On Building a Principled Framework for Evaluating and Testing Evolutionary Algorithms: A Continuous Landscape Generator

Bo Yuan Marcus Gallagher
School of Information Technology and Electrical Engineering
The University of Queensland
QLD 4072, Australia
{boyuan, marcusg}@itee.uq.edu.au

Abstract - In this paper, we address some issues related to evaluating and testing evolutionary algorithms. A landscape generator based on Gaussian functions is proposed for generating a variety of continuous landscapes as fitness functions. Through some initial experiments, we illustrate the usefulness of this landscape generator in testing evolutionary algorithms.

1 Introduction

Evolutionary Algorithms (EAs) have been successful at solving many challenging problems. Driven by their success, much effort has been devoted to designing new algorithms or improving existing algorithms. By contrast, little work has been done to answer the question: How can we effectively evaluate the performance of these algorithms? EAs are commonly evaluated based on some canonical test problems. But if an algorithm works well on a set of test problems, it does not necessarily mean that it will also perform well on other problems[1]. It is insufficient to draw general conclusions according to simulation results based on *ad hoc* test problems[2].

The reason is that people often regard these test problems as black-boxes and do not have very good knowledge about their landscapes. It is very likely that they are not representative of the classes of problems that people attempt to solve with EAs. As a result, the predictive value of those experiments can be weak. After all, algorithms operate on landscapes, not on problems. If there is little or no information about how an algorithm interacts with landscapes and the relationship between its performance and the properties of landscapes, it would be very difficult to predict its performance on other problems. In the mean time, people have already identified some disadvantages of current test problems. For example, many of them are separable, which can be often readily solved by local search methods[3].

Another issue is that people often spend a lot of time manually tuning the parameters of an algorithm in the hope that it can outperform other algorithms but finding a set of optimal parameters for a specific problem can itself be a difficult problem[4]. It is unfair to compare algorithms with careful and thorough tuning to other algorithms with no or little tuning[5]. After all, researchers are more interested in understanding when and why an algorithm works well or badly than simply knowing how well it can be on a specific problem.

According to the NFL theorems[6], no algorithm is better than others when averaged over all possible problems. This tells us that when comparing different algorithms, we must focus our attention on a restricted set of problems. Unfortunately, even if we concentrate on a very small set of problems, we still cannot completely rely on theoretical analysis to describe or predict the performance of an algorithm on a specific problem because the performance of an algorithm is tightly related to the specific problem and our current theoretical work usually can only apply to some very general or idealized situations. As a result, there is an urgent need for improving approaches to experimental evaluation of algorithms in research.

Recently, there has been some progress towards conducting systematic evaluation using test problem generators[1, 7-10]. The most significant advantage of problem generators is that they allow researchers to run controlled experiments in which the properties of problems can be systemically varied[8]. Another advantage is that they can remove the opportunity to hand-tune algorithms to a specific problem and, by allowing a large umber of problem instances to be randomly generated, the predictive power of the simulation results can also be increased[1, 8].

In this paper, we propose a new continuous multimodal landscape generator based on a sum of Gaussian functions. It can generate a wide range of landscapes, parameterized in terms of structural features of a landscape, such as the number of local optima, the basin size, the position and the height of each peak and the dependences among variables.

We also test two optimization algorithms by this landscape generator: a real-valued GA and a continuous PBIL (Population-Based Incremental Learning)[11, 12]. We demonstrate the different behaviors of these two algorithms under a large number of random landscapes in order to have a whole image of the variety of performance. We show that our landscape generator is a useful tool for evaluating and testing EAs and can help people have a deep understanding of the performance of various algorithms under different situations.

In the next section, we talk about some general issues related to evaluation and testing of EAs and briefly review some existing test problem generators. Section 3 gives the implementation details of our landscape generator. Section 4 describes the real-valued GA and the continuous PBIL algorithm used in our simulations and presents some simulation results. In the last section, we summarize our work and discuss directions of future work.

2 Evaluating Evolutionary Algorithms

2.1 Playing with Landscapes

Traditionally, researchers try to understand and evaluate EAs by empirically testing them on a set of benchmark problems. In general, an unconstrained optimization problem can be specified as:

$$\min_X f(X), \qquad X = (x_1, ..., x_n) \tag{1}$$

The function *f*(X) is defined over n-dimensional vectors and the goal of optimization is to find a vector X^*, which yields the minimum function value.

One advantage of EAs is that they only require minimum knowledge about the problem. Only the fitness value of X is needed to conduct searching. The common property of various EAs is that, in each generation, they will visit a population of points and calculate their fitness values. Then, a new population of points is generated and this process repeats until stopping criteria are met. The difference is how to generate a set of new points from current points. Different algorithms use different heuristics to do this task and each heuristic used represents the assumption of that algorithm about the structure of the problem (i.e., the landscape). For example, crossover operators usually assume that high-level building blocks can be created by combining low-level building blocks. So, hopefully, better individuals can be generated through combination of genes on good individuals. Certainly, this is not always the truth in practice.

Obviously, the properties of landscapes have a great influence on the performance of optimization algorithms. If the heuristic of an algorithm matches the structure of a problem, it would be reasonable to expect that it will perform well on this problem.

Furthermore, people also need to be aware of the reverse effect. For example, in a continuous space, if a hill-climbing algorithm uses a large step size, it may jump over local optima where the basin size is small compared to the step size. So, with a large step size, a hill-climbing algorithm can effectively smooth the landscape and as a result, some local optima will no longer exist for that algorithm. This tells us that algorithms actually search on a landscape defined by their operators and the fitness function[13].

It is possible to conduct evaluation and testing of algorithms based on landscapes with different properties. Furthermore, each EA is also coupled with a set of parameters. So, if we take into account all kinds of landscapes and all kinds of algorithms with different specified parameters, we describe a very large "space of experiments", which contains all possible experiments over all feasible values of both landscapes and algorithm parameters. Running experiments on a small number of test problems and with some hand-tuned parameters is like sampling a small cluster of points in this experimental space and any general conclusion about the space drawn from these few points is incomplete and misleading.

2.2 Real World Problems

The No Free Lunch (NFL) theorems show that, in a general sense, all optimization algorithms are equal across all possible problems[6]. However, no claims are made concerning the performance of an algorithm in practice. In fact, the real world problems that we are interested in are just a subset of the set of all possible problems on which NFL establishes its conclusion.

It is hard to quantify the structure of real-world problems. However, we would like to point out some general differences between real-world problems and arbitrary problems. Among all possible problems, many of them will look like noise functions because there is no consistent relationship between two neighboring points in terms of fitness value. If the fitness of a point can be some arbitrary value, having no relationship with the fitness values of its neighboring points, we can expect that the performance of all EAs would be very bad because the landscape has no exploitable structure.

Fortunately, this is not true in our physical world where changes often occur gradually or smoothly in terms of landscape. Certainly, due to many different factors, the micro-landscapes of real-world problems may not be very smooth. However, if we look at them from a high level, we can expect to find some global smoothness. Another property of real-world problems is that some of the parameters are often dependent of each other, which means that it is impossible to tune one parameter independently of others.

2.3 Good Test Problems

Previous research has identified a number of properties that are considered to be desirable for test optimization problems when evaluating EAs[3, 9, 14, 15]:

P1. Difficult to solve using simple methods such as hill-climbing algorithms;
P2. Nonlinear, non-separable and non-symmetric;
P3. Scalable in terms of problem dimensionality;
P4. Scalable in terms of time to evaluate the cost function;
P5. Tunable by a small number of user parameters;
P6. Can be generated at random and are difficult to reverse engineer;
P7. Exhibit an array of landscape-like features.

Generally speaking, test problems for EAs should be capable of exposing the strengths and weaknesses of these algorithms. For example, it does not make much sense to test a GA on a unimodal landscape with no dependence among variables, which can be easily solved by a large number of other algorithms like hill-climbing algorithms. Furthermore, test problems should resemble real-world problems, at least to some extent. As mentioned in the previous section, one expected feature of real-world problems is global smoothness. In terms of landscape structure, this may be the "hills", "valleys" and the like. Certainly, in high-dimensional spaces, these features are not obvious.

2.4 Landscape Generators

2.4.1 Landscape Generators vs. Problem Generators

Given the discussion above, it is clear that the number of optimization problems that might be considered relevant to real-world problems is huge and making some general conclusion about the performance of an algorithm based on very limited experiments on some classical handcrafted test problems is far from sufficient. This means that we need to be able to run experiments on a wide range of test problems to make our conclusion significant.

In recent years, there has been some progress towards using test case generators as the ground for optimization algorithms evaluation and testing. One common character of these test case generators is that they are all able to generate a number of random test problems. However, it is necessary to classify them into two categories: *problem generators* and *landscape generators*. In general, problem generators are used to generate a set of problem instances belonging to a specific problem class, for example, Boolean satisfiability (SAT) problems[1]. The control variables of these generators are usually the parameters of these problems themselves. In SAT they may include the number of Boolean variables and the number of disjunctive clauses. Problem generators are useful for investigating the performance of optimization algorithms on a specific class of problems. Another property of these problem generators is that they are high-level abstractions of problems that are not intended to provide information about the resulting fitness landscape structure, which is very important for algorithm evaluation and testing. If we are indeed interested in the performance of an EA across widely different landscapes, these black-box problems are of little direct value.

In contrast, landscape generators do not take into account any specific class of problems. Test cases generated are simply landscapes on which an algorithm will conduct searching. Unlike black-box problems, the structure of these problems can be explicitly specified, which allows people to have not only a whole image of the algorithm's performance but also a good understanding about the relationship between the performance of algorithms and the properties of landscapes.

2.4.2 A Brief Review of Existing Landscape Generators

Below are some landscape generators that have been used in previous work.

A. *NK Landscape Generator*[1] :

$$f(chromosome) = \frac{1}{N}\sum_{i=1}^{N} f(locus_i) \qquad (2)$$

B. *Bit-string Multimodal Landscape Generator* [8]:

$$f(x) = \tfrac{1}{L}\max_{i=1}^{P}\{L - HAMMING(x, Peak_i)\} \qquad (3)$$

C. *Continuous Multimodal Landscape Generator* [7]:

$$f(x) = \tfrac{1}{N}\min_{i=1}^{P}\left\{\sum_{j=1}^{N}(sigmoid(x_j) - Peak_i^j)^2\right\} \qquad (4)$$

D. *Dynamic Landscape Generator – DF1* [10]:

$$f(x, y) = \max_{i=1}^{P}\left\{H_i - R_i \cdot \sqrt{(x - X_i)^2 + (y - Y_i)^2}\right\} \qquad (5)$$

E. *Continuous Test-case Generator* [9]:

$$f_k(x) = a_k\left(\prod_{i=1}^{N}(u_i^k - x_i)(x_i - l_i^k)\right)^{\frac{1}{N}} \qquad (6)$$

In the NK landscape generator (A), N is the length of the binary string (chromosome) and K represents the number of linkages that each gene has to other genes. The fitness contribution of each bit (gene), $f(locus_i)$, is determined by searching a table with an index consisting of the value of i^{th} gene and the values of other K related genes. One practical difficulty related to this landscape generator is the storage issue: we need N tables with each table containing 2^{K+1} elements. Another concern is that since the table is often randomly generated, we cannot explicitly control the properties of the landscapes. For example, by changing the value of K, one can expect to get landscapes with different levels of complexity but it is difficult to quantitatively specify them.

As a contrast, other landscape generators (B to E) generate landscapes with predefined structure. The bit-string multimodal landscape generator (B) randomly chooses P L-bit individuals as peaks in the search space. All peaks have equal fitness value 1.0. Then, the fitness of a string is decided by its Hamming distance to the closest peak. The continuous multimodal landscape generator (C) is an extension of (B). It creates P peaks on N dimensions, uniformly distributed in the interval [0.0, 1.0]. All variables are restricted to this interval with a sigmoid function. Again, the fitness value is decided by the distance between the individual to be evaluated and the closest peak. In this case, the fitness value is to be minimized and has the lowest value 0. The dynamic landscape generator (D) is used to study EAs in changing environments. It employs P cones to construct landscapes in continuous spaces. Each cone has three parameters: position, height (H_i) and slope (R_i). The fitness value of each point in the search space is decided by its distance to those cones, the height and the slope of each cone. The general idea of the continuous landscape generator (E) is to divide the search space into a number of disjoint subspaces (N-dimensional cubes) and to define a unimodal function f_k for each cube. u_i^k and l_i^k are the upper and lower boundaries of i^{th} dimension of k^{th} subspace respectively while a_k is a predefined positive constant. It is easy to see that the optimum of each subspace locates in the middle of the cube. This landscape generator also employs a set of constraints by further dividing each subspace into feasible and infeasible parts.

2.4.3 Problems of Existing Landscape Generators

Although the above landscape generators (B to E) differ from each other in terms of binary vs. continuous, dynamic vs. static and constrained vs. unconstrained, they have one common shortcoming: even if the whole landscapes have some good properties as described in Section 2.3, local landscape features around optima do not reflect characteristics of real-world problems in that they are both symmetric and separable. If we examine the local area around an optimum, we would find that the landscape is symmetric because if a point moves away from that optimum, the amount of fitness loss is independent of the dimension along which it moves. This means that each variable has equal contribution to the fitness value or in other words, the fitness function is equally sensitive to all variables, which is obviously not true in real-world problems. Furthermore, the landscape is separable because the optimal value of each variable can be decided separately. In the above fitness functions, there is no explicit dependence among variables in each local area around an optimum. As a result, as long as an optimization algorithm can find the basin of a global optimum, the problem can be readily solved by local algorithms such as hill-climbing algorithms.

3 A Gaussian Landscape Generator

3.1 Landscapes: Mixture of Gaussians

In this paper we develop a landscape generator for problems with continuous variables. The basic "building-block" of this landscape generator is a Gaussian Function (GF) and each landscape may contain one or more GFs, each constituting a "hill" in the landscape. The probability density function of an n-dimensional Gaussian distribution is given by [16]:

$$G(X)=\frac{1}{(2\pi)^{n/2}\left|\sum\right|^{1/2}}\exp\left\{-\frac{1}{2}(X-\mu)^T\sum^{-1}(X-\mu)\right\} \quad (7)$$

In Eq. 7, $\sum$ is the covariance matrix and X is the parameter vector to be evaluated. We can see that, for each GF, the probability of X is governed by $\sum$ and μ. The fitness value of X in a landscape with m GFs can be calculated based on the mixture of GFs:

$$F_1(X)=\sum_{i=1}^{m}\omega_i G_i(X), \qquad \sum_{i=1}^{m}\omega_i=1 \quad (8)$$

In this Gaussian mixture model (Eq. 8), each GF is assigned a weight ω_i and the fitness value of X is the sum of the weighted probabilities across all GFs. Normally the sum of the weights should be 1 but for the purpose of our landscape generator, it does not matter. The mixture model is well known to be capable of representing a wide range of densities and is smooth and differentiable at every value of X. However, it is not easy to identify the global optimum or optima because of the way the weighted component GFs combine in the summation in Eq. 8. To avoid this, we propose a landscape generator based on the maximum value as below:

$$F_2(X)=\max_i G_i(X), \qquad i=1...m \quad (9)$$

According to Eq. 9, given a point X in the search space, we first calculate its value with regard to each GF and its fitness is set to the highest value returned by all GFs. So, the mean vector of each GF corresponds to an optimum and the mean vector of the GF that has the highest peak value corresponds to the global optimum. Note that one possible exception is when one GF is covered or swallowed by another GF. That is, the peak of the GF is beneath another GF.

In summary, each landscape generated by this Gaussian landscape generator consists of a set of GFs, which create hills, valleys and other landscape features. The parameters of our landscape generator are the total number of GFs (*m*), the dimensionality of the landscape (*n*), the range of the search space, the mean vector, variances and the covariance matrix of each GF, which decides the shape of each GF. In practice, any of these parameters can be varied or held constant, depending on the goals of the experiments and the amount of available CPU time.

3.2 Rotation

The covariance matrix $\sum$ determines the dependence among variables in each GF. The surfaces of constant fitness (height) in a GF are hyperellipsoids. The principal axes of the hyperellipsoids are given by the eigenvectors of $\sum$ and the corresponding eigenvalues give the variances along the respective principal directions[16]. For the purpose of algorithm evaluation and testing, dependence means that it is impossible to optimize one variable while setting other variables to some arbitrary values. Although multimodal landscapes usually have inherent dependence among variables, using a non-diagonal $\sum$ introduces additional difficulty to the landscapes generated because the micro-landscape around each peak will also become non-separable provided that the elements on the diagonal are not identical. However, randomly generating a valid covariance matrix itself is not an easy task because the matrix must be positive definite.

Alternatively, a Gaussian with arbitrary valid covariance structure can be conveniently generated through a series of rotations of the variable coordinate system[17]. The basic idea is to first assume that each variable is independent of others. So, the n-dimensional multivariate Gaussian is simply the product of each 1D GF. Next, a rotation matrix is generated for each GF and the coordinate system is rotated according to the rotation matrix so that the principal directions of each GF in its own new coordinate system are not aligned with the coordinate axes (Figures 1&2). Hence, $\sum$ is parameterized by rotation angles and variance values. Again, each angle can be manually chosen or randomly generated from a predefined range.

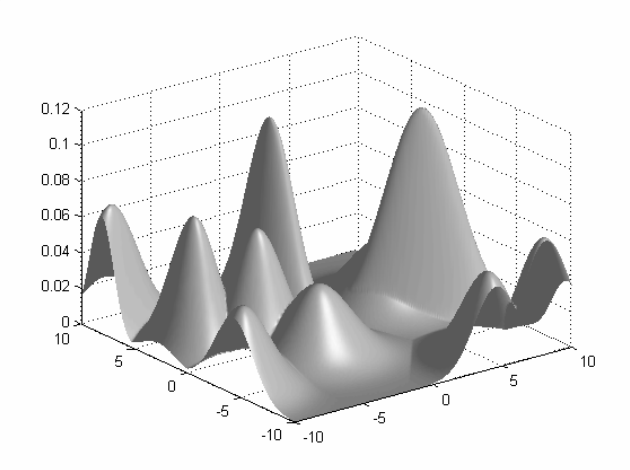

Figure 1: A sample fitness landscape

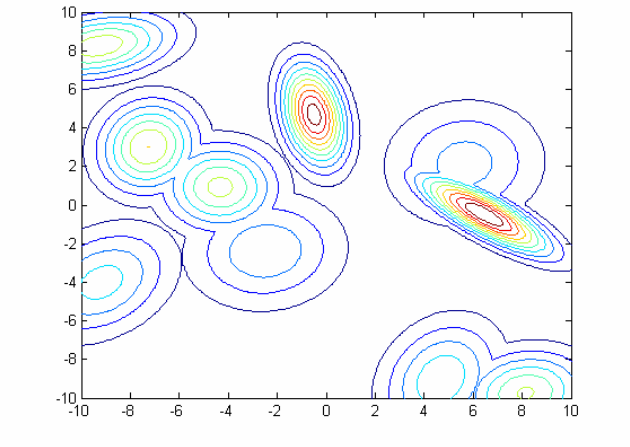

Figure 2: The contour of the fitness landscape

One thing we should point out is that this kind of rotation of coordinate system does not change the micro-structure of the landscape (e.g., the shape of each GF). After rotation, each GF is still a GF but with additional dependences among variables. However, since each GF is rotated independently, the whole landscape will be changed because the spatial relation among GFs will be changed. Another issue is that the main coordinate system will not be rotated. Instead, for each GF, we create a new coordinate system with the mean vector as its origin and rotate this new coordinate system. One advantage is that the position of each mean vector will be unchanged in the main coordinate system so that we know exactly the position of each optimum in the landscape after rotation. Another advantage is that if we rotate the main coordinate system, the global optimum may be left outside the search space and it may be difficult to locate the new global optimum in the new search space.

3.3 Extensions

3.3.1 Scalable GF

One potential shortcoming of the landscape generator discussed so far is that since it is based on GFs, there is some inherent restriction with regard to the shape of each GF. Since the volume under a GF is equal to 1, peaks in the landscapes generated with relatively large basin sizes will have low fitness values and vice versa.

In order to overcome this disadvantage, we may scale up or down each GF by a predefined or random scalar. By doing so, we can generate many more different landscapes and have more control on the properties of landscapes.

3.3.2 Noise

The landscapes of most real-world problems are unlikely to be completely smooth. Furthermore, due to the influence of some random factors, it is possible that we may get some different results when we try to evaluate the same set of parameters at different times. A simple method to simulate this property is to add some Gaussian noise on the landscapes generated to make them rugged. This kind of noise is stochastic in that each time a point is evaluated, a different value may be returned. Certainly, we may also apply some deterministic noise, which actually combines with the original fitness function and generates a new deterministic function. However, when applying noise, we need to be careful because this may change the global optimum if the difference between the global optimum and local optima is not large enough compared to the magnitude of the noise.

4 Experiments

4.1 Motivation and Purpose

As mentioned before, a “space of experiments” can be defined across the given algorithms and problems of interest, with each algorithm or problem parameter being a variable in this experimental space. This space is quite complex in that the total number of parameters of algorithms and problems is very large and the parameters may be of different types (e.g., binary, integer or real-valued). An exhaustive investigation of this space is infeasible. Here, we do not present extensive experiments to draw any general conclusion with regard to the optimization algorithms under testing. Instead, we focused on the landscape generator itself and the motivation of our experiments was twofold. Firstly, we wanted to demonstrate that our landscape generator is capable of generating different landscapes in terms of problem difficulty. Secondly, we wanted to show that our landscape generator can differentiate between algorithms.

4.2 Methodology

4.2.1 Specification of Landscape Generator

We fixed the dimensionality to 2 and only explicitly changed the number of GFs in each set of experiments. Other parameters were randomly chosen. The range of rotation angles was set to [-PI/4, PI/4] and the search space was bounded to [-10, 10] in each dimension while the range of variance values was set to [0.25, 5.25] in order to avoid very sharp peaks and many flat areas. For simplicity, we did not scale the GFs and did not apply noise.

4.2.2 Optimization Algorithms

We used two algorithms: a real-valued GA and a continuous PBIL[12]. For the PBIL, we simply adopted some common parameter values: learning rate (α) = 0.05; fixed standard deviation = 1.0; population size = 50. In each generation, the mean vector of the PBIL was updated by a combination of the best, the second best and the worst individuals in the population:

$$X^{t+1} = (1-\alpha)\cdot X^{t} + \alpha\cdot(X^{best,1} + X^{best,2} - X^{worst}) \quad (11)$$

The real-valued GA used a traditional one-point crossover, which exchanges genes after that point on two parents. For efficiency, we applied Tournament Selection in this GA. The mutation operator was based on a Gaussian with mean 0 and standard deviation 1.0. Crossover rate and mutation rate were set to 0.8 and 0.5 respectively. For the convenience of comparison, the population size was set to 50. The GA also used elitism to keep the current best individual to the next generation.

We are aware that different parameter settings may have more or less influence on experiment results. Thus, it is not reasonable to claim that one algorithm outperforms another one in general based on results obtained with a single instance of chosen parameter values. However, if we regard each algorithm instance as having a fully specified set of parameter values, we can analyze the performance of these concrete instances.

4.2.3 Performance Criteria

We used the fitness value of the overall best individual found to evaluate the performance. Since different landscapes may have different global optima in terms of fitness value, in order to facilitate the comparison across landscapes, each raw fitness value was divided by the value of the corresponding global optimum, producing a normalized value between 0 and 1 (i.e., all fitness values are positive).

Another issue is when to stop the evolution. Some algorithms may achieve rapid improvement in the beginning stage and then flatten out while others may work in other ways. Furthermore, if EAs are allowed to run for a very long time, we can expect that they would perform (approximately) equally well because of their global searching ability. Since our major purpose was to demonstrate the performance difference between these two algorithms and the performance difference across different landscapes, we set the number of generations in each trial to 100, which allowed 5,000 function evaluations per trial. The first reason was, according to some preliminary experiments, both of these two algorithms often experienced a long stable period without any significant improvement after 100 generations. The second reason was, if we allowed them to run longer, say, 10,000 function evaluations, random searching or brute force methods may also generate good results. As a result, it may not make much sense to compare two algorithms under this situation.

4.3 Experiment Results

We used three sets of landscapes with 1 GF, 10 GFs and 100 GFs respectively. These sets represented landscapes with one optimum, a small number of local optima and a large number of local optima (i.e., the number of GFs is the upper boundary of the number of optima). Each set contained 20 randomly generated landscapes. Each algorithm was run on each landscape for 50 trials in order to have a good understanding of its performance. We presented each algorithm's performance on each set of landscapes using box-plots with each box showing the performance distribution of an algorithm on a concrete landscape over 50 trials. The box has lines at the lower quartile, median and upper quartile values. There are also lines extending from both ends of the box to show the extent of the rest of the data within 1.5 times of the interquartile range. Outliers are marked with '+'.

In each of the following figures, the horizontal axis represents landscapes while the vertical axis stands for the normalized fitness values of the best individual at the end of evolution. For the PBIL, it represents the best individual found so far because the best individual may not be in the current population. In order to have a better comparison between these two algorithms, we sorted the landscapes based on the performance of the PBIL (i.e., the median of each set of results). The sequence of landscapes in each GA plot is the same as that in its corresponding PBIL plot.

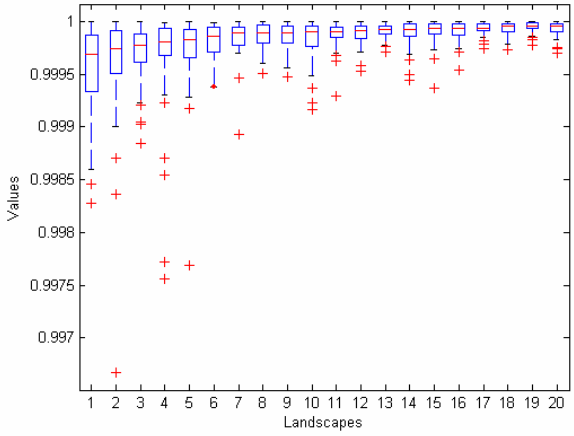

Figure 3: Performance of PBIL with 1 GF

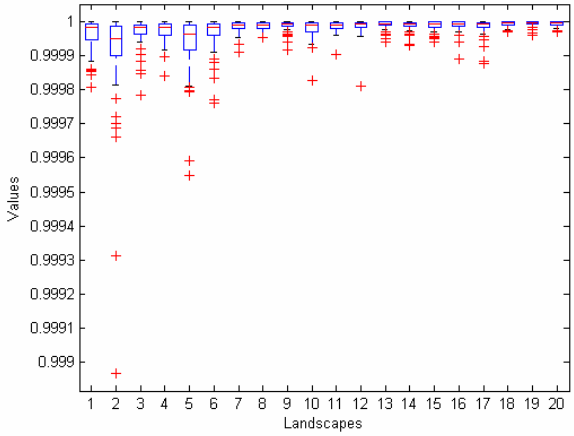

Figure 4: Performance of GA with 1 GF

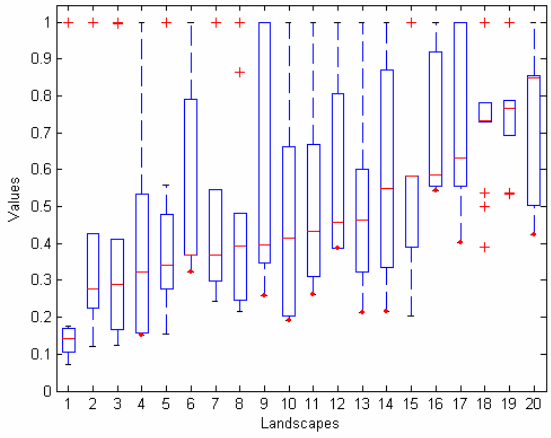

Figure 5: Performance of PBIL with 10 GFs

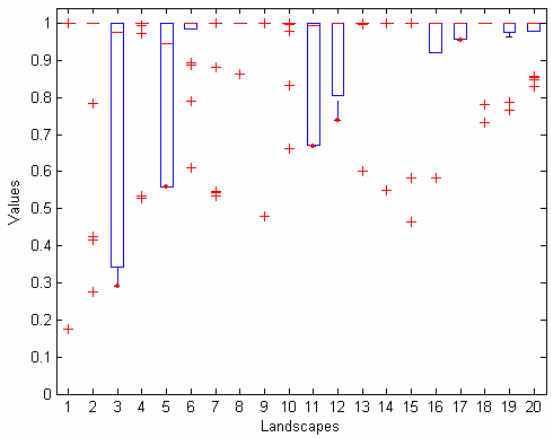

Figure 6: Performance of GA with 10 GFs

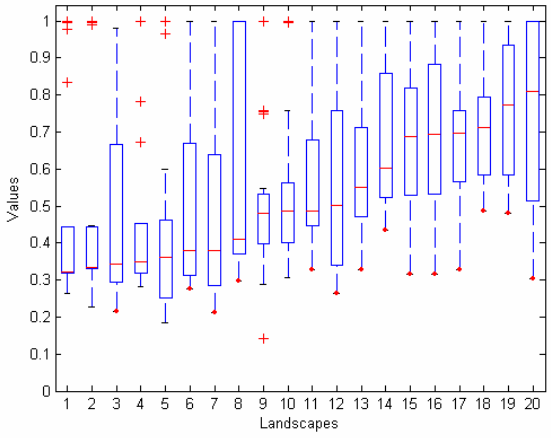

Figure 7: Performance of PBIL with 100 GFs

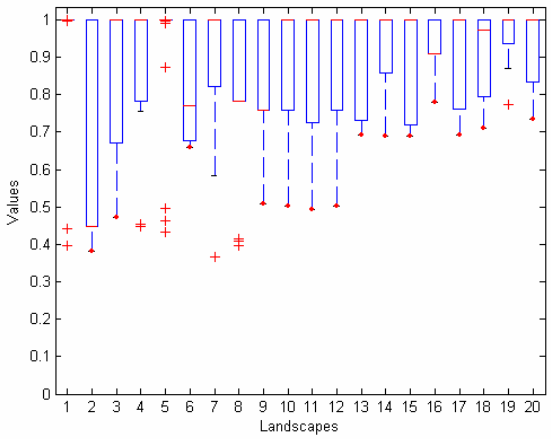

Figure 8: Performance of GA with 100 GFs

Figures 3&4 show the performance of these two algorithms on landscapes with only one GF. Unsurprisingly, both of them worked very well.

From Figures 5&6 it is clear that the increasing number of GFs did have some influence on the performance of these two algorithms. Sometimes, they got stuck somewhere far from the global optimum in terms of fitness value. Furthermore, the GA still maintained high success rate on most of these landscapes. By contrast, many landscapes were difficult for the PBIL in which only very few trails found the global optimum or found an individual comparable to it.

This contrast still held in results of experiments on landscapes with 100 GFs (Figures 7&8). Note that in a landscape with a large number of GFs, it is very likely that quite a few local optima are comparable to the global optimum in terms of fitness value. This may explain why there was no further significant decreasing in the performance of the PBIL (i.e., Figure 5 vs. Figure 7).

Now, it is clear that, as the number of GFs increases, which corresponds to increasing multimodality of landscapes, the performance of these two algorithms will all decrease. However, in our experiments, we found that the PBIL was more sensitive to the multimodality of landscapes than the GA. It suffered a lot in multimodal landscapes while the GA was relatively robust. This may be partially due to the Gaussian model it employs.

4.4 Discussion

The experimental work above is limited and far from an extensive comparison of two EAs. For example, we only considered the fitness value but did not examine how close the best individual was to the global optimum in each trial. We know that in some landscapes, it may be quite difficult to locate the global optimum but it may not be very hard to find an acceptable solution because of the existence of several local optima of high quality. Another issue is that we evaluated each algorithm after 100 generations but we did not investigate each algorithm's online performance in each generation.

However, our current interest is not to show which algorithm is superior to another one. Instead, we hope that these initial experiments can serve as a good demonstration of the usefulness of our landscape generator. In the above experiments, we have seen that by changing one of the properties of the landscape generator (i.e., the number of GFs), we could systematically generate a large number of landscapes with different levels of difficulty. In the mean time, we can see that different algorithms performed differently on these landscapes, which provides a basis for empirically comparing these algorithms in the future. Furthermore, since the structure of each landscape was completely parameterized, it is also possible for us to relate the performance of an algorithm to

the specific properties (e.g., the basin size of each local optimum, the distribution of local optima and the quality of local optima) of the landscape. By doing so, we can expect to have a deep understanding towards the interaction between algorithms and landscapes and it may also help us investigate when and why an algorithm may work well.

5 Conclusion

The major motivation of this paper was to investigate how to conduct systematic, empirical evaluation and testing for optimization algorithms. The ultimate goal is to help people understand the difference among various algorithms and provide some useful indication on choosing the right algorithm for an unknown problem. A landscape generator based on a sum of Gaussian functions was proposed to aid in the conducting of controlled experiments for continuous, unconstrained problems. Furthermore, we demonstrated the usefulness of this landscape generator through some experimental study of two EAs: a real-valued GA and a continuous PBIL. By systematically varying the properties of the landscape, we could not only have a whole image about the performance of an algorithm on a large number of random problems but also understand how a specific property of the landscape could influence its performance.

Compared to traditional methods based on canonical test problems, this landscape generator can give us direct control on the structure of the problems. By contrast, classical test problems are often regarded as black-box problems, which can only tell us whether an algorithm works well on them but do not give us much insight into the internal mechanism and behavior of the algorithm. This advantage is especially significant in research where people often wish to know exactly when and why an algorithm will be successful or fail. In practice, although problems generated by the landscape generator cannot completely approximate real-world problems, if we have good knowledge about on what kind of landscapes an algorithm can be expected to work well, it would still help us choose the right algorithm for a concrete problem.

A direct extension of the current work is to conduct more extensive experiments to fully investigate the performance of an algorithm. However, the number of experiments required may be huge if one wants to change parameters of the algorithm and parameters of the landscape generator simultaneously. If we regard each of these parameters as a variable and the performance of the algorithm as another variable, we would actually get a performance landscape of high dimensionality. A greedy exploration of this landscape can be computationally prohibitive and some statistical methods may be needed to estimate the structure of this landscape.

Acknowledgement

This work was partially supported by the Australian Postgraduate Award granted to Bo Yuan.

References

[1] De Jong, K.A., Potter, M.A., and Spears, W.M. "Using Problem Generators to Explore the Effects of Epistasis". In *Seventh International Conference on Genetic Algorithms*, T. Bäck Ed., Morgan Kauffman, pp. 338-345, 1997.

[2] Eiben, A.E. and Jelasity, M. "A Critical Note on Experimental Research Methodology in EC". In *Congress on Evolutionary Computation*, Honolulu, Hawaii, IEEE, pp. 582-587, 2002.

[3] Whitley, D., Mathias, K., Rana, S. and Dzubera, J. "Evaluating Evolutionary Algorithms", *Artificial Intelligence*, **85**(1-2): pp. 245-276, 1996.

[4] Johnson, D.S. "A Theoretician's Guide to the Experimental Analysis of Algorithms". In *5th and 6th DIMACS Implementation Challenges*, Goldwasser, et al. Eds., 2002.

[5] Hooker, J.N. "Testing heuristics: We have it all wrong", *Journal of Heuristics*, **1**(1): pp. 33-42, 1995.

[6] Wolpert, D.H. and Macready, W.G. "No Free Lunch Theorems for Optimization", *IEEE Transactions on Evolutionary Computation*, **1**(1): pp. 67-82, 1997.

[7] Kennedy, J. "Continuous Valued Multimodality Generator for Evolutionary Algorithms", 1997. Retrieved from: www.cs.uwyo.edu/~wspears/multi.kennedy.html (25 April 2003)

[8] Kennedy, J. and Spears, W.M. "Matching Algorithms to Problems: An Experimental Test of the Particle Swarm and Some Genetic Algorithms on the Multimodal Problem Generator". In *International Conference on Evolutionary Computation*, IEEE, pp. 78-83, 1998.

[9] Michalewicz, Z., Deb, K., Schmidt, M. and Stidsen, T. "Test-case Generator for Nonlinear Continuous Parameter Optimization Techniques", *IEEE Transactions on Evolutionary Computation*, **4**(3): pp. 197-215, 2000.

[10] Morrison, R.W. and De Jong, K.A. "A Test Problem Generator for Non-Stationary Environments". In *Congress on Evolutionary Computation*, IEEE, pp. 2047 - 2053, 1999.

[11] Baluja, S. and Caruana, R. "Removing the Genetics from the Standard Genetic Algorithm", Tech. Report CMU-CS-95-141, Carnegie Mellon University, 1995.

[12] Sebag, M. and Ducoulombier, A. "Extending Population-Based Incremental Learning to Continuous Search Spaces". In *Parallel Problem Solving from Nature-PPSN V*, A.E. Eiben, et al. Eds., Amsterdam, Springer, pp. 418-427, 1998.

[13] Jones, T. "Evolutionary Algorithms, Fitness Landscapes and Search", PhD Thesis, University of New Mexico, 1995.

[14] Holland, J.H. "Building blocks, Cohort Genetic Algorithms, and Hyperplane-defined Functions", *Evolutionary Computation*, **8**(4): pp. 373-391, 2000.

[15] Gallagher, M. "Fitness Distance Correlation of Neural Network Error Surfaces: A Scalable, Continuous Optimization Problem". In *European Conference on Machine Learning (ECML 2001)*, L. De Raedt, et al. Eds., pp. 157-166, 2001.

[16] Bishop, C.M. *Neural Networks for Pattern Recognition*, Oxford University Press, 1995.

[17] Salomon, R. "Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions. A survey of some theoretical and practical aspects of genetic algorithms", *BioSystems*, **39**: pp. 263-278, 1996.