*International Journal of Innovative Computing, Information and Control*
Volume **10**, Number **2**, April **2014**

ICIC International ©2014 ISSN 1349-4198
pp. 459–479

# ASYNCHRONOUS PARALLEL ALGORITHMS FOR STRATEGIC HYBRID SEARCHING BASED ON A MIXTURE GAUSSIAN MODEL

Said Mohamed Said and Morikazu Nakamura

Information Engineering Department
University of the Ryukyus
1 Senbaru, Nishihara, Okinawa 903-0213, Japan
saidy87@hotmail.com; morikazu@ie.u-ryukyu.ac.jp

Received January 2013; revised May 2013

Abstract. *Technically exploring a solution space in an effective way helps not only to find good quality solutions, but also to reduce computation time. This paper proposes an optimization technique that utilizes hybridization, strategic search, parallelization, and asynchronous cooperation. A master-slave topology has been formulated in which the master strategically sorts out portions of the search space in four phases with the help of a clustering algorithm and assumes the role of an estimation of distribution algorithm to model the solution distribution within the space using a Gaussian mixture model without variable dependency. The new algorithm models a solution distribution by considering not only the mean vector of clustered solutions obtained from previous searches, as per the continuous univariate marginal distribution algorithm, but also by including information about the quality of solutions. With sorted probability distributions assigned by the master, slaves use genetic algorithms to extensively explore the solution space. The effect of our proposal has been experimentally analyzed in continuous domains, and the resultant algorithm shows significant improvements both in finding relatively good solutions and in reducing computation time.*

**Keywords:** Evolutionary algorithms, Genetic algorithms, Estimation of distribution algorithms, Parallel processing, $k$-means clustering, Synchronous/asynchronous algorithms, Continuous population-based incremental learning, Continuous univariate marginal distribution algorithm

1. **Introduction.** The modern era of computational intelligence has seen much use of evolutionary algorithms (EAs) with great success [1, 2, 3]. The familiarity and efficiency of these algorithms is due to their underlying principles of natural inspiration and implementation simplicity. Hybrid algorithms used by modern researchers can explore the benefits of two or more EAs; hence, they have become important and more powerful tools in solving optimization problems [4].

Among the well-known classes of EAs are genetic algorithms (GAs), which are robust, easy to use, and applicable in diverse areas in machine learning, combinatorial optimization, and numerical optimization [5]. An abundance of successful research has been performed using estimation of distribution algorithms (EDAs), which were introduced as a novel EA technique by [6]. Both GAs and EDAs have been used with a great degree of success in a variety of problem domains. The work in [7] cites some examples.

Technological innovation has led to a constant need for increasingly powerful algorithms as the size and complexity of real life application problems has increased enormously. Together with intelligent algorithms, the evolution of computers with multiple processor cores provides an additional tool for "too expensive" problem-solving scenarios, through the use of parallel processing techniques. The schemes proposed in [8, 9] are examples of

algorithms utilizing this parallel processing power. However, for large-scale, complicated problems, further reduction in computation time is still required, and this requires other powerful techniques. Practical implementation of parallel processing can be achieved in either a shared memory or distributed memory architecture. The former uses multiple processor cores in a single processing unit and is characterized by the ability for all processors to access all memory as a single addressing space. The latter uses processors distributed in multiple processing units and requires a communication network to connect inter-processor memory.

Asynchronous algorithms have also replaced synchronous algorithms with the same goal of improving computation efficiency. These tools have been proved successful by many researchers, including [10, 11], who achieved a significant reduction of their algorithms computation times. With asynchronous algorithms, nodes operate at different rates and hence spend more time in an active mode than in a waiting mode.

This paper broadly analyzes the performance of architecturally formulated master-slave hybrid approaches to GAs and EDAs using different optimization techniques. This is a continuation of the research in [12, 13, 14]. The approach aims not only to obtain good quality solutions, but also to do so in a reasonably small computation time. The basic idea of this algorithm is to extensively explore the solution space by progressively sorting out the possible regions where optimal solutions are likely to occur. This search space monitoring and evaluation process is performed in four strategically arranged search phases, which are executed in a top-down fashion. The search is performed by slaves using GAs; search space exploration and supervision is done by the master using an EDA. The master's work of EDA probabilistic estimation relies on the solutions obtained by the slaves GAs during the process. This collaborative work is performed strategically in four different sequentially executed phases that aim to identify the exact location of the global optimal solution in a solution space.

In this paper, we improve the estimation of the distribution part of the master's role, one of the most important functions in our hybrid approach. In our previous work, we used the simplest probability distribution model for the continuous univariate marginal distribution algorithm (UMDAc), which works using only the mean of clustered solutions. The distribution was taken to be Gaussian without variable dependency. A slight modification of this distribution model is used in our new algorithm by considering not only each cluster's mean but also the quality of solutions from previous searches to model the probability distribution, taking into account that the global optimal solution might occur in a region far from the cluster center. This sort of enhanced distribution, which tries to avoid characteristics of non-promising solutions and at the same time to retain characteristics of promising solutions through learning, is a typical example of machine learning in relation to an EA.

Parallelization has also been improved. In our previous approach [12, 13, 14], all processes perform their roles synchronously. The synchronous execution allowed easier control of the master-slave collaboration and of the implementation of the optimization program, but at a cost of waiting delays. Therefore, in our new algorithm, we employ asynchronous execution. We have also parallelized the $K$-means clustering algorithm used in the master's EDA. Parallel dynamic $K$-means clustering classifies the optimal solutions sent from the slaves into $K$ groups. The number of groups, $K$, is dynamically determined in the algorithm.

The performance of our resultant algorithm is compared in this paper with results from [12, 13, 14] and other algorithms by using Real-Parameter Black-Box Optimization Benchmarking 2009 (hereinafter, BBOB) [16] with 24 continuous functions. We experimentally analyzed the contribution of every optimization technique used in our approach.

The combined benefits are experimentally observed in our resultant algorithm in terms of the quality of solutions and the algorithm's computation time.

In Section 2, we explain fundamental concepts, such as the basic evolutionary algorithms used in our approach. Section 3 gives a detailed explanation of our proposed scheme. The details of the experiments and an analysis of the results are reported in Section 4. Section 5 concludes the paper by giving some remarks about the algorithm and the results.

## 2. **Preliminaries.**

2.1. **Mathematical notations and function optimization.** $X_i$ is used to denote a random variable. An instance of $X_i$ is denoted by $x_i$. $\mathbf{X} = (X_1, X_2, \ldots, X_n)$ denotes an $n$-dimensional random variable, and $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is one of its instances. For a continuous variable $X_i$, we denote the density function of $X_i$ by $f(X_i)$. Since, for our case, all the variables in $\mathbf{X}$ are continuous, we denote the joint density function by $f(\mathbf{x})$.

In mathematics and computer science, function optimization is the process of finding the best solution from all feasible solutions to minimize or maximize a specific function. Function optimization can be divided into two categories, depending on whether the variables are continuous or discrete. The case with discrete variables is known as combinatorial optimization; it is continuous function optimization if the variables are continuous. In this paper, we perform minimization of continuous functions; the standard form of optimization is

*Minimize* $f(x)$
*Subject to* $g_i(x) \leq 0,\ i = 1, \ldots, m$
$h_i(x) \leq 0,\ i = 1, \ldots, p$

where

- $f(x) : \mathbb{R}^n \to \mathbb{R}$ is the **objective function** to be minimized over the variable $x$,
- $g_i(x) \leq 0$ are called **inequality constraints**, and
- $h_i(x) \leq 0$ are called **equality constraints**.

2.2. **Evolutionary algorithms.** GAs are population-based EAs that simulate genetic selection and natural elimination processes of biological evolution. Holland [17] conducted pioneering work in this field. GAs are useful for searching very general or poorly defined spaces [18]. The three main components of GAs are crossover, mutation, and selection. The pseudocode in Figure 1 summarizes the basic structure of GAs.

With recombination operators in GAs, good solutions can be easily destroyed, even with elitism. EDAs, as explained in [6, 19], have their theoretical foundation in probability theory. They perform iterative evolutionary computation without using any recombination operators. EDAs work by probabilistic estimation and sampling. The best solutions are selected from the population and a distribution is constructed from the selected solutions. Using the distribution, sampling is done to obtain new members of the population. The pseudocode of EDAs is shown in Figure 2.

The univariate marginal distribution algorithm (UMDA) was described by Mühlenbein in 1997 [20], who provides the theoretical foundation for the field of investigation in general and for the algorithm specifically. UMDA is the simplest model of EDA that does not involve dependency between variables. UMDA models a distribution by just calculating the average value of each gene representation in its relative position among all selected genotypes. Mühlenbein also introduced an incremental version of UMDA (IUMDA) that is described as equivalent to a population-based incremental learning (PBIL) algorithm [21]. The PBIL approach bridges the gap between machine learning and EAs; PBIL explicitly constructs an intensional description of the optima, expressed as a distribution on the solution space. Continuous PBIL (PBILc), as introduced by [22], uses a Gaussian

distribution to sample the population and updates the distribution from the population. By doing so, the memory of good regions of the solution space is inherited from generation to generation.

```
1: P ← GenerateInitialPopulation();
2: Evaluate(P);
3: while termination condition is not met do
4:    P' ← Recombine(P);
5:    P'' ← Mutate(P');
6:    Evaluate(P'');
7:    P ← Select(P, P'');
8: end while
9: Output the best solution found;
```

FIGURE 1. Pseudocode for GAs

```
1: P ← GenerateInitialPopulation();
2: Evaluate(P);
3: while termination condition is not met do
4:    P_sel ← ChooseFrom(P);
5:    p(x) ← EstimateProbabilityDistribution(P_sel);
6:    P ← SampleBasedOn(p(x));
7: end while
8: Output the best solution found;
```

FIGURE 2. Pseudocode for EDAs

3. **Asynchronous Hybrid Master-Slave Scheme.** Our algorithm is a master-slave parallel processing approach that hybridizes GA and EDA to explore the solution space in an asynchronous fashion using a specific strategy. A probabilistic estimation of EDA uses a Gaussian mixture model without variable dependency.

3.1. **Estimation on a mixture model without dependency.** For simple computations, univariate models are used in EDAs. These models assume that the $n$-dimensional joint probability distribution function is a product of $n$ univariate and independent probability distributions. For more powerful estimation, multiple-dependency models are used. These models show good performance for target problems with a complicated landscape, but their computational costs are very high.

In our approach, we use the mixture model and perform parallel computation based on master-slave cooperation. Each slave generates an initial population for the GA according to a simple model without dependency. However, the total system executes, in parallel, a search based on the mixture model.

The EDA plays the role of a master for efficient control of search by slaves; hence, the probabilistic estimation process has to be as accurate as possible to attain more efficient control. We focus on the Gaussian mixture model, with the master estimating a distribution of good solutions in the search space as a set of clusters and assuming that good solutions in each cluster follow a Gaussian distribution.

We consider, as the probabilistic model, the joint Gaussian probabilistic density function

$$f_l(\mathbf{X}) = \prod_{i=1}^{n} f_l(x_i) \tag{1}$$

where $f_l(x_i)$ denotes the sum of component distributions at the $l$-th iteration:

$$f_l(x_i) = \sum_{j=1}^{K_l} \pi_{l,j} f_l(x_i|j). \tag{2}$$

Here, $K_l$ is the number of mixture components and $\pi_{l,j}$ is the coefficient for mixing the $j$-th component. Since we do not assume any dependency among variables, $f_l(x_i|j) \equiv \mathcal{N}(x_i; \mu^l_{i,j}, \sigma^2_{i,j,l})$ – that is, the univariate Gaussian density function corresponds to the $j$-th component.

The EDA, which serves as the master process, starts with a clustering algorithm that partitions solutions are sent from the slaves into $k_l$ clusters. In the previous works [12, 13, 14], after clustering, the master estimates the mean probabilistic vector for every cluster using UMDAc as the Gaussian mixture model without dependency between variables. Let us denote the mean vector by

$$\mu^l_j = (\mu^l_{1,j}, \mu^l_{2,j}, \ldots, \mu^l_{n,j}), \tag{3}$$

$$\mu^l_{i,j} = \frac{1}{m^l_j} \sum_{y=1}^{m^l_j} x^{l,y}_{i,j}, \tag{4}$$

where $m^l_j$ is the total number of solutions in the $j$-th cluster and $x^{l,k}_{i,j}$ is the $y$-th component of the $i$-th solution in the $j$-th cluster at the $l$-th iteration. Therefore, in the previous works, the vector is modeled using only the vector $\mu^l_j$, which is just a cluster center, and the population initialization for the next search is done using $\mu^l_j$ and the standard deviation of solutions within a cluster.

In an actual case, the best solutions might occur far away from a central point; thus, in this version of our approach, we enhance our probability distribution vector by considering not only the mean of the best solutions in a cluster but also the quality of solutions from previous searches. Hence, Equation (4) is replaced by the following equation with enhanced probability distribution vector $\hat{\mu}^l_j$ instead of $\mu^l_j$:

$$\hat{\mu}^l_{i,j} = (1-\alpha).\mu^l_{i,j} + \alpha.(X^{best,1} + X^{best,2} - X^{worst}), \tag{5}$$

where $\alpha$ is a learning rate, $X^{best,1}$ and $X^{best,2}$ are the cluster's best and second-best individuals, respectively, $X^{worst}$ is a cluster's worst solution, and $\mu^l_{i,j}$ is a cluster-mean probabilistic vector calculated by Equation (4). Figure 3 illustrates the relationship between $\mu$ and $\hat{\mu}$ using a vector representation in a cluster $j$ space. The figure shows that the best solutions can occur very far from any cluster mean $\mu$. If population sampling is done around only $\mu$, then it may cause the algorithm to be somehow kept away from the real best solutions. The enhanced mean, $\hat{\mu}$ tends to shift away from the worst solution and to occur somewhere not very far from both the cluster mean and the best solutions; this not only causes the new sampled solutions to be of the best quality but also results in a speedy search. This idea of employing solution quality in the probability distribution estimation used in this work was inspired by the field of machine learning, where evolution is considered to be similar to a *learning by query* process [30]. In the formula, we use only the first two best solutions and the one worst solution to represent the solution quality, but these numbers can be varied with little impact.

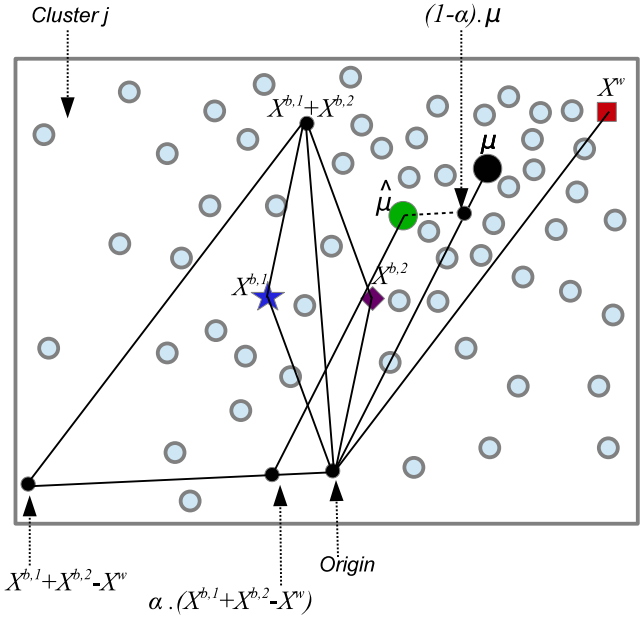


FIGURE 3. Representation of solution distribution and estimation in a cluster for the previous and present approaches

3.2. **Strategic master-slave framework.** The merits of a master-slave formulation are the following: i) the formulation is a simple transposition of the single-processor evolutionary algorithm onto a multiple-processor architecture, thus allowing reproducibility of results; ii) there is no permanent loss of information when a slave fails or is unreachable by the master; iii) the formulation is appropriate for networks of computers where availability is sometimes limited (e.g., available only during nighttime or when a screensaver is on), because nodes can be added or removed dynamically with no loss of information; and iv) the formulation consists of a centralized repository of the population which simplifies data collection and analysis, as elaborated in [23]. In changing our approach to be asynchronous, we maintained its basic master-slave architecture to retain the above-mentioned benefits. The whole algorithm runs in a fixed number of iterations, with the slaves independently performing GAs (i.e., no chromosome migration between slaves) while the master controls the GAs using the simplest model of EDA by instructing where each slave concentrates its search in the next iteration (i.e., the master probabilistically helps slaves to initialize their population members). The master process receives the best solutions from all slaves and stores them in database $DB$. These solutions are then clustered, and the master process calculates probabilistic estimation vectors $\mu$ from within the clusters and sends them to slaves as a message of "where to proceed" for their next search. In addition to imitating what the EDA does using already obtained best solutions, the master process uses a strategic control mechanism to effectively explore the search space. Figure 4 is a diagrammatic representation of the proposed scheme. Master needs to interact with database $DB$ to access the best solutions sent by slaves and has to follow a specific strategy to perform estimation. In the figure, $P.D.$ $K$-Means represents parallel dynamic $K$-means clustering, which is part of the EDA in our algorithm. Note that the role of the master is analogous to that of an EDA because the master generates probability vectors based on solutions already obtained and prepares a probabilistic distribution for the next-iteration population initialization. However, our method is different from EDAs in that it incorporates strategic search and parallel evolutionary computation instead of simple

solution sampling, as in the case of traditional EDAs. Although GAs are used as search algorithms by the slaves in our method, other population-based evolutionary algorithms can be used.

The order of probabilistic search space exploration (strategy) is of vital importance, as experimentally proved in [12]; hence the master performs this task in four sequentially arranged phases, as follows, to accomplish what we call "strategic search".

**Wide-range search (WRS)**: At the beginning of the search in early iterations, the GA population member initialization must be random to explore a much wider area of the search space (the location of the global optimum is entirely unknown).

**Outside clusters search (OCS)**: When there is a large enough number of solutions returned by the slaves GAs, parallel dynamic $K$-means clustering is performed. The results of clusters are used to form vectors $\mathbf{p}$ as per the EDA probabilistic model, calculated above as $\hat{\mu}$. In our past works cited above, UMDAc was used to model the vector as shown in Equation (4). In the present study, enhancements are made, and Equation (5) is used to calculate the probability distribution vector $\hat{\mu}$. The vector represents a specific area within the search space. In this phase, the master instructs slaves to perform initialization outside of clusters to avoid premature convergence (i.e., it uses complements of a determined vector). If $\mathbf{p} = (\hat{\mu}_{1,j}, \hat{\mu}_{2,j}, \ldots, \hat{\mu}_{n,j})$ is the probabilistic estimation vector of the $j$-th cluster, the master generates a complementary vector as $\mathbf{p} = (b - \hat{\mu}_{1,j}, b - \hat{\mu}_{2,j}, \ldots, b - \hat{\mu}_{n,j})$, where $b = \textit{upper limit of search domain if } \hat{\mu}_{i,j} > 0$, or $b = \textit{lower limit if } \hat{\mu}_{i,j} \leq 0 \textit{ in } [-b,\ b]$.

**Cumulative clustering (CC)**: In this phase, the same process as in OCS is used to get probabilistic vectors, but with an increased number of solutions to be clustered. Here, the master estimates new vectors at each iteration, and the number of solutions to be clustered in the current iteration is greater than in the previous iteration. The population initialization is done using a vector $\mathbf{p} = (\hat{\mu}_{\mathbf{1},\mathbf{j}}, \hat{\mu}_{\mathbf{2},\mathbf{j}}, \ldots, \hat{\mu}_{\mathbf{n},\mathbf{j}})$ (i.e., within the estimated area). Here, every slave is assigned a specific area to search within the larger search space.

**Best cluster focusing (BCF)**: Using the results of CC, the master can make a very good guess where the best solution lies. The qualities of solutions returned by all slaves that have been assigned by different clusters are compared. Hence, a final search is done by all slaves focusing on promising areas using only the $\mathbf{p}$ value that resulted in the best solutions.

In each of the phases described above, except WRS, every slave is assigned by one probability distribution vector; to initialize a new population for their respective GAs (sampling) they assume a Gaussian distribution with mean $\mathbf{p}$ and standard deviation calculated by the master for a cluster.

Regarding the master-slave model, some limitations are mentioned in [23], but our present approach accepts these limitations in favor of accuracy and algorithm performance (solution quality). When the master process is the backbone of the architecture, it enables better control of probabilistic estimation for the slave initialization process, and in case of failure the slaves will assume random initialization. According to previous studies, creating a network of slaves (migration of chromosomes) helps to boost search performance [24], but we tried to reduce algorithm complexity as well as communication overhead by completely removing inter-slave communication.

3.3. **Parallel processing.** In this research we have adopted a shared-memory parallel processing technique using a multi-core processing unit. We use master-slave cooperation to execute, in parallel, an asynchronous strategic hybrid algorithm using a Gaussian mixture model without variable dependency. We also perform, in parallel, dynamic $K$-means clustering in the EDA part of our algorithm during the solution space estimation

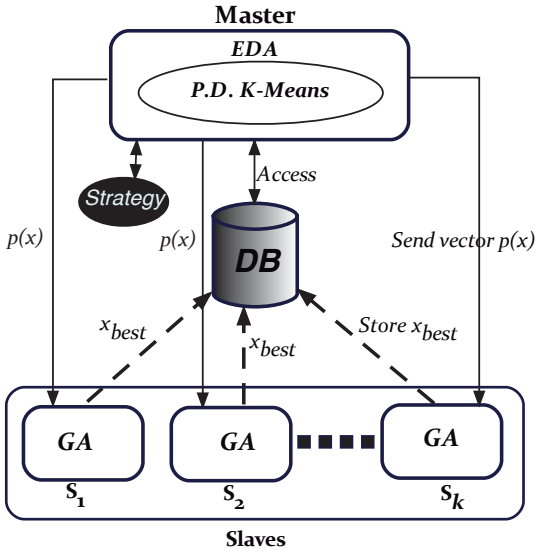


FIGURE 4. Hybrid model based on master-slave cooperation

process. However, on distributed memory systems, our asynchronous version can be efficiently implemented.

3.3.1. *Parallel dynamic K-means clustering.* In data mining, $K$-means clustering is a frequently used method for classifying data according to their particular characteristics [15]. Among its limitations is that it requires the number of clusters as an input. In an algorithm we used in [12], we specified the number of clusters to be the same as the number of slaves to make a one-to-one correspondence between the number of vectors and the total number of slave processes. To increase the efficiency of our algorithm, we decided to let the algorithm decide the optimal $K$ from a certain range, based on a validity measure. Various cluster validity measures have been proposed in [25, 26, 27, 28]; all aim to have clusters that are compact and well separated. In this research, we adopt the measure explained in [28], with the validity calculated as

$$validity = intra/inter \tag{6}$$

where *intra* denotes the total sum of all distances between points within a cluster and their respective cluster centers, and *inter* is the minimum of the distances between cluster centers. Because we want to maximize inter-cluster distances and minimize intra-cluster distances, the clustering module with the lowest value of the validity measure is the one that gives us the best value of $K$. We therefore executed several $K$-means algorithms, with the value of the number of clusters $K$ ranging from 2 to $2k-1$ where $k$ is the total number of slaves in the algorithm. By the validity obtained from each $K$-clustering module, the master process decides the best $K$. To compensate for the added complexity, these modules are executed in parallel. Hence, during clustering the master creates sub-master processes to perform dynamic clustering tasks concurrently. Figure 5 is a pseudocode fragment for dynamic $K$-means clustering.

After the clustering process, the master calculates the probabilistic vector and standard deviation of each cluster. All slaves need to initialize their populations for their GAs based on the probabilistic vectors returned by the master process. With the best $K$ determined in the above process, we might obtain a situation where $K$ does not correspond to the number of slaves (i.e., $k \neq K$). We might ask ourselves how the clusters (vectors) will

```
1: Access (DB);
2: for all K = 2...2k − 1 do
3:    C_K ←InitializeClusterCenters;
4:    repeat
5:       CalculateEuclideanDistance of solutions(DB);
6:       Group solutions to nearest cluster;
7:       C_K←RecalculateClusterCenters;
8:    until no change in clusters
9:    Calculate validity[K];
10:   Compare validity[K];
11: end for
12: Display K
13: Display cluster centers
```

FIGURE 5. Pseudocode for parallel dynamic $K$-means clustering

be assigned to each slave. We developed the simple algorithm shown by pseudocode in Figure 6 to take care of this kind of situation.

```
1: if K < k then
2:    Make more (K − k) copies of vectors from the best cluster and assign to remaining
      slaves;
3: else
4:    Use vectors only from the best k clusters.
5: end if
```

FIGURE 6. Pseudocode for probabilistic vectors assignment

3.3.2. *Synchronous vs. asynchronous models.* In a synchronized version of our approach all slaves performed GA searching for exactly the same number of generations during each iteration. When one slave finished its execution, it must wait for other slaves to finish so that all can start the next iteration at the same time. The master must also wait for the slaves to reach a specified number of iterations in order to start the estimation process. The purpose of this synchronization was to provide smooth timing and activity scheduling between the master and slaves. For example, the master knows the iteration in which it will start clustering and can order changing the phase after a specific number of iterations. As for the slaves, it is easy to determine the type of initialization (phase) they are going to use at a given iteration. Furthermore, a fixed number of iterations are observed in every phase (i.e., the total number of algorithm iterations is divided by 4 to get the number of iterations in each phase).

Figure 7(a) is an activity-time diagram that shows the sequence of events from the start to the end of the algorithm for the synchronous strategic hybrid approach. M and S represent synchronous master and slave processes, respectively. A round dot on the S lines indicates the starting point of synchronization (i.e., the time when the quickest slave has finished searching). Point $p$ marks the common restart of all slaves after the master's estimation. The phase change occurs at the dark dashed horizontal lines. Within the CC phase, synchronization and estimation occur at each iteration. The thick dark vertical lines on the M and S lines indicate time intervals when the master and slaves become active, respectively. The thin parts of the M and S lines are when they are idle. The master is inactive when it is waiting for slaves to complete their evolutionary computations

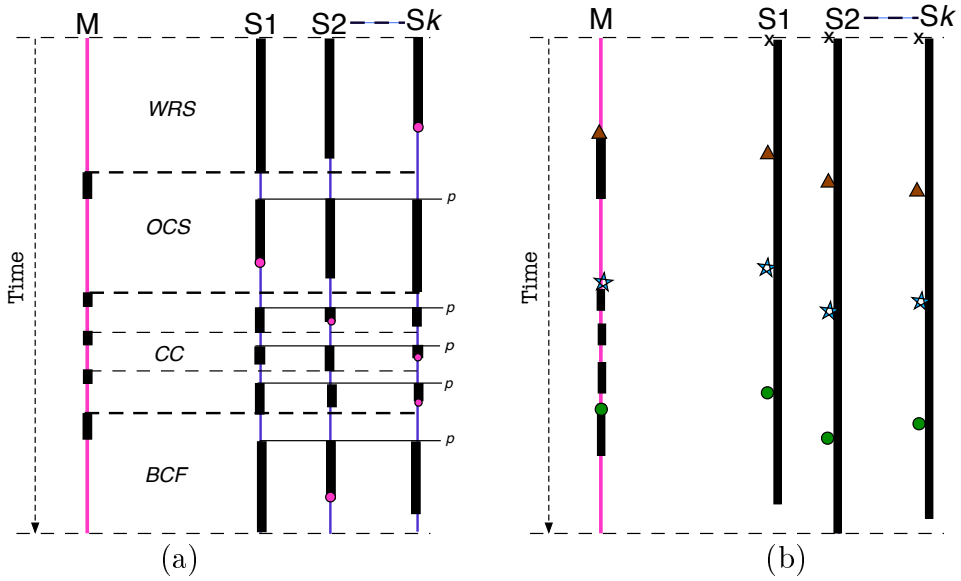


FIGURE 7. (a) Synchronous and (b) asynchronous activity-time sketch showing search, estimation, phases, and synchronization intervals

and then to return best solutions; slaves are idle when waiting for the master to complete estimation.

Our asynchronous version allows the slaves to operate at different speeds. If any slave meets the stopping criteria during the search process, it can terminate, regardless of the number of generations, and continue to the next initialization. In this algorithm, we also set a specific number of stored solutions to be reached for the master's estimation process to begin. Due to the difficulty of implementing the asynchronous version, we have set the following conditions to ensure a correct and effective estimation for the master as well as a smooth population initialization process for slaves.

(i) A slave can start the next phase even if the required estimation by the master has not finished. In this case, the best local optimal solution of the previous phase is used as an initialization vector.
(ii) The master must start the estimation process after a specified number of local optimal solutions are available in its database. This number must be smaller than the maximum number of available local optimal solutions during a complete phase of the synchronous version. This helps to reduce the size of the master's idle intervals.
(iii) Once the master has finished its estimation process, all the slaves must be in the new phase to use the newly calculated estimation vectors.

Figure 7(b) shows an activity-time representation of the asynchronous version, with M and S bearing the same meanings as in Figure 7(a). We have used symbols to mark the start of every phase for every slave: X for WRS, $\triangle$ for OCS, $\star$ for CC and $\bullet$ for BCF. For the master, the symbols mark the start of the estimation process for each respective phase. Slave $S_2$ is the slowest slave. The continuous thick dark vertical line on the S lines shows the complete removal of idle times for all slaves. On the M line we can see that estimation starts before some phases end. Early estimation ensures early completion of the next phase and decreases the master's idle time. The figure also shows that slaves finish searching at different points. Note that, time axis on two Figures 7(a) and 7(b) do not use the same time units, hence the algorithm in Figure 7(b) finishes execution earlier than in Figure 7(a).

The removal of wait times for slaves and the reduction in idle time for the master increase the algorithms search capability and mean that it completes its execution earlier than the previous synchronous model. With asynchronous parallelization, the implementation is easy on a distributed memory parallel processing platform using Message Passing Interface (MPI) programming; however, we should take into account the tradeoff with communication overhead.

```
1: for i = 0... MAXITERATIONS do
2:    Receive(p(x)) from master;
3:    P ← GenerateInitialPopulation(p(x));
4:    Evaluate(P);
5:    while Target fitness value is not reached do
6:       P' ← Recombine(P);
7:       P'' ← Mutate(P');
8:       Evaluate(P'');
9:       P ← Select(P, P'');
10:   end while
11:   Send the best solutions to master;
12: end for
```

FIGURE 8. Pseudocode for slaves

Figures 8 and 9 show pseudocode fragments that summarize the slave and master processes, respectively, in an asynchronous model. In the figures, $T$ means strategy and $x$ denotes a real gene of the chromosome in a population. In Figure 9, step 10 indicates the start of the EDA by using solutions stored in database $DB$. In step 13, the results of the clustering algorithm stored in database $DB$ are used to generate probabilistic vectors $(p_i(x))$ governed by strategy $T$.

```
1: Initialize(DB);
2: for i = 0...k − 1 do
3:    T ← WRS;
4:    p_i(x) ← GenerateProbabilityVector(T, DB);
5:    Send p_i(x) to Slave i;
6: end for
7: while Termination condition is not met do
8:    ReceiveSolutionFromSlaves(DB, i);
9:    if EnoughSolutions then
10:      EstimationOfDistribution(DB);
11:      T ← Strategy();
12:      Perform parallel K-means clustering(DB);
13:      p_i(x) ← GenerateProbabilityVector(T, DB);
14:      Send p_i(x) to Slave i;
15:   end if
16: end while
```

FIGURE 9. Pseudocode for master

4. **Experimental Evaluation.** In this work, we implemented our algorithm in the C programming language using the POSIX Threads (Pthreads) library on a Mac Pro computer (Intel Dual Quad Core: 8 cores in total; 3.0 GHz; 12 GB RAM). Implementation

is possible on any parallel-processing platform, such as OpenMP or MPI for a shared or distributed memory environment, respectively. The mentioned libraries are available for the C, C++, and Fortran programming languages.

To elaborate the significance of our proposed methodology, we compare our current algorithm with others to examine the following points:

i) Whether the asynchronous approach is more effective than the synchronous approaches in reducing overall computation time.
ii) Whether parallel processing is more effective than serial processing in reducing overall computation time.
iii) Whether the strategic hybrid algorithm improves solution quality and reduces computation time.
iv) Whether modification of the EDA probability distribution leads to improvement over the previous model.

In this experiment, nine algorithms were used to investigate the above-mentioned points. The performance analysis in this study was conducted by using the algorithms to solve 24 benchmark continuous functions used in BBOB. The algorithms used can be grouped into the following three categories.

- Parallel strategic hybrid algorithms, which include the synchronous hybrid algorithm (SH) from [13], the asynchronous hybrid approach using UMDAc estimation (AH) in [14], and the current version of the asynchronous hybrid algorithm whose EDA probability distribution model has been enhanced (eAH).
- Traditional parallel algorithms, which include the parallel continuous univariate marginal distribution algorithm (pUMDAc), parallel continuous population-based incremental learning (pPBILc), and parallel genetic algorithm (pGA).
- Traditional serial algorithms, which include the serial continuous UMDA algorithm (sUMDAc), serial continuous PBIL algorithm (sPBILc), and serial genetic algorithm (sGA).

4.1. **Functions and parameter settings.** We used all 24 functions in BBOB on a noiseless testbed; noiseless testbeds are described well in [16, 29]. Table 1 displays the definition of all functions with $D$ as dimension. For descriptions of $\mathbf{z}$, $f_{opt}$, $z$, $s_i$, $T_{osz}$, $f_{pen}(x)$, $\mathbf{T}$, $\mathbf{R}$ and $\mathbf{C}$, which are used in the function formulas, see [29]. The functions have been categorized into five subgroups: separable ($f_1$ – $f_5$), moderate ($f_6$ – $f_9$), ill-conditioned ($f_{10}$ – $f_{14}$), multi-modal ($f_{15}$ – $f_{19}$), and weakly structured multi-modal ($f_{20}$ – $f_{24}$). The dimensions 2, 3, 5, 10, 20 and 40 have been used for every function. The search space for all functions in all dimensions was limited to $[-5, 5]$.

The crossover and mutation probabilities used for GAs are 1 and 0.08, respectively. GAs used the stochastic remainder method for selection. In PBILc and eAH algorithms, the learning rate $\alpha = 0.009$ was used. 3 Pthreads were used to execute 3 slave processes, and 1 Pthread was used to execute the task of master and 1 slave (4 parallel Pthreads for 8 processor cores executing a master and 4 slave processes). We used three iterations (population re-initialization) in each tactic (phase), hence making a total of 12 sequentially-executed iterations in every instance. Due to the flexibility of asynchronous algorithms, phase change occurred at different times for different slaves. The clustering algorithm, which used values of $K$ from 2 to 7, was executed concurrently by 6 Pthreads created within master process. This $K$ range was selected after analysis showing that small $K$ values perform better when the data size is not huge. The numbers of local optimal solutions to be reached in the master's $DB$ before clustering and EDA estimation in the asynchronous algorithms were 400, 800, 1000, 1200 and 1400. For all algorithms, we maintained the maximum number of function evaluations for fair performance comparison.

TABLE 1. BBOB functions

| F | Name | Formula |
|---|---|---|
| $f_1$ | Sphere | $f_1(x) = \|\mathbf{z}\|^2 + f_{opt}$ |
| $f_2$ | Ellipsoidal | $f_2(x) = \sum_{i=1}^{D} 10^{6\left(\frac{i-1}{D-1}\right)} {z_i}^2$ |
| $f_3$ | Rastrigin | $f_3(x) = 10\left(D - \sum_{i=1}^{D}\cos(2\pi z_i)\right) + \|\mathbf{z}\|^2 + f_{opt}$ |
| $f_4$ | Büche-Rastrigin | $f_4(x) = 10\left(D - \sum_{i=1}^{D}\cos(2\pi z_i)\right) + \sum_{i=1}^{D} {z_i}^2 + 100 f_{pen}(x) + f_{opt}$ |
| $f_5$ | Linear Slope | $f_5(x) = \sum_{i=1}^{D} 5\lvert s_i\rvert - s_i z_i + f_{opt}$ |
| $f_6$ | Attractive Sector | $f_6(x) = T_{osz}\left(\sum_{i=1}^{D}(s_i z_i)^2\right)^{0.9} + f_{opt}$ |
| $f_7$ | Step Ellipsoidal | $f_7(x) = 0.1\max\left(\lvert\hat{z}_1\rvert/10^4, \sum_{i=1}^{D} 10^{2\frac{i-1}{D-1}} {z_i}^2\right) + f_{pen}(x) + f_{opt}$ |
| $f_8$ | Rosenbrock, original | $f_8(x) = \sum_{i=1}^{D-1}(100({z_i}^2 - z_{i+1})^2 + (z_i - 1)^2) + f_{opt}$ |
| $f_9$ | Rosenbrock, rotated | $f_9(x) = \sum_{i=1}^{D-1}(100({z_i}^2 - z_{i+1})^2 + (z_i - 1)^2) + f_{opt}$ |
| $f_{10}$ | Ellipsoidal | $f_{10}(x) = \sum_{i=1}^{D} 10^{6\frac{i-1}{D-1}} {z_i}^2 + f_{opt}$ |
| $f_{11}$ | Discus | $f_{11} = 10^6 {z_1}^2 + \sum_{i=2}^{D} {z_i}^2 + f_{opt}$ |
| $f_{12}$ | Bent Cigar | $f_{12}(x) = {z_1}^2 + 10^6 \sum_{i=2}^{D} {z_i}^2 + f_{opt}$ |
| $f_{13}$ | Sharp Ridge | $f_{13}(x) = {z_1}^2 + 100\sqrt{\left(\sum_{i=2}^{D} {z_i}^2\right)} + f_{opt}$ |
| $f_{14}$ | Different Powers | $f_{14}(x) = \sqrt{\left(\sum_{i=1}^{D} \lvert z_i\rvert^{2+4\frac{i-1}{D-1}}\right)} + f_{opt}$ |
| $f_{15}$ | Rastrigin | $f_{15}(x) = 10\left(D - \sum_{i=1}^{D}\cos(2\pi z_i)\right) + \|\mathbf{z}\|^2 + f_{opt}$ |
| $f_{16}$ | Weierstrass | $f_{16}(x) = 10\left(\frac{1}{D}\sum_{i=1}^{D}\sum_{k=0}^{D} 1/2^k \cos(2\pi 3^k(z_i + 1/2)) - f_0\right)^3 + \frac{10}{D} f_{pen}(x) + f_{opt}$ |
| $f_{17}$ | Schaffers F7 | $f_{17}(x) = \left(\frac{1}{D-1}\sum_{i=1}^{D-1}\sqrt{s_i} + \sqrt{s_i}\sin^2(50 {s_i}^{1/5})\right)^2 + 10 f_{pen}(x) + f_{opt}$ |
| $f_{18}$ | F7, condition 1000 | $f_{18}(x) = \left(\frac{1}{D-1}\sum_{i=1}^{D-1}\sqrt{s_i} + \sqrt{s_i}\sin^2(50 {s_i}^{1/5})\right)^2 + 10 f_{pen}(x) + f_{opt}$ |
| $f_{19}$ | Griewank-Rosenbrock F8F2 | $f_{19}(x) = \frac{10}{D-1}\sum_{i=1}^{D-1}\left(\frac{s_i}{4000} - \cos(s_i)\right) + 10 + f_{opt}$ |
| $f_{20}$ | Schwefel | $f_{20}(x) - \frac{1}{D}\sum_{i=1}^{D} z_i \sin(\sqrt{\lvert z_i\rvert}) + 4.189828872724339 + 100 f_{pen}(\mathbf{z}/100) + f_{opt}$ |
| $f_{21}$ | Gallagher 101-me Peaks | $f_{21}(x) = T_{osz}\left(10 - \max_{i=1}^{101} w_i \exp\left(-\frac{1}{2D}(x - y_i)^{\mathbf{T}}\mathbf{R}^{\mathbf{T}}\mathbf{C}_i\mathbf{R}(x - y_i)\right)\right)^2 + f_{pen}(x) + f_{opt}$ |
| $f_{22}$ | Gallagher 21-hi Peaks | $f_{22}(x) = T_{osz}\left(10 - \max_{i=1}^{21} w_i \exp\left(-\frac{1}{2D}(x - y_i)^{\mathbf{T}}\mathbf{R}^{\mathbf{T}}\mathbf{C}_i\mathbf{R}(x - y_i)\right)\right)^2 + f_{pen}(x) + f_{opt}$ |
| $f_{23}$ | Katsuura | $f_{23}(x) = \frac{10}{D^2}\prod_{i=1}^{D}\left(1 + i\sum_{j=1}^{32}\frac{\lvert 2^j z_i - \lfloor 2^j z_i\rceil\rvert}{2_j}\right)^{10/D^{1.2}} - \frac{10}{D^2} + f_{pen}(x)$ |
| $f_{24}$ | Lunacek bi-Rastrigin | $f_{24}(x) = \min\left(\sum_{i=1}^{D}(\hat{x}_i - \mu_0)^2, dD + s\sum_{i=1}^{D}(\hat{x}_i - \mu_1)^2\right) + 10\left(D - \sum_{i=1}^{D}\cos(2\pi z_i)\right) + 10^4 f_{pen}(x)$ |

4.2. **Results.** For every algorithm, in every function and every dimension, 15 trials are run to try to reach the target value $f_t = f_{opt} + \Delta f$. The **expected running time (ERT)** depends on $f_t$ and is computed over all relevant trials as the number of function

evaluations executed during each trial for which the best function value does not reach $f_t$, summed over all trials and divided by the number of trials that actually reach $f_t$ [16, 29]. The target value used to consider a trial a success is $f_t = f_{opt} + 10^{-8}$. The randomly generated value $f_{opt}$ was added to the target value in order to make it difficult for algorithms to guess the global optimum solution. Finally, $f_{opt}$ is eliminated, and if the obtained value is less than or equal to $10^{-8}$, then the trial is considered a success since all the functions are minimization problems with their global optimum at 0.

TABLE 2. Number of successfully solved functions (Max = 24)

| DIMENSION | 2 | 3 | 5 | 10 | 20 | 40 |
|---|---|---|---|---|---|---|
| SH | 20 | 11 | 9 | 3 | 1 | 0 |
| AH | 22 | 15 | 10 | 3 | 1 | 0 |
| eAH | **23** | **18** | **12** | **6** | **3** | **1** |
| pGA | 17 | 9 | 4 | 1 | 0 | 0 |
| pUMDAc | 5 | 3 | 2 | 1 | 1 | **1** |
| pPBILc | 8 | 4 | 4 | 1 | 1 | **1** |
| sGA | 20 | 12 | 5 | 1 | 0 | 0 |
| sUMDAc | 6 | 3 | 2 | 1 | 1 | **1** |
| sPBILc | 8 | 4 | 4 | 1 | 1 | **1** |

Table 2 summarizes the results on the ability of algorithms to reach the target value in all dimensions. Listed in the table is the number of functions for which an algorithm successfully reached the target value $f_t$. The table suggests that the ability of algorithms to solve functions decreases with an increase in dimension. Bolded values mark the highest number of successfully-solved functions in each dimension. The results also show that eAH is better able to solve many of the functions in all dimensions than other algorithms. Although the difference is not large between AH and SH algorithms, we will analyze the cause of this small difference in the following subsections. Non-strategic approaches generally gave poor performance, as shown by the table, since they were able to successfully solve only a few of the 24 functions in all dimensions. The algorithms pGA and sGA performed well in dimensions 2 and 3 compared with other non-strategic algorithms.

TABLE 3. Computation time percentages

| ALGORITHM | SH | AH | eAH | pUMDAc | pPBILc | pGA | sUMDAc | sPBILc | sGA |
|---|---|---|---|---|---|---|---|---|---|
| Computation Time (%) | 50.82 | 49.77 | 48.69 | 52.95 | 51.31 | 52.12 | 60.21 | 59.19 | 100 |

Fitness values of functions from each of 5 subgroups are presented in Table 4 for all algorithms in dimension 5. In the table, AVG is the average of the best fitness values obtained from 15 trials. MIN stands for the minimum fitness value among the best values in 15 trials, and STDEV is the standard deviation of the best fitness values. The bolded numbers mark the best of the best fitness values regarding minimization. The general indication from this solution quality table is that eAH outperformed all other algorithms for many functions in terms of obtaining high-quality solutions. Hybrid strategic searching gave notably better performance, outperforming traditional algorithms in both serial and parallel modes in all but a few cases.

4.2.1. *Serial versus parallel algorithms.* The aim of introducing parallelism has always been to improve computation time. The results obtained from our experiments are no

TABLE 4. Solution quality comparison in dimension 5

| ALG | | $f_1$ | $f_7$ | $f_{12}$ | $f_{19}$ | $f_{22}$ |
|---|---|---|---|---|---|---|
| SH | AVG | 3.5619E-09 | 5.8020E-02 | 1.8506E-01 | 8.6209E-02 | 5.1524E-02 |
| | MIN | 1.3195E-09 | 1.2031E-09 | 4.6595E-03 | 2.5394E-02 | 1.9348E-07 |
| | STDEV | 2.1450E-09 | 8.9749E-02 | 3.3509E-1 | 5.1667E-02 | 1.7745E-01 |
| AH | AVG | 2.5700E-09 | 8.4467E-02 | 7.5223E-02 | 1.2613E-01 | 2.2398E-03 |
| | MIN | **3.0880E-10** | 3.6630E-10 | 1.2058E-04 | 2.3562E-02 | 2.2508E-07 |
| | STDEV | 1.8572E-09 | 1.9095E-01 | 8.5335E-02 | 6.1482E-02 | 6.4706E-03 |
| eAH | AVG | **2.5360E-09** | **1.9661E-03** | **1.5160E-02** | 7.0060E-02 | **8.9911E-05** |
| | MIN | 7.8040E-10 | **1.4970E-10** | **1.9397E-05** | 1.4944E-02 | 8.1141E-09 |
| | STDEV | 1.7561E-09 | 5.3856E-03 | 1.7336E-02 | 3.6636E-02 | 1.8403E-04 |
| pGA | AVG | 2.14499E-08 | 1.0424E-01 | 8.4906E-02 | 1.5332E-01 | 2.4834E-03 |
| | MIN | 3.5901E-09 | 9.9537E-03 | 2.5819E-03 | 6.7322E-02 | **2.8590E-09** |
| | STDEV | 2.2057E-08 | 1.0271E-01 | 7.7475E-02 | 5.6923E-02 | 6.7231E-03 |
| pUMDAc | AVG | 1.5817E-04 | 1.7513E+00 | 3.0356E+01 | 2.1024E-02 | 1.7150E+01 |
| | MIN | 7.3161E-05 | 3.7411E-08 | 6.9705E+00 | 6.6291E-03 | 8.7747E-07 |
| | STDEV | 7.4731E-05 | 1.6203E+00 | 1.4132E+01 | 5.6148E-03 | 1.6666E+01 |
| pPBILc | AVG | 1.0873E-04 | 3.5251E-01 | 4.7317E+01 | 3.1755E-03 | 1.4667E+01 |
| | MIN | 1.3532E-05 | 5.9970E-10 | 5.9737E+00 | 4.3874E-04 | 1.1461E-08 |
| | STDEV | 1.0447E-04 | 9.7806E-01 | 3.2899E+01 | 1.7521E-03 | 1.6814E+01 |
| sGA | AVG | 4.6528E-09 | 1.1643E-01 | 1.4250E-01 | 1.0919E-01 | 1.2684E-03 |
| | MIN | 3.8410E-10 | 5.8984E-03 | 3.0856E-03 | 2.6584E-2 | 4.6275E-08 |
| | STDEV | 2.6029E-09 | 1.3988E-01 | 2.2171E-01 | 5.0943E-02 | 3.9949E-03 |
| sUMDAc | AVG | 2.1956E-04 | 2.7514E+00 | 3.7565E+01 | 2.0360E-02 | 1.7062E+01 |
| | MIN | 1.2584E-04 | 1.3055E-01 | 4.5708E+00 | 7.9231E-03 | 9.7095E-07 |
| | STDEV | 6.5895E-05 | 3.0673E+00 | 1.9363E+01 | 5.2269E-03 | 1.6595E+01 |
| sPBILc | AVG | 1.8239E-04 | 2.6681E-01 | 5.2831E+01 | **2.1273E-03** | 1.4055E+01 |
| | MIN | 1.0757E-04 | 1.5362E-09 | 1.2774E+01 | **4.4938E-05** | 1.9641E-08 |
| | STDEV | 7.4661E-05 | 7.8501E-01 | 2.7192E+01 | 2.3145E-03 | 1.7116E+01 |

exception. Parallel versions of GA, UMDAc, and PBILc have managed to reduce computation time by 47.88%, 12.06% and 13.31%, respectively, as shown in Table 3, which lists the overall computation time needed to complete algorithm execution for all 24 functions across all dimensions as a percentage of the slowest algorithm (i.e., sGA). From Tables 2 and 4, it is clear that there is no significant difference in solution quality between serial and parallel versions of these three algorithms. Therefore, we introduced the idea of the strategic parallel approach in our work in order to improve the quality of solutions while at the same time reducing computation time.

4.2.2. *Strategic versus non-strategic parallel approaches.* The introduction of strategies as explained in Section 3.2 has played a very big role in improving the quality of solutions as well as in reducing computation time. Table 4 shows that SH, AH and eAH obtained the best average fitness values for most cases in comparison to other algorithms. Table 2 also shows that strategic algorithms outperformed non-strategic algorithms in terms of their ability to reach the target value, except in dimension 40 for the UMDAc and PBILc algorithms. Another significant difference that can be seen from our experimental results is that every strategic approach finished execution earlier than any non-strategic approach.

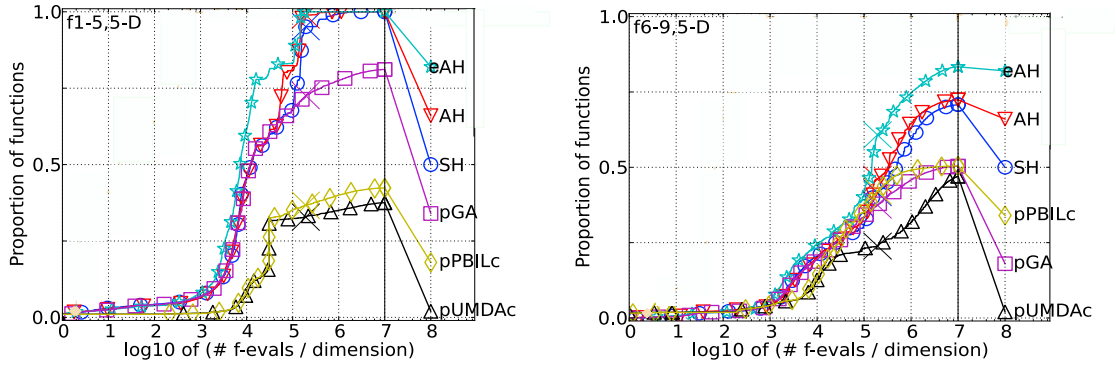


FIGURE 10. Bootstrapped empirical cumulative distribution of the number of objective-function evaluations divided by dimension (FEvals/D) for 50 targets in $10^{[-8..2]}$ for separable functions $f_1 - f_5$ (left subplot) and moderate functions $f_6 - f_9$ (right subplot) in 5-D

Figure 10 compares the performances of strategic and non-strategic algorithms using a bootstrapped empirical cumulative distribution in two subgroups of functions (separable and moderate) in dimension 5. The graphs express the proportion of solved problems rather than the reached function values; therefore, it is possible to meaningfully aggregate many graphs for several functions of the same class into one graph. The graphs show the ratio of the number of function evaluations in which the algorithms successfully reached target values to the number of function evaluations divided by dimension. Therefore, not only the ability to reach target values was considered but also the ability to reach them in a specified number of function evaluations. Higher proportions of function evaluations reaching the target values are noted for all strategic algorithms (SH, AH and eAH) compared with non-strategic algorithms (pGA, pUMDAc and pPBILc) for different numbers of function evaluations in both subplots.

4.2.3. *Performance comparison between synchronous and asynchronous algorithms.* After we introduced the idea of strategies to our parallel algorithms, we moved forward, looking for further reduction of computation time and improvement of solution quality. Since all of our parallel algorithms were synchronous parallel master-slave algorithms with the master and slaves waiting for each other to finish some tasks before starting others, we decided to introduce asynchronous evaluation, hoping for even more reduction in computation time. The results show no major difference in terms of solution quality; however, asynchronous parallel strategic algorithms (AH and eAH) were slightly better than the synchronous parallel strategic algorithm (SH). These results are shown in Tables 2 and 4.

Figure 11 shows a plot of the ratio of ERT for AH to ERT for SH versus $\log_{10}(\Delta f)$. Each subplot is for one function, which represents one of the five subgroups of functions. Ratios less than 1 indicate an advantage of AH; smaller values are always better. The line becomes dashed when, for any algorithm, ERT exceeds threefold the median of the trialwise overall number of $f$-evaluations of the function by the same algorithm. Filled symbols indicate the best $\Delta f$-value achieved by an algorithm (ERT is undefined to the right). The dashed line continues as the fraction of successful trials of the other algorithm, where 0 means 0% and the $y$-axis limit is 100%, values below zero for AH. The line ends when no algorithm reaches $\Delta f$ anymore. The number of successful trials is given only if it was in $\{1 \ldots 9\}$ for AH ($1^{\text{st}}$ number) and non-zero for SH ($2^{\text{nd}}$ number). One star indicates significance at $p < 0.05$; otherwise, $p < 10^{-\#\star}$, with Bonferroni correction within

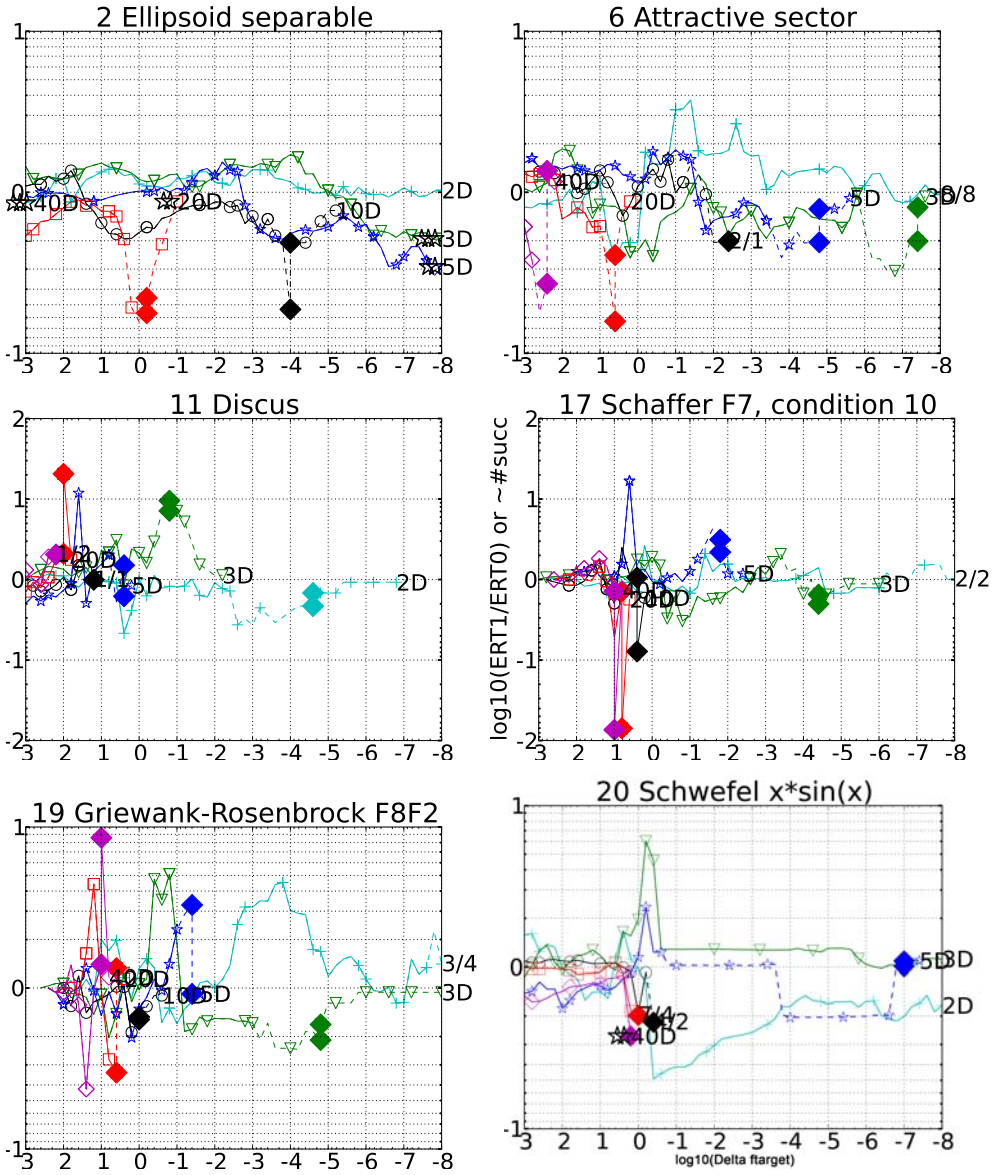


FIGURE 11. Ratio of ERT for AH to ERT for SH versus $\log_{10}(\Delta f)$ in dimensions 2:+, 3:▽, 5:⋆, 10:○, 20:□ and 40:◇

each figure. The figure shows the advantage of AH over SH for almost all the ratios in each function for all dimensions except dimension 5 in $f_7$ and dimension 40 in $f_{11}$.

CPU timing experiments were conducted to compare the computation speeds of SH and AH, by using function $f_8$ and restarted until at least 30 s. All experiments were conducted on a Mac Pro (Intel Xeon Dual Quad Core: 8 cores in total; 3.0 GHz; 12 GB RAM) running OS X 10.6.8. For SH, the results were 2.0; 2.0; 2.0; 2.0; 2.1, and $2.2 \times 10^{-5}$ s per function evaluation in dimensions 2; 3; 5; 10; 20 and 40, respectively. The results for AH were 1.8; 1.9; 2.0; 2.0; 2.0, and $2.0 \times 10^{-5}$ s per function evaluation in dimensions 2; 3; 5; 10; 20, and 40 respectively. Furthermore, the overall computation time for algorithm execution of all 24 functions in all dimensions was reduced by 2.07% for AH and by 4.19% for eAH as related to SH.

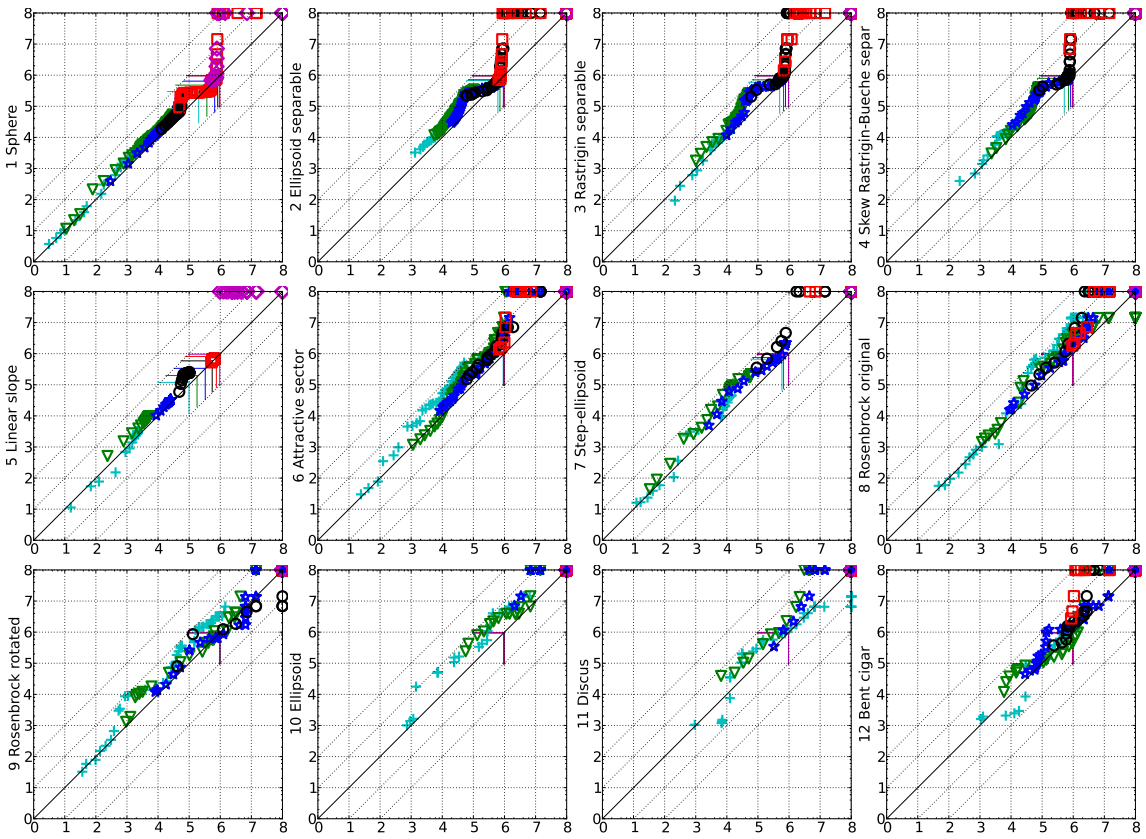


FIGURE 12. Expected running time (ERT in $\log_{10}$ of number of function evaluations) of eAH ($x$-axis) versus AH ($y$-axis) for 46 target values $\Delta f \in [10^{-8}, 10]$ in each dimension on functions $f_1 - f_{12}$. Markers on the upper or right edge indicate that the target value was never reached. Markers represent dimension: 2:+, 3:▽, 5:⋆, 10:○, 20:□, 40:◇.

4.2.4. *Performance of strategic hybrid approaches in different EDA models.* Since our strategic master-slave parallel algorithms rely on EDA probabilistic estimations and sampling for population initialization in late phases of searching, we tried to conduct experiments using two different EDA probabilistic estimation models and compare their performances. Both algorithms were asynchronous strategic parallel master-slave algorithms with one using the previous UMDAc to model distribution (AH) and the other using quality of solutions to enhance its distribution (eAH). Experimental results show that eAH slightly outperforms AH in terms of both solution quality and computation time. In Table 4, eAH has better AVG, MIN, and STDEV values in all presented functions than AH does. In terms of speed, eAH finished execution earlier than AH by 2.17%.

The search capabilities and speeds of AH and eAH are further compared using the data in Figures 12 and 13. In the figures, **ERT** (in $\log_{10}$ of the number of function evaluations) needed to reach certain target values for eAH ($x$-axis) and AH ($y$-axis) has been plotted for all 24 functions. From these figures we can see that it took a slightly larger number of function evaluations for AH to reach most of the targets in almost all functions than for eAH to do so. We can also spot that for almost all functions many markers are found on the upper edge rather than on the right edge, indicating that many of the target values were not reached by the AH algorithm.

Generally, it was easy for both algorithms to reach the target value $f_t$ in lower dimensions (2 and 3) for most of the functions. In higher dimensions (5–40), both AH and eAH

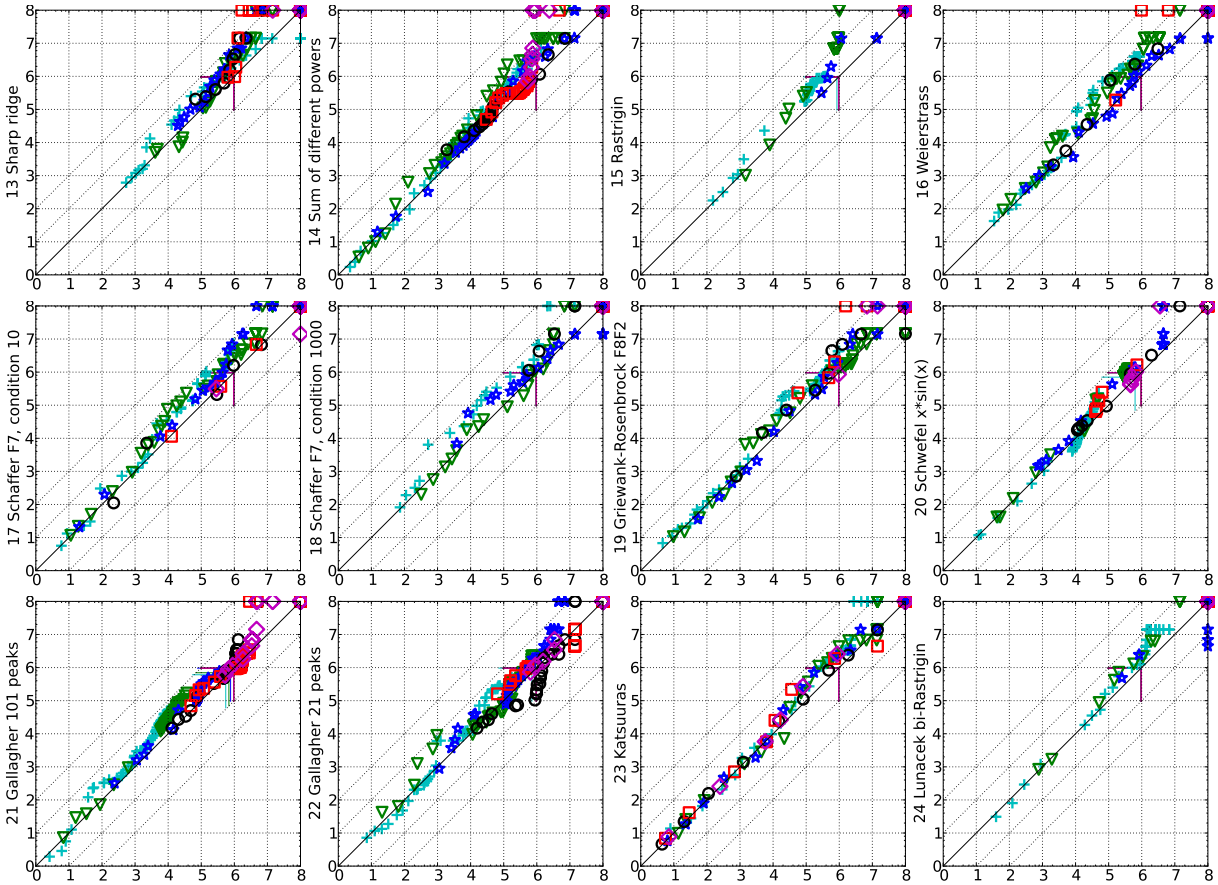


FIGURE 13. Expected running time (ERT in $\log_{10}$ of number of function evaluations) of eAH ($x$-axis) versus AH ($y$-axis) for 46 target values $\Delta f \in [10^{-8}, 10]$ in each dimension on functions $f_{13} - f_{24}$. Markers on the upper or right edge indicate that the target value was never reached. Markers represent dimension: 2:+, 3:▽, 5:⋆, 10:○, 20:□, 40:◇.

performed exceptionally well, with eAH outperforming AH. Furthermore, analysis of the results using the number of successful trials for all 24 functions indicates that functions from two subgroups, separable ($f_1 - f_5$) and weakly structured (especially $f_{20} - f_{22}$), were easily solved by both algorithms. For moderate functions, only $f_7$ was easily solved in dimensions 2-5. No algorithm was able to solve moderate functions in dimensions 10-40. Using all algorithms, the worst performance was observed for the ill-conditioned functions $f_{10} - f_{14}$ and the multimodal functions $f_{15} - f_{19}$, in which no algorithm reached the target value in any dimension for functions $f_{10}$ and $f_{18}$. However, AH and eAH did exceptionally well on these two subgroups by obtaining a good number of successful trials for some functions in dimension 2, with fewer successful trials in dimension 3.

5. **Concluding Remarks.** In this paper, we researched an effective way to explore a continuous search space using a strategic master-slave hybrid approach of GAs and EDA. Parallel processing techniques such as master-slave formulation and asynchronization were used to overcome the tradeoff between computation time and solution quality. The attempt was made by having the master progressively narrowing the search space to try to locate promising regions using a four-phase strategy. The modeling process within the regions is done using an enhanced UMDAc probabilistic distribution that includes information about the quality of solutions from previous searches. Using GAs, the slaves

explored the regions with estimated probability distributions to optimize some benchmark continuous functions. We performed progressive comparisons using nine different algorithms with BBOB continuous functions to try to analyze the effect of every technique used in our resultant approach. The combination of parallel processing, strategic searching, asynchronization and an enhanced probabilistic estimation model resulted in the powerful algorithm eAH, which not only improved solutions significantly but also cut the computation time by up to 51.31% in comparison with other algorithms. Moreover, apart from our general findings, we have seen that the ability of different algorithms differs between continuous functions as well as between subgroups of functions. For example, in Table 4, sPBILc was exceptional at finding the best values for function $f_{19}$, outperforming all other algorithms on this function.

In future work, we will focus mainly on strengthening the master part of our algorithm, and in particular, will replace the current univariate estimation models with multivariate models. We also plan to introduce automation into strategic search, which will involve automatic phase changes according to the search conditions. Since we plan to experiment with real-world application problems, the resultant algorithm will also need to have a fault-tolerance mechanism in case of the failure of either the master or slave processes.

## REFERENCES

[1] C. Li and S. Yang, An island based hybrid evolutionary algorithm for optimization, *Lecture Notes in Computer Science, Simulated Evolution and Learning*, vol.5361, pp.180-189, 2008.

[2] T. Nikam, B. Amiri, J. Olamaei and A. Arefi, An efficient hybrid evolutionary optimization algorithm based on PSO and SA for clustering, *JZU SC. A*, vol.4, pp.512-519, 2008.

[3] T. Tometzki and S. Engelli, Hybrid evolutionary optimization of two-stage stochastic integer programming problems, *Evolutionary Computation, Winter*, vol.17, no.4, pp.511-526, 2009.

[4] C. Grosan and A. Abraham, Hybrid evolutionary algorithms: Methodologies, architectures, and reviews, *Hybrid Evolutionary Algorithms*, pp.1-17, 2007.

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

[6] H. Mühlenbein and G. Paaß, From recombination in genes to the estimation of distribution I, Binary parameters, *Lecture Notes in Computer Science 1411: Parallel Problem Solving from Nature*, pp.178-187, 1996.

[7] L. Davis, *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991.

[8] J. Ocenasek, *Parallel Estimation of Distribution Algorithms*, Ph.D. Thesis, Brno University of Technology, 2002.

[9] A. Mendiburu-Alberro, *Parallel Implementation of Estimation of Distribution Algorithms Based on Probabilistic Graphical Models. Application to Chemical Calibration Models*, Ph.D. Thesis, The University of the Basque Country, 2006.

[10] S. S. Abdelwahed, M. F. Hassan and M. A. Sultan, Parallel asynchronous algorithms for optimal control of large scale dynamic systems, *J. Optimal Control Applications and Methods*, vol.18, no.4, pp.257-271, 1997.

[11] M. Golub and L. Budin, An asynchronous model of global parallel genetic algorithms, *The 2nd ICSC Symposium on Engineering of Intelligent Systems EIS2000*, pp.353-359, 2000.

[12] S. M. Said and M. Nakamura, A hierarchical hybrid evolutionary algorithm for continuous function optimization, *International Journal of Next Generation Computing*, vol.3, pp.13-28, 2012.

[13] S. M. Said and M. Nakamura, Parallel enhanced hybrid evolutionary algorithm for continuous function optimization, *Proc. of the 7th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, pp.125-131, 2012.

[14] S. M. Said and M. Nakamura, Asynchronous strategy of parallel hybrid approach of GA and EDA for function optimization, *Proc. of the 3rd International Conference on Networking and Computing ICNC'12, WCOP3*, pp.420-428, 2012.

[15] J. MacQueen, Some methods for classification and analysis of multivariate observations, *Proc. of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, pp.281-297, 1967.

[16] N. Hansen, A. Auger, S. Finck and R. Ros, Real-parameter black-box optimization benchmarking 2010: Experimental setup, *INRIA, no.RR-7215*, 2009.
[17] J. H. Holland, *Adaptation in Nature and Artificial Systems*, The University of Michigan Press, Reprinted by MIT Press, 1972.
[18] E. K. Prebys, The genetic algorithm in computer science, *MIT Undergrad. J. Math*, pp.165-170, 2007.
[19] P. Larrañaga and J. A. Lozano, *Estimation of Distribution Algorithms*, Kluwer Academic Publishers, 2002.
[20] H. Mühlenbein, The equation for response to selection and its use for prediction, *Evolutionary Computation*, vol.5, pp.303-346, 1997.
[21] S. Baluja, Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning, *Technical Report CMU-CS-94-163*, Carnegie Mellon University, 1994.
[22] M. Sebag and A. Ducoulombier, Extending population-based incremental learning to continuous search spaces, *Proc. of Parallel Problem Solving from Nature PPSN V*, pp.418-427, 1998.
[23] C. Gagne and M. Parizeau, A robust master-slave distribution architecture for evolutionary computations, *Proc. of Genetic and Evolutionary Computation Conference Late Breaking*, pp.80-87, 2003.
[24] Y. Gong and M. Nakamura, Migration effects of parallel genetic algorithms on line topologies of heterogeneous computing resources, *IECE Transactions on Fundamentals of Electronics, Communication and Computer Sciences*, vol.E91-A(4), pp.1121-1128, 2008.
[25] J. C. Bezdek and N. R. Pal, Some new indexes of cluster validity, *IEEE Trans. on Syst. Man. Cybern*, vol.28, pp.301-315, 1998.
[26] G. W. Milligan, Clustering validation: Results and implications for applied analyses, in *Clustering and Classification*, P. Arabie, L. J. Hubert and G. De Soete (eds.), Singapore, World Scientific, 1996.
[27] G. W. Milligan and M. C. Cooper, An examination of procedures for determining the number of clusters in a data set, *Psychometrika*, vol.50, pp.159-179, 1985.
[28] S. Ray and R. H. Turi, Determination of number of clusters in $K$-means clustering and application in colour image segmentation, *Proc. of the 4th International Conference on Advances in Pattern Recognition and Digital Techniques*, pp.137-143, 1999.
[29] N. Hansen, S. Finck, R. Ros and A. Auger, Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions, *INRIA, no.RR-6829*, 2009.
[30] T. M. Mitchell, *Machine Learning*, McGraw Hill, 1995.