Sun, Jianyong and Garibaldi, Jonathan M. and Zhang, Yongquan and Al-Shawabkeh, Abdallah (2016) A multi-cycled sequential memetic computing approach for constrained optimisation. Information Sciences, 340-341 . pp. 175-190. ISSN 1872-6291

A Multi-cycled Sequential Memetic Computing Approach for Constrained Optimisation

Jianyong Sun[a,b,*], Jonathan M. Garibaldi[c], Yongquan Zhang[d], Abdallah Al-Shawabkeh[e]

[a] *School of Computer and Software, Nanjing University of Information Science and Technology, China.*
[b] *School of Computer Science and Electrical Engineering, University of Essex, Colchester, CO4 3SQ, U.K.*
[c] *School of Computer Science, The University of Nottingham, Nottingham, NG8 1BB, U.K.*
[d] *Department of Applied Mathematics, China Jiliang University, Hangzhou 310018, ZheJiang, China.*
[e] *Faculty of Engineering and Science, University of Greenwich, Kent ME4 4TB, U.K.*

Abstract

In this paper, we propose a multi-cycled sequential memetic computing structure for constrained optimisation. The structure is composed of multiple evolutionary cycles. At each cycle, an evolutionary algorithm is considered as an operator, and connects with a local optimiser. This structure enables the learning of useful knowledge from previous cycles and the transfer of the knowledge to facilitate search in latter cycles. Specifically, we propose to apply an estimation of distribution algorithm (EDA) to explore the search space until convergence at each cycle. A local optimiser, called DONLP2, is then applied to improve the best solution found by the EDA. New cycle starts after the local improvement if the computation budget has not been exceeded. In the developed EDA, an adaptive fully-factorized multivariate probability model is proposed. A learning mechanism, implemented as the guided mutation operator, is adopted to learn useful knowledge from previous cycles.

The developed algorithm was experimentally studied on the benchmark problems in the CEC 2006 and 2010 competition. Experimental studies have shown that the developed probability model exhibits excellent exploration capability and the learning mechanism can significantly improve the search efficiency under certain conditions. The comparison against some well-known algorithms showed the superiority of the de-

*Corresponding author
Email address: j.sun@greenwich.ac.uk (Jianyong Sun)

veloped algorithm in terms of the consumed fitness evaluations and the solution quality.
Keywords: multi-cycled sequential memetic computing approach, estimation of distribution algorithm, constrained optimisation

1. Introduction

The goal of this paper is to develop a memetic algorithm for the constrained optimization problem which is also referred to as nonlinear programming (NLP) [3]. The NLP can be stated as follows:

$$\min f(\mathbf{x}), \mathbf{x} \in \mathcal{F} \subset \mathbb{R}^n$$

where $f(\mathbf{x})$ is the objective function, and $\mathcal{F}$ is the set of *feasible* solutions that satisfies:

$$\begin{cases} g_i(\mathbf{x}) & \leq & 0, & i = 1, \cdots, q; \\ h_j(\mathbf{x}) & = & 0, & j = q+1, \cdots, m. \end{cases}$$

Often, a solution $\mathbf{x}$ is regarded as *feasible*, if

$$\begin{cases} g_i(\mathbf{x}) & \leq & 0 & \forall i = 1, \cdots, q \\ |h_j(\mathbf{x})| - \varepsilon & \leq & 0 & \forall j = q+1, \cdots, m. \end{cases}$$

where ε is small positive real number. The NLP can then be restated as (cf. 1):

$$\min f(\mathbf{x}), \mathbf{x} \in \mathcal{F} = \{\mathbf{x} : \hat{g}_i(\mathbf{x}) \leq 0, 1 \leq i \leq m\} \tag{1}$$

where $\hat{g}_i = g_i, 1 \leq i \leq q, \hat{g}_j = |h_j| - \varepsilon, j = q+1, \cdots, m$. Many machine learning problems, such as image processing [69], ordinal regression [17, 18], robust clustering [56, 58], correlation analysis [59], and others, can be formulated as NLP.

One of the main concerns in developing evolutionary algorithms (EAs) for the NLP is on how to select promising parent individuals for offspring reproduction. An effective selection method, or essentially individual ranking, should balance the feasibility and the objective values of the individuals. Note that an individual with small objective function value might not even be feasible. Most of the selection strategies are based on the superiority of feasible solutions over infeasible solutions [50]. However, Jiao et

al. [26] found that global optimal solutions are more likely to be found on the boundary between the non-dominated and feasible sets.

Various constraint-handling techniques have been developed for effective ranking. The stochastic ranking (SR) method [52] ranks the individuals by balancing the objective function value and the penalty on constraint violations stochastically. An addition of ranking method developed in [21] ranks various numerical properties of the population such as the values of the objective functions, the constraint violations, and the number of constraint violations, respectively; and aggregates these rankings together as the final ranking criterion. Some authors, e.g. [1][11], proposed to rank individuals based on Pareto dominance relation in a multi-objective perspective. In [2], the authors proposed to adapt the penalty parameters. In [48], the authors proposed to first identify which constraints are effective and then use them to contribute to the fitness evaluation. In [60], the ε-constraint handling method was proposed in which an ε parameter is applied to control the relaxation of the constrains. A rough penalty method based on the rough set theory was proposed in [33]. The ensembles of these constraint-handling techniques were claimed to reduce the use of fitness evaluations and perform better than algorithms with a single constraint-handling technique in [39]. In [49], the authors studied several existing constraint-handling strategies and proposed several methodologies based on parent-centric and inverse parabolic probability distribution. The authors in [19] found that existing constraint-handling methods are applied to assist but not to guide the search process. They thus proposed the so-called constraint consensus methods to assist infeasible individuals to move towards the feasible region. Interested readers are referred to [42][44] for reviews, and [10][41] for recent advances on constraint-handling.

Another important issue in developing effective EAs for the NLP is on the offspring generation scheme. It is expected that the scheme should be able to explore feasible regions of the NLP in the early stages, and exploit for the global optimum later on. The search abilities of a range of EAs on the NLP (including genetic algorithms [22][64], evolution strategies [27], evolutionary programming [4], differential evolution [14], particle swarm optimisation [20, 13], and many others) have been extensively studied. To the best of our knowledge, the application of EDAs is very limited. In [16], two

EDAs coupled with different constraint-handling methods were compared but only on two test problems. The continuous Gaussian model was used in [54] for constrained optimisation.

Besides these research efforts, some researchers have made attempts to develop memetic computing (MC) approaches, i.e. the hybridisation of local optimization and EAs, for the NLP. The MC approach has been well acknowledged as a promising paradigm for dealing with various types of optimization problems [8]. In this paper, we develop a multi-cycled sequential MC framework, where an EDA and a classical constrained optimization algorithm is hybridised sequentially. Further, a simple learning scheme is proposed to learn useful information from previous cycles to improve the search efficiency in latter cycles.

In the rest of the paper, related work on MC is reviewed in Section 2. We then present the multi-cycled sequential MC framework in Section 3. The developed algorithm is presented in Section 4. The experimental results are summarised in Section 5. Section 6 concludes the paper and discusses future work.

2. Related Work

The development of the MC approaches has been proceeding in two main directions. On one hand, different meta-heuristics are combined to take advantages of their respective strengths. For examples, in [30], a combination of fuzzy logic and evolutionary programming is proposed to handle constraints. In [9], evolutionary programming is hybridized with GENOCOP [43] for the NLP. In [65] and [60], GAs are combined with simulated annealing and PSO, respectively, for the NLP. The integration of artificial bee colony and bees algorithm was presented in [62]. In [23], a novel variant of invasive weed optimization was combined as a local refinement procedure within differential evolution [23]. The combination of variability evolution [36] and CMA-ES [37] was proposed in [38] for the NLP.

On the other hand, classical numerical optimization approaches for the NLP have been hybridized in EAs. One of the main advantages of classical approaches is that they are usually very efficient in locating feasible local optimum, but the search efficiency

highly depends on the quality of the initial solution. Starting from a 'bad' solution, a classical approach could either only find an infeasible solution, or need a high computational cost to reach a feasible local optimum. Thus, effective strategies to address when and how to apply the classical approach should be the main considerations in designing a MC approach.

In recent literature (see e.g. [8][46]), the authors considered the MC approaches as a broader subject of memetic algorithms (MAs). They stated that a MA is composed by an evolutionary framework that integrates one or more local search components within the generation cycle of the evolutionary framework; while a MC is simply a hybrid algorithm without a specific structure. As summarised in [45][47], basic MAs can be considered as local minimizer(s) acting on evolutionary population, in which local optimiser(s) is applied to every single individual. From the view of computational cost, it is highly likely that such an indiscriminate strategy will result in a high computational cost. An obvious reason is that some individuals with low fitness cannot survive from the selection operation in the evolution procedure, which means that the improvement efforts will be wasted. Obviously, more uses of local improvements imply more efforts on exploitation. As a result, too much emphasises on exploitation could be placed on the existing EAs, at least in some cases. In other words, the balance of exploration and exploitation may be shifted too much in favour of exploitation.

Some efforts have been made to address this shortcoming. One way is to apply the local optimizers only on a proportion of promising individuals at each generation [47]. However, one can criticise that it is not fair to the other individuals when the selection operation is performed. This is because a local search on a low-quality solution does not necessarily lead to a low-quality local optima, especially in the constraint optimization context [26].

Another way is to apply the local search only after the EA has converged. Under this strategy, to obtain a good algorithm performance, the hope is that the best solution found by the EA is located in the attraction basin of a high-quality solution. Unfortunately, this is not always the case. No scheme in this strategy is provided to escape from the found optimum if it is not global.

In recent literature, a sequential memetic computing (SMC) approach has been

implemented [25, 57]. In such structure, components in the evolutionary framework and the local optimiser(s) are all considered as operators. The evolution procedure can be considered as a connected structure of those operators. The structure simplifies the MA structure, and has the potential to alleviate the aforementioned problems existed in the MAs. In [25], a single solution evolves till convergence and a parametrised local search improves the solution at different stages with different parameter settings. In [7], a meta-heuristic is first applied to find a promising solution and to compute a separability index; two heuristic local optimisers are then selected according to the index to improve the promising solution. In our work [57], an EDA is hybridized with a classical local optimizer under a SMC structure. These papers have shown that a simple SMC approach is highly potential to improve the search efficiency.

3. The Multi-cycled Sequential Memetic Computing Structure

We observe that existing SMC approaches do not take an EA as a single operator. Rather, the EA operators and local optimizer(s) are connected sequentially within an evolutionary framework. Obviously, an EA takes some inputs (e.g. fitness function, algorithmic parameters, etc.) and outputs some solutions, which is similar to what other operators (such as crossover and mutation operators in GA) do. In this paper, we propose to use a complete EA as an operator; and connect it with local optimizer(s) sequentially. Further, we propose to employ the combination of EA and local optimizer(s) multiple times until the computational budget has been reached. If we consider the composition of EA and local optimiser as a cycle, we end up with a multi-cycled SMC structure.

Under the multi-cycled SMC structure, firstly, we do not apply local optimisers to any individual during the EA search procedure. This avoids wasting computational resources on unpromising individuals under the MA structure. Secondly, the multiple-cycle structure can provide a mechanism to improve search efficiency. That is, we can gradually accumulate useful knowledge from previous cycles, and apply them in later cycles to either escape from previously found local optima, or to accelerate the exploitation. To the best of our knowledge, no MC-based algorithms have been pro-

posed to take the multi-cycled structure, which means no learning mechanisms have ever been studied. Moreover, no efforts have been carried out to apply the multi-cycled SMC structure for the NLP.

The above multi-cycled SMC approach for optimisation problems can be summarised in Alg. 1. In the algorithmic framework, Θ_1 and Θ_2 are the parameters of the EA and the local optimiser, respectively, and c is the cycle index. At each cycle, **EvolutionaryAlgorithm**($\Theta_1, history$) **takes the history information into account, and returns the best solution found (denoted as** $\mathbf{x}_c$) in line 3. In the first cycle, no history information is available, we thus set $history = \emptyset$ (line 1). **LocalSearch**($\mathbf{x}_c, \Theta_2$) improves $\mathbf{x}_c$ to a local optimum $\mathbf{x}_c^*$ (which is called the cycle best solution) in line 4. The global best solution $\mathbf{x}^*$ is updated after the local improvement (line 5). Useful information S is then learned from the current cycle by **LearningFromHistory**() (line 6). The cycle index and the history information are updated hereafter (line 7). A new cycle starts if the computational budget has not been exceeded. The global best solution $\mathbf{x}^*$ is returned on termination.

Algorithm 1 Multi-cycled Sequential Memetic Computing Framework

Require: parameters Θ_1 and Θ_2

Ensure: The best solution found x^*

1: Initialization. Set $c = 0$, $history = \emptyset$;
2: **while** computational budget has not been exceeded, **do**
3: $\mathbf{x}_c =$ **EvolutionaryAlgorithm**(Θ_1, *history*);
4: $\mathbf{x}_c^* :=$ **LocalSearch**(x_c,Θ_2);
5: $\mathbf{x}^* := \min\{\mathbf{x}_j^*, 1 \leq j \leq c\}$;
6: $S :=$ **LearningFromHistory**();
7: $c := c + 1$; *history* $:= history \bigcup S$.
8: **end while**

4. The Working Algorithm

In this section, we present a simple working algorithm according to the generic scheme proposed in the above section. An estimation of distribution algorithm (EDA)

is proposed as the EA operator. As well known, in an EDA, offspring are generated by sampling from a probability model, which is constructed from selected promising individuals, at each generation. The probability model is to represent the statistical information extracted from the selected promising solutions. The way to construct the probability model differentiates the EDA instantiations. Readers are referred to [29] for detailed descriptions of these EDAs.

4.1. Adaptive Probability Model & Multiple Sampling Strategy

In existing EDAs, the probability model for real variables is usually assumed to be a Gaussian distribution [29], a Gaussian mixture [5], or a histogram [63][68]. In this paper, we propose to construct a full-factorised adaptive multivariate model. That is, we assume $p(\mathbf{x};t) = \prod_{i=1}^{n} p(x_i;t)$ where $\mathbf{x} = (x_1, \cdots, x_n)^\intercal$. The construction of $p(\mathbf{x};t)$ is presented in Alg. 2, where a selected population containing a set of K individuals $\mathcal{P}^s(t)$ is the input. First, the range of the i-th variable in $\mathcal{P}^s(t)$ is sought, denoted as $[\ell_i^{\min}, \ell_i^{\max}]$ (line 1); then the range is expanded with a small positive number ϵ_i; different probabilities are assigned to the range interval $[\ell_i^{\min}, \ell_i^{\max}]$ and the expanded intervals ($[\ell_i^{\min} - \epsilon_i, \ell_i^{\min}]$ and $[\ell_i^{\max}, \ell_i^{\max} + \epsilon_i]$) (line 3).

The developed EDA exhibits several new features in model construction and sampling. First, a uniform distribution is assumed over the range interval $[\ell_i^{\min}, \ell_i^{\max}]$. This is meant to preserve the diversity during the search. Second, the expansion intervals $[\ell_i^{\min} - \epsilon_i, \ell_i^{\min}]$ and $[\ell_i^{\max}, \ell_i^{\max} + \epsilon_i]$ are meant to address the premature convergence problem. Finally, we propose to use a *multiple sampling strategy* to make the sampling more effective, which is meant to address the sampling noise problem.

To the best of our knowledge, in almost all EDAs, the number of sampled offspring from the probability model $p(\mathbf{x};t)$ is usually less than, or equal to, the population size. However, it is well known that to accurately characterise $p(\mathbf{x};t)$, a large sampling size is needed [51]. Therefore, statistically speaking, a small sample size will result in high sampling noise, which might falsely guide the evolutionary search. That is, the search may be leaded to possibly non-promising areas. The problem will become much serious when $p(\mathbf{x};t)$ is complex. To address this problem, we propose to generate a number of offspring which is $k(> 1)$ (we call it the *sampling factor*) times of the

population size.

Algorithm 2 Multivariate Adaptive Probability Model

Require: The selection population $\mathcal{P}^s(t) = \left\{\mathbf{x}^1(t), \cdots, \mathbf{x}^K(t)\right\}$

Ensure: The probability $p(\mathbf{x}; t)$.

1: **for** $1 \leq i \leq n$ **do**

2: Find $\ell_i^{\min}(\ell_i^{\max}) = \min(\max)\{x_i^k(t), 1 \leq k \leq K\}$;

3: Assign a small probability to the intervals $[\ell_i^{\min} - \epsilon_i, \ell_i^{\min}]$ and $[\ell_i^{\max}, \ell_i^{\max} + \epsilon_i]$, and a big probability to $[\ell_i^{\min}, \ell_i^{\max}]$.

4: **end for**

4.2. The Learning from Previous Cycles

An important contribution of the proposed framework is that it enables the learning from previous cycles to improve the search efficiency in latter cycles. This section presents a simple learning method.

The most important message we obtained from previous cycles is the location information of the global best $\mathbf{x}^*$. This information should be incorporated in the new cycle. One possible way to take advantage of the location information is to combine it in the sampling procedure by using the guided mutation method [67]. Alg. 3 describes the guided mutation in detail, where $\lceil A \rceil$ represents the rounding of A to the nearest integers greater than or equal to A. Basically speaking, the guided mutation generates an offspring by copying a part of $\mathbf{x}^*$, and filling the other part by sampling from a probability model.

The underlying rationale behind the guided mutation is closely related to the so-called proximity optimality principle (POP) [15], which has been explicitly or implicitly applied in almost all meta-heuristics. The POP states that good solutions have similar structure. By using the guided mutation operator, some elements of the the global best solution is statistically retained during the search. Under certain conditions, retaining these location information will improve the algorithmic search efficiency. We will discuss the condition in Section 4.5.

Algorithm 3 Guided Mutation Operator

Require: a template solution $\mathbf{x}^*$, a real number $0 \leq \alpha \leq 1$ and a probability model $p(\mathbf{x}; t) = \prod p(x_j; t)$.

Ensure: An offspring $\mathbf{x} = (x_1, \cdots, x_n)^\intercal$.

1: Set $U = \{1, 2, \cdots, n\}$ and $N := \lceil \alpha n \rceil$; Randomly select a set of indices $V \subset U$ with $|V| = N$;

2: For an index $i \in V$, set $x_i := x_i^*$; For an index $j \in U \setminus V$, sample a value y from the probability model $p(x_j; t)$, set $x_j := y$;

3: Return $\mathbf{x}$;

4.3. Selection and Replacement

The selection process has been widely studied in the constrained evolutionary optimisation literature, mostly based on constrain-handling techniques. Selection methods based on penalty methods will bias the search, while those based on multi-objective approaches will not. However, as stated in [53], the unbiased search does not necessarily improve the search efficiency. Since local optimisers usually work better on a feasible solution than on an infeasible solution, we prefer to use a selection method that favours feasible solutions. Here, the selection method, called the over-penalised approach in [53], is adopted in this paper.

In the over-penalised approach, the feasible individuals are ranked higher than the infeasible individuals. The feasible solutions are sorted according to their objective function values f. The infeasible individuals are ranked according to the penalty function values ψ, which is defined as $\psi(x) = f(\mathbf{x}) + \sum_j g_j^+(\mathbf{x})^\beta$ where $g_j^+(\mathbf{x}) = \max\{0, g_j(\mathbf{x})\}$, and $\beta = 2$.

Regarding replacement, we again adopt the over-penalised selection approach to form new population. At each generation, the best individuals are used to construct the probability model and passed to the new population, while the rest of the new population is replaced by the best offspring sampled from the constructed probability model.

4.4. The Local Optimiser

We adopt a classical optimisation method developed for the NLP, called DONLP2 (abbreviation for 'DO NonLinear Programming') [55] to improve the best solution found by the EDA. DONLP2 is based on the sequential quadratic programming method (SQP), in which fully regularised mixed constrained sub-problems are used to deal with non-regular constraints. It incorporates techniques including a slightly modified Pantoja-Mayne update for the Hessian of the Lagrangian, a variable dual scaling and an improved Armijo-type step size algorithm to improve the search efficiency of the SQP.

The most important algorithmic parameters of the DONLP2, i.e. Θ_2 in Alg. 1, include τ_0 which gives a bound describing how much the unscaled penalty-term (the L_1-norm of the constraint violation) may deviate from zero and δ_0 which is a binding constraint. In our experimental simulations, we set $\tau_0 = 1.0$ and $\delta_0 = 0.2$ as suggested in [55]. Moreover, we do not calculate the analytical form of the gradients and Hessian of the Lagrangian, but using numerical differentiation. The used NFEs for computing the differentials are included in the calculation of the overall NFEs in the sequel reports.

4.5. Remarks on the Algorithmic Framework and the Working Algorithm

In this section, we discuss the pros and cons of the algorithmic framework, and the condition that the working algorithm will be effective.

4.5.1. The Algorithmic Framework

In the sequel, we assume that there are a limited number of feasible local optima[1] $\mathbf{x}_i^*, 1 \leq i \leq M$ in terms of the fitness function $\lambda(\mathbf{x}) = f(\mathbf{x})$. They can be sorted in a descending order, denoted as $\mathbf{x}_1^*, \mathbf{x}_2^*, \cdots, \mathbf{x}_M^*$ where $\lambda(\mathbf{x}_1^*) \geq \lambda(\mathbf{x}_2^*) \geq \cdots \geq \lambda(\mathbf{x}_M^*)$.

In the sequel, we define

$$\phi(\mathbf{x}_i^*) = \left\{\mathbf{x}^* | \exists j \in \{1, 2, \cdots, n\} \setminus \{i\}, \text{s.t. } |x_j^* - x_{ij}^*| < \varepsilon\right\} \tag{2}$$

[1]Readers that are interested in the theoretical analysis on the collaboration between global search and local search please refer to [34, 35], in which the concept of local search zones are defined and studied. Here, we only consider local optima rather than the local search zones since local optimiser is considered as a black-box in the SMC structure, and its application only result in local optima.

where ε is very small positive number. That is, $\phi(\mathbf{x}_i^*)$ contains the optima that is close to the i-th optimal solution. Further, we introduce the condition

$$\phi(\mathbf{x}_i^*) \bigcap \phi(\mathbf{x}_j^*) \neq \emptyset \text{ for } j > i \tag{3}$$

which indicates that the two optima $\mathbf{x}_i^*$ and $\mathbf{x}_j^*$ have common elements. This can be seen as the mathematical formalisation of the POP. Note that the condition (3) also implies that $\phi(\mathbf{x}_i^*) \neq \emptyset$ for all $i, 1 \leq i \leq M$. This can be proved by using contradiction as follows. Suppose for some i, $\phi(\mathbf{x}_i^*) = \emptyset$. Then for all $j > i$, $\phi(\mathbf{x}_i^*) \bigcap \phi(\mathbf{x}_j^*) = \emptyset$, which contradicts the condition.

Moreover, we can see that if such a condition holds, a solution path exists under the proposed multi-cycled SMC structure. That is, starting from a local optimum $\mathbf{x}_{i_1}^*$, $i_1 \in \{1, \cdots, M\}$, a better local optimum $\mathbf{x}_{i_2}^*, i_2 > i_1$ can be found at further cycles since $\phi(\mathbf{x}_{i_1}^*) \bigcap \phi(\mathbf{x}_{i_2}^*) \neq \emptyset$. Applying the evolutionary cycle K times, we will end up with a sequence of local optima $\mathbf{x}_{i_1}^*, \cdots, \mathbf{x}_{i_K}^*$, or a 'solution path', with

$$\phi(\mathbf{x}_{i_j}^*) \bigcap \phi(\mathbf{x}_{i_k}^*) \neq \emptyset, i_j < i_k; \text{ and } \lambda(\mathbf{x}_{i_k}) \leq \lambda(\mathbf{x}_{i_j}).$$

Hence, we call Eq. 3 as the *"solution path"* condition.

The above discussion suggests that the multi-cycled SMC structure will be effective on problem instances that satisfy the solution path condition. It also suggests that if $\phi(\mathbf{x}_i^*) \bigcap \phi(\mathbf{x}_j^*) = \emptyset$, the effectiveness of the framework is thus doubtful on those problem instances since the information learned from history has no help for future search.

4.5.2. The Working Algorithm

According to [66], an EDA with truncation selection converges if the truncation threshold (i.e. the percentage of individuals being selected to the next generation) is less than 1. The over-penalised selection approach can be considered as a truncation selection with adaptive threshold. The threshold will be always smaller than 1 since not all individuals will be passed to the new generation. Therefore, we can conclude that the proposed EDA converges to a solution $\mathbf{x}_c(t)$ at the t-th cycle. Under the proposed structure, $\mathbf{x}_c(t)$ is improved by the DONLP2 to obtain $\mathbf{x}_c^*(t)$ at generation t. It has

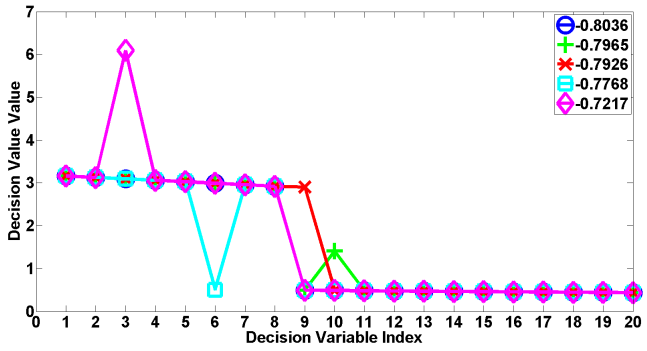


Figure 1: The demonstration of the solution path using the problem instance g_{02} as an example.

been proved in [55] that the DONLP2 holds a local convergence property. Therefore, $\mathbf{x}_c^*(t)$ is a local optimum.

According to previous discussion, if for some problem instances, the solution path condition holds, we can see that the guided mutation operator will be very efficient in finding a better optimum in the solution path since it can stochastically retain some location information of the present local optimum.

Fig. 1 shows the solutions (four local optima and the global optimum) found by the developed algorithm on g_{02} in five cycles. The objective function values of the local optima are shown in the legend. From Fig. 1, we can see that many variables (including x_{1-2}, x_{4-5}, x_{7-8}, x_{11-20}) of the local optima take similar values to the global optimum. In latter cycles, the rest variables ($x_{3,6,9,10}$) are gradually modified. This example shows that the "solution path" condition holds for g_{02}.

5. Experimental Results

In the developed algorithm, called the multi-cycled evolutionary (MCEA) algorithm, the parameters (i.e. Θ_1) of the EDA include the population size M, the selection size K, the sampling factor k, the guided mutation parameter α, and the expansion parameter ϵ. The expansion in each dimension of the search space is set to be $\epsilon_i = \frac{b_i - a_i}{M}, 1 \leq i \leq n$ where a_i and b_i are the lower and upper bound, respectively. This setting is to eliminate the effect of the variable scales in different coordinates.

Regarding the criterion to decide when a new cycle starts, in our implementation, new cycle starts if the number of generations is more than 30; and in consecutive 5 generations[2], there are no better solutions found.

In this section, firstly we analyse the effects of the proposed EDA components to the algorithmic performance and the algorithm's sensitivity to the parameters. We then compare the developed algorithm with some well-known algorithms including the winners of the CEC 2006 and 2010 competitions. Readers are referred to [31] and [40] for detailed problem definitions.

5.1. Comparison Metrics

The comparison metrics include the success rate (#succ_run), the average number of fitness evaluation consumed (NFE), and the average number of cycles (#cycle). Suppose that in total T runs, there are K successful runs. For each run i, the consumed NFEs is N_i (if not successful, N_i is the maximum NFEs allowed) and the number of cycles is C_i, then the success rate is defined as #succ_run $= K/T$, the average NFEs is computed as NFE $= \sum_{i=1}^{T} N_i/K$, and the average number of cycles is defined as #cycle $= \sum_{i=1}^{T} C_i/T$.

5.2. Component Analysis

The two aspects that mostly affect the performance of the proposed algorithm are the exploration capability of the probabilistic model, and the learning capability of the guided mutation operator. The component analysis aims to investigate their respective contributions. Moreover, we intend to study the effect of the constraint-handling techniques to the algorithmic performance. The CEC 2006 test problems are used for the analysis. The experimental configurations are set as follows: the positive number to relax the equality constraints is $\varepsilon = 0.0001$, the number of runs is 25 and the maximum number of fitness evaluations (NFEs) is 500,000. At each run, the NFEs needed to find a solution satisfying $f(\mathbf{x}) - f(\mathbf{x}^*) < \varepsilon$ are recorded.

[2]These values used here were chosen based on experiments we carried out for the test problems.

5.2.1. The Probability Model

The effect of the probability model can be carried out by adopting different probability models in the proposed EDA. In this study, we compare the histogram model and the Gaussian model with the proposed adaptive model. The resultant algorithms are called MCEH (with histogram model), and MCEG (with Gaussian model), respectively. Those probability models are all fully-factorised multivariate models. In the histogram model, the bound of each variable is divided into 10 subintervals (as suggested in [68]), and the histogram of the selected individuals is normalised to be the probability distribution over these subintervals. The Gaussian model assumes that the selected individuals at each variable follows a Gaussian distribution.

The parameter settings of these algorithms are $M = 2n$, $k = 1$, and $\alpha = 0.3$. Table 1 summarises the comparison metrics obtained by the compared algorithms for the test problems except g_{20} and g_{22} (since they do not have feasible solutions).

Table 1: Experimental results obtained by the MCEA, the MCEH and MCEG on the test problems except g_{20} and g_{22}.

function	MCEA			MCEH			MCEG		
	# succ_run	NFE	#cycle	#succ_run	NFE	#cycle	#succ_run	NFE	#cycle
g_{01}	1.00	**5,774**	4.16	1.00	8,606	8.20	1.00	7,232	7.12
g_{02}	1.00	74,030	33.16	1.00	**59,712**	30.76	0.00	500,000	112.36
g_{03}	1.00	**1,326**	1.00	1.00	3,346	2.36	1.00	3,346	2.36
g_{04}	1.00	**412**	1.00	1.00	433	1.00	0.92	40,470	86.08
g_{05}	1.00	417	1.00	1.00	398	1.00	1.00	**381**	1.00
g_{06}	1.00	467	1.00	0.96	20,385	58.12	1.00	**196**	1.00
g_{07}	1.00	2,207	1.00	1.00	**1,432**	1.00	1.00	1,368	1.00
g_{08}	1.00	458	2.56	1.00	**416**	2.08	1.00	450	2.56
g_{09}	1.00	1,030	2.08	1.00	**416**	1.00	1.00	1480	1.00
g_{10}	1.00	**13,359**	1.00	1.00	37,859	1.48	1.00	33533	1.32
g_{11}	1.00	174	1.00	1.00	**155**	1.00	1.00	174	1.00
g_{12}	1.00	**196**	1.00	1.00	198	1.00	1.00	197	1.00
g_{13}	1.00	1,065	1.88	1.00	**506**	1.00	1.00	1,501	3.00
g_{14}	1.00	**3,642**	2.68	1.00	7,590	9.40	0.00	500,000	731.24
g_{15}	1.00	292	1.00	1.00	303	1.08	1.00	**277**	1.00
g_{16}	0.96	22,058	26.56	0.96	**21,587**	23.44	1.00	25,311	31.12
g_{17}	0.80	**257,115**	73.24	0.76	266,865	63.04	0.92	353,855	35.76
g_{18}	1.00	7,657	7.04	1.00	4,150	3.72	1.00	**3,893**	2.92
g_{19}	1.00	4,078	1.00	1.00	**2,617**	1.00	1.00	2,846	1.00
g_{21}	1.00	**34,152**	8.12	0.88	180,512	39.16	0.56	359,339	54.72
g_{23}	1.00	4,321	1.36	1.00	**3,403**	1.00	1.00	2,388	1.60
g_{24}	1.00	181	1.00	1.00	**164**	1.00	1.00	239	1.40

In Table 1, entries in bold typeset indicate the least NFEs consumed by the algo-

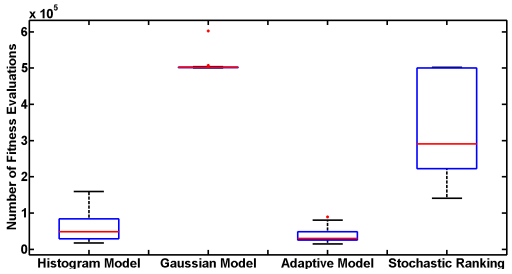


Figure 2: Comparison of different probability model and selection methods in terms of NFEs on g_{02}.

rithms. From Table 1, we see that in 8 out of 20 test problems, the MCEA consumed fewer NFEs than the MCEH; while in 10 out of 22 test problems, the MCEH requires fewer NFEs than that of the MCEA. In 4 out of 22 test problems, the MCEG performs better than the other two. Though it seems that the histogram model performs better in general, it can be seen from Table 1 that the success rates obtained by the proposed model on functions $g_{06,17,21}$ are higher than those obtained by the histogram model. Moreover, if we focus on those functions (including $g_{02,07-09,11,13,16,19,23,24}$) that the histogram model has a better performance, it can be seen that except functions g_{02}, g_{08} and g_{16}, the average numbers of cycles used by the MCEH and the MCEA to reach the global optima are all one, which means that it is fairly easy for the histogram and the adaptive model to obtain high-quality initial solutions.

Since the main difference between the compared algorithms is on the probabilistic model used in the EDA, we may conclude that the proposed adaptive model can result in better exploration capability than the others. Fig. 2 shows the boxplots of the NFEs consumed by the MCEA, MCEH and MCEG, respectively, on g_{02}.

5.2.2. The Contribution of the Constraint-handling

We now study the effects of the over-penalised selection and the stochastic ranking selection to the algorithm performance. To carry out the comparison, we build an algorithm, called MCES, in which the stochastic ranking selection is used.

In the experiments, the same algorithmic parameters as above are used by the MCEA. For the MCES, the stochastic ranking parameter is set to 0.45 as suggested in [52]. Table 2 lists the comparison metrics obtained by the two algorithms for the test

Table 2: The experimental results obtained by the MCEA and the MCES.

function	MCEA			MCES		
	#_run	NFE	#cycle	#_run	NFE	#cycle
g_{01}	1.00	**5,774**	4.16	1.00	15,712	10.00
g_{02}	1.00	**74,030**	33.16	0.64	336,461	114.24
g_{03}	1.00	**1,326**	1.00	0.92	118,536	77.84
g_{04}	1.00	**412**	1.00	1.00	458	1.00
g_{05}	1.00	417	1.00	1.00	**398**	1.00
g_{06}	1.00	**467**	1.00	0.92	40,264	108.68
g_{07}	1.00	2,207	1.00	1.00	**1,576**	1.00
g_{08}	1.00	**458**	2.56	1.00	584	2.80
g_{09}	1.00	**1,030**	1.00	1.00	1,888	1.00
g_{10}	1.00	**13,359**	1.00	1.00	20,939	1.20
g_{11}	1.00	**174**	1.00	1.00	213	1.00
g_{12}	1.00	196	1.00	1.00	196	1.00
g_{13}	1.00	1,065	1.88	1.00	**627**	1.04
g_{14}	1.00	**3,642**	2.68	1.00	51,218	38.36
g_{15}	1.00	**292**	1.00	1.00	349	1.00
g_{16}	0.96	**22,058**	26.56	0.96	24,184	23.32
g_{17}	0.80	**257,115**	73.24	0.76	289,937	85.64
g_{18}	1.00	7,657	7.04	1.00	**2,145**	1.72
g_{19}	1.00	**4,078**	1.00	1.00	3,595	1.00
g_{21}	1.00	**34,152**	8.12	1.00	110,405	29.96
g_{23}	1.00	**4,321**	1.36	1.00	3,522	1.00
g_{24}	1.00	**181**	1.00	1.00	422	2.44

problems except g_{20} and g_{22}. Entries in bold typeset are the least NFEs obtained by the compared algorithms.

From Table 2, we can see that in 16 out of 22 test problems, the over-penalised selection approach performs better than the stochastic ranking selection in terms of the NFEs consumed. In terms of the success rate, it can be seen that the over-penalised approach can obtain higher rates than the stochastic ranking approach in all the test problems, except for g_{16} where the success rates are both one. We thus may conclude that the over-penalised constraint-handling technique is more effective than that of the stochastic ranking under the proposed framework.

Moreover, in comparison with the results obtained by the MCEH shown in Table 1, one can see that the MCES performs even worse than that of the MECH on most of the test problems. This shows that the exploration capability of the developed EDA does not benefit from the application of the stochastic ranking. Particularly, we can also observe this from the last column in Fig. 2. It shows that the NFEs consumed by the MCES are even more than that of the MCEH.

5.3. Sensitivities to the Algorithmic Parameters

The main parameters of the working algorithm include the population size N, the sampling factor k and the guided mutation parameter α. In this section, we investigate the effects of these parameters on the performance of the algorithm.

5.3.1. The Sampling Factor

To test the effects of the sampling factor to the algorithmic performance, we run the algorithm by setting different $k \in \{0.5, 1, 1.5, 2, 2.5, 3\}$. The rest parameters are set as $M = 2n$, and $\alpha = 0.3$. Table 3 shows the results obtained. In the table, entries in bold typeset are the least NFEs consumed by the algorithm.

In Table 3, we omit the success rates for $k \geq 2$ since they are all one. On one hand, from Table 3, we can see that the MCEA with $k = 0.5$ performs the best on most (12 out of 22) of the test problems in terms of the consumed NFEs. However, as we discussed early in Section 5.2.1, those problems are fairly easy. The good performance of the MCEA with $k = 0.5$ might be due to the efficiency of the learning mechanism. On the other hand, if we focus on the functions $g_{06,16,17}$ which are considered as hard, we can see that the best performance is achieved by the MCEA with $k = 2$. This indicates that a large sampling size can indeed improve the search efficiency.

Regarding the MCEA with large sampling size (i.e. $k \geq 2$), it can be seen that in 19 out of 22 test problems, the MCEA with $k = 2$ requires the least NFEs than the MCEA with $k = 2.5$ and 3. This shows that a large sampling factor does not always lead to a competitive performance in terms of computational cost. The sampling factor should be carefully chosen to balance the search efficiency and the computational cost.

We further investigate the interaction between the population size M and the sample factor k, using g_{02} as an example. g_{02} is of high-dimensional ($n = 20$), non-linear in both the objective function and the constraints, and multi-modal. Fig. 3 summarises the obtained results. Fig. 3(a) shows the mean NFEs with varied $M \in \{20, 40, 60, 80, 100\}$ and $k \in \{0.5, 1, 1.5, 2\}$, while (b) shows the mean number of cycles. From Fig. 3(a), one can see that generally a small sampling size does not always result in a reduced NFEs. Specifically, we can see that in case $k = 1$, the consumed NFEs is fewer than that in case $k = 0.5$ when the population size is less than 80. This

Table 3: The experimental results obtained by the MCEA with different sampling factor k. Entries in bold typeset are the least NFEs obtained by the algorithm.

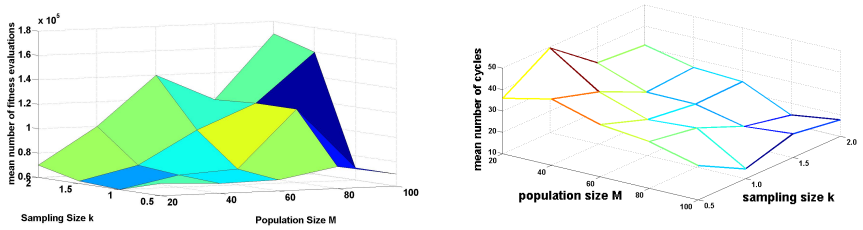
function	$k = 0.5$		$k = 1$		$k = 1.5$		$k = 2$	$k = 2.5$	$k = 3$
	#succ_rate	NFE	#succ_rate	NFE	#succ_rate	NFE	NFE	NFE	NFE
g_{01}	1.00	8,005	1.00	**5,774**	1.00	6549	6,728	5,877	7,136
g_{02}	1.00	73,131	1.00	74,030	1.00	68,968	**38,931**	55,270	69,043
g_{03}	1.00	4,829	1.00	**1,326**	1.00	3,200	1,895	3,969	3,052
g_{04}	1.00	**294**	1.00	412	1.00	591	691	950	1,034
g_{05}	1.00	**207**	1.00	417	1.00	514	722	861	1,463
g_{06}	0.88	60,175	1.00	467	1.00	526	366	**345**	536
g_{07}	1.00	**1,193**	1.00	2,207	1.00	2,292	2,373	3,198	3,573
g_{08}	1.00	**405**	1.00	458	1.00	1,407	444	656	1,116
g_{09}	1.00	756	1.00	**1,030**	1.00	1,092	1,198	1,458	1,700
g_{10}	1.00	13,619	1.00	13,359	1.00	**4,331**	5,570	32,087	12,716
g_{11}	1.00	**92**	1.00	174	1.00	246	302	394	471
g_{12}	1.00	**109**	1.00	196	1.00	325	396	487	573
g_{13}	1.00	**449**	1.00	1,065	1.00	658	1,406	3,739	3,406
g_{14}	1.00	4,227	1.00	3,642	1.00	3,542	**3,155**	11,242	6,818
g_{15}	1.00	**221**	1.00	292	1.00	336	474	669	966
g_{16}	0.92	41,822	0.96	22,058	0.96	21,738	**1,550**	3,000	44,211
g_{17}	0.80	257,115	0.92	181,885	0.92	145,887	**89,042**	93,971	110,189
g_{18}	1.00	**3,416**	1.00	7,657	1.00	4,264	4,338	2,189	6,158
g_{19}	1.00	**3,012**	1.00	4,078	1.00	5,078	5,733	6,949	7,843
g_{21}	1.00	44,588	1.00	34,152	1.00	48,749	48,944	**17,307**	52,991
g_{23}	1.00	**2,274**	1.00	4,321	1.00	3,444	3,315	4,322	5,964
g_{24}	1.00	**97**	1.00	181	1.00	404	297	689	500

observation justifies that more samples can reduce the sampling noise in case a small population size is employed, as claimed in Section 4.1. From Fig. 3(b), it can be seen that the number of cycles tends to decrease along with the increase of the population size and the increase of the sampling factor.

In summary, we may conclude that the multiple sampling strategy can indeed improve the search efficiency. But a sampling factor should be carefully chosen to balance the search efficiency and the computational cost. Moreover, the multiple sampling strategy is able to reduce the sampling noise in case a small population size is employed.

5.3.2. The Guided Mutation and the Population Size

In this section, we study the effect of the guided mutation by looking at the performance of the MCEA with different α and population size M. The population size M seriously affects the exploration capability of the proposed EDA, and α controls the



(a) The interactions of M and k in terms of NFEs. (b) The interactions of M and k in terms of #cycles.

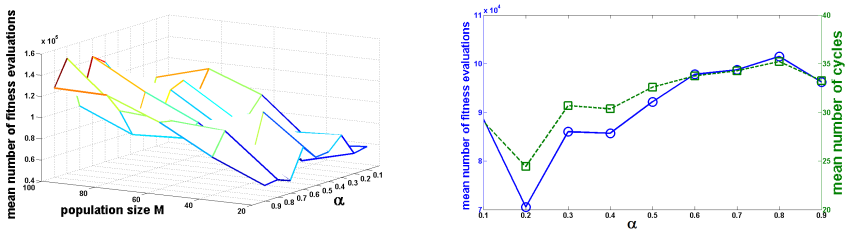
Figure 3: The study of the interactions of the algorithmic parameters M and k to the performance of the MCEA. (a) shows the results in terms of the NFEs; (b) is the results in terms of the number of cycles.

contribution of the learned information to further search.

We first test the performance of the MCEA by setting the population size M to be $j \times n$ where $j \in \{1, 2, \cdots, 6\}$. Table 4 lists the results obtained by the algorithm where entries with bold typeset are the least NFEs consumed. The other parameters are set as $k = 1$, and $\alpha = 0.3$. From Table 4, we can see that the MCEA with $M = 2n$ achieved the best performance in terms of the NFEs on almost all test problems, except for g_{12-15} where it is not as successful as the MCEA with $M = n$. In general, we may conclude that a population size $M = 2n$ is a good choice for an optimal algorithmic performance.

We further investigate how α and M interact to effect the algorithmic performance. An increased population size will increase the NFEs used in a cycle, but it also means an improved search ability. On the other hand, a large α value will result in an accelerated search speed, but also a quick loss of diversity, which will deteriorate the exploration ability and the possibility of escaping from local optima. Therefore, the optimal settings of M and α will be the settings that balance the search speed and diversity.

In our experiment, the study was carried out by varying $\alpha \in \{0.1, \cdots, 0.9\}$ and $M \in \{20, 40, 60, 80, 100\}$. Again, g_{02} is used as an example. Fig. 4 summarises the obtained results. Since for all the α settings, the MCEA can successfully locate the global optimum in all runs, we thus do not include the success rate results. Fig. 4(a)



(a) The interactions of M and α in terms of NFEs. (b) The mean NFEs and cycles against α.

Figure 4: The study of the interactions of the algorithmic parameters M and α to the performance of the MCEA. (a) shows the results in terms of the NFEs; (b) is the results in terms of the number of cycles.

shows the average NFEs consumed by the MCEA with different α and M values. From this figure, we can see that the consumed NFEs tend to decrease along with the decrease of the population size for any given α. This indicates for g_{02}, the loss of diversity due to small population size can be compensated by the learning scheme. Fig. 4(b) shows that the consumed NFEs and the number of cycles increase along with the increase of α. This indicates that a large α will limit the exploration ability of the MCEA due to the quick loss of diversity.

5.4. Summary on Component Study

In summary, we may conclude that (i) the adaptive model can improve the exploration ability of the proposed EDA; (ii) the learning strategy can compensate for the loss of diversity caused by employing a small population size; and (iii) the multiple sampling strategy can improve the search efficiency but need to seek balance with the population size for the best algorithmic performance. Regarding the parameters of the MCEA, it seems reasonable to choose $\alpha \in [0.1, 0.4]$, $k = 1$, and $M = 2n$ according to our empirical studies.

5.5. Comparison with EAs on the CEC'06 Benchmarks

In this section, we present the comparison of the MCEA with the algorithms in the CEC'06 competition. Since most of the compared algorithms were successful in all

Table 4: The experimental results obtained by the MCEA with different population size M. Entries in bold typeset are the least NFEs obtained.

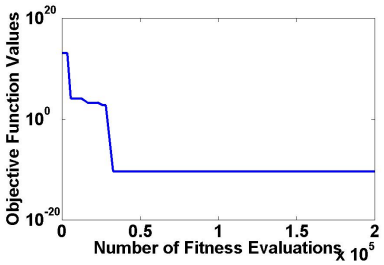
function	$M = n$		$M = 2n$		$M = 3n$		$M = 4n$		$M = 5n$		$M = 6n$	
	NFE	#cycle	NFE	#cycle	NFE	#cycle	NFE	#cycle	NFE	#cycle	NFE	#cycle
g_{01}	5,774	4.16	**3,745**	2.64	7,513	3.76	4,715	1.16	5,205	1.00	8,368	1.40
g_{02}	74,030	33.16	**57,433**	18.24	72,456	32.08	91,890	22.68	110,236	21.84	80,186	13.20
g_{03}	**1,326**	1.00	2,567	1.00	4,106	2.36	2,871	1.00	3,778	1.64	2,709	1.00
g_{04}	412	1.00	**1,104**	1.00	691	1.12	791	2.10	1,106	1.08	1,061	1.60
g_{05}	417	1.00	**353**	1.00	604	1.00	766	1.00	718	1.00	1,036	1.00
g_{06}	467	1.00	**426**	1.00	449	1.00	642	1.00	515	1.00	475	1.00
g_{07}	2,207	1.00	**2,162**	1.00	2,343	1.00	3,101	1.00	3,182	1.00	5,161	1.00
g_{08}	458	2.56	**176**	1.00	658	2.76	295	1.00	800	2.00	663	1.48
g_{09}	1,030	1.00	**841**	1.00	915	1.00	1,281	1.00	1,503	1.00	1,631	1.00
g_{10}	13,359	1.00	**12,452**	1.00	21,186	1.12	12,446	1.00	17,958	1.08	21,138	1.04
g_{11}	174	1.00	**153**	1.00	244	1.00	291	1.00	370	1.00	453	1.00
g_{12}	**196**	1.00	209	1.00	315	1.00	415	1.00	510	1.00	580	1.00
g_{13}	**1,065**	1.88	1,566	1.00	915	1.00	1,875	2.04	3,786	3.72	8,269	6.52
g_{14}	3,642	2.68	3,770	2.72	3,578	2.72	3,402	2.48	**3,114**	2.28	4,146	3.04
g_{15}	**292**	1.00	328	1.04	393	1.00	527	1.00	705	1.00	673	1.00
g_{16}	22,058	26.56	**1,326**	1.00	2,943	1.00	3,000	1.00	3,282	1.00	3,905	1.00
g_{17}	257,115	73.24	**160,063**	42.56	188,138	69.80	187,185	46.56	175,857	41.20	199,214	39.64
g_{18}	7,657	7.04	**1,525**	1.00	3,206	1.88	4,311	1.60	8,747	3.60	8,447	2.84
g_{19}	4,078	1.00	**3,878**	1.00	5,276	1.00	6,662	1.00	8,116	1.00	10,178	1.00
g_{21}	34,152	8.12	**23,749**	11.44	34,526	13.68	65,104	18.12	88,466	21.60	54,616	15.24
g_{23}	4,321	1.36	**3,402**	1.08	3,441	1.00	4,032	1.20	4,763	1.16	4,975	1.00
g_{24}	181	1.00	**165**	1.00	318	1.36	452	1.60	362	1.00	462	1.08

Table 5: The NFEs consumed by the MCEA, $\varepsilon-$DE, and the other algorithms on the CEC'06 test problems.

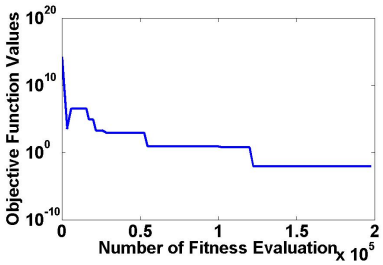
function	MCEA			$\varepsilon-$DE	Best (Alg.)
	min	mean	max		
g_{01}	3,284	6,728	17,917	59,309	25,115 (SaDE)
g_{02}	14,654	38,930	89,309	149,827	96,222 (MDE)
g_{03}	1,773	1,895	1,917	89,407	24,861 (MPDE)
g_{04}	512	691	891	26,216	15,281 (GDE)
g_{05}	592	724	933	97,430	21,306 (MDE)
g_{06}	270	366	444	7,381	5,202 (MDE)
g_{07}	2,035	2,373	3,159	74,304	26,578 (DMS)
g_{08}	331	444	896	1,139	918 (MDE)
g_{09}	1,133	1,198	1,235	23,121	16,152 (MDE)
g_{10}	5,071	5,570	5,805	105,234	25,520 (DMS)
g_{11}	292	362	585	16,420	3000 (MDE)
g_{12}	382	396	442	4,124	1,308 (MDE)
g_{13}	788	1,406	2998	31,096	21,723 (MDE)
g_{14}	2,601	3,155	3,987	113,439	25,220 (DMS)
g_{15}	447	474	533	83,655	10,458 (MDE)
g_{16}	1,635	1,550	2,270	19,122	8,730 (MDE)
g_{17}	19,345	85,478	155,419	98,860	26,364 (MDE)
g_{18}	1,948	4,338	7,924	59,153	28,261 (DMS)
g_{19}	4,561	5,732	6,958	356,350	21,830 (DMS)
g_{21}	14,513	48,944	92,988	135,142	38,217 (PCX)
g_{23}	2,212	3,135	3,751	200,763	129,550 (SaDE)
g_{24}	278	297	380	2,952	1,794 (jDE-2)

runs, we use the consumed NFEs as the criterion. The experimental results are summarised in Table 5. The minimal, mean and maximum number NFEs in 25 runs for the MCEA are shown in the 'min', 'mean' and 'max' columns, respectively. The column '$\varepsilon-$DE' shows the average NFEs used by $\varepsilon-$DE which is reproduced from [60]. The 'Best(Alg.)' column shows the least NFEs used by the algorithms appeared in the CEC'06 competition. These algorithms are SaDE [24], MDE [42], MPDE [61], GDE [28], PCX [12], DMS [32] and jDE-2 [6].

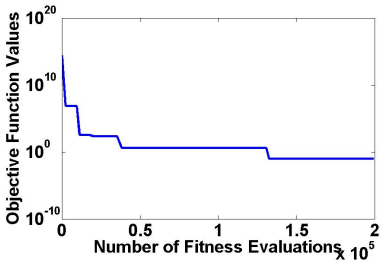
From Table 5, it can be seen that the MCEA consumed much fewer NFEs than all the other compared algorithm except for g_{17} where the average NFEs used by the MCEA is more than that of the MDE. However, we may still conclude that on average, the MCEA outperforms these compared EAs in terms of the consumed NFEs on these test problems.



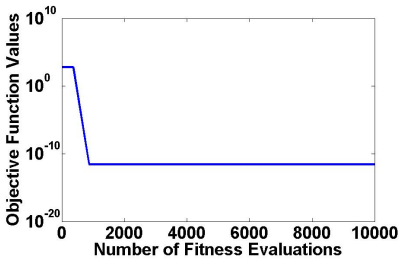
(a) C10



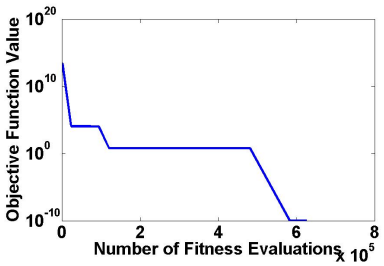
(b) C14



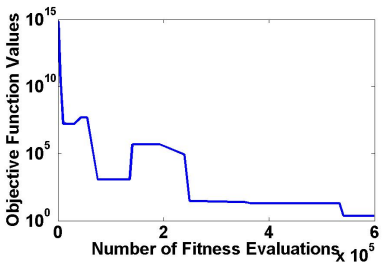
(c) C15



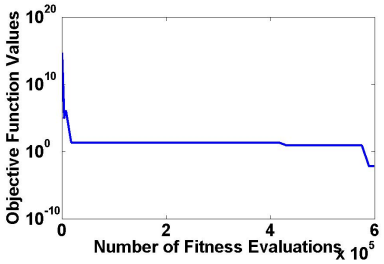
(d) C17



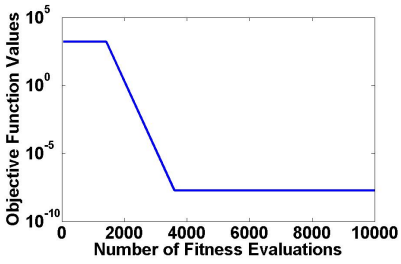
(e) C10



(f) C14



(g) C15



(h) C17

Figure 5: The convergence plots of the MCEA on C10, C14, C15 and C17. Plots (a)-(d) show the curves of the 10-D test problems; plots (e)-(h) shows the curves of the 30-D test problems.

5.6. Comparison on the CEC'10 Benchmark

In this section, we compared the MCEA with the winner of the CEC'10 competition, called ϵ-Constrained Differential Evolution (ϵ-DEg), on the CEC'10 test problems. The experimental configurations are the same as those used in CEC'06, except that the maximum NFEs are 200,000 for 10D, and 600,000 for 30D.

Table 6 summarises the statistics (including min, max and median) obtained by the two algorithms. The Wilcoxon rank sum is applied to carry out the hypothesis test at 5% significance level. In the table, we use notations "+", "-" and "$\sim$" to denote that the MCEA performs better, worse or similar to the ϵ-DEg in terms of solution quality.

From Table 6, it can be seen that for the 10D problems, the MCEA performs better than the ϵ-DEg on 8 problems; while the ϵ-DEg performs better on 4 test problems. For the rest problems, they perform similarly. For the 30D problems, the MCEA performs better than the ϵ-DEg on 12 test problems, and worse on 2 test problems. We may conclude that the MCEA performs better than the ϵ-DEg on average. Fig. 5 shows the convergence plots of the MCEA on C_{10}, C_{14}, C_{15} and C_{17}, respectively.

Furthermore, we observed that the best solutions found by the MCEA are worse than those found by the ϵ-DEg on $C_{09,14,15}$ at 10D, and $C_{09,14}$ at 30D. However, the worst solutions found by the MCEA are better than the ϵ-DEg on $C_{09,14,15}$ at 10D. This indicates that the performance of the MCEA is more stable than that of the ϵ-DEg. However, the MCEA performs worse than the ϵ-DEg on C_{09} and C_{14} at 30D, but better on C_{15}.

So far, the same parameters used for the CEC'06 test problems were applied on the CEC'10 benchmarks. We suspect that the degeneration performance of the MCEA on $C_{09,14,15}$ is because that these parameters are not well configured. To justify, we run the MCEA on $C_{09,14,15}$ at 10D and 30D with different M and α values in search of the optimal settings. We found that the optimal settings for C_{09} at 30D are $M = n$ and $\alpha = 0.2$; for C_{14} at 10D and 30D are $M = n$ and $\alpha = 0.4$, for C_{15} at 10D are $M = n$ and $\alpha = 0.1$. The experimental results are summarised in Table 7. From Table 7, we see that with appropriate parameters, the MCEA's performances were significantly improved as suggested by the hypothesis test. Unfortunately, we cannot find a common parameter setting that is able to achieve quality performance for all benchmark

Table 6: The comparison between the developed algorithm and ϵ-DEg on the CEC'10 test problems.

Prob.	ϵ-DEg			MCEA			hypo.
	min	median	max	min	median	max	test
$D = 10$							
C01	-0.747310	-0.747310	-0.738039	-0.747310	-0.747310	-0.747310	+
C02	-2.277710	-2.263489	-2.209323	-2.248475	-2.210387	-2.174758	-
C03	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	$\sim$
C04	-0.000010	-0.000010	0.003319	-0.000010	-0.000010	-0.000010	+
C05	-483.610625	-483.610625	-483.610625	-483.610625	-483.610625	-483.610625	$\sim$
C06	-578.658607	-578.652619	-578.645017	-588.442757	-588.382059	-584.697207	+
C07	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	$\sim$
C08	0.000000	10.572854	10.941538	0.000000	0.000000	0.000000	+
C09	0.000000	0.000000	142.078336	0.000000	7.931077	29.736517	+
C10	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	$\sim$
C11	-0.001523	-0.001523	-0.001523	-0.001523	-0.001523	-0.001523	$\sim$
C12	-570.089884	-426.511353	-0.199246	-158.383888	-59.906748	-39.453016	-
C13	-68.429365	-68.429365	-49.678547	-68.429365	-65.578466	-63.500458	+
C14	0.000000	0.000000	10.844283	0.000000	0.000705	0.0658532	-
C15	0.000000	0.000000	4.497445	0.079061	3.971311	4.180293	-
C16	0.000000	0.083187	0.847118	0.000000	0.000001	0.000002	+
C17	0.000000	0.015067	0.603958	0.000000	0.000000	0.000001	+
C18	0.000000	0.000000	0.000000	0.000000	0.000000	0.000002	$\sim$
$D = 30$							
C01	-0.821724	-0.820803	-0.819459	-0.821884	-0.821884	-0.818056	+
C02	-2.180058	-2.151956	-2.131994	-2.247491	-2.223187	-2.199613	+
C03	28.673466	28.673467	28.673767	0.000000	0.000000	0.000000	+
C04	0.003207	0.007317	0.029206	-0.000003	-0.000003	-0.000003	+
C05	-453.965250	-446.129938	-443.650912	-483.610624	-483.610620	-483.610542	+
C06	-528.706201	-527.747125	-527.149398	-590.183769	-572.525932	-492.219758	+
C07	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	$\sim$
C08	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	$\sim$
C09	0.000000	0.000000	85.465484	94.993156	261.731772	516.596882	-
C10	32.354417	33.129880	35.365672	0.000000	0.000000	0.000000	+
C11	-0.000337	-0.000288	-0.000238	-0.000392	-0.000392	-0.000392	+
C13	-66.724791	-65.453406	-64.275217	-68.429365	-68.429236	-65.577742	+
C14	0.000000	0.000000	0.000000	3.169514	22.578798	28.623660	-
C15	21.603509	21.603763	21.603913	0.010589	2.612249	21.603421	+
C16	0.000000	0.000000	0.000000	0.000000	0.000000	0.000003	$\sim$
C17	0.261621	3.331664	18.665782	0.000000	0.000000	0.000000	+
C18	0.805405	39.033089	825.543126	0.000000	0.000000	0.000071	+

Table 7: Further results on test problems $C_{09,14,15}$ at 10D and 30D. The first three columns list the results obtained by the common parameters, while the last three columns list the resuts with the optimized parameters.

Prob.	MCEA			MCEA with optimized parameters			hypo.
	min	median	max	min	median	max	test
$D = 10$							
C09	0.000000	7.931077	29.736517	0.000000	0.000000	4.408181	+
C14	0.000000	0.000705	0.065853	0.000000	0.000000	0.000036	+
C15	0.079061	3.971311	4.180293	0.000000	0.001066	0.073362	+
$D = 30$							
C09	94.993156	261.731772	516.596882	0.000000	66.931544	85.609162	+
C14	3.169514	22.578798	28.623660	0.000000	0.000037	0.005925	+

problems.

6. Conclusion and Future Work

In this paper, we presented a constrained evolutionary algorithm by combining an estimation of distribution algorithm (EDA) and a classical local optimizer under a multi-cycled sequential memetic computing (SMC) structure. Such structure regards a complete EA as an operator, and connects it with a local optimiser sequentially. It clearly decouples the EA and the local optimizer. It also enables the learning from previous cycles to improve the search efficiency of the latter evolutionary searches. In the experiments, we studied the components of the developed EDA to investigate its exploration capability, and investigated the advantages of the proposed learning strategy. The developed algorithm was extensively compared against the winning algorithms in the CEC 2006 and 2010 competition. The comparison results suggest that the proposed algorithm outperforms the compared algorithms on these benchmarks.

From the experimental study, it can be seen that the most significant components that influence the algorithmic performance under the proposed framework are the exploration capability of the EA and the learning capability. The EA should not consider much about the exploitation, but it should be designed to realise quick and broad exploration. The learning mechanism should be able to learn from history to facilitate effective search.

In the developed working algorithm, a full-factorized probability distribution model was developed, where the variable interconnections are not considered. Since the vari-

able interactions have a significant effect on the difficulties of the optimisation problem, it can be expected that a more sophisticated probabilistic model should result in a better performance.

The guided mutation operator is used as the learning mechanism. Our analysis showed that the learning approach can be effective when the "solution path" condition holds, which may not be effective for those that do not hold. In the future, an online learning algorithm which can learn the salience of the variables will be conducted. This could make the learning more intelligent, and the learned knowledge could be more effective in guiding the evolutionary search to promising areas.

Acknowledgment

JS was supported by the Natural Science Foundation of China (No. 11301494, 61175063, and 61573279), and by the Key Science and Technology Project of Wuhan under Grant No. 2014010202010108. JS and JMG were also supported by Centre for Plant Integrative Biology (CPIB), BBSRC/EPSRC and BB/D019613/1. YZ was supported by the National Natural Science Foundation of China (No. 11301494).

The authors would like to thank Prof. W. Pedrycz, and the anonymous reviewers for their constructive and helpful comments.

References

[1] Aguirre, A., Rionda, S., Coello, C., and Lizárraga, G. (2003). *Use of Multiobjective Optimization Concepts to Handle Constraints in Genetic Algorithms*, volume 2723 of *Lecture Notes in Computer Science*, pages 573–584. Springer.

[2] Ali, M. and Zhu, W. (2013). A penalty function-based differential evolution algorithm for constrained global optimisation. *Computational Optimization and Applications*, 54:707–739.

[3] Bertsekas, D. (1996). *Constrained Optimization and Lagrange Multiplier Methods*. Athena Scientific, Cambridge, MA.

[4] Bonyadi, M., Li, X., and Michalewicz, Z. (2014). A hybrid particle swarm with a time-adaptive topology for constrained optimization. *Swarm and Evolutionary Computation*, 18:22–37.

[5] Bosman, P. A. N. and Thierens, D. (2000). Expanding from discrete to continuous estimation of distribution algorithms: The IDEA. In Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Merelo, J. J., and Schwefel, H.-P., editors, *Parallel Problem Solving from Nature – PPSN VI. Lecture Notes in Computer Science 1917*, pages 767–776.

[6] Brest, J., Zumer, V., and Maućec, M. (2006). Self-adaptive differential evolution algorithm in constrained real-parameter optimization. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 215 – 222, Vancouver, BC, Canada.

[7] Caraffini, F., Neri, F., and Picinali, L. (2014). An analysis on separability for memetic computing automatic design. *Information Sciences*, 265:1–22.

[8] Chen, X., Ong, Y.-S., Lim, M., and Tan, K. (2011). A multi-facet survey on memetic computation. *IEEE Transactions on Evolutionary Computation*, 15(5):591–607.

[9] Chung, C.-J. and Reynolds, R. (1996). A testbed for solving optimization problems using culture algorithms. In *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming*, pages 225–236, Cambridge, Massachusetts. MIT Press.

[10] Datta, R. and Deb, K. (2015). *Evolutionary Constrained Optimization*. Springer.

[11] Deb, K. and Datta, R. (2010). A fast and accurate solution of constrained optimization problems using a hybrid bi-objective and penalty funcation approach. In *Proceedings of the 2010 IEEE Congress on Evolutionary Computation*, pages 1–8, Barcelona. IEEE.

[12] Deb, K., Sinha, A., and Aravind, S. (2006). A population-based, parent centric procedure for constrained real-parameter optimization. In *Proceedings of the*

2006 IEEE Congress on Evolutionary Computation, pages 239–245, Vancouver, BC, Canada.

[13] Dhadwal, M., Jung, S., and Kim, C. (2014). Advanced particle swarm assisted genetic algorithm for constrained optimization problems. *Comput Optim Appl*, 58:781–806.

[14] Dong, N. and Wang, Y. (2014). A memetic differential evolution algorithm based on dynamic preference for constrained optimization problems. *Journal of Applied Mathematics*. http://www.hindawi.com/journals/jam/2014/606019/.

[15] Glover, F. and Laguna, M. (1997). *Tabu Search*. Kluwer Academic Publisher, Norwell, MA.

[16] Grahl, J. and Rothlauf, F. (2004). PolyEDA: Combining estimation of distribution algorithms and linear inequality constraints. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1174–1185.

[17] Gu, B., Sheng, V., Tay, K., Romano, W., and Li, S. (2015a). Incremental support vector learning for ordinal regression. *IEEE Transactions on Neural Networks and Learning Systems*, 26(7):1403–1416.

[18] Gu, B., Sheng, V., Wang, Z., Ho, D., Osman, S., and Li, S. (2015b). Incremental learning for ν-support vector regression. *Neural Networks*, 67:140–150.

[19] Hamza, N., Sarker, R., Essam, D., Deb, K., and Elsayed, S. (2014). A constraint consensus memetic algorithm for solving constrained optimization problems. *Engineering Optimization*, 46(11):1447–1464.

[20] He, X., Liu, C., Dong, H., Pan, J., and Yan, Q. (2014). Particle swarm optimization-based augmented Lagrangian algorithm for constrained optimization problems. *Journal of Software Engineering*, 8(3):169–183.

[21] Ho, P. and Shimizu, K. (2007). Evolutionary constrained optimization using an addition of ranking method and a percentage-based tolerance value adjustment scheme. *Information Sciences*, 177:2985–3004.

[22] Homaifar, A., Qi, C., and Lai, S. (1994). Constrained optimization via genetic algorithms. *Simulation*, 62(4):242–253.

[23] Hu, Z., Cai, X., and Fan, Z. (2014). An improved memetic algorithm using ring neighbourhodd topology for constrained optimization. *Soft Computing*, 18:2023–2041.

[24] Huang, V., Qin, A., and Suganthan, P. (2006). Self-adaptive differential evolution algorithm for constrained real-parameter optimization. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 17–24, Vancouver, BC, Canada.

[25] Iacca, G., Neri, F., Mininno, E., Ong, Y.-S., and Lim, M. H. (2012). Ockham's razor in memetic computing: Three stage optimal memetic exploration. *Information Sciences*, 188:17–43.

[26] Jiao, L., Li, L., Shang, R., Liu, F., and Stolkin, R. (2013a). A novel selection evolutionary strategy for constrained optimization. *Information Sciences*, 239:122–141.

[27] Jiao, L., Li, L., Shang, R., Liu, F., and Stolkin, R. (2013b). A novel selection evolutionary strategy for constrained optimization. *Information Sciences*, 239:122–141.

[28] Kukkonen, S. and Lampinen, J. (2006). Constrained real-parameter optimization with generalized differential evolution. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 207–214, Vancouver, BC, Canada.

[29] Larrañaga, P. and Lozano, J. A. (2002). *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academic Publishers.

[30] Le, T. V. (1995). A fuzzy evolutionary approach to constrained optimization problems. In *Proceedings of the second IEEE Conference on Evolutionary Computation*, pages 274–278, Perth. IEEE.

[31] Liang, J., Runarsson, T., Mezura-Montes, E., Clerc, M., Suganthan, P., Coello, C., and Deb, K. (2006). Problem definitions and evaluation criteria for the CEC 2006 special session on constrained real-parameter optimization. Technical report, Nanyang Technological University, Sigapore.

[32] Liang, J. and Suganthan, P. (2006). Dynamic multi-swarm particle swarm optimizer with a novel constraint-handling mechanism. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 124–129, Vancouver, BC, Canada.

[33] Lin, C.-H. (2013). A rough penalty genetic algorithm for constrained optimization. *Information Sciences*, 241:119–137.

[34] Lin, J.-Y. and Chen, Y.-P. (2011). Analysis on the collaboration between global search and local search in memetic computation. *IEEE Transactions on Evolutionary Computation*, 15(5):608–623.

[35] Lin, J.-Y. and Chen, Y.-P. (2012). When and what kind of memetic algorithms perform well. In *Proceedings of the 2012 IEEE World Congress on Computational Intelligence*, pages 1–8. IEEE.

[36] Maesani, A., Fernando, P., and Floreano, D. (2014). Artificial evolution by viability rather than competition. *PLoS ONE*, 9(1):e86831.

[37] Maesani, A. and Floreano, D. (2014). Viability principles for constrained optimization using a (1+1)-CMA-ES. In *Parallel Problem Solving from Nature*, volume 8672, pages 272–281.

[38] Maesani, A., Iacca, G., and Floreano, D. (2015). Memetic viability evolution for constrained optimization. *IEEE Transactions on Evolutionary Computation*, PP(99). DOI 10.1109/TEVC.2015.2428292.

[39] Mallipeddi, R. and Suganthan, P. (2010a). Ensemble of constraint handling techniques. *IEEE Transactions on Evolutionary Computation*, 14(4):561–579.

[40] Mallipeddi, R. and Suganthan, P. (2010b). Problem definitions and evaluation criteria for the CEC 2010 competition on constrained real-parameter optimization. Technical report, Nanyang Technological University, Singapore.

[41] Mezura-Montes, E., editor (2009). *Constraint-Handling in Evolutionary Optimization*, volume 198 of *Studies in Computational Intelligence Series*. Springer.

[42] Mezura-Montes, E., Velázquez-Reyes, J., and Coello, C. C. (2006). Modified differential evolution for constrained optimization. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 25–32, Vancouver, BC, Canada.

[43] Michalewicz, Z. and Janikow, C. (1991). Handling constraints in genetic algorithms. In Belew, R. and Booker, L., editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 151–157, San Mateo, California. Morgan Kaufmann Publishers.

[44] Michalewicz, Z. and Schoenauer, M. (1996). Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1):1–32.

[45] Neri, F. and Cotta, C. (2012). Memetic algorithms and memetic computing optimization: A literature review. *Swarm and Evolutionary Computation*, 2:1–14.

[46] Neri, F., Cotta, C., and Moscato, P., editors (2012). *Handbook of Memetic Algorithms*. Studies in Computational Intelligence. Springer.

[47] Nguyen, Q. H., Ong, Y.-S., and Lim, M. H. (2009). A probabilistic memetic framework. *IEEE Transactions on Evolutionary Computation*, 13(3):604–623.

[48] Oh, S., Ahn, C., and Jeon, M. (2012). Effective constraints based evolutionary algorithm for constrained optimization problems. *International Journal of Innovative Computing, Information and Control*, 8(6):3997–4014.

[49] Padhye, N., Mittal, P., and Deb, K. (2015). Feasibility perserving constraint-handling strategies for real parameter evolutionary optimization. *Comput Optim Appl*. DOI 10.1007/s10589-015-9752-6.

[50] Powell, D. and Skolnick, M. (1993). Using genetic algorithms in engineering design optimization with nonlinear constraints. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 424–431, San Mateo, CA. Morgan Kaufmann.

[51] Richey, M. (2010). The evolution of Markov Chain Monte Carlo methods. *The American Mathematical Monthly*, 117(5):383–413.

[52] Runarsson, T. and Yao, X. (2002). Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 4(3):284–294.

[53] Runarsson, T. and Yao, X. (2005). Search biases in constrained evolutionary optimization. *IEEE Transactions on Systems, Man and Cybernetics – Part C: Applications and Reviews*, 35(2):233–243.

[54] Simionescu, P., Beale, D., and Dozier, G. (2004). Constrained optimization problem solving using estimation of distribution algorithms. In *2004 Congress on Evolutionary Computation*, volume 1, pages 296–302.

[55] Spellucci, P. (1998). An SQP method for general nonlinear programs using only equality constrained subproblems. *Mathematical Programming*, 82:413–448.

[56] Sun, J., Garibaldi, J., and Kenobi, K. (2012). Robust Bayesian clustering for datasets with repeated measures. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 9(5):1504–1514.

[57] Sun, J., Garibaldi, J., Krasnogor, N., and Zhang, Q. (2013). An intelligent multi-restart memetic algorithm for box constrained global optimization. *Evolutionary Computation Journal*, 21(1):107–147.

[58] Sun, J. and Kaban, A. (2010). A fast algorithm for robust mixtures in the presence of measurement errors. *IEEE Transactions on Neural Networks*, 21(8):1206–1220.

[59] Sun, J. and Keates, S. (2013). Canonical correlation analysis on data with censoring and error information. *IEEE Transactions on Neural Networks and Learning Systems*, 24(12):1909 – 1919.

[60] Takahama, T. and Sakai, S. (2006). Constrained optimization by the ε constrained differential evolution with gradient-based mutation and feasible elites. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 308–315, Vancouver, BC, Canada.

[61] Tasgetiren, M. F. and Suganthan, P. (2006). A multi-populated differential evolution algorithm for solving constrained optimization problems. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 33 – 40, Vancouver, BC, Canada.

[62] Tsai, H.-C. (2014). Integrating the artificial bee colony and bees algorithm to face constrained optimization problems. *Information Sciences*, 258:80–93.

[63] Tsutsui, S., Pelikan, M., and Goldberg, D. (2001). Evolutionary algorithm using marginal histogram models in continuous domain. In *Proceedings of the 2001 Genetic and Evolutionary Computation Conference Workshop*, pages 230–233, San Francisco, CA.

[64] Umbarkar, A., Joshi, M., and Sheth, P. (2015). Dual population genetic algorithm for solving constrained optimization problems. *Intelligent Systems and Applications*, 2:34–40.

[65] Wah, B. and Chen, Y. (2001). Hybrid constrained simulated annealing and genetic algorithms fornonlinear constrained optimization. In *Proceedings of the 2001 Congress on Evolutionary Computation*, volume 2, pages 925–932.

[66] Zhang, Q. and Mühlenbein, H. (2004). On the convergence of a class of estimation of distribution algorithms. *IEEE Transactions on Evolutionary Computation*, 8(2):127–136.

[67] Zhang, Q., Sun, J., and Tsang, E. (2005). Evolutionary algorithm with the guided mutation for the maximum clique problem. *IEEE Transactions on Evolutionary Computation*, 9(2):192–200.

[68] Zhang, Q., Sun, J., Tsang, E., and Ford, J. (2003). Hybrid estimation of distribution algorithm for global optimisation. *Engineering Computations*, 21(1):91–107.

[69] Zheng, Y., Jeon, B., Xu, D., Wu, Q., and Zhang, H. (2015). Image segmentation by generalized hierarchical fuzzy c-means algorithms. *Journal of Intelligent and Fuzzy Systems*, 28(2):961–973.