Chapter 1
Performance Analysis of Dynamic Optimization Algorithms

Amir Nakib and Patrick Siarry

Abstract. In recent years dynamic optimization problems have attracted a growing interest from the community of stochastic optimization researchers with several approaches developed to address these problems. The goal of this chapter is to present the different tools and benchmarks to evaluate the performances of the proposed algorithms. Indeed, testing and comparing the performances of a new algorithm to the different competing algorithms is an important and hard step in the development process. The existence of benchmarks facilitates this step, however, the success of these benchmarks is conditioned by their use by the community. In this chapter, we cite many tested problems (we focused only on the continuous case), and we only present the most used: the moving peaks benchmark , and the last proposed: the generalized approach to construct benchmark problems for dynamic optimization (also called benchmark GDBG).

1.1 Introduction

The dynamic optimization problems (DOPs) can be met in many real-world cases. A dynamic optimization problem can be formulated as follows:

$$\begin{aligned} &\min \ f(\mathbf{x},t) \\ &\text{s.t.} \ \ h_j(\mathbf{x},t) = 0 \text{ for } j = 1,2,...,u \\ &\qquad g_k(\mathbf{x},t) \leq 0 \text{ for } k = 1,2,...,v, \end{aligned} \tag{1.1}$$

where $f(\mathbf{x},t)$ is the objective function of a minimization problem, $h_j(\mathbf{x},t)$ denotes the j^{th} equality constraint, and $g_k(\mathbf{x},t)$ denotes the k^{th} inequality constraint.

Amir Nakib · Patrick Siarry
Université Paris Est Créteil, Laboratoire Images,
Signaux et Systèmes Intelligents (LISSI, EA 3956)
61, avenue du Général de Gaulle 94010 Créteil, France
e-mail: `nakib@u-pec.fr`

The function f is deterministic at any point in time, but is dependent on time t. Consequently the position of the optimum changes over the time. Thus, the algorithm dealing with DOPs should be able to continuously track this optimum rather than requiring a repeated restart of the optimization process. The algorithms dedicated to solve this kind of problem take into account, during the optimization process, the information from previous environments to speed up optimization after a change.

The naive approach after a change of the environment consists in formulating each change as the arrival of a new optimization problem that has to be solved from scratch. Indeed, if there is enough time to solve the problem, this is an efficient way. However, the time for restarting the optimization process is, in most cases, short, and it is based on the assumption of the identification of changing events, which is not always the case.

To increase the convergence speed after a change, one direction consists in finding the best way to use the previous known information about the search space to speed up the search after a detected change. For example, if the new optimum is *close* to the old one, one can reduce the search space to the neighborhood of the previous optimum. Of course, taking into account the information from the past depends on the nature of the change. If the change is drastic, and the new problem has little similarity to the previous problem, restart may be the only viable option, and use of information from the past would be misguiding rather than helping the search.

In most real-world problems, however, the changes are smooth and using knowledge from the past would be a good way to speed up the optimization. The difficult question is: *what information should be kept, and how it is used to accelerate search after the environment has changed?*

However, the information transfer does not guarantee that the optimization algorithm is flexible enough to follow the optimum over the changes. Most Dynamic Optimization Algorithms (DOAs) converge during the run, at least when the environment has been static for some time, without losing their adaptability. Thus, besides transferring knowledge, a successful DOA for dynamic optimization problems has to maintain adaptability.

In order to evaluate the performances of the dynamic optimization algorithms, many researchers have applied several dynamic test problems to them. The most known are: the *moving peaks* benchmark (MPB) proposed by Branke [3], the DF1 generator introduced by Morrison and De Jong [17], the single- and multiobjective dynamic test problem generator by dynamically combining different objective functions of existing stationary multiobjective benchmark problems, suggested by Jin and Sendhoff [6], Yang and Yao's exclusive-or (XOR) operator [22][24][25], Kang's dynamic traveling salesman problem (DTSP) [10], and dynamic multi knapsack problem (DKP), etc.

The goal of this chapter is to present the different methods to evaluate the performances of a DOA. In the following section, the performance analysis of optimization algorithms methods is presented. In section 3, the Moving Peak benchmark is summarized. In section 4, we present the generalized benchmark. In section 5, the dynamic multiobjective optimization benchmark is presented. The conclusion is presented in section 6.

1.2 Performance Analysis Tools of Optimization Algorithms

In this section, we present some metrics and tools that can be used to analyze the performance of an optimization algorithm. Here, we do not focus only on dynamic optimization but we present a set of tools to evaluate any optimization algorithm.

1.2.1 Fitness Value

The fitness of a solution is a numerical value that provides an indication on how well the solution meets the objective(s) of the problem. The concept of fitness is central to nature inspired algorithms (metaheuristics). The concept of fitness is applied in most metaheuristics. In the case of benchmark problems such as the Sphere function, the mathematic formulation and the location of the global optimum are known. In such cases the fitness function corresponds to the distance to the global optimum. The best fitness is known and is often zero. A solution that is closer to the global optimum has a smaller error and a best fitness than a solution farther away.

Another kind of problem that can be met is when the global optimum is unknown. It may not even be known whether or not a global optimum exists, and, if it does, whether there are multiple global optima. Most examples of this type of problem are NP-hard and the fitness score is a function of the system output(s). Furthermore, the fitness score may be a weighted function of output parameters. An example is a pipe renewal problem in drinking water networks, where the numbers and types of pipes at hand, the provided pressure at demand points, and the cost may all be weighted and incorporated into fitness values [4].

It is known that the three spaces of adaptation of an algorithm are: decision variable space, system output space, and fitness space. System output space is the space defined by the dynamic range(s) of the output variable(s). The fitness space is the space used to define the *goodness* of the solutions in the output space. It is recommanded to scale the fitness to values between 0 and 1, where 0 or 1 is the optimal value, depending on the optimization problem (minimization or maximization). Thus, system output and fitness generally do not coincide.

Furthermore, the numerical value of the fitness rarely has a meaning. In most cases, we only use fitness values to rank solutions. A proposed solution with a fitness value of 0.950 is rarely exactly twice as good as a solution with a fitness value of 0.760. We simply have a rank-ordered list of how good a solution is relatively to other solutions.

It is a common practice to vary parameters of the algorithm such as population size and attempt to see what value produces a better cost. For example, run the metaheuristic fifty times with one population size and fifty times with another population size. Due to the stochastic nature of the algorithm, we may very well get a different fitness value each time.

How do we determine which solution is better? If all of the fitness values for one population size are better than those for another population size, the situation is clear: use the solution that consistently produces the best fitness values. However, the situation is not always so simple. Especially, when we are fine-tuning parameters to maximize system performance, we can meet situations that are difficult to analyze and interpret.

1.2.2 Computational Analysis

The computational analysis of an optimization algorithm can be provided using a theoretical analysis or an empirical one. In the first case, the worst-case complexity of the algorithm is computed. Usually, the asymptotic complexity is not enough to represent the computational performances of metaheuristics. If the probability distribution of the input instances is available, then average-case complexity is recommended and it is more practical.

To perform empirical analysis, different measures of the computation time of the metaheuristic used to solve a given instance are presented. The computation time corresponds to the CPU time or wall clock time, with or without input/output and preprocessing/postprocessing time.

The main drawback of computation time measure is its dependency on the used hardware (e.g., processor, GPUs, memories, ...), operating systems, language, and compilers on which the metaheuristic is executed. Some indicators that are independent of the computer system may also be used, such as the number of objective function evaluations. It is an acceptable measure for time-intensive and constant objective functions. Using this metric may be problematic for problems where the evaluation cost is low compared to the rest of the metaheuristics or is not time constant since it depends on the solution evaluated and time. This appears in some applications with variable length representations (genetic programming, robotics, etc.) and dynamic optimization problems.

Different stopping criteria may be used: time to obtain a given target solution, time to obtain a solution within a given percentage from a given solution (e.g., global optimal, lower bound, best known), and number of iterations [21].

1.2.3 Classical Metrics

Two classical metrics for the effectiveness of metaheuristics were described by De Jong (1975). These metrics, however, are appropriate for only the algorithms that evolve a population of solutions. De Jong named these metrics off-line performance and on-line performance.

When an optimization algorithm is being run off-line, many system configurations can be evaluated (the fitness calculated) and the best configuration selected.

For on-line work, however, configurations must be evaluated in real time, therefore the usual goal is to develop an acceptable solution as early as possible.

The on-line performance, which measures the ongoing performance of a system configuration, is defined in Equation 1.2, where $\bar{f}_c(g)$ is the average population fitness for a system configuration c during generation g and G is the number (index) of the latest generation:

$$S_c^{\text{Online}} = \frac{1}{G} \sum_{g=1}^{G} \bar{f}_c(g), \tag{1.2}$$

The off-line performance measures convergence of the algorithm and is defined in Equation 1.3, where $f^*_{c(g)}$ is the best fitness of any population member in generation g for system configuration c. Off-line (convergence) performance is thus the average of the best fitness values from each generation up to the present.

$$S_c^{\text{Offline}} = \frac{1}{G} \sum_{g=1}^{G} \bar{f}^*_{c(g)}, \tag{1.3}$$

1.2.4 Sensitivity Analysis

The definition of the sensitivity of the optimization algorithms, also called robustness, is not standardized inside the community. Different alternative definitions were proposed for sensitivity. In general, it corresponds to insensitivity against small deviations in the input instances (data) or the parameters of the algorithm. The lower the variability of the obtained solutions, the better the sensitivity. Sensitivity analysis that is related to the applications of optimization algorithms sometimes focuses on the problem and/or solution domain.

The parameters of the metaheuristics play an important role in their search capacity. Indeed, the sensitivity of a metaheuristic with respect to its parameters is critical to its performance and, therefore, its successful applications. Using this approach, we consider the parameters of an optimization algorithm as the input values to the sensitivity analysis, and its performance values as the output values. Many indicators can be used to evaluate the sensitivity; here we propose:

- *Parameter sensitivity*: it measures the effect of a given parameter on a given output when all other parameters are constrained to be constant.
- *Performance sensitivity*: it takes into account the fact that the effect of a given parameter on a given output differs with varying the data set values. This indicator measures the average effect of a given parameter on a given output over a set of data.

For metaheuristics algorithms, the output values can include parameters such as fitness value, convergence rate, and the maximum generation required to reach a good enough solution. The input values may be different for different algorithms. For example, for genetic algorithms, the input values can be mutation rate, crossover rate, population size, and so on. For particle swarm optimization algorithms, the input values can be inertia weight w, cognitive and social coefficients c_1 and c_2, and so on.

1.2.5 Statistical Analysis

Different statistical tests may be carried out to analyze and compare the metaheuristics. The statistical tests are performed to estimate the confidence of the results to be scientifically valid. The selection of a given statistical hypothesis testing tool is performed according to the characteristics of the data.

Under some assumptions (normal distributions), the most widely used test is the paired *t-test*. Otherwise, a nonparametric analysis may be realized, such as the Wilcoxon test and the permutation test. For a comparison of more than two algorithms, ANOVA models are well-established techniques to check the confidence of the results. Multivariate ANOVA models allow simultaneous analysis of various performance measures (e.g., both the quality of solutions and the computation time). Kolmogorov-Smirnov test can be performed to check whether the obtained results follow a normal (Gaussian) distribution. Moreover, the Levene test can be used to test the homogeneity of the variances for each pair of samples. The Mann-Whitney statistical test can be used to compare two optimization methods. According to a p-value and a metric under consideration, this statistical test reveals if the sample of approximation sets obtained by a search method S_1 is significantly better than the sample of approximation sets obtained by a search method S_2, or if there is no significant difference between both optimization methods.

These different statistical analysis procedures must be adapted for nondeterministic (or stochastic) algorithms. Indeed, most metaheuristics belong to this class of algorithms. Many trials (100 runs is the most used number) must be carried out to derive significant statistical results. From this set of trials, many measures may be computed: mean, median, minimum, maximum, standard deviation, the success rate that the reference solution (e.g., global optimum, best known, given goal) has been attained, and so on.

Below are some considerations about the use of statistical tools for analysing the performances of metaheuristics or stochastic-based optimization algorithms:

- *t-tests* require that certain assumptions be made regarding the format of the data. The one sample t-test requires that the data have an approximately normal distribution, whereas the paired t-test requires that the distribution of the differences is approximately normal. The unpaired t-test relies on the assumption that the data from the two samples are both normally distributed, and has the additional requirement that the standard deviations (SDs) from the two samples are approximately equal.

  Formal statistical tests are performed to examine whether a set of data are normal or whether two SDs (or, equivalently, two variances) are equal, although results from these should always be interpreted in the context of the sample size and associated statistical power in the usual way. However, the t-test is known to be robust to modest departures from these assumptions, and so a more informal investigation of the data may often be sufficient in practice.

If assumptions of normality are violated, then appropriate transformation of the data may be used before performing any calculations. Similarly, transformations may also be useful if the SDs are very different in the unpaired case. Finally, these methods are restricted to the case where comparison has to be made between one or two groups. This is probably the most common situation in practice but it is by no means uncommon to want to explore differences through three or more methods. This requires an alternative approach that is known as analysis of variance (ANOVA).

- *The nonparametric tests* require very few or very limited assumptions to be made about the format of the data, and can therefore be used in situations where classical methods, such as t-tests, may be inappropriate. They can be useful for dealing with unexpected, outlying observations that might be problematic with a parametric approach. Moreover, these methods are intuitive and are simple to carry out by hand, for small samples at least. Indeed, nonparametric methods are often useful in the analysis of ordered categorical data in which assignation of scores to individual categories may be inappropriate. In contrast, parametric methods require scores to be assigned to each category, with the implicit assumption that the effect of moving from one category to the next is fixed.

  However, nonparametric methods may lack power as compared with more traditional approaches. This is of particular concern if the sample size is small or if the assumptions for the corresponding parametric method (e.g. normality of the data) hold. Moreover, these methods are geared toward hypothesis testing rather than estimation of effects. It is often possible to obtain nonparametric estimates and associated confidence intervals, but this is not generally simple. In many cases an adjustment to the statistic test may be necessary.
- The Kruskal-Wallis, Jonckheere-Terpstra, and Friedman tests can be used to test for differences between more than two groups or treatments when the assumptions for analysis of variance are not held.
- *The P-value* is the probability that an observed effect is simply due to chance; it therefore provides a measure of the strength of an association. Moreover, it does not provide any measure of the size of an effect and cannot be used in isolation to inform about the best optimization algorithm. Indeed, P-values are affected both by the magnitude of the effect and by the size of the study from which they are derived, and should therefore be interpreted with caution. In particular, a large P-value does not always indicate that there is no difference and, similarly, a small P-value does not necessarily signify a high difference. The subdivision P-values into *significant* and *non-significant* are poor statistical practice and should be avoided. Finally, exact P-values should always be presented, along with estimates of effect and associated confidence intervals.

The success rate: The success rate (*SR*) represents the ratio between the number of successful runs and the number of trials:

Table 1.1 MPB parameters in scenario 2.

Parameter	**Scenario 2**
Number of peaks N_p	10
Dimension d	5
Peak heights	$[30, 70]$
Peak widths	$[1, 12]$
Change cycle α	5000
Change severity s	1
Height severity	7
Width severity	1
Correlation coefficient λ	0
Number of changes N_c	100

$$SR = \frac{\text{NbSuc}}{\text{NbR}}, \tag{1.4}$$

where *NbSuc* is the number of successful runs and *NbR* is the total number of runs.

The performance rate: The performance rate (*PR*) takes into account the computational effort to find the solution by considering the number of objective function evaluations:

$$PR = \frac{\text{NbSuc}}{\text{NbR} \times \text{Nbfeval}}, \tag{1.5}$$

where $Nbfeval$ is the total number of evaluations of the objective function.

1.3 The Moving Peaks Benchmark

The most commonly used benchmark for continuous dynamic optimization is the Moving Peaks Benchmark (MPB) [3].

MPB is a maximization problem that consists of a number of peaks that randomly vary their shape, position, and height upon time. At any time, one of the local optima can become the new global optimum. MPB generates DOPs consisting of a set of peaks that periodically move in a random direction, by a fixed amount s (the change severity). The movements are autocorrelated by a coefficient λ, $0 \leq \lambda \leq 1$, where 0 means uncorrelated and 1 means highly autocorrelated. The peaks change position every α evaluations, and α is called *time span*. The fitness function used for the landscape of MPB is formulated in Equation 1.6:

$$f(\mathbf{x},t) = max_{i=1,\dots,N_p}\left(H_i(t) - W_i(t)\sqrt{\sum_{j=1}^{d}\left(x_j - X_{ij}(t)\right)^2}\right), \tag{1.6}$$

where N_p is the number of peaks, d is the number of dimensions, and $H_i(t)$, $W_i(t)$ and $\mathbf{X}_i(t)$ are the height, the width, and the position of the i^{th} peak at the time t, respectively.

In order to evaluate the performance, the *offline error* is used. The offline error (*oe*) is defined in Equation 1.7:

$$oe = \frac{1}{N_c} \sum_{j=1}^{N_c} \left(\frac{1}{N_e(j)} \sum_{i=1}^{N_e(j)} \left(f_j^* - f_{ji}^* \right) \right) \tag{1.7}$$

where N_c is the total number of fitness landscape changes within a single experiment, $N_e(j)$ is the number of evaluations performed for the j^{th} state of the landscape, f_j^* is the value of the optimal solution for the j^{th} landscape, and f_{ji}^* is the current best fitness value found for the j^{th} landscape. We can see that this measure has some weaknesses: it is sensitive to the overall height of the landscape, and to the number of peaks. It is important for an algorithm to find the global optimum quickly, thus minimizing the offline error. Hence, the most successful strategy is a multi-solution approach that keeps track of every local peak [19]. In [3], three sets of parameters, called scenarios, were proposed. It appears that the most commonly used set of parameters for MPB is scenario 2 (see Table 10.3).

Figure 10.1 illustrates an MPB landscape before and after a change (after one time span). More details about this benchmark will be given in the dedicated chapter.

Table 1.2 Comparison with competing algorithms on MPB using $s = 1, 2, ..., 6$.

Algorithm	**Offline error**					
	$s=1$	$s=2$	$s=3$	$s=4$	$s=5$	$s=6$
Moser and Chiong, 2010 [18]	0.25 ± 0.08	0.47 ± 0.12	0.49 ± 0.12	0.53 ± 0.13	0.65 ± 0.19	0.77 ± 0.24
Lepagnot *et al.* [9]	0.35 ± 0.06	0.60 ± 0.08	0.91 ± 0.10	1.23 ± 0.10	1.62 ± 0.13	2.00 ± 0.20
Lepagnot *et al.*, 2009 [7, 8]	0.59 ± 0.10	0.87 ± 0.12	1.18 ± 0.13	1.49 ± 0.13	1.86 ± 0.17	2.32 ± 0.18
Moser and Hendtlass, 2007 [18, 19]	0.66 ± 0.20	0.86 ± 0.21	0.94 ± 0.22	0.97 ± 0.21	1.05 ± 0.21	1.09 ± 0.22
Yang and Li, 2010 [23]	1.06 ± 0.24	1.17 ± 0.22	1.36 ± 0.28	1.38 ± 0.29	1.58 ± 0.32	1.53 ± 0.29
Liu *et al.*, 2010 [14]	1.31 ± 0.06	1.98 ± 0.06	2.21 ± 0.06	2.61 ± 0.11	3.20 ± 0.13	3.93 ± 0.14
Lung and Dumitrescu, 2007 [15]	1.38 ± 0.02	1.78 ± 0.02	2.03 ± 0.03	2.23 ± 0.05	2.52 ± 0.06	2.74 ± 0.10
Bird and Li, 2007 [1]	1.50 ± 0.08	1.87 ± 0.05	2.40 ± 0.08	2.90 ± 0.08	3.25 ± 0.09	3.86 ± 0.11
Lung and Dumitrescu, 2008 [16]	1.53 ± 0.01	1.57 ± 0.01	1.67 ± 0.01	1.72 ± 0.03	1.78 ± 0.06	1.79 ± 0.03
Blackwell and Branke, 2006 [2]	1.75 ± 0.06	2.40 ± 0.06	3.00 ± 0.06	3.59 ± 0.10	4.24 ± 0.10	4.79 ± 0.10
Li *et al.*, 2006 [13]	1.93 ± 0.08	2.25 ± 0.09	2.74 ± 0.09	3.05 ± 0.10	3.24 ± 0.11	4.95 ± 0.13
Parrott and Li, 2006 [20]	2.51 ± 0.09	3.78 ± 0.09	4.96 ± 0.12	5.56 ± 0.13	6.76 ± 0.15	7.68 ± 0.16

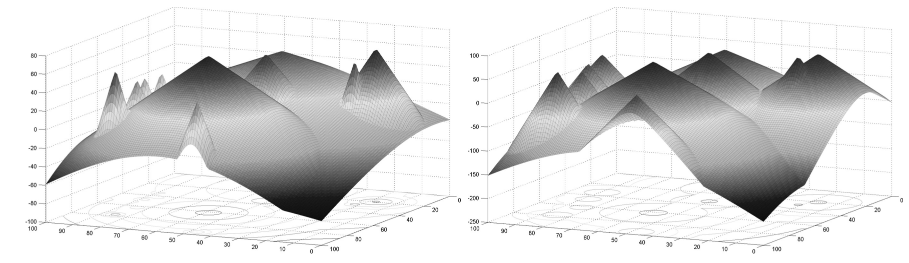

Fig. 1.1 An MPB landscape before and after a change.

Table 1.3 Static functions used to generate the GDBG problems.

Name	Function	Range
Sphere	$f(\mathbf{x}) = \Sigma_{i=1}^{d} x_i^2$	$[-100,100]^d$
Rastrigin	$f(\mathbf{x}) = \Sigma_{i=1}^{d} (x_i^2 - 10\cos(2\pi x_i) + 10)$	$[-5,5]^d$
Weierstrass	$f(\mathbf{x}) = \sum_{i=1}^{d} (\sum_{k=0}^{k_{max}} [a^k \cos(2\pi b^k (x_i + 0.5))]) - d \sum_{k=0}^{k_{max}} [a^k \cos(\pi b^k)]$ $a = 0.5, b = 3, k_{max} = 20$	$[-0.5,0.5]^d$
Griewank	$f(\mathbf{x}) = \frac{1}{4000} \Sigma_{i=1}^{d} (x_i)^2 - \Pi_{i=1}^{d} \cos(\frac{x_i}{\sqrt{i}}) + 1$	$[-100,100]^d$
Ackley	$f(\mathbf{x}) = -20\exp(-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}) - \exp(\frac{1}{d}\sum_{i=1}^{d}\cos(2\pi x_i)) + 20 + \exp(1)$	$[-32,32]^d$

Table 1.2 summarizes the last published scores of the dynamic optimization algorithms tested on MPB benchmark. Here, we will not comment on these results because it is out of the scope of this chapter.

1.4 The Generalized Dynamic Benchmark Generator

The Generalized Dynamic Benchmark Generator (GDBG) is the second benchmark described in this chapter, it was introduced in [11, 12]. It was provided for the CEC'2009 Special Session on Evolutionary Computation in Dynamic and Uncertain Environments. The functions used to create this benchmark are depicted in Table 10.2. These functions were rotated, composed, and combined to form six problems with different degrees of difficulty:

F_1: rotation peak function (with 10 and 50 peaks)
F_2: composition of Sphere's function
F_3: composition of Rastrigin's function
F_4: composition of Griewank's function
F_5: composition of Ackley's function
F_6: hybrid composition function

A total of seven dynamic scenarios with different degrees of difficulty was proposed:

T_1: small step change (a small displacement)
T_2: large step change (a large displacement)
T_3: random change (Gaussian displacement)
T_4: chaotic change (logistic function)
T_5: recurrent change (a periodic displacement)
T_6: recurrent with noise (the same as above, but the optimum never returns exactly to the same point)
T_7: changing the dimension of the problem

Table 1.4 GDBG parameters used during the CEC'2009 competition.

Parameter	Value
Dimension d (fixed)	10
Dimension d (changed)	$[5, 15]$
Change cycle α	$10000 \times d$
Number of changes N_c	60

The basic parameters of the benchmark are given in Table 1.4.

There are 49 test cases that correspond to the combinations of the six problems with the seven change scenarios (indeed, function F_1 is used twice, with 10 and 50 peaks, respectively). For each of them, the average best (Equation 1.8), average mean (Equation 1.9), average worst (Equation 1.10) values, and the standard deviation (Equation 1.11) of the absolute error are recorded:

$$Avg_{best} = \sum_{i=1}^{runs} \min_{j=1}^{N_c} \frac{E_{ij}}{runs} \tag{1.8}$$

$$Avg_{mean} = \sum_{i=1}^{runs} \sum_{j=1}^{N_c} \frac{E_{ij}}{runs \times N_c} \tag{1.9}$$

$$Avg_{worst} = \sum_{i=1}^{runs} \max_{j=1}^{N_c} \frac{E_{ij}}{runs} \tag{1.10}$$

$$STD = \sqrt{\frac{\sum_{i=1}^{runs} \sum_{j=1}^{N_c} (E_{ij} - Avg_{mean})^2}{runs \times N_c}}, \tag{1.11}$$

where $E_{ij} = \left| f_j^* - \tilde{f}_{ji}^* \right|$, f_j^* is the value of the global optimum for the j^{th} landscape, $\tilde{f}_{ji}^*$ is the value of the best solution found during the i^{th} run of the tested algorithm, for the j^{th} landscape, and *runs* is the number of runs of the tested algorithm on the benchmark, equal to 20 in our experiments.

The convergence graphs, showing the relative error $r_i(t)$ of the run with median performance for each problem, are also computed. For the maximization problem F_1, the formula used for $r_i(t)$ is defined in Equation 1.12, and for the minimization problems F_2 to F_6, it is defined in Equation 1.13:

$$r_i(t) = \frac{f_i(t)}{f_i^*(t)} \tag{1.12}$$

$$r_i(t) = \frac{f_i^*(t)}{f_i(t)}, \tag{1.13}$$

where $f_i(t)$ is the value of the best found solution at time t since the last occurrence of a change, during the i^{th} run of the tested algorithm, and $f_i^*(t)$ is the value of the global optimum at time t.

A mark is calculated for each test case using the formula given in Equation 1.14:

$$mark_{pct} = mark_{max} \times \sum_{i=1}^{runs} \sum_{j=1}^{N_c} \frac{r_{ij}}{runs \times N_c}, \tag{1.14}$$

The sum of all marks $mark_{pct}$ gives a score, denoted by op, that corresponds to the overall performance of the tested algorithm. The maximum value of this score is 100. In Equation 1.14, r_{ij} is calculated using the formula defined in Equation 1.15, and $mark_{max}$ is a coefficient that defines the percentage of the mark of each test case in the final score. It is also the maximal mark that can be obtained by the tested algorithm on each test case.

$$r_{ij} = \frac{r_i(t_j + \alpha)}{1 + \sum_{s=1}^{\frac{\alpha}{s_f}} \frac{1 - r_i(t_j + s_f \times s)}{\frac{\alpha}{s_f}}}, \tag{1.15}$$

where t_j is the time at which the j^{th} change occurred, s_f is the sampling frequency, here equal to 100, and α is the change cycle (as for MPB, it corresponds to the number of evaluations that makes a time span).

1.4.1 Illustration of Some Test Functions in GDBG

F_3: Composition of Rastrigin's function

Basic functions: $f_1 - f_{10}$ =Rastrigin's function

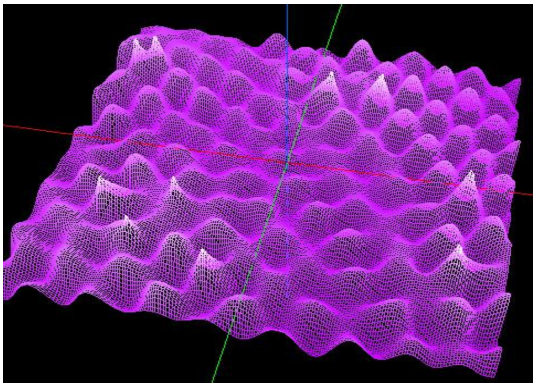

Fig. 1.2 3-D map for 2-D function of F_3.

Properties

Multi-modal
Scalable

Rotated
A huge number of local optima
$x \in [-5,5]^n$, Global optimum $x^*(t) = \mathbf{O_i}, F(x^*(t)) = H_i(t), H_i(t) = min_j^m H_j$

F_5:Composition of Ackley's function

Basic functions: $f_1 - f_{10}$ =Ackley's function

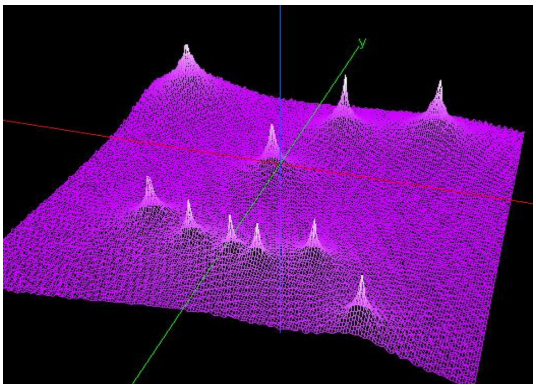

Fig. 1.3 3-D map for 2-D function of F_5.

Properties

Multi-modal
Scalable
Rotated
A huge number of local optima
$x \in [-5,5]^n$, Global optimum $x^*(t) = \mathbf{O_i}, F(x^*(t)) = H_i(t), H_i(t) = min_j^m H_j$

Figure 1.4 presents the scores of recent dynamic optimization algorithm on GDBG benchmark.

1.5 Dynamic Multiobjective Optimization Benchmark

In this section, the dynamic multiobjective optimization benchmark (DMOB) proposed by Farina *et al.* [5] is presented. Indeed, unlike in the single-objective dynamic optimization problems, where the ordering criterion in decision space is trivial, here, we are dealing with two distinct yet related spaces where an ordering criterion has to be considered: decision variable space and objective space. Such an increased complexity holds true for static problems and even more for dynamic problems, where there are four possible ways a problem can demonstrate a time-varying change:

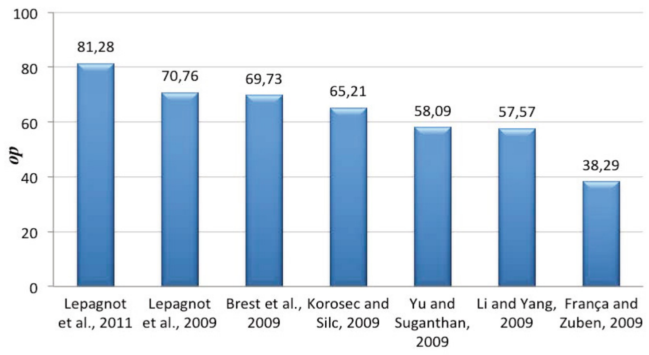

Fig. 1.4 Comparison with competing algorithms on GDBG.

1. The POS (optimal decision variables) changes, whereas the POF (optimal objective values) does not change.
2. Both POS and POF change.
3. POS does not change but POF varies.
4. The problem changes but POF and POS do not vary.

In this bechmark five different scenarios were proposed and a test suite for continuous problems was also proposed (FDA1 to FDA5). Based on the proposed scenarios, the five test problems involve ZDT and DTLZ test problems. However, these test problems can be used as a representative set of test problems in a study. The authors do not provide more functions but provide the procedure to build other more interesting problems.

1.6 Conclusions

In this chapter a suggested tool has been studied to analyse the performance of stochastic-based optimization algorithms. Moreover, a description of two continuous dynamic optimization benchmarks was presented. We also reported a review of scores of different algorithms on these benchmarks. We ended this chapter by a benchmark for dynamic multiobjective algorithms. We hope this chapter will help the researchers to speed up their development of new algorithms.

References

[1] Bird, S., Li, X.: Using regression to improve local convergence. In: Proc. Congr. Evol. Comput., Singapore, pp. 592–599. IEEE (2007)

[2] Blackwell, T., Branke, J.: Multi-swarms, exclusion and anti-convergence in dynamic environments. IEEE Transactions on Evolutionary Computation 10(4), 459–472 (2006)

[3] Branke, J.: The Moving Peaks Benchmark website (1999), `http://www.aifb.unikarlsruhe.de/~jbr/MovPeaks`

[4] Eberhart, R.C., Shi, Y.: Computational intelligence: concepts to implementation. Elsevier (2007)

[5] Farina, M., Deb, K., Amato, P.: Dynamic multiobjective optimization problems: test cases, approximations, and applications. IEEE Transactions on Evolutionary Computation 8(5), 425–442 (2004)

[6] Jin, Y., Sendhoff, B.: Constructing Dynamic Optimization Test Problems Using the Multi-objective Optimization Concept. In: Raidl, G.R., Cagnoni, S., Branke, J., Corne, D.W., Drechsler, R., Jin, Y., Johnson, C.G., Machado, P., Marchiori, E., Rothlauf, F., Smith, G.D., Squillero, G. (eds.) EvoWorkshops 2004. LNCS, vol. 3005, pp. 525–536. Springer, Heidelberg (2004)

[7] Lepagnot, J., et al.: Performance analysis of MADO dynamic optimization algorithm. In: Proc. IEEE Adaptive Computing in Design and Manufacturing. Int. Conf. on Intelligent Systems Design and Applications, Pisa, pp. 37–42. IEEE (2009)

[8] Lepagnot, J., Nakib, A., Oulhadj, H., Siarry, P.: A new multiagent algorithm for dynamic continuous optimization. International Journal of Applied Metaheuristic Computing 1(1), 16–38 (2010)

[9] Lepagnot, J., Nakib, A., Oulhadj, H., Siarry, P.: Brain cine-MRI registration using MLSDO dynamic optimization algorithm. In: IXth Metaheuristics International Conference, pp. S1–25–1–S1–25–9 (2011)

[10] Li, C., Yang, M., Kang, L.: A New Approach to Solving Dynamic Traveling Salesman Problems. In: Wang, T.-D., et al. (eds.) SEAL 2006. LNCS, vol. 4247, pp. 236–243. Springer, Heidelberg (2006)

[11] Li, C., Yang, S.: A Generalized Approach to Construct Benchmark Problems for Dynamic Optimization. In: Li, X., Kirley, M., Zhang, M., Green, D., Ciesielski, V., Abbass, H.A., Michalewicz, Z., Hendtlass, T., Deb, K., Tan, K.C., Branke, J., Shi, Y. (eds.) SEAL 2008. LNCS, vol. 5361, pp. 391–400. Springer, Heidelberg (2008)

[12] Li, C., Yang, S., Nguyen, T.T., Yu, E.L., Yao, X., Jin, Y., Beyer, H.-G., Suganthan, P.N.: Benchmark generator for CEC 2009 competition on dynamic optimization. Technical report, University of Leicester, University of Birmingham, Nanyang Technological University (2008)

[13] Li, X., Branke, J., Blackwell, T.: Particle swarm with speciation and adaptation in a dynamic environment. In: Proc. Genetic Evol. Comput. Conf., Seattle, Washington, USA, pp. 51–58. ACM (2006)

[14] Liu, L., Yang, S., Wang, D.: Particle swarm optimization with composite particles in dynamic environments. IEEE Trans. Syst. Man. Cybern. Part B 40(10), 1634–1648 (2010)

[15] Lung, R.I., Dumitrescu, D.: Collaborative evolutionary swarm optimization with a Gauss chaotic sequence generator. Innovations in Hybrid Intelligent Systems 44, 207–214 (2007)

[16] Lung, R.I., Dumitrescu, D.: ESCA: A new evolutionary-swarm cooperative algorithm. SCI, vol. 129, pp. 105–114 (2008)

[17] Morrison, R.W., De Jong, K.A.: A test problem generator for non-stationary environments. In: Proc. Congr. Evol. Comput., pp. 2047–2053 (1999)

[18] Moser, I., Chiong, R.: Dynamic function optimisation with hybridised extremal dynamics. Memetic Computing 2(2), 137–148 (2010)

[19] Moser, I., Hendtlass, T.: A simple and efficient multi-component algorithm for solving dynamic function optimisation problems. In: Proc. Congr. Evol. Comput., pp. 252–259. IEEE, Singapore (2007)

[20] Parrott, D., Li, X.: Locating and tracking multiple dynamic optima by a particle swarm model using speciation. IEEE Transactions on Evolutionary Computation 10(4), 440–458 (2006)

[21] Talbi, E.-G.: Metaheuristics: from design to implementation. John Wiley and Sons Inc. (2009)

[22] Yang, S.: Non-stationary problem optimization using the primal-dual genetic algorithm. In: Proc. Congr. Evol. Comput., pp. 2246–2253. IEEE, Canberra (2003)

[23] Yang, S., Li, C.: A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments. IEEE Transactions on Evolutionary Computation (2010)

[24] Yang, S., Yao, X.: Experimental study on population-based incremental learning algorithms for dynamic optimization problems. Soft Computing – A Fusion of Foundations, Methodologies and Applications 9(11), 815–834 (2005)

[25] Yang, S., Yao, X.: Population-based incremental learning with associative memory for dynamic environments. IEEE Transactions on Evolutionary Computation 12(5), 542–562 (2008)