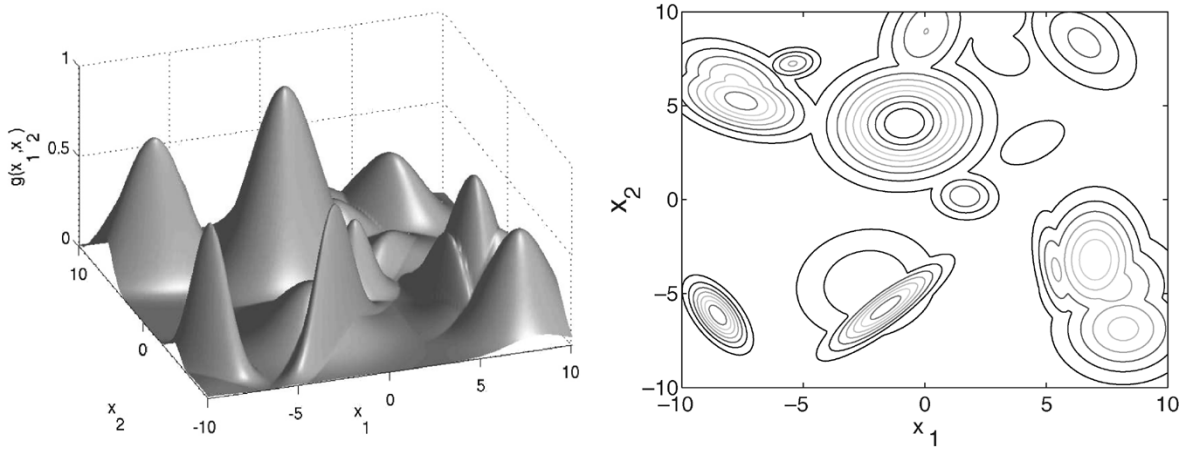


Fig. 1. Example of 2-D landscape (continuous test optimization problem) generated by the max-sum of Gaussians landscape generator. (a) Surface plot. (b) Contour plot.

unlikely to influence the performance of any algorithm. In this paper, we refer to a landscape generator based on the general form described here as a max-set of Gaussians (MSG) landscape generator.

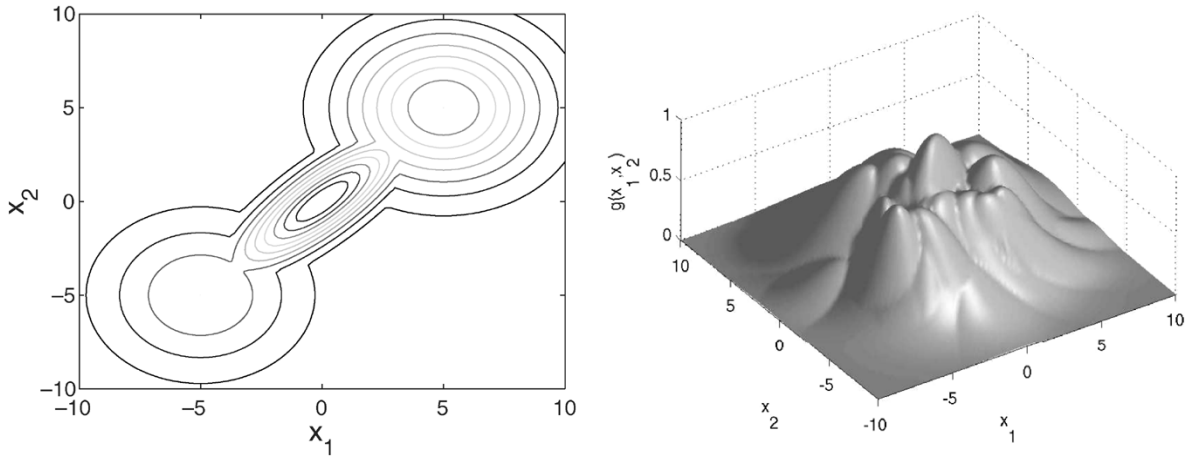
B. Parameterizations of a Max-Set of Gaussians Landscape

There are many possibilities for the parameterization of the max-sum of Gaussians landscape defined above. In particular, we make choices that allow the landscape to be tunable according to certain geometrical features, as well as being probabilistic (i.e., for a given specification of the parameters of the landscape generator, an arbitrary number of random landscapes can be generated that have similar properties). In this paper, we discuss a basic set of landscape parameters.

- The number of Gaussian components m. Clearly, $m = 1$ results in a unimodal landscape. As m increases, the number of peaks and hence local optima in the landscapes also increases. As mentioned above, the actual number of local optima will be at most (but likely less than) m because of the possibility of the peaks of components overlapping.
- The mean vector $\boldsymbol{\mu}_i$ of each component. There are at least two obvious possibilities for the specification of the mean vectors of each component. First, if the experimenter wishes to manually construct a landscape with specific features then these vectors can be predetermined and fixed. An example is given in [52] where a two-peaked landscape is used to analyze the behavior of a continuous population-based incremental learning algorithm [1]. Fig. 2(a) shows a landscape formed by three components positioned in the feasible region of the (2-D) search space, where a narrow, globally optimal peak lies between two broader basins containing local optima.

  Alternatively, mean vectors can be generated from a suitable probability distribution, provided that the samples are constrained to lie within the feasible search space. Possibilities include generating each variable value in a mean vector independently from a uniform distribution (as for the landscape shown in Fig. 1) or from a Gaussian distribution over the search space. As shown in Fig. 2(b), using a mean-zero Gaussian distribution to generate the component mean vectors results in a high concentration of mean vectors (and hence hills) around the origin, with fewer hills as the distance to the origin is increased. Note that this naturally gives rise to a "big valley" [6] or "Massif Central" [26] landscape structure, which has been found in the analysis of a number of artificial and combinatorial optimization landscapes (see, e.g., [6] and [41]).
- The covariance matrix Σ_i of each component. The covariance determines the dependence between variables in each Gaussian component. The contours of constant height for a single Gaussian are hyperellipsoids. The principal axes of these hyperellipsoids are given by the eigenvectors of Σ_i, and the corresponding eigenvalues give the variances along the principal axes. For a diagonal covariance matrix, the principal axes are parallel to the coordinate axes. Off-diagonal terms indicate that the variables are correlated, or are linearly dependent.

  For our landscape generator, the covariance matrices could also be specified manually by the experimenter if desired. In Fig. 2(a), the Gaussian component that produces the global optimum has a large positive covariance, produced by rotating a component with $\sigma^2_{x_1} = 5$ and $\sigma^2_{x_2} = 0.5$ by 45°. Alternatively, it would be desirable to specify a parameter to control the "degree of dependence" typically seen in a landscape component (together these factors determine the amount of dependence structure seen on the overall landscape). A convenient way to generate arbitrary valid covariance structures (i.e., symmetric positive definite matrices) is via a series of rotations of the coordinate system given an initially diagonal matrix. A simple approach used in the adaptation of the mutation distribution in ESs [43] (and adopted here) is to generate a random rotation matrix for each Gaussian component, so that each hill will have a random orientation. An alternative would be to use a different distribution for rotation angle generation. Figs. 1 and 2(b) show examples of landscapes generated with Gaussian components of random covariance structure. Note that the overlaps of components can effectively create more complex dependency structures in the landscape.



(a) (b)

Fig. 2. (a) Contour plot of an example 2-D landscape specifically "designed" using the max sum of Gaussians idea. The global optimum is at (0,0) $(G(\mathbf{x}) = 1.0)$ and the local optima are $(G(\{5, 5\}) = 0.8)$ and $(G(\{-5, -5\}) = 0.4)$. (b) A randomly generated 2-D landscape with 100 Gaussian components. Each element of the component mean vectors is generated from a Gaussian distribution, $\mathcal{N}(0, 2)$.

- Component weights and global optimum hill selection/threshold. As mentioned above, the component weight values w_i allow for the scaling of each component. A generating probability distribution (e.g., uniform) and an appropriate range must be specified. One useful feature of scaling is that it allows us to make the peak of any particular component the global optimum by scaling this peak to be higher than all other peaks by some factor. The global optima of the landscapes shown in Figs. 1 and 2 are at least 25% higher than the next best local optimum. Having knowledge of the location of the global optimum and control of its height is very useful in experiments, as demonstrated in Section V. One method of producing weights (described in Section V-B) is to specify a maximum (threshold) value for local optima t and the fitness value of the global optimum.

A MSG landscape generator with the above parameters can be specified by extending the problem tuples in Section III-C. The general form is

$$\langle \mathrm{MSG}, [-c, +c]^n, n, m, D_\mu, \{D_\Sigma\}, \{t, G^*\}\rangle$$

where we specify in turn the form of the problem generator, the boundary constraints of $\mathcal{S}$, the dimensionality of the search space, the number of Gaussian components, the distribution used to generate mean vectors of components, the distribution/procedures used to generate covariances of components, the threshold for local optima, and the fitness value of the global optimum. The landscape shown in Fig. 1 is then specified as

$$\langle \mathrm{MSG}, [-10, +10]^2, 2, 20, \mathcal{U}[-10, 10], \{\mathcal{U}[0.1, 2.1], \mathcal{U}(-45^\circ, 45^\circ)\}, \{0.8, 1.0\}\rangle$$

while the landscape shown in Fig. 2(b) is specified as

$$\langle \mathrm{MSG}, [-10, +10]^2, 2, 100, \mathcal{N}(0, 2), \{\mathcal{U}[0.1, 2.1], \mathcal{U}(-45^\circ, 45^\circ)\}, \{0.8, 1.0\}\rangle.$$

A designed landscape such as that in Fig. 2(a) would require a somewhat different specification.

C. Comparison of Max-Sum of Gaussians Landscape Generator With Previous Approaches

It would seem that other existing problem and landscape generators all have some useful features for different classes of optimization problems. It is unfortunate that they have not seen more extensive use and evaluation in the literature. We do not claim that the properties of the MSG generator proposed here cannot be found in other generators. Nevertheless, we believe that the MSG generator offers a novel combination of features that make it both generally applicable and useful as a tool in experimental research on optimization algorithms. Specifically, the MSG generator is applicable to continuous optimization problems with bound constraints. It uses a Gaussian as a basis or kernel function; sums of Gaussian functions are widely used in statistics and machine learning and are flexible in the kinds of functions they can approximate. It is randomized, allowing for large sets of landscapes with given features to be easily produced. Finally, it is parameterizable in such a way that makes it easy to apply and easy to relate the parameters to the geometrical properties of the landscape. Varying the values of these parameters causes the landscape to vary in a relatively smooth, predictable way.

With reference to the list of desirable properties of test problems given in Section III-C above, the parameterization of the MSG generator described here satisfies all properties P1–7, with the exception of P4. Although this is no guarantee of the effectiveness of the MSG generator, it at least provides some suggestion that it is well aligned with practical requirements.

V. Experimental Results

In this section, we provide illustrations of the usage of the MSG landscape generator. The primary aim of these experiments is to evaluate the properties of the landscape generator itself, through the application of a number of specified algorithms. We aim to show that the landscapes generated can be discriminative in terms of an appropriate performance criterion (i.e., a range of different results are produced) and that the generator can be used to explore specific properties of algorithms. This is seen with respect to a single specified algorithm, or in comparing the performance of more than one algorithm.

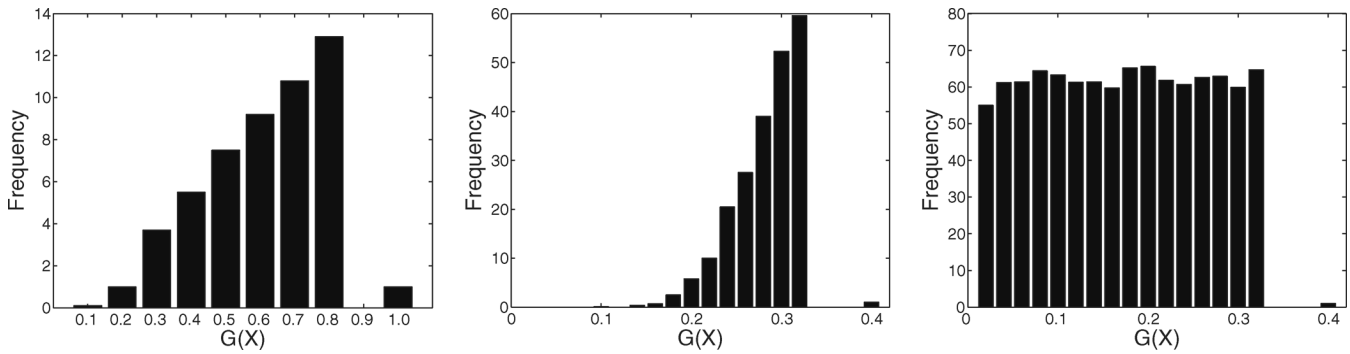


Fig. 3. Performance distributions for an idealized hill-climbing algorithm on some generated landscapes: (a) 2-D landscape, 100 Gaussian components; (b) 2-D landscape, 1000 Gaussian components; and (c) 10-D landscape, 1000 Gaussian components.

A. Illustration: An Idealized Hill-Climbing Algorithm

In order to demonstrate the nature of the results produced by the landscape generator, we consider a simple "algorithm" which represents a kind of idealized hill-climber. Specifically, we assume the existence of a greedy sequentially searching algorithm that starts from a random point in the search space and follows the gradient of the landscape until it reaches the nearest local optimum, where it terminates. Repeating the execution of this algorithm from many random starting positions, the terminating points of these experimental trials will produce a sample of the optima of the landscape. In reality, optima would appear in this sample with a frequency proportional to the size of their basin of attraction in the landscape. Here, we neglect this, effectively assuming that all basin sizes are equal.

First, consider a landscape specified as

$$\langle \mathrm{MSG}, [-10,+10]^2, 2, 100, \mathcal{U}[-10,10], \{\mathcal{U}[0.1,2.1], \mathcal{U}(-45^\circ, 45^\circ)\}, \{0.8,1.0\}\rangle .$$

That is, a 2-D landscape with 100 Gaussian components, the mean vector of each Gaussian generated uniformly in the feasible search space. The variance of each variable in each Gaussian component is generated uniformly in the range [0.1,2.1], while each component is rotated $\binom{n}{2}$ times at a random angle between -45° and 45°. In addition, one optimum is selected at random to be the global optima, scaled to have fitness equal to 1.0, while all other optima are restricted to have values in the range [0,0.8].

Fig. 3(a) shows a histogram produced by counting the optima of this landscape (i.e., approximating the hill-climbing algorithm as described above). Apart from a spike at the global optimum, this histogram reveals a tendency for relatively higher values to dominate the landscape. This is because, although the height of each Gaussian is determined uniformly, higher components will "dominate" lower components when their volume covers the peak of a lower component. This effect increases as the ratio of the number of components to the dimensionality of the landscape increases, as illustrated in Fig. 3(b) for a 2-D landscape with 1000 Gaussian components

$$\langle \mathrm{MSG}, [-10,+10]^2, 2, 1000, \mathcal{U}[-10,10], \{\mathcal{U}[0.1,2.1], \mathcal{U}(-45^\circ, 45^\circ)\}, \{0.8,1.0\}\rangle .$$

As a result, the landscape tends toward a roughly flat landscape as this ratio increases. On the other hand, if this ratio decreases, the components will tend to be well-separated from each other on the landscape. This leads to a more uniform distribution of performance for the idealized hill-climber [as shown in Fig. 3(c) for a ten-dimensional (10-D) 1000-component landscape]

$$\langle \mathrm{MSG}, [-10,+10]^{10}, 10, 1000, \mathcal{U}(-10,10), \{\mathcal{U}[0.1,2.1], \mathcal{U}(-45^\circ, 45^\circ)\}, \{0.8,1.0\}\rangle .$$

It is important to produce results of this kind as a "baseline" for experimental data resulting from the usage of the landscape generator. The reason is that the distributions observed in Fig. 3 are a direct result of the structure of the generated landscapes (i.e., the max-sum of Gaussian functions).

B. Evaluating a Continuous EDA Using the MSG Landscape Generator

In this section, we present the results of experiments where a continuous EDA termed real-coded estimation of distribution algorithm (RECEDA) [38] was applied to a number of generated problems. In this algorithm, a full multivariate Gaussian distribution is the model used to guide the search process. This model is reestimated based on selected individuals at each generation of the algorithm. Sampling from the model is achieved via a Cholesky decomposition of the covariance matrix. The algorithm can be simply parameterized as

$$\langle RECEDA, popsize, selectionrate\rangle$$

where $popsize$ is the size of the population and $selectionrate$ is a value between zero and one, specifying the fraction of the population selected by truncation selection to construct the model. Truncation selection is also used to create the new population from the previous population and the new points generated from the model. The RECEDA algorithm was tested on several sets of landscapes which, roughly speaking, have a global big valley structure [such as that shown in Fig. 2(b)]; in other words, the mean vectors of each component were themselves generated from a Gaussian distribution. The specifications for each set of problems were the following.

- Set 1

$$\langle \mathrm{MSG}, [-10,+10]^{10}, 10, 100, \mathcal{N}(0,4), \{\mathcal{U}[0.2,4.2], \mathcal{U}(-45^\circ, 45^\circ)\}, \{0.8,1.0\}\rangle .$$

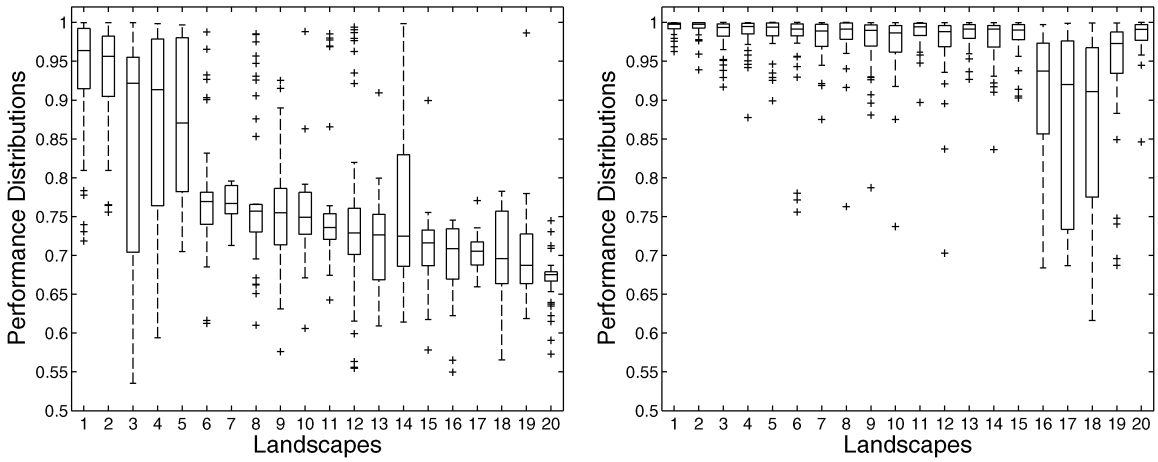


Fig. 4. Experimental results (over 50 trials) on big valley landscapes using RECEDA. (a) Set 1: big valley landscape with global optimum component selected from a wide Gaussian. (b) Set 2: big valley landscapes with global optimum typically much closer to the center of the "valley."

A rough big valley structure is implied. As the global optimum is randomly selected as the mean of one component, it can be located far from the origin (the center of the big valley). In this sense, the global structure can be deceptive. The weight value w_i for each component is calculated based on its original peak value and its assigned fitness value (i.e., for the global peak $w_i g_i = 1$, while other fitness values are normalized in the range [0,0.8]).

- Set 2

$$\langle \mathrm{MSG}, [-10,+10]^{10}, 10, 100, \{\mathcal{N}_l(0,4), \mathcal{N}_g(0,1)\}, \{\mathcal{U}[0.2,4.2], \mathcal{U}(0,45^\circ)\}, \{0.8,1.0\}\rangle.$$

The landscapes in this set are similar to those in Set 1, except that the mean values of the component that determines the global optimum are generated from $\mathcal{N}(0,1)$. This means that the global optimum will typically be much closer to the origin, so the global structure of the landscape tends to give a better indication of the location of the global optimum.

- Set 3

$$\langle \mathrm{MSG}, [-10,+10]^{10}, 10, 100, \{\mathcal{N}_l(0,4), \mathcal{N}_g(0,0)\}, \{\mathcal{U}[0.2,4.2], \mathcal{U}(-45^\circ,45^\circ)\}, \{0.8,1.0\}\rangle.$$

The landscapes in this set are the same as those in Set 1, except that the mean values of the component that determines the global optimum are set equal to zero (i.e., the global optimum is at the origin). In these landscapes, the global structure is even more indicative of the location of the global optimum than those in Set 2.

- Set 4

$$\langle \mathrm{MSG}, [-10,+10]^{10}, 10, 100, \{\mathcal{N}_l(0,4), \mathcal{N}_g(0,0), \mathcal{N}_f(0,4)\}, \{\mathcal{U}[0.2,4.2], \mathcal{U}(-45^\circ,45^\circ)\}, \{0.8,1.0\}\rangle.$$

The landscapes in this set are the same as those in Set 3, except that the fitness values of the local optima are generated proportionally according to the closeness of the component to the origin. The peak values are generated from a symmetric Gaussian-shaped function with mean at the origin and a standard deviation value $\delta = 4$, i.e.,

$$\mathrm{Peak}(\mathbf{x}) = 0.8 \cdot e^{-\frac{\frac{1}{n}\sum_{i=1}^{n} x_i^2}{2\delta^2}}.$$

This means that the closer the peaks are to the origin (and the global optimum), the higher their fitness values.

These sets of experiments attempt to capture a global big valley structure, with this structure becoming stronger from Set 1 through to Set 4.

Twenty different landscapes were generated in each set of experiments. The algorithm used is specified as

$$\langle RECEDA, 100, 0.3\rangle.$$

Fifty trials from random initializations were run for 20 generations. The best objective function value found during each run was recorded.

The results for these experiments are summarized in Figs. 4 and 5. Each box[3] in the figures summarizes the distribution of best solutions found over the 50 repeated trials for each of the 20 landscapes. The boxes (landscapes) in Fig. 4(a) have been sorted according to median values, and the boxes in Figs. 4(b) and 5 are arranged in the same order. Apart from the differences mentioned above (i.e., the global optimum in Sets 2 and 3, and the peak values in Set 4), the landscapes are the same. Therefore, "landscape i" in Fig. 4(a) can be qualitatively compared with "landscape i" in the other plots.

Fig. 4(a) shows a wide range of performance for the algorithm across these landscapes. This indicates that the landscape generator is able to produce landscapes of varying difficulty. As we compare the results across Figs. 4 and 5, it is clear that the performance of the algorithm improves as we move from Set 1 to Set 4 (i.e., as the quality of the big valley structure in the landscape improves). The improvement is most noticeable from Fig. 4(a) to Fig. 4(b) and from Fig. 4(b) to Fig. 5(a). Overall,

[3]The box shows the median and interquartile range (IQR) of the data, with "whiskers" (lines) extending to 1.5 $*$ IQR. Data points outside this range are shown explicitly.

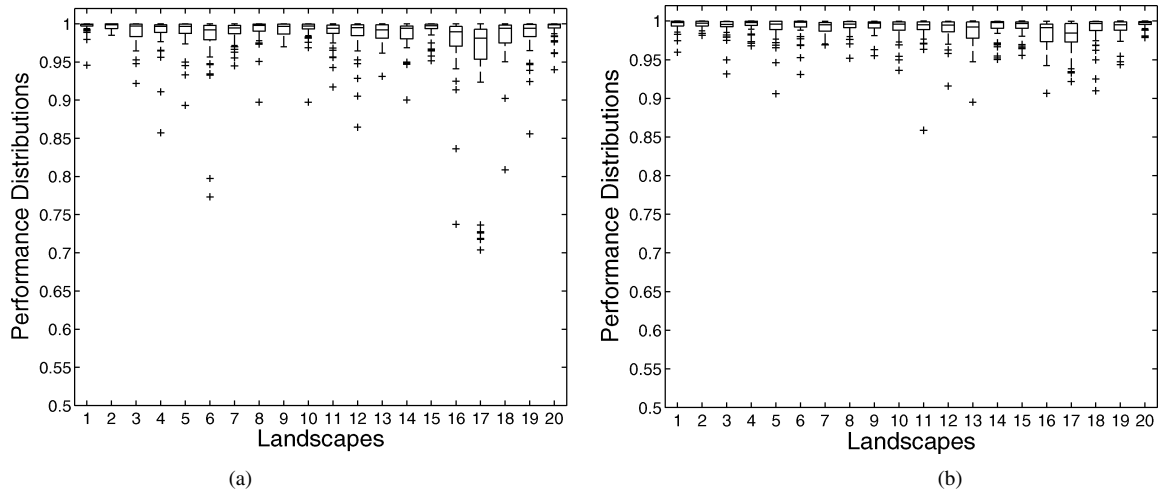


Fig. 5. Experimental results (over 50 trials) on big valley landscapes using RECEDA. (a) Set 3: big valley landscapes with global optimum fixed at origin. (b) Set 4: big valley structure with local optima fitness values closely following the big valley structure.

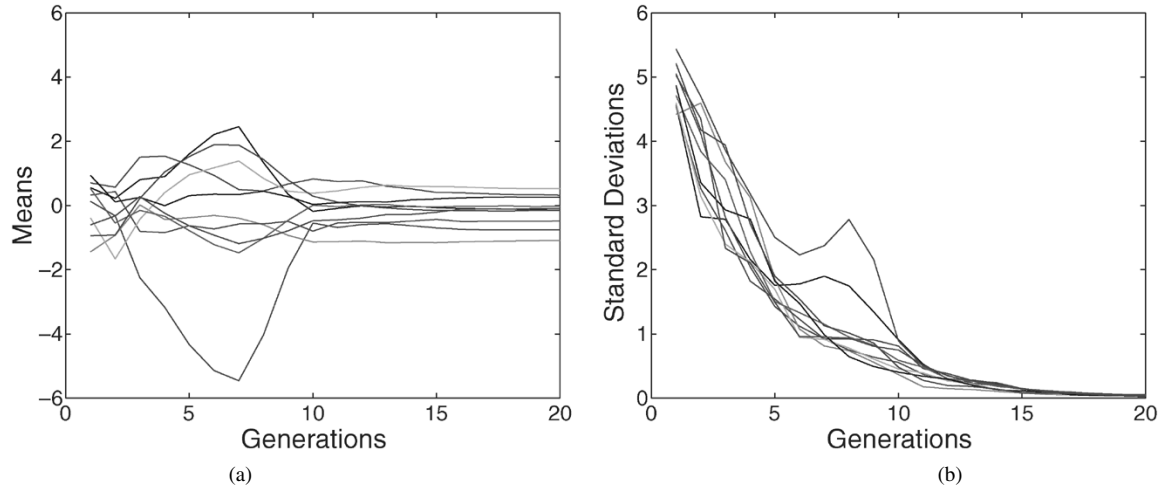


Fig. 6. Evolution of the model parameters of RECEDA on one particular randomly generated landscape. (a) Mean parameters. (b) Standard deviation parameters.

these experiments allow us to hypothesize that RECEDA performs significantly better in the presence of a global (nondeceptive) big valley structure.

Examination of results from these experimental configurations revealed further insight into the dynamics of the algorithm. For one particular landscape (identified during preliminary trials), the algorithms' performance was relatively poor compared to others. One advantage of EDAs is that the parameters of the probabilistic model can be examined directly. Fig. 6 shows the result of a typical experimental trial of RECEDA on this landscape. The results show that during the search, the algorithm encountered landscape structure that caused a noticeable change in the direction [Fig. 6(a)] and scale of the distribution that drives the search process [Fig. 6(b)]. At generation 7, the mean values changed their direction and converged to a different set of values. Around the same time, we see a significant increase in two of the standard deviation values (corresponding to the mean parameters that have the largest change in value). This result leads us to hypothesize that the algorithm has navigated from one basin of attraction into another. Since the structure of the generated landscapes is transparent, we examined the components in the landscape, and found that there was a component in the landscape with mean vector close to the values of the model at generation 7 in Fig. 6(a). The fitness value associated with this component was relatively large, as were its standard deviation parameter values, indicating that this component produces a local optimum with a relatively large basin of attraction on this landscape.

To further verify our hypothesis and reproduce the observed behavior, a 2-D landscape was explicitly constructed (Fig. 7). The landscape has three optima, where the worst optimum has the largest basin of attraction, overlapping with the (smaller) basin of a better optimum, which finally overlaps with the (again smaller) basin of the global optimum. The result of a trial run on this algorithm is shown in Fig. 8. It can be seen clearly from the mean parameter for variable x_1 that the algorithm model progresses through the three basins of attraction during the search. The evolution of the standard deviation parameter for x_1 increases on two occasions, corresponding to the algorithm learning the structure of these "newly encountered" basins. Note that all the algorithm parameter settings will contribute to the convergence behavior, so the behavior seen will not necessarily be seen for all parameter values. In addition, while we discovered this behavior through random trials, we only observed this

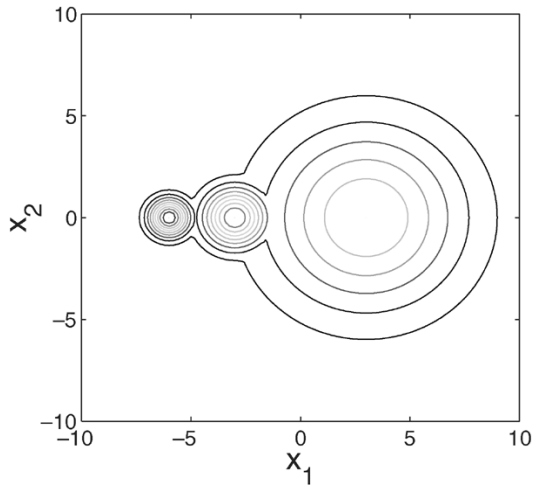


Fig. 7. Contour plot of a 2-D landscape constructed to examine the convergence of RECEDA, inspired by the results on a randomly generated landscape shown in Fig. 6.

behavior clearly for one landscape that we produced (and none of the 20 landscapes in Figs. 4 and 5). Nevertheless the results give an informative insight into the workings of the algorithm, which matches the conceptual intention of its design but nevertheless has not (to our knowledge) previously been demonstrated empirically.

C. The Effect of Dependency Modeling in Continuous EDAs

1) Arbitrary Dependencies in a Unimodal Landscape: The influence of the multimodality of landscapes on RECEDA has been demonstrated above. In this section, we focus on a unimodal landscape, constructed by a single Gaussian function, to investigate the connection between its structure and the performance of RECEDA. As mentioned in Section IV-B, dependence structure can be conveniently introduced by conducting coordinate system rotation. In this experiment, a five-dimensional (5-D) landscape was created by a multivariate Gaussian function with variances [10, 2, 0.4, 0.08, 0.016] and mean vector at the origin. In this case, there are $\binom{5}{2} = 10$ rotation matrices; and within each matrix, the rotation angle was fixed to 45°.

In these experiments, RECEDA was compared with UMDA$_c^G$, a variant of the univariate marginal distribution algorithm (UMDA) [30]. The probabilistic model in UMDA$_c^G$ is a factorized product of univariate Gaussian density functions, and is therefore not capable of explicitly modeling any dependence information in a problem. Both algorithms were tested on the unimodal landscape with population size $= 500$ and number of generations $= 40$. Fig. 9(a) shows the convergence behavior of a typical trial of these algorithms in terms of the standard deviations of their Gaussian models. It is clear that RECEDA achieved a much faster convergence speed. Both algorithms were run for 50 independent trials with the same parameter setting and the mean fitness values of the population in each generation were plotted in Fig. 9(b). This indicates that, on this problem, RECEDA is consistently able to find good solutions more efficiently than UMDA$_c^G$.

2) Restricted Dependencies in a Unimodal Landscape and the MIMIC Algorithm: A significant amount of previous work in developing EDAs has focused on the degree of dependence information incorporated into the probabilistic model [30]. Unfortunately, in some experimental studies it is not possible to compare the dependency structure of the problem with the modeling ability of the algorithm. As a result, it is difficult to relate the performance of the algorithm to its dependency modeling ability. In the following, we demonstrate how the MSG generator can be used to directly align and compare the dependency structure in a test problem with the modeling in the algorithm.

A Gaussian network is a probabilistic graphical model where conditional dependences between problem variables are represented as edges connecting nodes in a graph. The network represents a conditional factorization of the full (assumed Gaussian) multivariate distribution of the problem variables. Given a Gaussian network and its parameter values (i.e., unconditional means, conditional variances and coefficients), there is a unique corresponding Gaussian function whose mean vector and covariance matrix could be found by an algorithm [37]. In order to create a landscape using a Gaussian function with certain predefined dependence structure, an appropriate Gaussian network needs firstly to be specified. For the experiments described in this section, we generated a test problem based on a Gaussian network with five variables assuming the following factorization:

$$G(x_1, x_2, x_3, x_4, x_5) = G(x_1)G(x_2|x_1)G(x_3|x_2)G(x_4|x_3)G(x_5|x_4).$$

This factorization corresponds to a chain-structured Gaussian network of pairwise dependencies. The order of variables is chosen arbitrarily. The conditional variances were set to [5, 1, 0.2, 0.04, 0.008] and all coefficients were set to 0.8. The mean vector was set at the origin. The Gaussian function was scaled to have a maximum fitness value of 1.0.

In these experiments, we used RECEDA, UMDA$_c^G$, and MIMIC$_c^G$, a continuous version of the mutual information maximization for input clustering (MIMIC) algorithm [30]. The model used in this algorithm is a Gaussian network where the dependencies are restricted to a chain of pairwise dependencies. The aim of this experiment is to find out whether MIMIC$_c^G$ is able to capture and exploit the chain structure inherent in the problem. Since MIMIC$_c^G$ has this advantage over UMDA$_c^G$, it would be expected to offer improved performance. On the other hand, RECEDA should also be able to capture the problem structure, although it has the capacity to find more complex models. For each algorithm, a population size of 500 was used, and each trial was run for 30 generations. Each algorithm was run for 50 independent trials.

Fig. 10(a) compares the typical convergence behavior for a single run of RECEDA, MIMIC$_c^G$ and UMDA$_c^G$ in terms of the standard deviation parameters of their respective Gaussian models. It is clear that the convergence speeds of RECEDA and MIMIC$_c^G$ are much faster than that of UMDA$_c^G$, indicating that the ability to capture dependences in MIMIC$_c^G$ provides some benefit. The resulting chain structures discovered by the MIMIC$_c^G$ were also examined, which confirmed that MIMIC$_c^G$ could successfully discover the correct chain structure present in the landscape.

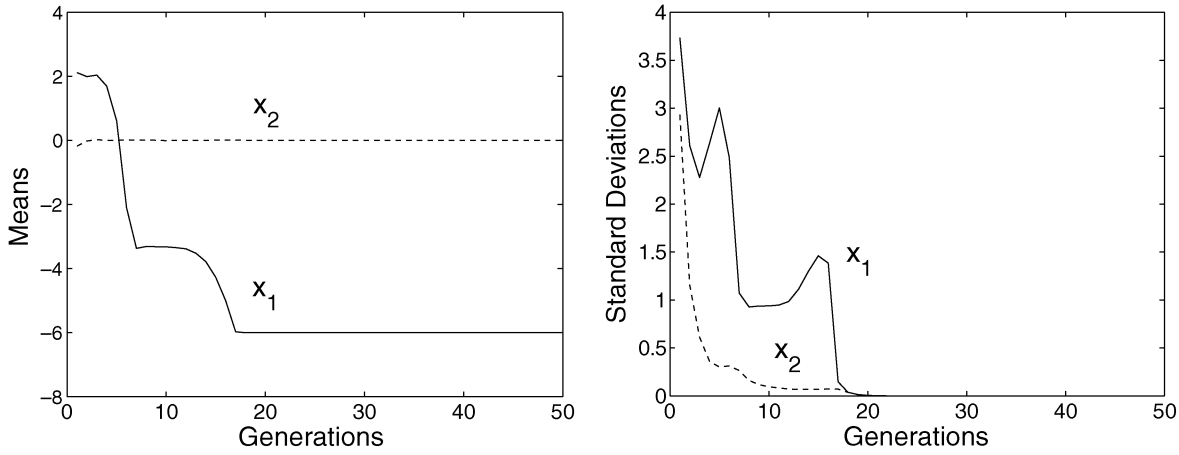
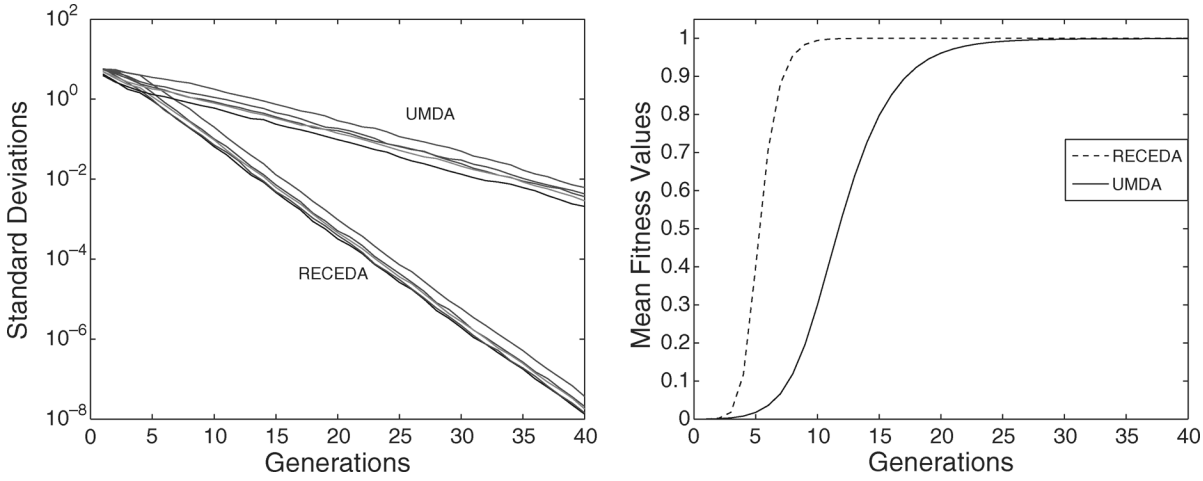
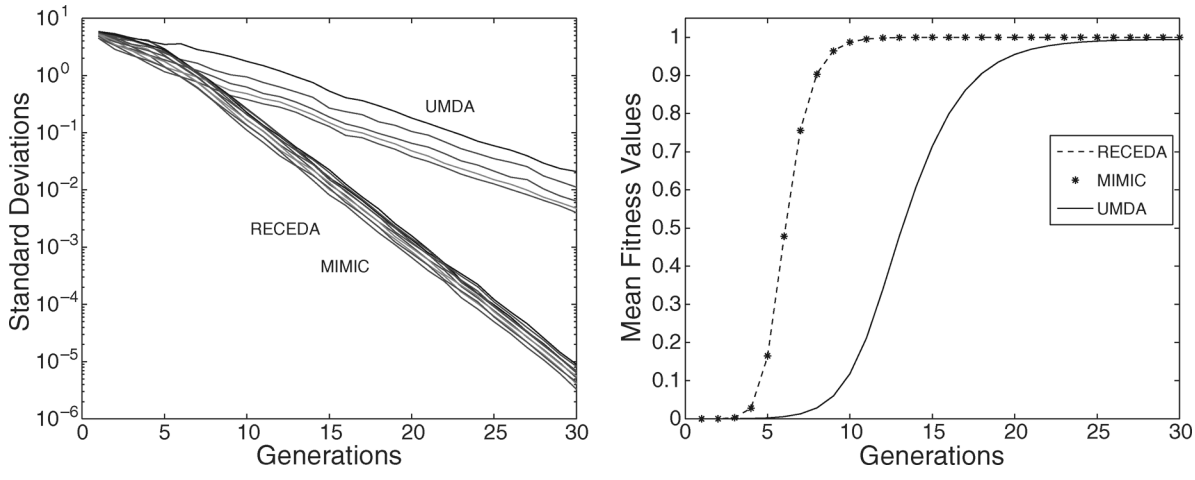


Fig. 8. Evolution of the model parameters of RECEDA on the landscape shown in Fig. 7. (a) Mean parameters. (b) Standard deviation parameters.



(a) (b)

Fig. 9. Results for RECEDA and UMDA$_c^G$. (a) Standard deviation model parameter values (5-D landscape) for each algorithm as a function of generations on a typical trial landscape. (b) Mean fitness values over 50 trials for each algorithm as a function of generations.



(a) (b)

Fig. 10. Comparison of results for UMDA$_c^G$, MIMIC$_c^G$, and RECEDA on a problem with chain-like pairwise dependency structure. (a) Typical convergence of standard deviation model parameters for UMDA$_c^G$ and MIMIC$_c^G$ on a single trial. (b) Mean fitness values for all three algorithm over 50 trials as a function of generations. The curves for MIMIC$_c^G$ and RECEDA are indistinguishable in this graph.

The (average of the) mean fitness values of the population (over the 50 trials) in each generation are shown in Fig. 10(b). Again, RECEDA and MIMIC$_c^G$ show significant advantage over UMDA$_c^G$. No significant difference can be observed between MIMIC$_c^G$ and RECEDA [the two performance curves overlap very closely in Fig. 10(b)]. This shows that there is no need to capture additional dependence information in this situation. Note that for a comprehensive comparison, other factors would

need to be considered, such as the computational requirements of each algorithm, the sensitivity of performance to algorithm parameters (in particular population size, which impacts directly on the estimation of the probabilistic model), and the complexity of implementation.

VI. Conclusion and Future Work

In this paper, a randomized test problem generator has been proposed for the evaluation of continuous metaheuristic algorithms. The generator is based on a set of Gaussian functions and is parameterized by a small number of parameters. These parameters can be related to geometric properties of a problem using the metaphor of a fitness landscape or performance surface over the feasible search space. While other test problem generators exist, we believe that the MSG generator proposed here offers a novel combination of simultaneously desirable features for researchers wishing to perform experimental evaluations and comparisons of metaheuristic algorithms. Experiments have been presented to demonstrate possible uses of the landscape generator as well as providing some understanding of the nature of the results generated by its use. In addition, experiments on continuous EDAs demonstrate that the landscape generator can be used to gain new insights into the behavior of particular algorithms and to conduct comparisons of algorithms.

The most important avenue for future work is the utilization of the landscape generator in conducting thorough empirical studies of MHAs. An implementation of the landscape generator in Matlab will be provided online[4] to encourage further usage. These experiments may also help to evaluate the usefulness of the landscape generator, or to identify its limitations. One potential issue is that the amount of time required to evaluate the fitness of a point on a landscape may become significant when a very large number of Gaussian components are used in the landscape.

There are also a number of ways in which we believe the MSG generator could be extended to produce classes of landscapes with different features from those described here. An obvious possibility is to use different generating distributions for relevant landscape parameters; we have only considered uniform and Gaussian distributions. Another extension would be to introduce a degree of noise into the landscapes. One way of doing this is to add a noise term ϵ to (3)

$$F(\mathbf{x}) = \max_i \left[w_i g_i(\mathbf{x}) \right] + \epsilon \tag{4}$$

where the distribution of the noise model must be specified. Finally (as mentioned in Section IV-C), we have not considered the issue of producing landscapes that are tunable in terms of the time to evaluate the objective function (property P4 in Section III-C).

We hope that the MSG generator will serve as a useful tool for the MHA research community. Clearly, however, it is only a small part of the effort required to introduce more rigorous and comprehensive experimental work in the evaluation and comparison of metaheuristic algorithms.

[4]See http://www.itee.uq.edu.au/~marcusg/msg.html.

References

[1] S. Baluja, "Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning," School of Computer Science, Carnegie Mellon Univ., Pittsburgh, PA, Tech. Rep. CMU-CS-94-163, 1994.

[2] R. S. Barr, B. L. Golden, and J. P. Kelly, "Designing and reporting on computational experiments with heuristic methods," *J. Heurist.*, vol. 1, pp. 9–32, 1995.

[3] T. Bartz-Beielstein, "Experimental analysis of evolution strategies—Overview and comprehensive introduction," Universität Dortmund, Tech. Rep. Reihe CI 157/03, SFB 531, 2003.

[4] T. Bartz-Beielstein, K. E. Parsopoulos, and V. N. Vrahatis, "Design and analysis of optimization algorithms using computational statistics," *Appl. Numer. Anal. Comp. Math.*, vol. 1, no. 2, pp. 413–433, 2004.

[5] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford, U.K.: Oxford Univ. Press, 1995.

[6] K. D. Boese, "Models for iterative global optimization," Ph.D. dissertation, Univ. of California, Los Angeles, CA, 1996.

[7] M. A. Carreira-Perpiñán and C. K. I. Williams, "On the number of modes of a Gaussian mixture," in *Lecture Notes in Computer Science*, L. D. Griffin and M. Lillholm, Eds. Berlin, Germany: Springer-Verlag, 2003, vol. 2695, Proc. 4th Int. Conf. Scale Space Methods in Computer Vision, pp. 625–640.

[8] R. Chelouah and P. Siarry, "Enhanced continuous tabu search: an algorithm for the global optimization of multiminima functions," in *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, S. Voss *et al.*, Eds. Norwell, MA: Kluwer, 1999, ch. 4, pp. 49–61.

[9] M. Coffin and M. J. Saltzman, "Statistical analysis of computational tests of algorithms and heuristics," *INFORMS J. Comput.*, vol. 12, no. 1, pp. 24–44, 2000.

[10] K. A. De Jong, M. A. Potter, and W. M. Spears, "Using problem generators to explore the effects of epistasis," in *Proc. 7th Int. Conf. Genetic Algorithms*, T. Bäck, Ed., 1997, pp. 338–345.

[11] K. Deb, "Multi-objective genetic algorithms: problem difficulties and construction of test problems," *Evol. Comput.*, vol. 7, no. 3, pp. 205–230, 1999.

[12] A. E. Eiben and M. Jelasity, "A critical note on experimental research methodology in EC," in *Proc. 2002 Congr. Evol. Comput.*, 2002, pp. 582–587.

[13] I. P. Gent, S. A Grant, E. MacIntyre, P. Prosser, P. Shaw, B. M. Smith, and T. Walsh, "How not to do it," School of Computer Studies, Univ. of Leeds, Leeds, U.K., Tech. Rep. 97.27, May 1997.

[14] M. Farina, K. Deb, and P. Amato, "Dynamic multi-objective optimization problems: Test cases, approximations and applications," *IEEE Trans. Evol. Comput.*, vol. 8, no. 5, pp. 425–442, 2004.

[15] D. B. Fogel, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. Piscataway, NJ: IEEE Press, 1995.

[16] O. François and C. Lavergne, "Design of evolutionary algorithms—A statistical perspective," *IEEE Trans. Evol. Comput.*, vol. 5, no. 2, pp. 129–148, 2001.

[17] M. Gallagher, "Fitness distance correlation of neural network error surfaces: A scalable, continuous optimization problem," in *Eur. Conf. Machine Learning (ECML 2001)*, vol. 2167, Lecture Notes in Artificial Intelligence, L. De Raedt and P. Flach, Eds., Singapore, 2001, pp. 157–166.

[18] J. He and X. Yao, "From an individual to a population: An analysis of the first hitting time of population-based evolutionary algorithms," *IEEE Trans. Evol. Comput.*, vol. 6, no. 5, pp. 495–511, 2002.

[19] F. Herrera, M. Lozano, and J. Verdegay, "Tackling real-coded genetic algorithms: Operators and tools for behavioral analysis," *Artif. Intell. Rev.*, vol. 12, pp. 265–319, 1998.

[20] R. R. Hill, "An analytical comparison of optimization problem generation methodologies," in *Proc. Winter Simulation Conf.*, D. J. Dedeiros *et al.*, Eds., 1998, pp. 609–615.

[21] J. N. Hooker, "Testing heuristics: We have it all wrong," *J. Heurist.*, vol. 1, pp. 33–42, 1996.

[22] C. Igel and M. Toussaint, "On classes of functions for which no free lunch results hold," *Inf. Process. Lett.*, vol. 86, pp. 317–321, 2003.

[23] R. H. F. Jackson, P. T. Boggs, S. G. Nash, and S. Powell, "Guidelines for reporting results of computational experiments. Report of the ad hoc committee," *Math. Program.*, vol. 49, pp. 413–425, 1991.

[24] R. H. F. Jackson and J. M. Mulvey, "A critical review of comparisons of mathematical programming algorithms and software (1953–1977)," *J. Res. Nat. Bur. Stand.*, vol. 83, no. 6, pp. 563–584, Jun. 1978.

[25] D. Johnson, "A theoretician's guide to the experimental analysis of algorithms," in *Proc. 5th and 6th DIMACS Implementation Challenges, Data Structures, Near Neighbor Searches, Methodology*, M. H. Goldwasser, D. S. Johnson, and C. C. McGeoch, Eds., Providence, 2002, pp. 215–250.
[26] S. Kauffman, *The Origins of Order*. Oxford, U.K.: Oxford Univ. Press, 1993, ch. 2, pp. 33–67.
[27] J. Kennedy. (1997) Continuous valued multimodality generator for evolutionary algorithms. [Online]. Available: http://www.cs.uwyo.edu/~wspears/multi.kennedy.html
[28] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Networks*, New York, 1995, pp. 1942–1948.
[29] J. Kennedy and W. M. Spears, "Matching algorithms to problems: an experimental test of the particle swarm and some genetic algorithms on the multimodal problem generator," in *Proc. Int. Conf. Evol. Comput.*, 1998, pp. 78–83.
[30] P. Larrañaga and J. A. Lozano, Eds., *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Norwell, MA: Kluwer, 2002.
[31] C. C. McGeoch, "Toward an experimental method for algorithm simulation," *INFORMS J. Comput.*, vol. 8, no. 1, pp. 1–15, 1996.
[32] ——, "Experimental analysis of optimization algorithms," in *Handbook of Applied Optimization*, P. M. Pardalos and M. Resende, Eds. Oxford, U.K.: Oxford Univ. Press, 2002, ch. 24, pp. 1044–1052.
[33] Z. Michalewicz, K. Deb, M. Schmidt, and T. Stidsen, "Test-case generator for nonlinear continuous parameter optimization techniques," *IEEE Trans. Evol. Comput.*, vol. 4, no. 3, pp. 197–215, 2000.
[34] D. Mitchell, B. Selman, and H. Levesque, "Hard and easy distributions of SAT problems," in *Proc. 10th Nat. Conf. Artificial Intelligence*, 1992, pp. 459–465.
[35] R. W. Morrison and K. A. De Jong, "A test problem generator for nonstationary environments," in *Proc. Congr. Evol. Comput.*, 1999, pp. 2047–2053.
[36] R. Myers and E. R. Hancock, "Empirical modeling of genetic algorithms," *Evol. Comput.*, vol. 9, no. 4, pp. 461–493, 2001.
[37] R. E. Neapolitan, *Learning Bayesian Networks*. Upper Saddle River, NJ: Pearson Prentice-Hall, 2004.
[38] T. K. Paul and H. Iba, "Real-coded estimation of distribution algorithm," in *Proc. 5th Metaheuristics Int. Conf.*, 2003, pp. 61–66.
[39] R. L. Rardin, C. Tovey, and M. Pilcher, "Analysis of a random cut test instance generator for the tsp," in *Complexity in Numerical Optimization*, P. Pardalos, Ed. Singapore: World Scientific, 1993.
[40] R. L. Rardin and R. Uzsoy, "Experimental evaluation of heuristic optimization algorithms: A tutorial," *J. Heurist.*, vol. 7, pp. 261–304, 2001.
[41] C. R. Reeves, "Landscapes, operators and heuristic search," *Ann. Oper. Res.*, vol. 86, pp. 473–490, 1999.
[42] J.-M. Renders and S. P. Flasse, "Hybrid methods using genetic algorithms for global optimization," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 26, no. 2, pp. 243–258, 1996.
[43] G. Rudolph, "On correlated mutations in evolution strategies," in *Parallel Problem Solving From Nature: PPSN-II*, R. Männer and B. Manderick, Eds. Berlin, Germany: Springer-Verlag, 1992, pp. 107–116.
[44] J. D. Schaffer, R. Caruana, L. Eshelman, and R. Das, "A study of control parameters affecting online performance of genetic algorithms for function optimization," in *Proc. 3rd Int. Conf. Genetic Algorithms*, J. D. Schaffer, Ed., 1989, pp. 51–60.
[45] H.-P. Schwefel, *Evolution and Optimum Seeking*. New York: Wiley, 1995.
[46] R. Storn and K. Price, "Differential evolution—A simple and efficient adaptive scheme for global optimization over continuous spaces," International Computer Science Inst., Tech. Rep. TR-95-012, Mar. 1995.
[47] B. E. Stuckman, "A global search method for optimizing nonlinear systems," *IEEE Trans. Syst., Man, Cybern.*, vol. 18, no. 6, pp. 965–977, 1988.
[48] D. Whitley, K. Mathias, S. Rana, and J. Dzubera, "Building better test functions," in *Proc. 6th Int. Conf. Genetic Algorithms*, L. J. Eshelman, Ed., 1995, pp. 239–247.
[49] ——, "Evaluating evolutionary algorithms," *Artif. Intell.*, vol. 85, no. 1–2, pp. 245–276, 1996.
[50] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67–82, 1997.
[51] B. Yuan and M. Gallagher *et al.*, "On building a principled framework for evaluating and testing evolutionary algorithms: A continuous landscape generator," in *Proc. Congr. Evol. Comput.*, R. Sarkar *et al.*, Eds., 2003, pp. 451–458.
[52] ——, "Playing in continuous spaces: Some analysis and extension of population-based incremental learning," in *Proc. Congr. Evol. Comput.*, R. Sarkar *et al.*, Eds., 2003, pp. 443–450.
[53] ——, "Statistical racing techniques for improved empirical evaluation of evolutionary algorithms," in *Lecture Notes in Computer Science*, vol. 3242, Proc. Parallel Problem Solving Nature (PPSN VIII), X. Yao *et al.*, Eds.. Berlin, Germany, 2004, pp. 172–181.
[54] Q. Zhang and H. Muhlenbein, "On the convergence of a class of estimation of distribution algorithms," *IEEE Trans. Evol. Comput.*, vol. 8, no. 2, pp. 127–136, 2004.

Marcus Gallagher (M'99) received the B.Comp.Sc. and Grad.Dip.Sc. degrees from the University of New England, Australia, in 1994 and 1995, respectively, and the Ph.D. degree from the University of Queensland, Queensland, Australia, in 2000.

He is currently a Lecturer in the School of Information Technology and Electrical Engineering, University of Queensland. His main research interests are metaheuristic optimization and machine learning algorithms, in particular, techniques based on statistical modeling. He is also interested in the biologically inspired algorithms, methodology for empirical evaluation of algorithms, and the visualization of high-dimensional data.

Bo Yuan (S'02) received the B.E. degree from Nanjing University of Science and Technology, China, in 1998 and the M.Sc. degree from the University of Queensland, Queensland, Australia, in 2002. He is currently working towards the Ph.D. degree at the University of Queensland, where his doctoral work is in evolutionary computation.

His research interests include estimation of distribution algorithms, experimental methodology, machine learning, and statistical techniques.