Information Sciences 253 (2013) 34–55

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Performance analysis of the coarse-grained parallel model of the artificial bee colony algorithm

Alper Basturk [a,*], Rustu Akay [b]

[a] Dept. of Computer Engineering, Erciyes University, Kayseri 38039, Turkey
[b] Graduate School of Natural and Applied Sciences, Erciyes University, Kayseri 38039, Turkey

ARTICLE INFO

Article history:
Received 9 November 2012
Received in revised form 14 June 2013
Accepted 16 August 2013
Available online 26 August 2013

Keywords:
Artificial bee colony optimization algorithm
Global optimization
Parallel computing
Message passing interface

ABSTRACT

Despite the efficiency of evolutionary algorithms is prominent for large scale problems, their running times in terms of CPU time are quite large. Multi processing units served by recent hardware developments can be employed to overcome this drawback reducing the running time and sharing the total workload. However, evolutionary algorithms cannot be directly distributed to processing units due to their cooperative working models. These models need to be modified to be able to run them on distributed environments without causing deterioration in performance. In this study, a detailed performance analysis of a parallel model for the artificial bee colony algorithm, which is one of the recently developed swarm based evolutionary algorithms and a promising numerical optimization tool, is proposed. For this purpose large-scale benchmark problems are solved by the proposed model and also its original sequential counterpart model. The model is also applied to a real-world problem: training of neural networks for classification purposes. Comparative results show that the artificial bee colony algorithm is very suitable to use in parallel architectures since it has the ability to produce high quality solutions with small populations due to its perturbation operator. The proposed model decreases the running time in addition to improving the performance and convergence rate of the algorithm. It can be said that the speedup gained over its sequential counterpart is almost linear.

1. Introduction

Evolutionary algorithms (EAs) model natural evolution and are used to solve a wide range of problems [11]. Many various versions of EAs based on different models have been proposed in the literature [4,36].

The artificial bee colony (ABC) algorithm [15] introduced by Karaboga is a recent swarm-based evolutionary computation technique and models the foraging behavior of honey bees. The algorithm has been further studied by Karaboga and Basturk [9,20,2,21,17,1], and has been found to be a good optimization tool especially for high dimensional and multi-modal problems. The ABC algorithm combines exploration and exploitation in a balanced manner and has just a few parameters including the population size, the number of cycles and the control parameter limit which determines some solutions to be abandoned. Several modifications to the original ABC algorithm have been presented to improve its performance for certain specific problems [30,14,28,24,37]. Karaboga and Akay [17] compared the ABC algorithm with genetic algorithm, particle swarm optimization algorithm, differential evolution algorithm and evolution strategies. They pointed out the differences and similarities between the ABC algorithm and the others. Their study showed that the performance of the ABC algorithm

* Corresponding author.
E-mail addresses: ab@erciyes.edu.tr (A. Basturk), akay@erciyes.edu.tr (R. Akay).

0020-0255/$ - see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ins.2013.08.035

is better than or similar to those of other population-based algorithms with the advantage of employing fewer control parameters [17].

Like other EAs, the running time of the ABC algorithm can be reduced while solving large-scale problems by using parallel implementations. However, most of the population-based algorithms including the ABC algorithm work sequentially in nature. To make them work in a parallel manner and communicate the units working in parallel with each other, some modifications need to be introduced into the original algorithm without loss of efficiency.

In the literature, some recent parallel ABC algorithm models based on different parallelization strategies are available. The first parallel implementation of the ABC algorithm was used through a shared memory architecture [27]. In that study, a whole colony was divided into local populations, each of which has a local memory, and a global solution is stored in a global shared memory. The approach provided speedup while it did not change the performance of the original algorithm. Luo et al. proposed a ripple communication strategy and investigated the effect of high population sizes and also different migration sizes [23]. Subotic et al. assigned different threads to separate the subpopulations and proposed different types of communications among these subpopulations [32,33]. Benitez and Lopes presented two parallel approaches for the ABC algorithm: a master–slave and a hybrid-hierarchical approach [10]. The hybrid-hierarchical approach improved the quality of solutions while the master–slave approach did not produce satisfactory results. Three parallel models, which are the master–slave, multi-hive with migrations, and hybrid hierarchical, were examined in [29,6]. The multi-population approaches generated better quality with less computational effort. These studies introduced low speedups since they employed a relatively small number of CPUs. Another issue related to these studies is that they used low dimensional problems and did not analyze benchmark problems requiring high computational effort. Very recently, Basturk and Akay implemented a parallel synchronous ABC algorithm based on a master–slave model and the results showed that the proposed model is as efficient as the asynchronous version while it requires much less time to solve large problems [8].

To the best of our knowledge, any study on the coarse-grained model-based parallel ABC algorithm which investigates the effects of the number of subpopulations, migration interval and different migration topologies is not available in the literature yet. One of the aims of this study is to conduct such an analysis. The other purpose of the study is to employ the proposed parallel algorithm to a real-world problem in addition to large-scale benchmark problems. The real-word problem considered in the study is training of an artificial neural network (ANN). The ANN is a computing system that can learn on its own and has been widely used in many real-world applications. The design of the ANNs is a difficult and also an important optimization problem since the success of the ANNs depends greatly on the training process and the training algorithm used [35]. Many optimization algorithms have been applied to train the ANNs. The ABC algorithm has also been applied successfully for training the ANNs [16,18,19].

In this study, a coarse-grained model based parallel ABC algorithm (PABC) is implemented through the message passing interface (MPI) [12] among multi processing units. For this purpose, the whole population is divided into small size subpopulations and each subpopulation is evolved independently on a different core. The evolved subpopulations interact with each other and the information regarding a subpopulation migrates to its neighbors in some periods. Throughout the proposed study, some modifications such as the number of food sources in subpopulations, the migration interval, the migration size and migration topology which controls the amount of information exchanged among the subpopulations, are introduced to control the search behavior of the algorithm. The effect of these control parameters are analyzed to determine the parameter set that yields the best solutions in terms of solution quality and CPU time.

The performance of the PABC algorithm is compared to its sequential counterpart's to check the level of improvement. Results show that the modification in the original model provides a decrease in CPU time in addition to an improvement in solution quality. The reason for the decrease in CPU time is parallel implementation of the algorithm and distribution of processing load to multi processing units. Also, the reason for improved solution quality is that the ABC algorithm can work better with low size populations due to its perturbation operator which changes one parameter at a time.

The rest of the paper is organized as follows. Section 2 provides a description of the original sequential ABC algorithm. The proposed parallel implementation is explained in Section 3. Experimental results are reported in Section 4 in detail. Finally, conclusions and future research lines are provided in Section 5.

2. Artificial bee colony algorithm

The ABC algorithm is inspired by the intelligent foraging behavior of a honey bee colony in which bees try to maximize the nectar amount loaded into the hive by bees assigned a specific task. In the foraging behavior, there are three types of bees, each of which has a different search characteristic: these are employed bees, onlooker bees and scout bees. The employed bees are responsible for exploiting the sources in their memory whereas the onlooker bees go to exploit potentially rich sources depending on the information taken by communicating with the employed bees. Scout bees are responsible for exploring undiscovered sources by a random motivation or external clue. The ABC algorithm mimics this foraging behavior of honey bees in the context of a global optimization algorithm. From the perspective of a meta-heuristic, a population of solutions corresponds to the food sources to be discovered and the optimization task is to find the most profitable source by using the different search characteristics of the employed bees, onlooker bees and scout bees. Each type of bee is a phase of the algorithm as given in Algorithm 1. The block diagram of the algorithm is given in Fig. 1.

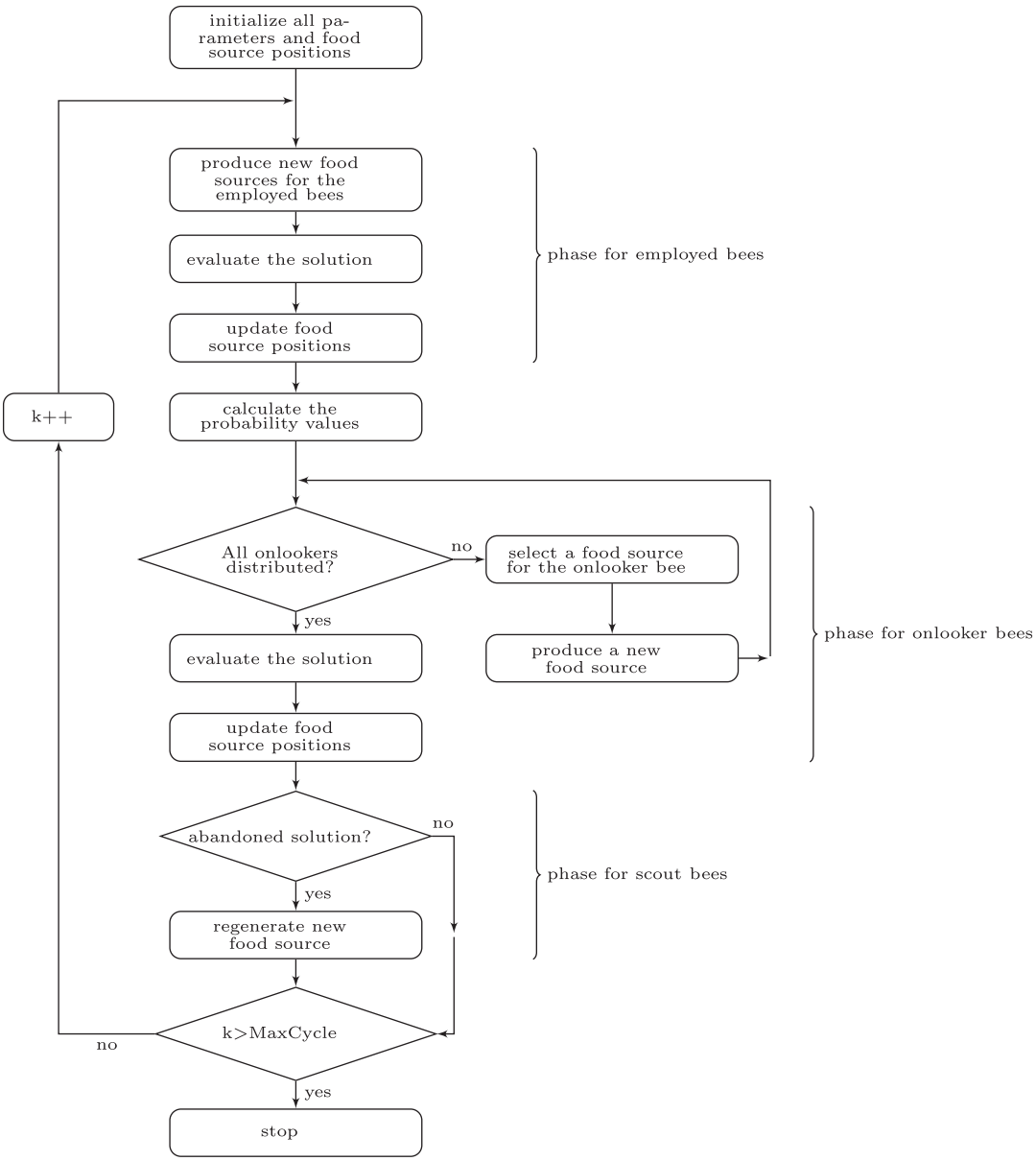

Fig. 1. Block diagram of the ABC algorithm.

In the initialization of the algorithm, a food source population is generated randomly using Eq. (1) within the range of parameters to be optimized:

Algorithm 1. Pseudo code of the basic ABC algorithm

1: Initialize the population of solution x_{ij}
2: Evaluate the population
3: Cycle = 1
4: **repeat**
5: Produce new solution x'_{ij} for the employed bees using Eq. (2)
6: Evaluate the population
7: Apply the greedy selection process between x_{ij} and x'_{ij}

8: Calculate the probability values p_i using Eq. (3)
9: Produce new solution x'_{ij} for the onlookers using Eq. (2) depending on p_i
10: Evaluate the population
11: Apply the greedy selection process between x_{ij} and x'_{ij}
12: Determine the abandoned solution, if exists, and replace it with a new randomly produced solution x_i for the scout using Eq. (1)
13: Memorize the best food source position achieved so far
14: Cycle = Cycle + 1
15: **until** Cycle = Maximum Cycle Number or time = Maximum CPU time

Algorithm 2. Pseudo code of the PABC algorithm

```
Initialize algorithm's constants, migration size and migration interval;
Generate m initial subpopulations pop_1, pop_2,...,pop_m;
for iter ← 1 to MaxCycle do
    Evaluate the population in pop_1, pop_2,...,pop_m;
    for i ← 1 to m do
        Evolve the pop_i;
    end
    if an interval of migration interval generations is reached then activate the migration
    procedure then
        for i ← 1 to m do
            For pop_i select best individuals from the other m − 1 subpopulation;
            Denote as set emigrants_i ;
            Set pop_i^n to pop_i ∪ emigrants_i ;
            Discard the worst individuals in pop_i^n ;
        end
    end
    Memorize the best individual;
end
```

$$x_{ij} = x_j^{min} + rand(0,1)\left(x_j^{max} - x_j^{min}\right). \tag{1}$$

In the phase for employed bees, for each solution, a new solution is produced in the neighborhood of the source by Eq. (2) as follows:

$$x'_{ij} = x_{ij} + \phi_{ij}(x_{ij} - x_{kj}) \tag{2}$$

where $k \in [1, CS]$ is a uniform random index and it has to different from i, CS is the number of food sources, $j \in [1, D]$ is the uniform random index, and D is the dimension of the problem. This perturbation strategy provides the solution to move towards promising regions of the search space. This is the positive feedback of the ABC algorithm. By a greedy selection, the new solution is kept in the memory and the current one is discarded if the new solution is better; otherwise, it is discarded, and the current one is retained in the population and a counter associated with the current solution is incremented by one in order to count the number of nonprogressive local searches in the neighborhood of the current solution. This counter will be used to determine the exhausted sources to be used in the phase for scout bees.

In the phase for employed bees, the local searches are conducted for all solutions in turn, while in the phase for onlooker bees they are carried out for the solutions chosen probabilistically. This probabilistic selection provokes high quality solutions to be chosen more, but also allows poor solutions to be selected. This selection scheme is also another positive feedback of the ABC algorithm. In the basic ABC algorithm, probabilistic selection uses the probability values p_i calculated by Eq. (3):

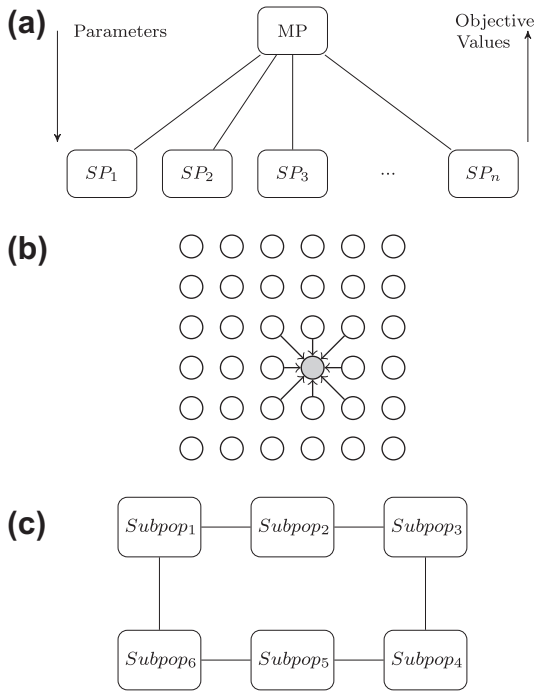

Fig. 2. Parallel computing models, (a) Master–slave, (b) Fine-grained, and (c) Coarse-grained.

$$p_i = \frac{fitness_i}{\sum_{i=1}^{CS} fitness_i} \tag{3}$$

where $fitness_i$ is the fitness value of the solution i.

Once an onlooker bee chooses a source, it searches the neighborhood of the solution by Eq. (2) and a greedy selection is applied to select a better one as in the phase for employed bees. Likewise, the counter holding the number of local searches is updated in this phase.

When the food sources are exploited through the searches in the phase for employed bees and the onlooker bees, some food sources may be exhausted as in nature, due to over exploitation. An exhausted source is determined by checking whether the associated counter holding the number of local searches in its neighborhood exceeds the control parameter limit. If a solution is labeled as exhausted, a new random solution (by Eq. (1)) is substituted for the solution. Abandoning is the negative feedback which counterbalances the positive feedback, and generating a random solution instead of the exhausted solution causes a fluctuation in the population of the ABC algorithm.

3. Coarse-grained parallel implementation of the artificial bee colony algorithm

The computation time of an algorithm can be reduced without distorting its performance through parallel computing environments [5]. EAs can be efficiently parallelized. The parallelization of an EA can be performed based on one of three well-known parallel computing models available in the literature or by hybridization of the models which are categorized depending on the synchronization and the communication behaviors of subtasks. These models are master–slave, fine-grained, and coarse-grained (subpopulation) models [3].

In the master–slave model which is shown in Fig. 2a, where MP is master processor and SP is slave processor, each objective function of an EA is evaluated on a different slave processor independently and a master collects information from all slaves to create offsprings based on the selection, recombination and mutation operators. The created offsprings are distributed to the slaves to be evaluated. The only communication between the slaves and masters occurs at sending the offsprings and receiving their fitness values [7,38]. Although the approach is easy to implement and to keep the load balance, the parallel speedup is less than ideal speedup.

The second model designed for working with massively parallel machines is called the fine-grained model, which is shown in Fig. 2b, and uses only one population. Parallelism and communication are realized within some individuals in the population based on a given topology which defines the neighborhood structure. The topology of the neighborhood and the number of individuals in the neighborhood affect the performance of the parallel algorithm.

In the last model, called the coarse-grained method which is shown in Fig. 2c, each subpopulation in one processor executes the algorithm independently and exchanges the results of the algorithm with other subpopulations by migrations performed in a certain migration interval based on a neighborhood topology using a certain migration size which is the number of individuals to migrate [25,31]. Although the communication overheads in this model are much less as compared to the other models, some new control parameters such as the number of individuals to migrate, the migration interval that controls the migration frequency, the selection/replacement of individuals, and the topology need to be set carefully.

All parallelization models have some advantages and disadvantages, but the coarse-grained method is probably the most preferred method among them due to its suitability for message passing parallel systems and its easiness to implement. While all parallelization models provide gain in the evaluation of expensive fitness functions such as the evaluation of real world problems, the coarse-grained model stands out amongst the others when the fitness function is computationally inexpensive. While the serial algorithm uses only one random seed at the initial step, the coarse-grained model uses a number of random seeds equal to the number of suppopulations. This mechanism increases the diversity of the population. So the enhancement of the exploration process on search space increases the probability to hit the optimal solution and improves the convergence rate. Another advantage of this model is that it can be well-adapted to many topologies (mesh, ring, hypercube, etc.). Furthermore, it needs less communication between nodes, so its efficiency may be better than those of other models. However, when an EA is parallelized based on the coarse-grained model, more control parameters are introduced to the approach, as mentioned before. In this model, the speedup may be predictable in systems with less communication. However, the speedup and performance of the algorithms based on the coarse-grained model may significantly change due to the differences in the behaviors of algorithms. Therefore, the relationship between migration interval, subpopulation size, and migration topologies are investigated for each optimization algorithm.

In our model, the coarse-grained parallelization model is used as shown in Fig. 2c to implement the parallel ABC algorithm. Each node in the structure, which runs the algorithm independently, is initialized by its own set and does not require a master node for the operations in the global population. The migration of individuals among subpopulations is performed in a certain interval to encourage the exchange of the experience of each independent algorithm among the subpopulations. The proposed model adopted some topologies such as ring, ring + 1 + 2, cube and grid. In the ring topology, the worst individual in the mth subpopulation is replaced with the best individual of each $(m-1)$ th subpopulation. In the ring + 1+2 topology the best individual of each $(m-1)$ th and $(m-2)$ th subpopulation is chosen. Cube topology has a tree connection with its neighbors, while grid topology has four connections with both its vertical and horizontal neighbors. These topologies have some advantages and disadvantages. Ring topologies possess an advantage in terms of communication efficiency, for example an m subpopulation ring only sends m messages at each migration time whereas a grid topology sends $4m$ messages. On the other hand, the convergence rate is different for each topology. The convergence is generally better when the subpopulations communicate more frequently compared to a situation which has less communication. The pseudo code of the PABC algorithm is shown in Algorithm 2 and the investigated topologies in this study are given in Fig. 3.

Some performance metrics to measure the performance of a parallel algorithm are available in the literature such as speedup and efficiency. Speedup is the ratio of sequential execution time to parallel execution time and the optimal value for the speedup is equal to the number of processors. Parallel efficiency is the ratio of speedup to number of processors and the ideal value for it is 1. If a parallel algorithm running on m processors runs m times faster than its sequential counterpart, then the efficiency equals 1, which is known as a linear speedup. Speedup and efficiency can be calculated by using Eqs. (4) and (5), respectively:

$$\text{speedup} = \frac{\text{execution time on a single processor}}{\text{execution time on } m \text{ processors}}, \tag{4}$$

$$\text{efficiency} = \frac{\text{speedup}}{m}. \tag{5}$$

4. Experimental results

The experimental setup for this study includes three parts. The first part compares the performances of ABC and PABC algorithms in terms of solution quality and investigates the control parameters of the PABC algorithm; the second part conducts a comparison in terms of computation times and the third part employs the proposed algorithm to train different ANN structures for classification purposes.

The first and the second part of the experiments are performed on CEC'2008 benchmark functions [34] and some other commonly used benchmark functions with different characteristics, which are specifically designed to benchmark large-scale optimization algorithms. Characteristics of these functions, such as multimodality, separability and parameter ranges are listed in Table 1, where the $z = x - o$, for some offset o, are shifted objective variables, and $b_1, \cdots, b_6$ are fixed biases. These variables are introduced to obtain more challenging problems that are randomly located, asymmetrical and multimodal. o defines the positions of global and local optima and b_i values define the global optimum [34].

The value of the "limit", which is a parameter of the ABC algorithm to determine exhausted sources and controls the change in the diversity of the population, is set to 2000. It means that if a solution cannot be improved for 2000 cycles, a new random solution is generated instead of it. The number of function evaluations is chosen to be $5000 \times D$ for all

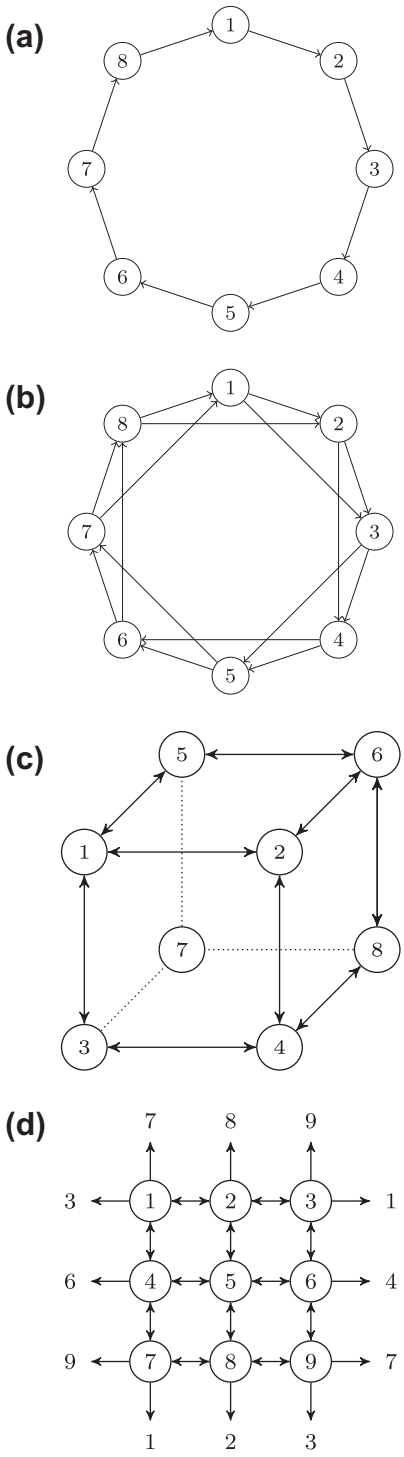

Fig. 3. Investigated migration topologies in the study, (a) Ring, (b) Ring + 1+2, (c) Cube, and (d) Grid.

algorithms and functions. Experiments are evaluated for the dimensions 100, 500 and 1000 and all the runs are repeated 30 times to compare the algorithms statistically. Function values less than 10^{-14} are assumed to be 0.

The proposed algorithm is implemented in C++ using the MPI library. Our experimental simulations are executed on AMD Opteron 6172 cores, and 128 GB 1333 DDR3 RAM computers having scientific Linux operating system at TUBITAK ULAKBIM High Performance Computing Center.

Table 1
Numerical benchmark functions used for comparison (C: Characteristic, U: Unimodal, M: Multimodal, S: Separable, N: Non-Separable, D: Dimension).

	Name	Bounds	C	Equation
1	Shifted Sphere	$[-100,100]^D$	US	$f_{cec1}(x)=\sum_{i=1}^{D} z_i^2 + b_1$
2	Shifted Schwefel 2.21	$[-100,100]^D$	UN	$f_{cec2}(x)=\max_i\{\lvert z_i\rvert + b_2, 1\leqslant i\leqslant D\}$
3	Shifted Rosenbrock	$[-100\,100]^D$	UN	$f_{cec3}(x)=\sum_{i=1}^{D-1}[100(z_{i+1}-z_i^2)^2+(z_i-1)^2+b_3]$
4	Shifted Rastrigin	$[-5,5]^D$	MS	$f_{cec4}(x)=\sum_{i=1}^{D}[z_i^2-10\cos(2\pi z_i)+10+b_4]$
5	Shifted Griewank	$[-600,600]^D$	MN	$f_{cec5}(x)=\frac{1}{4000}\sum_{i=1}^{D} z_i^2-\prod_{i=1}^{D}\cos\left(\frac{z_i}{\sqrt{i}}\right)+1+b_5$
6	Shifted Ackley	$[-32,32]^D$	MN	$f_{cec6}(x)=-20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} z_i^2}\right)-\exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi z_i)\right)+20+e+b_6$
7	Step	$[-100,100]^D$	US	$f_7(x)=\sum_{i=1}^{D}(\lfloor x_i+0.5\rfloor)^2$
8	Quartic	$[-1.28,1.28]^D$	US	$f_8(x)=\sum_{i=1}^{D} i x_i^4 + random[0,1)$
9	Schwefel 2.22	$[-10,10]^D$	UN	$f_9(x)=\sum_{i=1}^{D}\lvert x_i\rvert+\prod_{i=1}^{D}\lvert x_i\rvert$
10	Dixon Price	$[-10,10]^D$	UN	$f_{10}(x)=(x_1-1)^2+\sum_{i=2}^{D} i(2x_i^2-x_{i-1})^2$
11	Penalized	$[-50,50]^D$	MN	$f_{11}(x)=\frac{\pi}{D}\{10\sin^2(\pi y_1)+\sum_{i=1}^{D-1}(y_i-1)^2[1+10\sin^2(\pi y_{i+1})]$ $+(y_D-1)^2\}+\sum_{i=1}^{D} u(x_i,10,100,4)\quad y_i=1+\frac{1}{4}(x_i+1)$ $u(x_i,a,k,m)=\begin{cases}k(x_i-a)^m, & x_i>a\\ 0, & -a\leqslant x_i\leqslant a\\ k(-x_i-a)^m, & x_i<-a\end{cases}$
12	Penalized2	$[-50,50]^D$	MN	$f_{12}(x)=0.1\{\sin^2(\pi x_1)+\sum_{i=1}^{D-1}(x_i-1)^2[1+\sin^2(3\pi x_{i+1})]+(x_D-1)^2[1+\sin^2(2\pi x_D)]\}$ $+\sum_{i=1}^{D} u(x_i,5,100,4)$

Table 2
Function error values of the solutions obtained by PABC and ABC algorithms for the benchmark functions (NP = 120, D = 100).

	Function	f_{cec1}			f_{cec2}			f_{cec3}		
	CPU (s)	Mean	StdDev	Time (s)	Mean	StdDev	Time (s)	Mean	StdDev	Time (s)
ABC	1	**0**	0	1.1464	8.48E+01	2.79E+00	1.0204	1.06E+01	8.39E+00	1.5892
PABC	2	**0**	0	0.9156	7.13E+01	9.72E+00	0.8472	9.74E+00	9.75E+00	1.1712
	3	**0**	0	0.5944	5.25E+01	1.66E+01	0.5404	1.19E+01	1.37E+01	0.7524
	4	**0**	0	0.4376	4.81E+01	1.45E+01	0.3984	1.17E+01	1.12E+01	0.5504
	5	**0**	0	0.3532	3.48E+01	1.63E+01	0.3200	8.58E+00	6.72E+00	0.4476
	10	**0**	0	0.1764	5.27E+00	8.31E+00	0.1596	8.73E+00	8.66E+00	0.2228
	15	**0**	0	0.1200	2.87E+00	8.27E+00	0.1072	4.20E+00	1.42E+01	0.1484
	20	**0**	0	0.0912	**2.14E+00**	6.42E+00	0.0828	**3.12E−02**	1.45E−01	0.1144
		f_{cec4}			f_{cec5}			f_{cec6}		
ABC	1	2.01E+00	8.93E−01	5.5356	**0**	0	10.5568	1.25E−06	1.18E−06	5.5372
PABC	2	3.58E−01	5.54E−01	3.0980	**0**	0	5.7032	5.93E−07	1.96E−07	3.0928
	3	3.67E−09	3.57E−09	2.0416	**0**	0	3.7648	5.35E−07	2.01E−07	2.0544
	4	4.36E−07	9.56E−06	1.5192	**0**	0	2.8060	4.85E−07	2.84E−07	1.5232
	5	1.07E−09	2.93E−09	1.2144	**0**	0	2.2788	6.06E−07	3.53E−07	1.2292
	10	5.54E−10	1.24E−09	0.6092	**0**	0	1.1536	3.84E−07	3.55E−07	0.6104
	15	4.57E−10	1.93E−09	0.3972	**0**	0	0.7756	8.14E−08	1.20E−07	0.4016
	20	**1.18E−10**	4.01E−10	0.2964	**0**	0	0.5704	**8.89E−09**	1.74E−08	0.2916
		f_7			f_8			f_9		
ABC	1	**0**	0	1.0312	3.46E−01	4.64E−02	8.3464	7.92E−09	5.43E−09	0.9948
PABC	2	**0**	0	0.8480	3.05E−01	7.67E−02	5.6348	7.15E−09	3.26E−09	0.8900
	3	**0**	0	0.5504	2.83E−01	5.15E−02	3.1896	7.52E−09	2.72E−09	0.5816
	4	**0**	0	0.4088	2.46E−01	6.88E−02	2.4700	7.40E−09	2.92E−09	0.4304
	5	**0**	0	0.3220	2.46E−01	7.52E−02	2.2240	6.42E−09	3.53E−09	0.3376
	10	**0**	0	0.1612	1.46E−01	5.51E−02	1.0632	8.53E−09	7.09E−09	0.1672
	15	**0**	0	0.1104	8.14E−02	5.08E−02	0.6508	1.47E−09	1.86E−09	0.1140
	20	**0**	0	0.0832	**6.48E−02**	3.45E−02	0.5500	**2.20E−10**	3.22E−10	0.0872
		f_{10}			f_{11}			f_{12}		
ABC	1	4.04E−03	2.40E−03	1.0640	**0**	0	10.5580	**0**	0	9.1788
PABC	2	3.52E−03	2.61E−03	0.9264	**0**	0	5.7144	**0**	0	5.0472
	3	**3.20E−03**	2.41E−03	0.6036	**0**	0	3.8104	**0**	0	3.3588
	4	2.29E−02	9.76E−02	0.4472	**0**	0	2.8772	**0**	0	2.5148
	5	4.59E−03	7.10E−03	0.3552	**0**	0	2.2964	**0**	0	2.0336
	10	3.70E−01	2.04E−01	0.1760	**0**	0	1.2488	**0**	0	1.0904
	15	8.29E−02	1.82E−01	0.1216	**0**	0	0.7868	**0**	0	0.6956
	20	3.23E−01	2.36E−01	0.0932	**0**	0	0.6092	**0**	0	0.5188

Table 3
Function error values of the solutions obtained by PABC and ABC algorithms for the benchmark functions (NP = 120,D = 500).

	Function	f_{cec1}			f_{cec2}			f_{cec3}		
	CPU (s)	Mean	StdDev	Time (s)	Mean	StdDev	Time (s)	Mean	StdDev	Time (s)
ABC	1	**0**	0	25.9008	1.47E+02	4.91E+00	22.5136	2.94E+01	1.75E+01	37.0484
PABC	2	**0**	0	21.7876	1.29E+02	6.99E+00	19.8284	2.25E+01	1.77E+01	27.4868
	3	**0**	0	13.1108	1.16E+02	1.34E+01	11.8676	3.25E+01	2.81E+01	16.9996
	4	**0**	0	9.9664	1.05E+02	1.65E+01	8.9900	3.50E+01	2.58E+01	12.6888
	5	**0**	0	7.8460	9.18E+01	2.04E+01	7.0304	4.62E+01	3.72E+01	10.0952
	10	**0**	0	3.8996	2.40E+01	2.70E+01	3.4888	3.34E+01	3.80E+01	5.0056
	15	**0**	0	2.6416	8.45E+00	1.71E+01	2.4052	1.96E+01	3.80E+01	3.3960
	20	**0**	0	2.0020	**8.14E+00**	1.73E+01	1.7796	**1.63E+01**	4.42E+01	2.5332
		f_{cec4}			f_{cec5}			f_{cec6}		
ABC	1	3.35E+00	1.91E+00	135.7208	**0**	0	258.2264	2.12E−06	1.43E−06	134.3148
PABC	2	4.41E−01	1.06E+00	86.2980	**0**	0	162.2008	1.15E−06	3.67E−07	81.9508
	3	7.96E−02	2.70E−01	49.4464	**0**	0	94.5052	9.93E−07	4.72E−07	49.3076
	4	2.95E−09	2.24E−09	37.8920	**0**	0	69.7668	9.23E−07	4.21E−07	36.6544
	5	2.46E−09	1.81E−09	29.7524	**0**	0	55.4632	9.06E−07	5.03E−07	29.3552
	10	4.51E−09	5.16E−09	14.7192	**0**	0	27.9260	6.06E−07	7.72E−07	14.6476
	15	3.36E−09	8.37E−09	9.7228	**0**	0	19.7424	4.55E−07	1.24E−06	9.7388
	20	**1.94E−09**	9.50E−09	7.0936	**0**	0	14.3176	**6.28E−08**	2.10E−07	7.0940
		f_7			f_8			f_9		
ABC	1	**0**	0	20.7136	1.99E+00	1.85E−01	210.0348	3.93E−08	7.89E−09	21.8872
PABC	2	**0**	0	17.4424	1.27E+00	2.43E−01	119.1044	6.05E−08	1.17E−08	18.2272
	3	**0**	0	11.9344	1.11E+00	2.43E−01	77.9584	4.75E−08	1.57E−08	12.1964
	4	**0**	0	8.7572	8.96E−01	3.45E−01	64.4848	4.55E−08	1.85E−08	9.0944
	5	**0**	0	6.8312	9.28E−01	2.75E−01	48.6756	4.60E−08	2.39E−08	7.2616
	10	**0**	0	3.4516	6.55E−01	3.22E−01	27.8716	8.71E−08	8.46E−08	3.6280
	15	**0**	0	2.3228	3.49E−01	2.86E−01	19.2760	1.63E−08	2.46E−08	2.4428
	20	**0**	0	1.7616	**2.16E−01**	2.34E−01	13.2276	**1.60E−09**	2.44E−09	1.8376
		f_{10}			f_{11}			f_{12}		
ABC	1	9.06E+00	1.45E+01	23.0052	**0**	0	262.3164	**0**	0	225.3816
PABC	2	1.09E+01	1.30E+01	19.1668	**0**	0	140.8940	**0**	0	121.4256
	3	2.06E+01	2.35E+01	12.9116	**0**	0	93.8384	**0**	0	81.7076
	4	2.20E+01	1.96E+01	9.5096	**0**	0	71.0836	**0**	0	60.7684
	5	3.12E+01	3.04E+01	7.5720	**0**	0	56.2500	**0**	0	48.8300
	10	1.32E+00	2.76E+00	3.8616	**0**	0	29.6920	1.47E−13	1.95E−13	26.0992
	15	9.94E+00	2.70E+01	2.5820	**0**	0	19.3000	**0**	0	16.2668
	20	**5.00E−01**	1.33E−07	1.9376	**0**	0	14.7284	**0**	0	12.2462

4.1. Experiments: Part #1

We utilize the PABC algorithm by using a different number of subpopulation, migration interval, and migration topologies. Their performance, convergence ability, and running time are investigated.

In the experiments, the population size (NP) is investigated for two different cases: 60 and 120. Serial ABC algorithms using a single CPU use all individuals as a single population while the PABC algorithm variants divide the whole population into subpopulations made up of a different number of individuals which are calculated based on the number of CPUs by Eq. (6) as follows:

$$NP_{\text{subpopulation}} = \frac{NP}{\text{number of CPUs}} \tag{6}$$

where NP represents the number of food sources.

In the experiments, each subpopulation is evaluated on a different CPU. Therefore, the number of CPUs used is equal to the number of subpopulations used in the study. To analyze of the solution quality of parallelism, the population of the algorithm is divided into 20, 15, 10, 5, 4, 3, and 2 subpopulations. The migration interval which controls the frequency of the migration is set to different values (10, 20, 30, 40, 50, 100, 250, 500, 1000 and 2500) and the migration topologies are investigated for four different cases: ring, ring + 1+2, cube and grid. The number of individuals exchanged in each migration between the nodes (migration size) is 1. This means that only the best solutions migrate between the subpopulations.

The impact of the number of subpopulations and the statistical indicators (mean and standard deviation values) for 30 runs of the PABC and the serial ABC algorithms are given in Table 2 for D = 100, Table 3 for D = 500 and Table 4 for D = 1000 for the population size NP = 120 and in Table 5 for D = 100, Table 6 for D = 500 and Table 7 for D = 1000 for the population size NP = 60. In the tables, the best results are given in bold face. The values given in Tables 2–7 which cover the

Table 4
Function error values of the solutions obtained by PABC and ABC algorithms for the benchmark functions ($NP = 120, D = 1000$).

	Function	f_{cec1}			f_{cec2}			f_{cec3}		
	CPU (s)	Mean	StdDev	Time (s)	Mean	StdDev	Time (s)	Mean	StdDev	Time (s)
ABC	1	**0**	0	104.3556	1.74E+02	1.76E+00	90.1604	5.96E+01	2.43E+01	151.7932
PABC	2	**0**	0	77.8280	1.45E+02	8.89E+00	70.7512	6.00E+01	3.55E+01	102.2572
	3	**0**	0	51.9900	1.41E+02	6.78E+00	46.8096	4.53E+01	4.54E+01	68.1604
	4	**0**	0	39.7460	1.27E+02	1.24E+01	35.5920	7.16E+01	5.16E+01	50.4792
	5	**0**	0	31.4548	1.14E+02	1.88E+01	28.1272	9.68E+01	6.38E+01	40.7664
	10	**0**	0	15.7560	4.57E+01	4.05E+01	14.0340	5.64E+01	6.15E+01	20.1520
	15	**0**	0	10.6624	1.35E+01	2.22E+01	9.5548	1.13E+01	5.40E+01	13.5592
	20	**0**	0	8.1292	**1.19E+01**	2.14E+01	7.3128	**6.78E+00**	2.97E+01	10.4288
		f_{cec4}			f_{cec5}			f_{cec6}		
ABC	1	3.34E+00	1.78E+00	552.8196	**0**	0	1034.5350	6.97E−06	8.37E−06	551.6256
PABC	2	4.98E−01	1.32E+00	298.8188	**0**	0	564.8704	1.81E−06	8.62E−07	301.0700
	3	1.02E−08	1.85E−08	200.8688	**0**	0	382.3564	1.41E−06	3.72E−07	198.7972
	4	7.16E−09	1.39E−08	148.1864	**0**	0	282.3304	1.24E−06	5.30E−07	148.2776
	5	5.95E−09	1.28E−08	120.3636	**0**	0	221.1360	1.12E−06	6.09E−07	119.3652
	10	**1.60E−09**	2.02E−09	58.0448	**0**	0	114.2372	1.31E−06	1.21E−06	60.4572
	15	2.34E−09	5.52E−09	38.3532	**0**	0	76.7292	3.61E−07	8.72E−07	39.9092
	20	1.84E−09	8.99E−09	28.9712	**0**	0	58.6756	**2.67E−07**	8.74E−07	29.2816
		f_7			f_8			f_9		
ABC	1	**0**	0	81.8068	4.28E+00	4.87E−01	900.8444	9.10E−08	2.44E−08	86.6232
PABC	2	**0**	0	70.1240	2.38E+00	1.63E−01	483.6008	1.27E−07	2.65E−08	72.6752
	3	**0**	0	45.7120	1.90E+00	5.06E−01	343.7648	2.84E−07	5.39E−07	48.2096
	4	**0**	0	34.3100	1.91E+00	4.43E−01	249.7428	1.13E−07	3.08E−08	36.2532
	5	**0**	0	27.7984	1.58E+00	4.63E−01	217.8600	1.12E−07	3.80E−08	29.6628
	10	**0**	0	13.7496	1.43E+00	5.32E−01	122.4044	1.21E−07	9.94E−08	14.4292
	15	**0**	0	9.1996	1.05E+00	6.24E−01	86.8560	2.35E−08	3.17E−08	9.8888
	20	**0**	0	6.9240	**4.24E−01**	4.08E−01	70.1292	**3.75E−09**	6.61E−09	7.4160
		f_{10}			f_{11}			f_{12}		
ABC	1	1.94E+02	8.14E+01	92.2916	**0**	0	1047.9415	**0**	0	901.0845
PABC	2	2.14E+02	8.51E+01	75.8212	**0**	0	566.2028	**0**	0	488.9364
	3	2.27E+02	8.58E+01	50.6492	**0**	0	383.3388	**0**	0	325.4100
	4	2.30E+02	9.95E+01	37.9188	**0**	0	287.6076	**0**	0	251.0040
	5	2.21E+02	1.12E+02	31.9036	**0**	0	228.7836	**0**	0	195.9408
	10	1.41E+01	2.88E+01	15.0328	**0**	0	120.9584	2.09E−13	2.91E−13	106.9828
	15	1.66E+02	2.01E+02	10.3936	**0**	0	79.4680	**0**	0	66.6856
	20	**5.00E−01**	2.30E−07	7.7272	**0**	0	60.6864	**0**	0	49.5788

experimental results of the PABC algorithm, are obtained by using the ring topology and for the migration interval have a value of 10.

The two-sided Wilcoxon rank-sum test [13] is used to check whether the differences, if any, are statistically significant, because the results failed the Kolmogorov–Smirnov normality test [22]. Statistical test results for 30 runs of the PABC and the serial ABC algorithms are given in Table 8 for Tables 2–4 and in Table 9 for Tables 5–7. If the result of the hypothesis test is 0, the null hypothesis at significance level cannot be rejected, and if the result is 1, the null hypothesis at significance level can be rejected. In Tables 8 and 9, arrow notation is used. Directions of the arrows show the winner algorithm and represent the result (1) of the test. The value of (0) indicated that there is no statistically significant difference.

The results show that both the PABC with $NP = 60$ and the serial ABC with $NP = 60$ seem to produce better results than the PABC with $NP = 120$ and than the serial ABC with $NP = 120$. The PABC algorithm has a better performance compared to the serial ABC algorithm for both $NP = 60$ and $NP = 120$. Using a two-sided Wilcoxon rank-sum test with significance level of 0.05, pairwise comparisons between the ABC and PABC are conducted to test the difference between them. The Wilcoxon rank-sum test indicated that the PABC and the ABC for f_{cec2}, f_{cec3}, f_{cec4}, f_{cec6}, f_8 and f_{10} are statistically different while there is no significant difference for f_{cec1}, f_{cec5}, f_7, f_9, f_{11} and f_{12}. Moreover, as the number of subpopulations increases, the performance of the PABC algorithm also increases due to its perturbation operator. It has been seen that the maximum subpopulation number with minimum number of individuals produces the best results in the experiments.

The improvement in the performance of the PABC algorithm may vary depending on the function. From the results in Tables 2–7, the PABC algorithm can obtain better results than the ABC algorithm for all benchmarks. Especially, it is clear from the results for Rastrigin function that the success rate of the PABC algorithm is high because migrating the best individuals between the subpopulations improves the convergence rate of the algorithm much more for this function which requires more cycles to reach the global optima.

The impact of migration interval versus different number of subpopulations is analyzed for all dimensions and for all benchmark functions considered in the study. In this experiment, the ring topology is used, and NP is 120. Because of similar

Table 5
Function error values of the solutions obtained by PABC and ABC algorithms for the benchmark functions ($NP = 60, D = 100$).

	Function	f_{cec1}			f_{cec2}			f_{cec3}		
	CPU (s)	Mean	StdDev	Time (s)	Mean	StdDev	Time (s)	Mean	StdDev	Time (s)
ABC	1	**0**	0	1.1184	5.80E+01	5.39E+00	1.0036	9.40E+00	1.19E+01	1.5420
PABC	2	**0**	0	0.8460	4.18E+01	1.46E+01	0.7996	4.94E+00	5.48E+00	1.0728
	3	**0**	0	0.5576	2.43E+01	1.34E+01	0.5252	1.20E+01	1.69E+01	0.7092
	4	**0**	0	0.4796	1.26E+01	8.53E+00	0.8548	**4.64E+00**	6.15E+00	0.7744
	5	**0**	0	0.3392	5.45E+00	6.41E+00	0.3188	1.13E+01	1.50E+01	0.4448
	10	**0**	0	0.1748	**8.94E−01**	2.06E+00	0.1636	6.75E+00	1.58E+01	0.2236
		f_{cec4}			f_{cec5}			f_{cec6}		
ABC	1	**0**	0	4.9792	**0**	0	10.5100	**0**	0	4.5296
PABC	2	**0**	0	2.8032	**0**	0	5.8332	**0**	0	2.5392
	3	**0**	0	1.8784	**0**	0	3.9284	**0**	0	1.6996
	4	**0**	0	1.4248	**0**	0	4.2468	**0**	0	2.5956
	5	**0**	0	1.2396	**0**	0	2.3408	**0**	0	1.0272
	10	**0**	0	0.5924	**0**	0	1.1968	**0**	0	0.5032
		f_7			f_8			f_9		
ABC	1	**0**	0	0.9296	2.09E−01	2.52E−02	8.4588	**0**	0	0.9632
PABC	2	**0**	0	0.8024	1.77E−01	4.40E−02	4.5992	**0**	0	0.8392
	3	**0**	0	0.5336	1.61E−01	3.30E−02	3.1068	**0**	0	0.5572
	4	**0**	0	0.3956	1.38E−01	4.92E−02	2.3700	**0**	0	0.4092
	5	**0**	0	0.3176	1.21E−01	4.88E−02	1.9068	**0**	0	0.3256
	10	**0**	0	0.1616	**6.01E−02**	3.50E−02	1.2704	**0**	0	0.1676
		f_{10}			f_{11}			f_{12}		
ABC	1	6.33E−06	2.82E−06	1.0328	**0**	0	10.4424	**0**	0	9.0420
PABC	2	3.60E−06	2.82E−06	0.8736	**0**	0	6.5288	**0**	0	5.7052
	3	**2.34E−06**	1.56E−06	0.5796	**0**	0	4.3708	**0**	0	3.7984
	4	5.76E−06	5.42E−06	0.4328	**0**	0	3.2260	**0**	0	2.7852
	5	2.43E−05	1.02E−04	0.3480	**0**	0	2.6572	**0**	0	2.3164
	10	1.38E−05	3.12E−05	0.1780	**0**	0	1.4612	**0**	0	1.2944

Table 6
Function error values of the solutions obtained by PABC and ABC algorithms for the benchmark functions ($NP = 60, D = 500$).

	Function	f_{cec1}			f_{cec2}			f_{cec3}		
	CPU (s)	Mean	StdDev	Time (s)	Mean	StdDev	Time (s)	Mean	StdDev	Time (s)
ABC	1	**0**	0	25.4808	1.48E+02	4.18E+00	22.3832	7.17E+00	1.11E+01	36.2892
PABC	2	**0**	0	18.8356	1.08E+02	1.05E+01	17.4048	9.41E+00	1.57E+01	24.9596
	3	**0**	0	12.5460	8.29E+01	2.16E+01	11.5824	**7.07E+00**	8.85E+00	16.6152
	4	**0**	0	15.4088	6.93E+01	2.47E+01	11.1008	1.23E+01	1.68E+01	16.3212
	5	**0**	0	7.9180	5.05E+01	2.76E+01	7.0280	9.88E+00	1.58E+01	10.1036
	10	**0**	0	4.0148	**4.70E+00**	1.01E+01	3.6264	1.35E+01	3.42E+01	5.0404
		f_{cec4}			f_{cec5}			f_{cec6}		
ABC	1	1.44E−02	6.83E−02	123.0180	**0**	0	258.3985	**0**	0	109.2296
PABC	2	**0**	0	67.4912	**0**	0	152.8000	**0**	0	61.4892
	3	**0**	0	45.6104	**0**	0	102.6984	**0**	0	41.4908
	4	**0**	0	36.9488	**0**	0	76.3732	**0**	0	31.3644
	5	**0**	0	28.0732	**0**	0	61.7716	**0**	0	24.8200
	10	**0**	0	14.8504	**0**	0	31.6432	**0**	0	12.6820
		f_7			f_8			f_9		
ABC	1	**0**	0	20.5644	1.34E+00	9.99E−02	216.7608	**0**	0	21.2448
PABC	2	**0**	0	17.0164	8.23E−01	2.27E−01	144.8392	**0**	0	18.1712
	3	**0**	0	11.3348	6.09E−01	2.04E−01	79.8152	**0**	0	11.8512
	4	**0**	0	8.4472	6.22E−01	2.42E−01	68.0408	**0**	0	8.9452
	5	**0**	0	6.8244	4.59E−01	2.33E−01	53.5332	**0**	0	7.1892
	10	**0**	0	3.5048	**2.99E−01**	1.95E−01	27.9042	**0**	0	3.6464
		f_{10}			f_{11}			f_{12}		
ABC	1	1.13E+01	1.26E+01	22.5864	**0**	0	262.0268	**0**	0	224.7420
PABC	2	1.01E+01	1.78E+01	19.1028	**0**	0	170.5044	**0**	0	146.4996
	3	1.11E+01	1.59E+01	12.5092	**0**	0	112.8336	**0**	0	97.9168
	4	1.34E+01	2.05E+01	9.4320	**0**	0	83.5308	**0**	0	71.7384
	5	1.53E+01	2.11E+01	7.6388	**0**	0	68.3536	**0**	0	59.0912
	10	**6.09E−02**	1.62E−01	3.9388	**0**	0	36.8312	**0**	0	32.3021

Table 7
Function error values of the solutions obtained by PABC and ABC algorithms for the benchmark functions ($NP = 60, D = 1000$).

	Function	f_{cec1}			f_{cec2}			f_{cec3}		
	CPU (s)	Mean	StdDev	Time (s)	Mean	StdDev	Time (s)	Mean	StdDev	Time (s)
ABC	1	**0**	0	103.4800	1.74E+02	1.68E+00	90.2932	3.68E+00	1.19E+01	146.9744
PABC	2	**0**	0	75.4668	1.34E+02	9.02E+00	70.3208	**2.58E+00**	3.15E+00	101.3628
	3	**0**	0	49.9508	1.19E+02	1.25E+01	46.1760	3.69E+00	3.42E+00	66.6512
	4	**0**	0	50.4108	1.07E+02	1.34E+01	36.7624	9.63E+00	1.87E+01	51.1048
	5	**0**	0	30.3428	8.04E+01	2.53E+01	27.8800	1.71E+01	2.80E+01	40.0144
	10	**0**	0	15.4132	**1.67E+01**	2.28E+01	14.2192	3.07E+00	1.49E+01	20.4388
		f_{cec4}			f_{cec5}			f_{cec6}		
ABC	1	1.14E−01	3.09E−01	491.8496	**0**	0	1028.2540	1.42E−11	5.06E−13	441.0643
PABC	2	**0**	0	273.1324	**0**	0	597.2936	1.19E−11	4.64E−13	251.4340
	3	5.73E−09	2.80E−08	186.3280	**0**	0	409.9897	1.12E−11	4.01E−13	166.0632
	4	**0**	0	142.8060	**0**	0	304.1204	1.08E−11	6.65E−13	127.6840
	5	**0**	0	115.6564	**0**	0	250.3748	**0**	0	102.6228
	10	**0**	0	58.8032	**0**	0	125.7680	**0**	0	51.3828
		f_7			f_8			f_9		
ABC	1	**0**	0	81.0272	3.07E+00	2.44E−01	870.9340	**0**	0	84.1308
PABC	2	**0**	0	67.7736	1.59E+00	5.86E−01	536.0856	**0**	0	71.1132
	3	**0**	0	46.1768	1.35E+00	3.56E−01	381.1568	3.95E−08	1.39E−07	47.6668
	4	**0**	0	34.2916	1.24E+00	2.82E−01	238.4484	2.45E−13	9.26E−13	36.0776
	5	**0**	0	27.9248	1.04E+00	4.09E−01	209.8348	**0**	0	29.4784
	10	**0**	0	13.5988	**7.38E−01**	4.49E−01	128.9576	**0**	0	14.2608
		f_{10}			f_{11}			f_{12}		
ABC	1	7.01E+01	5.26E+01	89.4792	**0**	0	1039.5071	**0**	0	896.1972
PABC	2	1.48E+02	6.42E+01	74.5208	**0**	0	686.3208	1.95E−13	6.69E−13	612.3076
	3	1.47E+02	8.08E+01	49.9952	**0**	0	457.2236	**0**	0	408.4172
	4	1.49E+02	9.88E+01	37.9764	**0**	0	342.8152	**0**	0	295.2672
	5	1.73E+02	1.10E+02	30.7008	**0**	0	282.5940	**0**	0	249.0580
	10	**2.01E−01**	2.44E−01	15.2404	**0**	0	147.7784	**0**	0	130.0928

Table 8
Statistical comparison of ABC and PABC algorithms for $NP = 120$ using two-sided Wilcoxon rank-sum test with significance level 0.05.

Algorithm	CPU (s)	Functions	PABC CPUs											
			20			15			10			5		
			100	500	1000	100	500	1000	100	500	1000	100	500	1000
ABC	1	f_{cec2}	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑
		f_{cec3}	↑	↑	↑	↑	↑	↑	0	0	0	0	0	←
		f_{cec4}	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑
		f_{cec6}	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑
		f_8	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑
		f_{10}	←	0	↑	←	0	0	←	0	↑	0	←	0
PABC	20	f_{cec2}				←	0	0	←	←	←	←	←	←
		f_{cec3}				←	←	←	←	←	←	←	←	←
		f_{cec4}				←	←	←	←	←	↑	←	←	←
		f_{cec6}				←	←	←	←	←	←	←	←	←
		f_8				0	←	←	←	←	←	←	←	←
		f_{10}				0	0	←	←	←	0	↑	←	←
	15	f_{cec2}							←	←	←	←	←	←
		f_{cec3}							←	←	←	←	←	←
		f_{cec4}							←	←	↑	←	↑	←
		f_{cec6}							←	←	←	←	←	←
		f_8							←	←	←	←	←	←
		f_{10}							←	0	↑	↑	←	←
	10	f_{cec2}										←	←	←
		f_{cec3}										0	0	←
		f_{cec4}										←	0	←
		f_{cec6}										←	←	0
		f_8										←	←	0
		f_{10}										↑	←	←

Table 9
Statistical comparison of ABC and PABC algorithms for $NP = 60$ using two-sided Wilcoxon rank-sum test with significance level 0.05.

			PABC CPUs											
			10			5			4			3		
Algorithm	CPU (s)	Functions	100	500	1000	100	500	1000	100	500	1000	100	500	1000
ABC	1	f_{cec2}	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑
		f_{cec3}	↑	←	↑	0	0	←	0	0	←	0	0	←
		f_{cec4}	0	↑	↑	0	↑	↑	0	↑	↑	0	↑	↑
		f_{cec6}	0	0	↑	0	0	↑	0	0	↑	0	0	↑
		f_8	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑
		f_{10}	←	↑	↑	←	0	←	0	0	←	↑	0	←
PABC	10	f_{cec2}				←	←	←	←	←	←	←	←	←
		f_{cec3}				←	↑	←	↑	↑	←	←	↑	←
		f_{cec4}				0	0	0	0	0	0	0	0	←
		f_{cec6}				0	0	←	0	0	←	0	0	←
		f_8				←	←	←	←	←	←	←	←	←
		f_{10}				0	←	←	0	←	←	0	←	←
	5	f_{cec2}							←	←	←	←	←	←
		f_{cec3}							0	0	0	0	0	0
		f_{cec4}							0	0	0	0	0	←
		f_{cec6}							0	0	←	0	0	←
		f_8							0	←	←	←	←	←
		f_{10}							0	0	0	0	0	0
	4	f_{cec2}										←	←	←
		f_{cec3}										0	0	0
		f_{cec4}										0	0	←
		f_{cec6}										0	0	0
		f_8										0	0	0
		f_{10}										↑	0	0

Table 10
Results of the PABC algorithm with different migration intervals for the test function f_{cec4} ($NP = 120, D = 1000$).

Mig. Int.	Number of subpopulations								
	20			15			10		
	Mean	StdDev	Time (s)	Mean	StdDev	Time (s)	Mean	StdDev	Time (s)
10	**1.84E−09**	8.99E−09	28.9712	**2.34E−09**	5.52E−09	38.3532	**1.60E−09**	2.02E−09	58.0448
20	2.41E−07	8.18E−07	28.7352	2.34E−08	6.50E−08	37.8456	7.70E−09	1.44E−08	57.3692
30	1.24E−06	6.09E−06	28.4864	1.08E−07	3.06E−07	37.4684	1.91E−08	3.93E−08	57.4924
40	1.57E−06	7.71E−06	28.2744	1.93E−07	6.40E−07	37.5928	2.65E−08	4.37E−08	57.4800
50	3.99E−02	1.95E−01	28.1576	7.54E−07	3.70E−06	37.4420	9.62E−08	2.23E−07	57.6540
100	5.48E−04	2.12E−03	28.4180	1.20E−06	4.18E−06	38.2720	5.24E−07	1.19E−06	57.7624
250	2.00E−01	9.81E−01	29.0464	1.79E−01	6.98E−01	38.9188	1.26E−03	5.98E−03	57.8852
500	4.86E−01	2.38E+00	30.1652	8.54E−02	4.18E−01	39.9504	2.11E−01	3.97E−01	57.9680
1000	5.73E−01	2.81E+00	32.1024	2.15E+00	4.25E+00	42.6564	1.08E+00	1.87E+00	59.4780
2500	6.50E−01	3.19E+00	32.5540	8.44E−01	2.87E+00	45.9732	1.14E+00	1.95E+00	60.2776
No mig.	1.31E+02	3.75E+02	29.2480	5.95E+01	4.82E+02	39.0934	1.59E+02	2.73E+02	58.6302
	5			4			3		
10	**5.95E−09**	6.60E−08	120.3636	**7.16E−09**	1.39E−08	148.1864	**1.02E−08**	1.85E−08	200.8688
20	6.79E−09	2.92E−08	119.1304	1.89E−06	9.16E−06	146.2060	2.42E−05	1.18E−04	198.5944
30	9.57E−09	2.63E−08	119.2236	1.10E−06	5.31E−06	146.6260	2.15E−06	1.03E−05	198.5200
40	1.36E−08	5.55E−08	120.2628	4.35E−08	1.38E−07	147.3084	3.98E−02	1.95E−01	199.9364
50	4.80E−08	3.31E−07	118.3836	5.27E−08	1.33E−07	147.0672	6.86E−08	1.19E−07	197.6716
100	7.06E−08	5.78E−07	119.3840	8.45E−06	4.10E−05	147.8084	8.07E−02	2.74E−01	198.8756
250	7.96E−02	9.95E−01	119.7688	2.02E−01	5.62E−01	146.7620	3.06E−01	6.60E−01	197.1824
500	4.74E−01	2.99E+00	119.2164	5.96E−01	9.31E−01	147.2972	6.86E−01	8.48E−01	198.0704
1000	3.19E−01	2.09E+00	119.4588	8.82E−01	1.20E+00	147.0524	8.39E−01	8.76E−01	197.4976
2500	1.82E+00	8.41E+00	118.9416	1.94E+00	2.70E+00	147.3548	2.29E+00	2.05E+00	199.5236
No mig.	1.29E+01	1.70E+01	118.7964	7.81E+00	1.05E+01	146.1000	4.32E+00	2.02E+00	196.7580

results, only the results of f_{cec4} ($D = 1000$) and f_{cec6} ($D = 500$) are shown in Tables 10 and 11, respectively. Columns of 20/15/10/5/4/3 in Tables 10 and 11 correspond to different number of subpopulations and each row corresponds to different migration interval. Also, the row "No mig." corresponds the case of no migration is conducted.

Table 11
Results of the PABC algorithm with different migration intervals for the test function f_{cec6} (NP = 120,D = 500).

Mig. Int.	Number of subpopulations								
	20			15			10		
	Mean	StdDev	Time (s)	Mean	StdDev	Time (s)	Mean	StdDev	Time (s)
10	**6.28E−08**	2.10E−07	7.0940	4.55E−07	1.24E−06	9.7388	**6.06E−07**	7.72E−07	14.6476
20	1.17E−07	3.17E−07	6.9368	8.87E−07	1.74E−06	9.7396	8.51E−07	1.32E−06	14.4500
30	3.30E−07	7.47E−07	6.9544	8.35E−07	1.53E−06	9.5808	2.08E−06	1.91E−06	14.5212
40	1.40E−07	5.67E−07	6.8096	9.27E−07	2.18E−06	9.5944	1.32E−06	1.48E−06	14.4804
50	1.88E−07	8.28E−07	6.7860	7.37E−07	1.56E−06	9.5656	1.51E−06	2.06E−06	14.4156
100	1.21E−07	5.10E−07	6.7364	9.89E−07	1.94E−06	9.4804	1.28E−06	2.18E−06	14.3488
250	1.21E−07	5.40E−07	6.9060	1.06E−06	2.13E−06	9.5500	3.31E−06	3.74E−06	14.4900
500	4.57E−07	2.15E−06	7.0168	**2.21E−07**	9.08E−07	9.5768	3.08E−06	5.43E−06	14.4760
1000	4.11E−07	1.90E−06	7.3248	1.49E−06	3.31E−06	9.7992	5.69E−06	5.57E−06	14.6160
2500	1.44E−06	6.02E−06	7.7524	8.00E−06	2.12E−05	10.2412	5.11E−06	7.65E−06	14.9104
No mig.	5.59E−01	9.67E+00	8.0302	2.12E−01	1.49E+00	10.5091	1.75E−01	1.15E+00	15.1580
	5			4			3		
10	**9.06E−07**	5.03E−07	29.3552	**9.23E−07**	4.21E−07	36.6544	**9.93E−07**	4.72E−07	49.3076
20	1.61E−06	8.80E−07	29.2772	1.52E−06	8.58E−07	36.7072	1.41E−06	5.63E−07	49.3012
30	2.02E−06	9.47E−07	29.2036	1.87E−06	8.16E−07	36.6532	1.74E−06	4.41E−07	49.2512
40	2.26E−06	1.01E−06	29.1992	1.60E−06	9.32E−07	37.0000	1.84E−06	4.85E−07	49.1928
50	2.24E−06	1.41E−06	29.1212	1.87E−06	7.98E−07	36.5652	2.11E−06	8.48E−07	49.2840
100	3.06E−06	9.08E−07	29.1060	2.37E−06	6.22E−07	36.5388	1.68E−06	8.21E−07	49.1772
250	3.29E−06	1.51E−06	29.4152	2.81E−06	1.08E−06	36.8796	2.01E−06	9.15E−07	49.2316
500	2.91E−06	1.76E−06	29.1460	3.02E−06	1.33E−06	36.5944	2.31E−06	6.66E−07	49.1092
1000	3.46E−06	2.05E−06	29.1812	2.70E−06	8.89E−07	36.6544	2.45E−06	1.11E−06	49.1664
2500	4.16E−06	2.28E−06	29.2688	3.81E−06	2.00E−06	37.3604	3.10E−06	1.74E−06	49.4312
No mig.	1.23E−05	2.29E−05	29.1385	6.60E−06	1.56E−05	36.5717	3.93E−06	6.92E−06	48.8221

Table 12
Function error values of the solutions obtained using the ABC algorithm and PABC algorithms having different migration topologies for the benchmark functions (D = 100).

	Function	f_{cec1}			f_{cec2}			f_{cec3}		
	Topology	Mean	StdDev	Time (s)	Mean	StdDev	Time (s)	Mean	StdDev	Time (s)
ABC	–	**0**	0	1.0753	7.64E+01	7.16E+00	1.0020	1.03E+01	1.18E+01	1.5357
PABC	T_1	**0**	0	0.1224	1.44E+00	3.32E+00	0.1080	1.55E+00	2.62E+00	0.1409
	T_2	**0**	0	0.1286	6.24E−01	1.37E+00	0.1147	2.02E+00	8.42E+00	0.1504
	T_3	**0**	0	0.1330	5.46E−01	1.31E+00	0.1223	5.49E+00	1.41E+01	0.1551
	T_4	**0**	0	0.1410	**4.46E−01**	9.65E−01	0.1283	**1.26E−01**	1.18E−01	0.1583
		f_{cec4}			f_{cec5}			f_{cec6}		
ABC	–	4.26E−01	5.39E−01	5.3707	**0**	0	10.4910	3.79E−09	2.39E−09	4.8823
PABC	T_1	7.39E−13	1.75E−13	0.3787	**0**	0	0.7980	1.06E−10	2.08E−10	0.3563
	T_2	**5.68E−13**	1.10E−13	0.3977	**0**	0	0.8200	8.76E−11	1.90E−10	0.3653
	T_3	6.82E−13	1.11E−13	0.4157	**0**	0	0.8400	4.83E−11	8.44E−11	0.3770
	T_4	6.25E−13	1.05E−13	0.4273	**0**	0	0.8603	**1.83E−11**	2.73E−11	0.3817
		f_7			f_8			f_9		
ABC	–	**0**	0	1.0220	2.90E−01	3.54E−02	9.2270	1.55E−11	6.82E−12	1.0073
PABC	T_1	**0**	0	0.1091	5.18E−02	3.31E−02	0.6530	2.37E−13	2.94E−13	0.1107
	T_2	**0**	0	0.1095	3.63E−02	2.32E−02	0.6713	4.10E−13	7.86E−13	0.1180
	T_3	**0**	0	0.1099	2.65E−02	2.21E−02	0.6883	1.13E−12	4.12E−12	0.1247
	T_4	**0**	0	0.1132	**2.36E−02**	2.31E−02	0.7040	**2.11E−13**	3.74E−13	0.1313
		f_{10}			f_{11}			f_{12}		
ABC	–	**6.89E−04**	2.80E−04	1.0354	**0**	0	9.1827	**0**	0	8.2243
PABC	T_1	2.09E−03	1.94E−02	0.1032	**0**	0	0.7413	**0**	0	0.6357
	T_2	4.50E−03	2.45E−02	0.1113	**0**	0	0.7543	**0**	0	0.6543
	T_3	3.58E−03	7.30E−02	0.1154	**0**	0	0.7817	**0**	0	0.6823
	T_4	4.61E−02	1.43E−01	0.1177	**0**	0	0.8087	**0**	0	0.6967

Tables 10 and 11 show that low values of migration interval improve the performance for the f_{cec4} and f_{cec6} functions. The results show that low-frequency-migrations on high intervals should be avoided, because they make islands to exchange individuals slowly so that the subpopulations do not contribute to others sufficiently in terms of high quality solutions. If

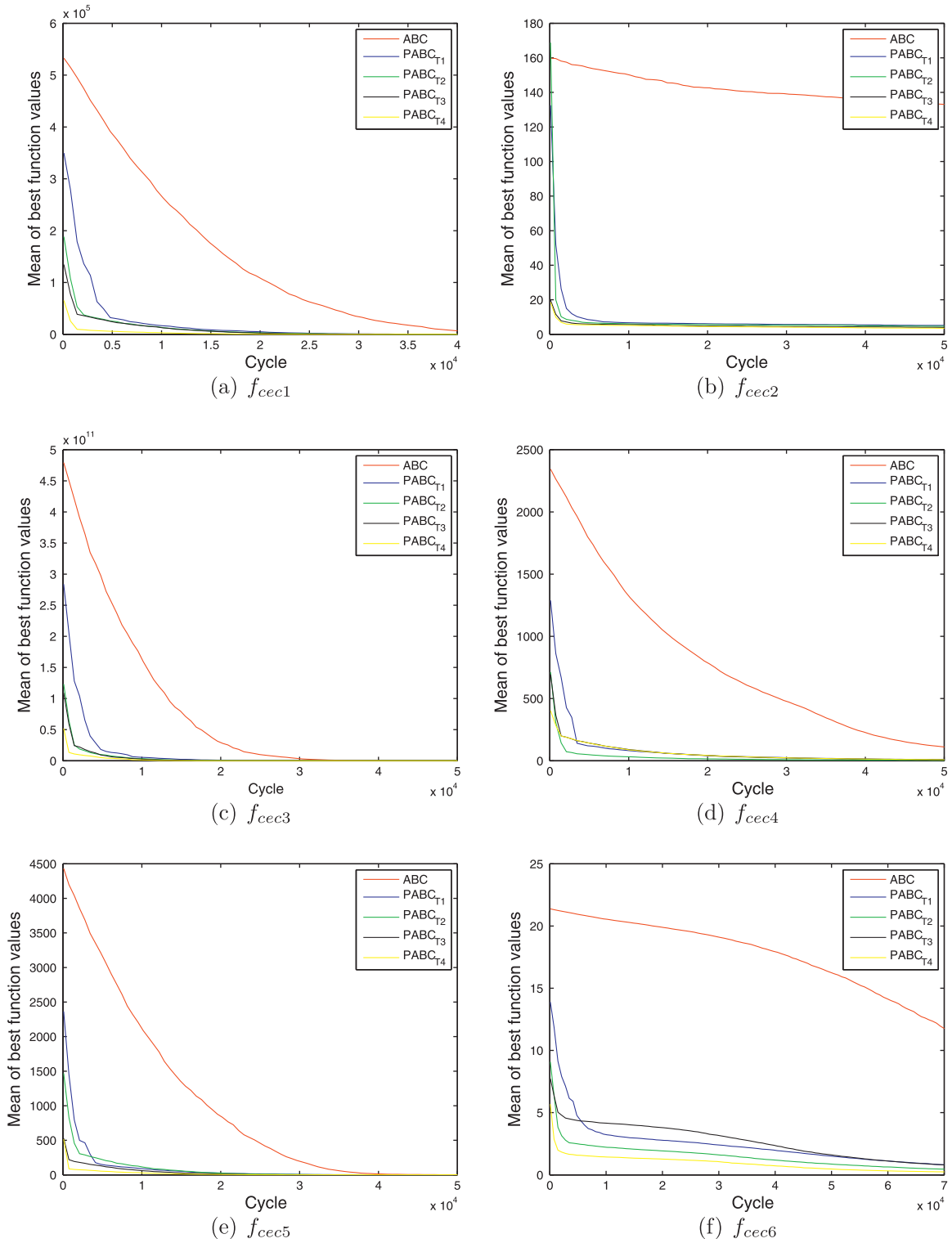

Fig. 4. Convergence behavior of the ABC algorithm and PABC algorithms having different migration topologies for the benchmark functions (a) f_{cec1}, (b) f_{cec2}, (c) f_{cec3}, (d) f_{cec4}, (e) f_{cec5}, and (f) f_{cec6}.

there is no migration, an island model is nothing more than a set of separate runs. The experiments show that to achieve a better solution quality, migration should be employed between islands. Besides, increasing the number of subpopulations

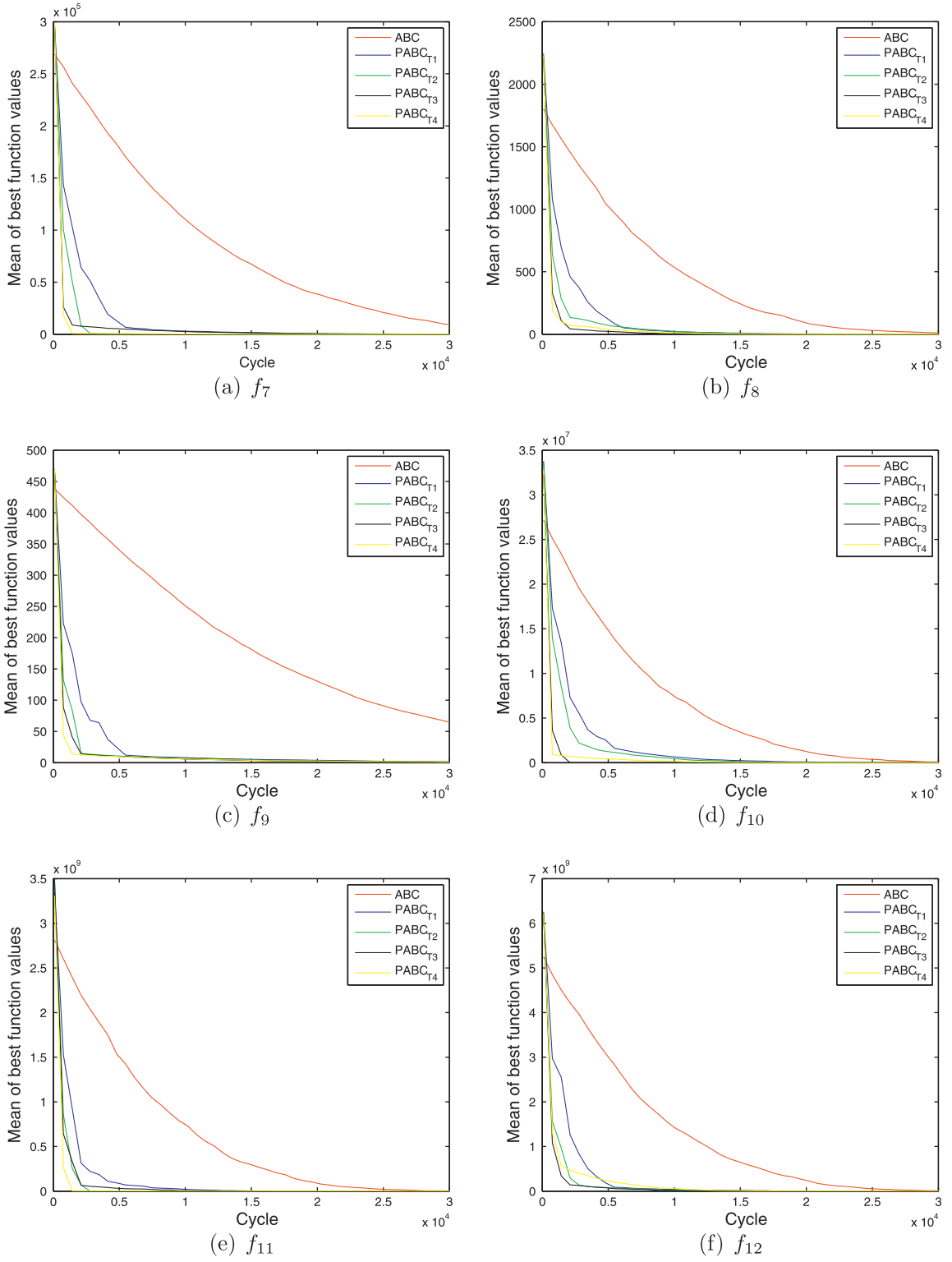

Fig. 5. Convergence behavior of the ABC algorithm and PABC algorithms having different migration topologies for the benchmark functions (a) f_7, (b) f_8, (c) f_9, (d) f_{10}, (e) f_{11}, and (f) f_{12}.

affects the importance of migration interval considerably. It can be concluded that the value 10 for migration interval is generally better for the experiments performed in this study.

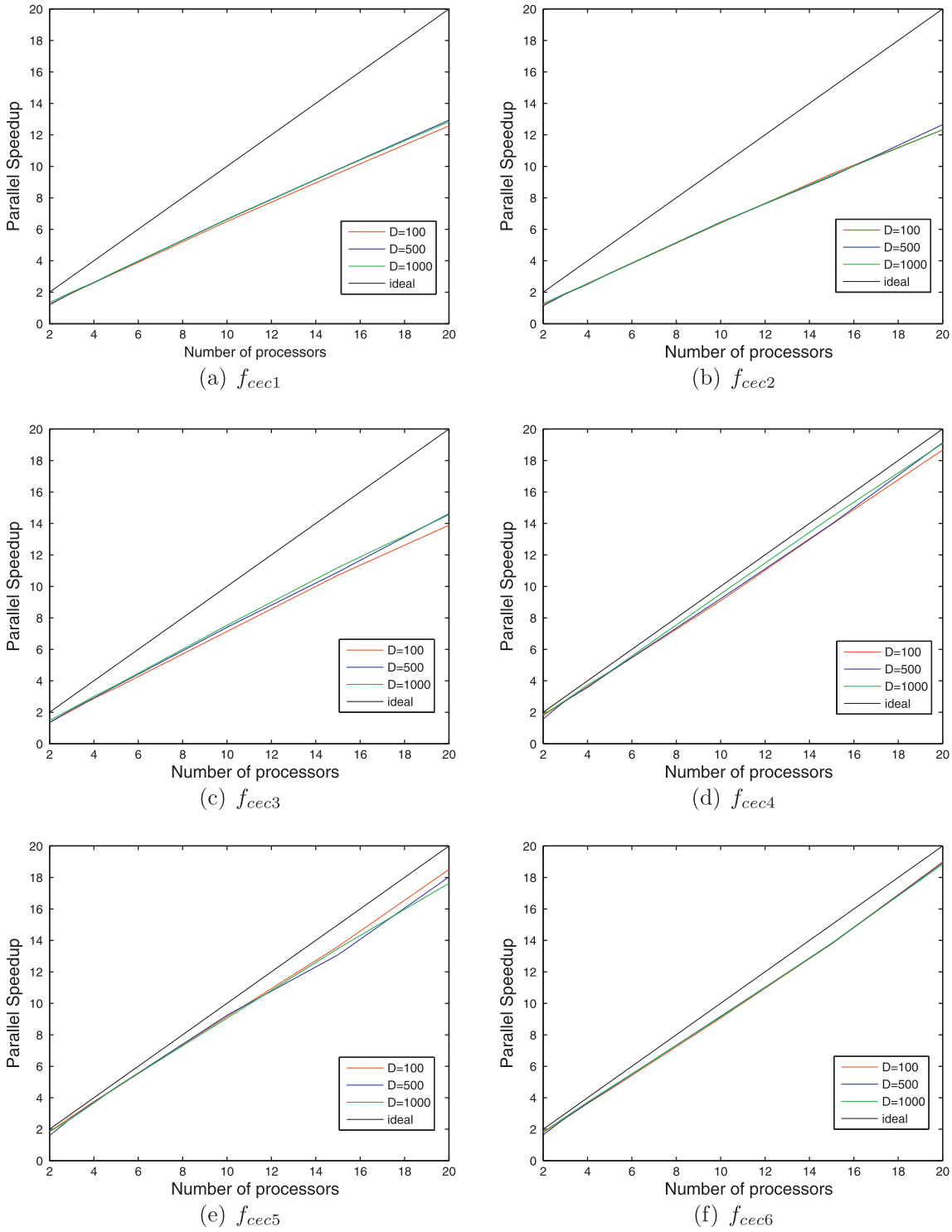

Fig. 6. Speedup results of the PABC algorithm for the benchmark functions (a) f_{cec1}, (b) f_{cec2}, (c) f_{cec3}, (d) f_{cec4}, (e) f_{cec5}, and (f) f_{cec6}.

The impact of migration topology, ring (T_1), ring + 1 + 2 (T_2), cube (T_3), and grid (T_4) topologies are investigated. In this experiment, we use 16 islands, the *NP* is 96 and value of the migration interval is set to 10. Results of the topologies T_1, T_2, T_3 and T_4 are shown in Table 12.

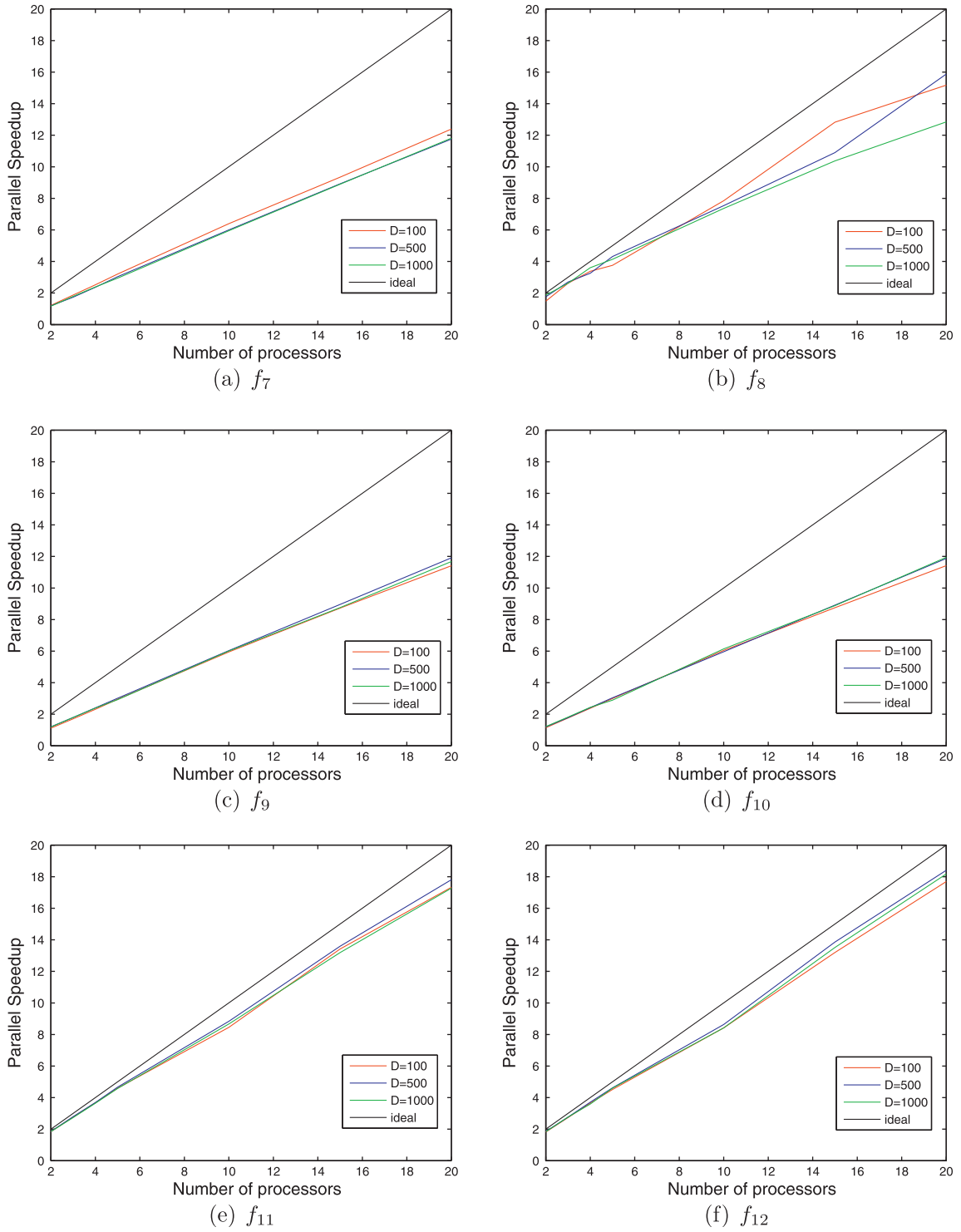

Fig. 7. Speedup results of the PABC algorithm for the benchmark functions (a) f_7, (b) f_8, (c) f_9, (d) f_{10}, (e) f_{11}, and (f) f_{12}.

Generally, migration topology is an important constituent for a parallel algorithm which uses the coarse-grained parallelization model. From the experimental results given in Table 12, it can be said that migration topology not only have significant influence on the results, but also it provides much faster convergence. Besides, both performance and convergence

Table 13
Properties of the datasets using in the study: Number of whole, training and testing data pairs, number of inputs for the classifier and number of the classes in each dataset.

	Data	Train	Test	Input	Class
Balance	625	469	156	4	3
Cancer-Int	699	524	175	9	2
Glass	214	161	53	9	2
Iris	150	112	38	4	3
Wine	178	133	45	13	3

Table 14
The average total training and output time values of ANN structures trained by the ABC algorithm and the PABC algorithms which use 10 and 20 CPUs for each classification purpose.

Dataset	ANN Structure	ABC	PABC			
		1 CPU	10 CPUs		20 CPUs	
		Time (s)	Time (s)	Speedup	Time (s)	Speedup
Balance	S_1	41.0667	4.2137	9.75	2.1673	18.95
	S_2	86.6643	8.8953	9.74	4.5430	19.08
	S_3	125.8160	12.8057	9.82	6.5187	19.30
Cancer-Int	S_1	55.3120	5.6813	9.74	3.2540	17.00
	S_2	119.2567	12.1877	9.79	6.1856	19.28
	S_3	182.7007	18.7480	9.75	9.9090	18.44
Glass	S_1	16.9503	1.7847	9.50	1.0683	15.87
	S_2	37.0583	3.7550	9.87	1.9017	19.49
	S_3	56.2490	5.7587	9.77	3.2493	17.31
Iris	S_1	9.8160	1.0560	9.30	0.5247	18.71
	S_2	20.7807	2.1387	9.72	1.1007	18.88
	S_3	30.0313	3.0483	9.85	1.5717	19.11
Wine	S_1	19.3147	1.9877	9.72	1.0040	19.24
	S_2	43.6847	4.4977	9.71	2.4097	18.13
	S_3	65.0247	6.6180	9.83	3.3987	19.13

Table 15
Average CEP and MSE values of different ANN structures trained by the ABC algorithm and the PABC algorithms which use 10 and 20 CPUs for each classification purpose.

Dataset	ANN Structure	ABC			PABC					
		1 CPU			10 CPUs			20 CPUs		
		CEP (Test)	CEP (Train)	MSE	CEP (Test)	CEP (Train)	MSE	CEP (Test)	CEP (Train)	MSE
Balance	S1	11.11	11.33	1.53E−01	10.92	10.98	1.48E−01	**9.96**	10.02	1.40E−01
	S2	10.21	9.96	1.42E−01	8.95	9.28	1.33E−01	**8.40**	7.65	1.10E−01
	S3	10.23	9.57	1.38E−01	8.61	8.91	1.24E−01	**7.54**	6.92	1.06E−01
Cancer-Int	S1	3.77	2.03	3.81E−02	**2.53**	2.37	4.25E−02	3.05	3.07	4.97E−02
	S2	3.81	1.90	3.56E−02	**1.92**	1.92	3.55E−02	2.90	2.80	4.78E−02
	S3	4.09	1.83	3.47E−02	**1.79**	2.24	3.88E−02	1.87	1.18	2.48E−02
Glass	S1	6.35	2.63	5.01E−02	4.03	5.01	6.48E−02	**2.38**	2.30	4.58E−02
	S2	5.79	2.03	3.91E−02	4.65	4.61	5.82E−02	**2.45**	1.66	3.32E−02
	S3	5.41	2.01	3.75E−02	2.58	2.15	4.15E−02	**1.76**	1.84	3.10E−02
Iris	S1	3.77	2.92	5.55E−02	2.89	2.85	5.15E−02	**1.58**	2.35	3.51E−02
	S2	2.98	2.38	4.21E−02	2.46	2.71	3.68E−02	**2.10**	1.85	3.03E−02
	S3	3.42	2.08	3.65E−02	**1.75**	1.76	3.47E−02	2.11	1.49	2.53E−02
Wine	S1	8.30	5.01	9.99E−02	5.04	3.78	7.08E−02	**2.44**	1.85	3.85E−02
	S2	7.26	3.93	8.29E−02	4.66	4.34	7.29E−02	**2.30**	1.85	3.71E−02
	S3	7.85	2.81	6.88E−02	2.89	2.13	4.01E−02	**0.96**	0.43	1.44E−02

rate of the PABC algorithm with grid topology is relatively better than the PABC algorithms with other migration topologies. The convergence curves for the algorithms are shown in Figs. 4 and 5.

Overall, for large-scale problems, the PABC algorithm can generate efficient results when the number of the subpopulations is high, the migration interval is low, and the grid topology is used.

4.2. Experiments: Part #2

In the second part of the experiments, the efficiency and the speedup of the PABC algorithm are analyzed using the computational cost values in Tables 2–4. The speedup results of parallel experiments for twelve test functions are shown in Figs. 6 and 7.

It is observed that each implementation of the PABC algorithm gains speedup compared to the its sequential counterpart. The speedup of the PABC algorithm is almost linear. Besides, the efficiency is less for the functions requiring less computation times and much more for the functions requiring high computation times.

The running times of coarse-grained parallel EAs are expected to increase when the migration occurs more while the PABC algorithm is not significantly affected when more communication occurs between subpopulations on the parallel network architecture we used. Tables 10 and 11 show the running time of the PABC algorithm with respect to the migration intervals from 10 to 2500 for different number of CPUs (20, 15, 10, 5, 4, and 3) for the functions f_{cec4} and f_{cec6}. As seen from these results, migration interval does not have a great influence on the running time of the PABC algorithm.

Similarly, when the results in Table 12 are examined, it is seen that different migration topologies do not significantly affect the running time of the PABC algorithm.

Insensitivity of running time of the PABC algorithm to migration interval and migration topology is due to high speed of the computation grid used. Besides, when the population size is decreased in the PABC algorithm, probability of being mutated of the individuals will be much more and the numbers of their trials get close to the control parameter, limit. Therefore, random solution production in the scout procedure works much more and this affects the running time of the algorithm. However, when the number of migrations is high, scout works less, and when the number of migrations is low, scout works more. The contributions of total migration times and total time of scout calls to the overall running time counteract each other because the number of scout calls can be low due to new solutions coming from other subpopulations via migration. Consequently, it can be said that the contributions of the migration and the scout procedure are relatively very small on the total running time.

Our observation from these experiments is that the number of subpopulations is playing much bigger role on the running time than the migration interval and migration topology does. Also, various topologies and interval values does not significantly affect the running time of the PABC algorithm in this study.

4.3. Experiments: Part #3

In the last part of the experiments, we present several results related to the optimization of feed-forward neural networks for five datasets which are commonly used in the literature for classification. The properties of each dataset are presented in Table 13. Details of the problem datasets can be obtained from reference [26]. The success and the running time of ANNs largely depends on the structure. For this purpose, three network structures are used for all problems. The three structures for each problem include one hidden layer with $S_1 = 2$, $S_2 = 5$, and $S_3 = 8$ hidden nodes. The classification error percentage (CEP) measure is used to evaluate the performance of algorithms, which is defined by Eq. (7) as follows

$$\mathrm{CEP} = 100 \cdot \frac{\text{misclassified examples}}{\text{size of the data set}}. \tag{7}$$

In this study, the multilayer perceptron neural network architecture is used for classification purposes. In the training phase, mean squared error (MSE) function is chosen as performance function and data is shuffled randomly before training of the network. For the ABC and PABC algorithms, population size, limit value and the total number of evaluations are chosen as 120, 1000 and 20,000, respectively.

In Tables 14 and 15, the average total training and output time values, and the average CEP and MSE values of the ANNs with different structures (S_1,S_2 and S_3) trained using the ABC algorithm and PABC algorithms which use 10 and 20 CPUs are presented, respectively. The results show that the PABC algorithm has better performance compared to its serial counterpart for almost all datasets and ANNs with different structures in terms of running time and solution quality. The PABC algorithm provides an almost linear speedup on training of ANNs as in the case of optimizing numerical benchmark problems.

5. Discussion and conclusion

In this study, a parallel ABC algorithm based on a coarse-grained parallel computing model is presented. Performance of the proposed algorithm is compared to performance of its sequential counterpart in terms of solution quality and running time. The characteristics of the algorithm are discussed and its performance is analyzed based on computational experiments on benchmark problems with various complexities and specialities. We also investigate the performance of the proposed algorithm for training three different architectures of ANNs in a variety of classical classification problems. This study represents the first study in the literature both analyzing the parameters of the PABC algorithm and the application of this algorithm in training of ANNs for classification purposes.

It is found that the performance and convergence of the PABC algorithm are superior to those of the ABC algorithm. Dividing the population into smaller parts produces better results as compared to large populations and as the migration interval

is reduced the performance is improving significantly and migration topology does not have a marked influence on the results but achieves good convergence on the problems considered. This observation made us assign proper values for the number of subpopulations, migration interval and migration topology. Implementing a parallel ABC algorithm on a coarse-grained model gives efficient results in addition to reducing running time significantly. We obtained almost linear speedups for both optimizing the benchmark functions and classification of datasets using ANNs.

A future development of this work will aim at further investigating different parallelization models of the ABC algorithm and the impact of programming models (master–slave, fine-grained and hybrid) on the performance of the algorithm on different real-world problems.

Acknowledgements

The numerical calculations reported in this paper are performed at TUBITAK ULAKBIM, High Performance and Grid Computing Center (TR-Grid e-Infrastructure). This study is a part of the Project Numbered FBD-10-3117. The author thanks Erciyes University's Scientific Research Projects Unit.

References

[1] B. Akay, D. Karaboga, Artificial bee colony algorithm for large-scale problems and engineering design optimization, Journal of Intelligent Manufacturing 23 (2012) 1001–1014.
[2] B. Akay, D. Karaboga, A modified artificial bee colony algorithm for real-parameter optimization, Information Sciences 192 (2012) 120–142.
[3] E. Alba, J.M. Troya, A survey of parallel distributed genetic algorithms, Complexity 4 (1999) 31–52.
[4] T. Back, Evolutionary Algorithms in Theory and Practice, Oxford University Press, New York, 1996.
[5] M. Baker, Cluster Computing White Paper, Technical Report, University of Portsmouth, UK, 2001.
[6] A. Banharnsakun, T. Achalakul, B. Sirinaovakul, Artificial bee colony algorithm on distributed environments, in: Second World Congress on Nature and Biologically Inspired Computing, Fukuoka, Japan, 2010.
[7] J.A. Bank, Code Importing Techniques for Fast, Safe Client/Server Access. Technical Report. Technical Report MIT/LCS/TR-702, 1996.
[8] A. Basturk, R. Akay, Parallel implementation of synchronous type artificial bee colony algorithm for global optimization, Journal of Optimization Theory and Applications 155 (2012) 1095–1104.
[9] B. Basturk, D. Karaboga, An artificial bee colony (ABC) algorithm for numeric function optimization, in: Proc. IEEE Swarm Intelligence Symposium, Indianapolis, USA, 2006.
[10] C.M.V. Benitez, H.S. Lopes, Parallel artificial bee colony algorithm approaches for protein structure prediction using the 3DHP-SC model, Intelligent Distributed Computing IV (2010) 255–264.
[11] A. Eiben, J. Smith, Introduction to Evolutionary Computing, 2nd ed, Springer, 2007.
[12] E. Gabriel, G.E. Fagg, G. Bosilca, T. Angskun, J.J. Dongarra, J.M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R.H. Castain, D.J. Daniel, R.L. Graham, T.S. Woodall, Open MPI: Goals, concept, and design of a next generation MPI implementation, in: 11th European PVM/MPI Users' Group Meeting Proceedings, Budapest, Hungary, 2004, pp. 97–104.
[13] J. Gibbons, S. Chakraborti, Nonparametric Statistical Inference, Chapman and Hall/CRC Press, Taylor and Francis Group, 2011.
[14] F. Kang, J. Li, Z. Ma, Rosenbrock artificial bee colony algorithm for accurate global optimization of numerical functions, Information Sciences 181 (2011) 3508–3531.
[15] D. Karaboga, An Idea Based on Honey Bee Swarm for Numerical Optimization. Technical Report TR06. Erciyes University, Engineering Faculty, Computer Engineering Department, 2005.
[16] D. Karaboga, B. Akay, Artificial bee colony (ABC) algorithm on training artificial neural networks, in: Signal Processing and Communications Applications, SIU 2007. IEEE 15th, 2007, pp. 1–4.
[17] D. Karaboga, B. Akay, A comparative study of artificial bee colony algorithm, Applied Mathematics and Computation 214 (2009) 108–132.
[18] D. Karaboga, B. Akay, C. Ozturk, Artificial bee colony (ABC) optimization algorithm for training feed-forward neural networks, in: V. Torra, Y. Narukawa, Y. Yoshida, (Eds.), MDAI, 2007, pp. 318–329.
[19] D. Karaboga, B. Akay, C. Ozturk, A novel clustering approach: Artificial bee colony (ABC) algorithm, Applied Soft Computing 11 (2011) 652–657.
[20] D. Karaboga, B. Basturk, A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm, Journal of Global Optimization 39 (2007) 459–-471.
[21] D. Karaboga, B. Basturk, On the performance of artificial bee colony (ABC) algorithm, Applied Soft Computing 8 (2008) 687–697.
[22] H. Lillieforsa, On the kolmogorov-smirnov test for normality with mean and variance unknown, Journal of the American Statistical Association (1967) 62.
[23] R. Luo, S.T. Pan, P.W. Tsai, J.S. Pan, Parallelized artificial bee colony with ripple-communication strategy, in: Fourth International Conference on Genetic and Evolutionary Computing, Shenzhen, China, 2010.
[24] V. Manoj, E. Elias, Artificial bee colony algorithm for the design of multiplier-less nonuniform filter bank transmultiplexer, Information Sciences 192 (2012) 193–203.
[25] W.N. Martin, J. Lienig, J.P. Cohoon, Island (migration) Models: Evolutionary Algorithms Based on Punctuated Equilibria, IOP Publishing and Oxford University Press, 1997.
[26] P.M. Murphy, D.W. Aha, UCI repository of machine learning databases, University of California, Department of Information and Computer Science, Irvine, CA, 1994. <http://www.ics.uci.edu/mlearn/MLRepository.html>.
[27] H. Narasimhan, Parallel artificial bee colony (PABC) algorithm, in: World Congress on Nature and Biologically Inspired Computing, NaBIC 2009, India, 2009.
[28] Q. Pana, M. Tasgetiren, P. Suganthan, T. Chuad, A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem, Information Sciences 181 (2011) 2455–2468.
[29] R.S. Parpinelli, C.M.V. Benitez, H.S. Lopes, Parallel Approaches for the Artificial Bee Colony Algorithm, Springer-Verlag, Berlin/Heidelberg, 2011. pp. 329–345 (Chapter Handbook of Swarm Intelligence).
[30] H. Quan, X. Shion, On the analysis of performance of the improved artificial-bee-colony algorithm, in: Proc. Fourth International Conference on Natural Computation (ICNC'08), Jinan, China, 2008.
[31] Z. Skolicki, K.D. Jong, The influence of migration sizes and intervals on island models, in: Proceedings of the 2005 Conference on Genetic and Evolutionary Computation, New York, USA, 2005, pp. 1295–1302.
[32] M. Subotic, M. Tuba, N. Stanarevic, Parallelization of the artificial bee colony (ABC) algorithm, in: Recent Advances in Neural Networks, Fuzzy Systems and Evolutionary Computing.
[33] M. Subotic, M. Tuba, N. Stanarevic, Different approaches in parallelization of the artificial bee colony algorithm, International Journal of Mathematical Models and Methods in Applied Sciences 5 (2011) 755–762.

[34] K. Tang, X. Yao, P.N. Suganthan, C. MacNish, Y.P. Chen, C. Chen, Z. Yang, Benchmark Functions for the cec 2008 Special Session and Competition on Large Scale Global Optimization, Technical Report, Nature Inspired Computation and Applications Laboratory, USTC, China, 2007. <http://nical.ustc.edu.cn/cec08ss.php>.
[35] X. Yao, Evolutionary artificial neural networks, International Journal of Intelligent Systems 4 (1993) 539–567.
[36] X. Yao, Evolutionary Optimization, vol. 48, Springer, US, 2003. pp. 27–53 (Chapter Evolutionary Computation).
[37] A.R. Yildiz, A new hybrid artificial bee colony-based approach for optimization of multi-pass turning operations, Information Sciences 220 (2013) 399–407.
[38] W. Zhou, A. Goscinski, An analysis of the web-based clientserver computing models, in: Proceedings of the Asia-Pacific Web Conference, 1998, pp. 343—348.