Parallelized Multiple Swarm Artificial Bee Colony Algorithm (MS-ABC) for Global Optimization

Milos SUBOTIC, Milan TUBA

Faculty of Computer Science, Megatrend University,
29, Bulevar Umetnosti, Belgrade, 11070, Serbia,
milos.subotic@gmail.com, tuba@ieee.org

Abstract: Swarm intelligence metaheuristics have been successfully used for hard optimization problems. After the initial introduction phase such algorithms are further improved by modifications and hybridizations. Parallelization is usually introduced for performance improvement and better resources utilization. In this paper we present an improved parallelized artificial bee colony (ABC) algorithm with multiple swarm inter-communication and learning that not only significantly improves computational time, but also improves the results. Proposed algorithm was tested on large set of standard benchmark functions and it outperformed the state-of-art ABC algorithm.

Keywords: Artificial bee colony, Optimization metaheuristics, Swarm intelligence, Parallelized algorithms, Nature inspired algorithms.

1. Introduction

Swarm intelligence algorithms became very popular in the past 20 years for efficiently finding suboptimal solutions to intractable optimization problems. Swarm intelligence is the collective behaviour of decentralized, self-organized systems, natural or artificial. Swarm is defined as any loosely structured collection of agents that interact among each other. Swarm intelligence algorithms are trying to model social behaviour of real life agents such as viruses, birds, fish, ants, honeybees etc. Among the oldest swarm intelligence algorithm is the Ant Colony Optimization (ACO) inspired by behaviour of ants in the colony that was proposed by Dorigo [1] and is still being investigated and improved [2], [3], [4]. Also very successful and very well researched among older algorithms is Particle Swarm Optimization (PSO) that simulates social behaviour of flock of birds or school of fish. PSO was a start-up point for honey bee based optimization algorithms where Artificial Bee Colony (ABC) introduced by Karaboga [5] is a relatively new, but very successful [6], [7] optimization metaheuristics. Many new algorithms are introduced regularly like Firefly Algorithm (FA) [8], [9], Seeker Optimization Algorithm (SOA) [10], [11] etc. as the research area continues to be active.

We are witnessing a dramatic change in computer architecture due to the multicore paradigm shift and every electronic device from cell phones to supercomputers confronts parallelism of unprecedented scale [12]. In general, a system of n parallel processors, each of speed k, is less efficient than one processor of speed $n * k$. However, the parallel system is usually much cheaper to build and its' power consumption is significantly smaller. Problems caused by higher clock speeds are excessive power consumption, heat dissipation and current leakage. Power consumption and heat dissipation problems are critical for mobile devices, which are getting more important every year. To that end research in parallelization is of great importance. Seymour Cray used to say: "Would you rather plough a field with two strong oxen or 1024 chicken?", but today's hardware looks more like chicken. It has increasing number of low power cores.

Swarm intelligence algorithms have excessive potential for parallelization, either in terms of better results, faster convergence or shorter time for completing the run of an algorithm. Swarm intelligence algorithms always had long execution times since complex functions with large number of parameters are optimized. Every swarm unit has to evaluate objective function multiple times, so this makes them very appropriate for parallelization. By dividing the population into several processing threads, parallel implementations of population based algorithms produce quality results in a reasonable computational time. Since bio-inspired algorithms are not-deterministic, it is advisable to run them multiple times in order to get more accurate results. Any shortening of execution time of an algorithm that usually takes long to execute and that should be run multiple times, is welcome.

Algorithm can be parallelized in different manners. Pedemonte [13] provides the

following taxonomy for parallel ant colony optimization (Table 1).

Table 1. Parallelization approaches taxonomy

Parallel approach				
no. colonies	One		Many	
cooperation	no	yes	no	yes
model	master slave	cellular	parallel independent runs	multi-colony

Very similar taxonomy can be applied to all population based heuristics. Speeding up the search is not the only incentive for parallelization of those algorithms; parallelization also provides a new search form often exceeding result quality of serial implementation of the same algorithm. Basically, approaches with built-in cooperation are aiming at better quality of results, while approaches without cooperation are aiming at shorter execution time of an algorithm.

Level of granularity can vary from finer grained parallelization such as the one where every agent has its dedicated thread, up to a very course, where whole population is divided into just few threads. Too fine grained implementation has one major disadvantage. There is a rather small portion of work for each agent, so extensive use of CPU time for creating threads and their synchronizations exceeds the benefits of parallel execution of the algorithm.

Master-slave models have one central process (master) that controls and delegates tasks to several other processes (slaves). Complexity of the tasks depends on the granularity of an approach, and can vary from evaluation of objective function and constraints to applying algorithm to the part of search space.

Historically, cellular models were initially designed for massively parallel computers and only latter, with popularization of multicore devices, adapted for execution on those devices. Cellular model has a very fine granularity; usually one population unit is assigned to one execution thread. In order to avoid high excess in communications and synchronization, all interaction between individual population units happens between neighbours.

Parallel independent runs approach is trying to speed up multiple execution of an algorithm. In this approach, there is no collaboration or communication between colonies. In independent parallel run approach, every run of an algorithm is independent thread, so in multicore environment, algorithm is running almost n times faster where n is the number of cores. This approach has no influence on the quality of results. There is no communication between runs in this method of implementation.

Multi-colony approach is much different from previous approaches. Execution time reduction is not the aim of this approach, although it can occur. Main goal is to improve quality of the results. In this approach, every colony has dedicated process. Multiple processes are running same serial version of the algorithm at the same time and at some point, communication among colonies occurs. Depending on the migration policy, colonies exchange some of the results (best, worst, random or combination), and then continue with algorithm execution containing results from other colonies. It has been shown that n colonies with k agents, with different random seeds for each colony, produce better results than one population that has n x k agents. We can observe synergetic effect in this way.

With growing popularity of multi-core computers, many of the population-based algorithm ware parallelized like Ant Colony Optimization [14], [15], [16], Particle Swarm Optimization [17], [18], [19], and Differential Evolution. There are several papers on parallelization of Artificial Bee Colony algorithm, some of which aimed on shortening execution times [20], [21] while others showed application of parallelized ABC algorithm [22]. Researchers in [23], [24] used multi colony model to obtain better quality of results. The latest analysis of coarse-graded parallelized ABC algorithm is [25].

2. Original ABC

Original Artificial Bee Colony algorithm is a relatively new metaheuristic optimization technique inspired by foraging behaviour of honey bees. Possible solutions are represented not by individual bees but by food sources and that is one of the key features of the ABC algorithm. Nectar amount of the food source stands for quality of solution and it is

represented by fitness value of the objective function. There are three kinds of bees in the algorithm; employed bees, onlooker bees and scout bees. Employed bees and onlookers are in charge of exploitation process, while scout bees conduct exploration phase. Colony contains equal number of employed and onlooker bees and that number is equal to the number of food sources. One employed bee is attached to every food source. Employed bees search for food around their food source and they share information about richness of their food source with onlooker bees through wiggle dance. Based on that information, onlooker bees increase search around good food sources. Occasionally, food source become depleted and then employed bee abandons it and converts to scout bee. Scout bees randomly search for new food sources. Like other population inspired algorithms, ABC is iteration based algorithm and search process occurs through numerous iterations. Phases of each iteration for optimization of continuous unconstrained functions are presented through short pseudo code given below.

```
initialize the population of solutions
evaluate the population
while maxCycle is not reached:
    produce new solutions for the employed bees
    apply the greedy selection process
    calculate the probability values
    produce the new solutions for the onlookers
    apply the greedy selection process
    send scout bees
    memorize the best solution achieved so far
end while
```

3. MS-ABC (Our Modification)

Population-based algorithms are fairly easy for parallelization; Artificial Bee Colony is not an exception. Considering this, the aim of the work presented herein was to examine implementation of parallelization on the ABC algorithm for unconstrained optimization problems.

There are few issues like granularity of parallelization, communication pattern and exchange policy that must be resolved when population-based algorithm is parallelized.

Granularity of parallelization is the issue with the most significant impact, both on quality of the results and computational performance of the parallelized algorithm. Granularity can vary from very fine grained, where one unit is attached to one execution process, up to very coarse grained, where whole population is divided into few sub colonies, and each is bound to one execution process. Fine grained parallelization is mostly used when objective function is very complicated and/or when algorithm is implemented on slow execution cores like ones in GPU. There is significant computational overhead if fine grained parallelization model is implemented for simpler objective functions and when modern CPU cores are used. Since modern processors are very fast, only small amount of computational time is used for algorithm calculations, and in the case of not so complicated evaluation function, small amount is also used for calculating evaluation function. Hence more work is spent on creating and synchronizing large number of threads than on the algorithm and objective function calculation. On the other hand, modern GPUs have great number of slow cores, which makes them very suitable for fine grained parallelization.

In this study, very course-grained parallelization is used and it is implemented on the colony level. This parallelization model is also known as island based model. Main parallelization method used in this study was to divide main population into N subpopulations and to run serial version of the algorithm simultaneously. In this paper the sub-colonies (sub-populations) are called swarms. Hence this modification is called the multiple swarm artificial bee colony algorithm (MS-ABC).

There are two main communication types, synchronous and asynchronous. Synchronous parallelization is based on the number of execution steps, which means that at certain point of an algorithm each process is waiting until all other processes finish previous tasks and come to the same point in an algorithm and then all processes can together continue with execution. Asynchronous parallelization does not need to wait until all processes synchronize. In this study, synchronous communication model is used.

Migration policy represents the way solutions migrate among swarms. Quality of obtained results can show significant fluctuation due to chosen exchange (migration) policy. Migration policy based on exchanging best solutions is used in this study.

MS-ABC starts by dividing whole population into swarms of equal size. All swarms execute

serial version of the original ABC algorithm. Swarms start search process with different random seeds in order to cover wider range of search space, increasing a chance to find global optimum and to avoid being trapped in local minimum. These subpopulations exchange solutions among themselves. There are different techniques for solution exchange, but the most common one is to exchange best solutions so that every subpopulation has best solution found in the whole population.

In MS-ABC the following migration policy was used: solutions from all swarms were collected, and then *n* best solutions from all swarms were chosen; in the final phase, *n* randomly chosen solutions in each swarm were replaced with *n* best solutions from all swarms. Our experiments showed that the best results were obtained when *n* was equal to 5. In this way, after every solution exchange, all swarms have the best solutions in their solution matrixes, and they could continue their search from that point.

Beside parallelization, a well know concept from Genetic Algorithms called elitism is introduced. Original ABC uses *limit*, a parameter that prevents replacing solution which is still improving with random solution in the scout bee phase. This means that if some solution is not improved for certain number of steps, and this number is called *limit*, it will be replaced with a random one. This prevents exploitation equation from being performed on solution that cannot be improved further because it is trapped in local minimum or already is an optimum solution for the problem. However, this also means that currently best solution could be replaced if it is not improved more than *limit* times, i.e. if algorithm needs more cycles to try to improve the solution in exploitation phase, currently best solution will be destroyed by being replaced with a random one.

In this study, elitism was used to prevent this from happening. Elitism is a well-known technique widely used in Genetic Algorithms. It is used to preserve best genes and ensures their presence in the next generation. In GA, best individuals are copied to the next generation and the rest of population is filled by individuals created in process of mutation and crossover. In MS-ABC elitism ensures the presence of the best solutions in the next cycle by preventing the best solution to reach limit and to be replaced with random solutions. This means that best solutions can be improved more times than other solutions. At the end of each cycle *max_trial* parameter for *EF* best solutions is decreased by one, where *EF* is elitism factor. This would be dangerous in the original ABC, since there is a possibility for colony to be stacked into local optimum, but in MS-ABC, chances for this to happen are smaller. Other swarms prevent colony to get stacked into local optimum by exchanging solutions among them. If one swarm is trapped in local minimum, there is no need to further improve best solutions from that swarm and solutions from other swarms will replace them next time when communication between swarms occurs.

Original ABC algorithm has very few control parameters: colony size, total number of iterations and limit. MS-ABC keeps all these parameters and adds few more. MS-ABC has two important parameters regarding swarm communication. The first one is the First Exchange Cycle (*FEC*). *FEC* denotes how many cycles of the algorithm will pass until the first communication among swarms occurs. Another important parameter is the Exchange Cycle Rate (*ECR*) which determines how many cycles are completed between two communications among swarms.

If *FEC* is too small, swarms do not have time to find useful solutions, and computational time will be used for exchanging poor quality solutions. If *ECR* is too small, i.e. if swarms communicate too often, the risk of premature convergence increases. On the other hand, if these two parameters are too big, the algorithm loses the power of searching multiple spaces and becomes similar to multiple independent runs approach. Consequently, there should be a good balance between those two demands.

Pseudo code for MS-ABC algorithm is:

```
initialize two swarms with different random seeds
for every swarm:
    evaluate population
    while MFEC not reached:
        send employers
        calculate probabilities
        send onlookers
        send scouts
        reset trial for best bees
        if (cn > FEC and cn mod ECR = 0) exchange ();
        memorize Best
    end while
end for
Choose the best solution from swarms
```

4. Benchmark Functions

Original ABC algorithm [16] was tested on extensive set of 50 benchmark functions. We tested our proposed MS-ABC algorithm on the same set, but we report here the results for 12 benchmark functions where there is some difference. For other functions both algorithms reached known global optimum. Selected benchmark functions with parameter range, dimensionality, characteristics and formulation are given in the Table 2 below.

Table 2. Benchmark functions

No	Range	D	C	Function	Formulation
5	[-1.28, 1.28]	30	US	Quartic	$f(x)=\sum_{i=1}^{n} ix_i^4 + random\ [0,1)$
9	[-10, 10]	4	UN	Colville	$f(x)=100\left(x_1^2-x_2\right)^2+\left(x_1-1\right)^2+\left(x_3-1\right)^2+90\left(x_3^2-x_4\right)^2+10.1\left(\left(x_2-1\right)^2+\left(x_4-1\right)^2\right)+19.8\left(x_2-1\right)\left(x_4-1\right)$
12	[-5, 10]	10	UN	Zakharov	$f(x)=\sum_{i=1}^{n} x_i^2+\left(\sum_{i=1}^{n} 0.5ix_i\right)^2+\left(\sum_{i=1}^{n} 0.5ix_i\right)^4$
13	[-4, 5]	24	UN	Powell	$f(x)=\sum_{i=1}^{n/k}\left(x_{4i-3}+10x_{4i-2}\right)^2+5\left(x_{4i-1}-x_{4i}\right)^2+\left(x_{4i-2}-x_{4i-1}\right)^4+10\left(x_{4i-3}-x_{4i}\right)^4$
16	[-30, 30]	30	UN	Rosenbrock	$f(x)=\sum_{i=1}^{n-1}\left[100\left(x_{i+1}-x_i^2\right)^2+\left(x_i-1\right)^2\right]$
33	[-5, 5]	4	MN	Kowalik	$f(x)=\sum_{i=1}^{11}\left(a_i-\frac{x_1\left(b_i^2+b_i x_2\right)}{b_i^2+b_i x_3+x_4}\right)^2$
37	[-D, D]	4	MN	Perm	$f(x)=\sum_{k=1}^{n}\left[\sum_{i=1}^{n}\left(i^k+\beta\right)\left(\left(x_i/i\right)^k-1\right)\right]^2$
38	[0, D]	4	MN	PowerSum	$f(x)=\sum_{k=1}^{n}\left[\left(\sum_{i=1}^{n} x_i^k\right)-b_k\right]^2$
46	[0, 10]	5	MN	Langerman5	$f(x)=-\sum_{i=1}^{m} c_i\left(\exp\left(-\frac{1}{\pi}\sum_{j=1}^{n}\left(x_j-a_{ij}\right)^2\right)\cos\left(\pi\sum_{j=1}^{n}\left(x_j-a_{ij}\right)^2\right)\right)$
47	[0, 10]	10	MN	Langerman10	$f(x)=-\sum_{i=1}^{m} c_i\left(\exp\left(-\frac{1}{\pi}\sum_{j=1}^{n}\left(x_j-a_{ij}\right)^2\right)\cos\left(\pi\sum_{j=1}^{n}\left(x_j-a_{ij}\right)^2\right)\right)$
49	[-π, π]	5	MN	FletcherPowell5	$f(x)=\sum_{i=1}^{n}\left(A_i-B_i\right)^2$ $A_i=\sum_{j=1}^{n}\left(a_{ij}\sin\alpha_j+b_{ij}\cos\alpha_j\right)$ $B_i=\sum_{j=1}^{n}\left(a_{ij}\sin x_j+b_{ij}\cos x_j\right)$
50	[-π, π]	10	MN	FletcherPowell10	$f(x)=\sum_{i=1}^{n}\left(A_i-B_i\right)^2$ $A_i=\sum_{j=1}^{n}\left(a_{ij}\sin\alpha_j+b_{ij}\cos\alpha_j\right)$ $B_i=\sum_{j=1}^{n}\left(a_{ij}\sin x_j+b_{ij}\cos x_j\right)$

5. Settings

In this paper our proposed Multiple Swarm Artificial Bee Colony Algorithm was compared against the original Artificial Bee Colony Algorithm presented by Karaboga and Akay [16]. Control parameters used in original ABC algorithm are the colony size, maximum number of cycles and limit. In Karaboga's paper, maximum number of objective function evaluations was set to 500,000. In order to make clear comparison between ABC and MS-ABC, control parameters that are specific for MS-ABC, like number of swarms and swarm size had to be set within this limitation. In [16] colony size was set to 50, and since there is an equal number of employed and onlooker bees in the colony, colony size has to be even number. In Karaboga's paper 10,000 cycles was used. If we used the same colony size as Karaboga, by dividing colony into two swarms, each swarm should have 25 bees, and since it has to be even number, size of 24 bees could be used. When colony is divided into three swarms, colony size of 16 could be used. For four swarms, population of each swarm should be 12, but this number is too small for finding any meaningful results. Our experimental studies showed that within these limitations, best results are obtained with four swarms when each of them has a population of 16 bees. In order to use the same number evaluations, we decreased the number of cycles to 7,812. In this way Karaboga had 50 bees times 10,000 cycles, which is equal to 500,000 evaluations, and we have 4 swarms x 16 bees x 7,812 cycles, which is equal to 499,968 evaluation calls and very close to 500,000 function evaluation calls used by Karaboga.

MS-ABC and the original ABC have one more parameter used in both algorithms. It is called *limit*. Limit denotes maximum number of cycles used for improving food source. After the *limit* is reached, food source is abandoned. Then scout bee is sent to replace that food source. Both algorithm use Eq. (1) for calculating the limit.

$$limit = SN * D \quad (1)$$

where D is the dimension of the problem and SN is the number of food sources. There are three additional parameters that have to be set for MS-ABC. First and the most important one is *FEC* and it is calculated according to Eq. (2).

$$FEC = MCN * FF \quad (2)$$

where *MCN* is maximum cycle number and *FF* is *FEC* factor. Our experiments showed that best results were obtained when *FF* is around 0.4, so *FEC* is around 40% of the total number of cycles. In this study *FEC* is set to 3,000. Another important parameter is *ECR*, exchange cycle rate, which shows how many cycles pass between two communications among swarms. According to our experiments for best results *ECR* should be 150. When elitism is used, there is a question for how many solutions trial numbers should be decreased. Our experiments show that quality of results is the highest when *EF* is 1. This means that in every cycle for one best solution of each swarm, trial number instead to be increased for one, is actually decreased by one. In this way, good solutions are saved from being abandoned in the scout bee phase. By using elitism, we are increasing the number of tries for good solutions.

6. Results

30 independent runs with different pseudo random seeds were performed for each of the 12 test problems. Table 3 shows optimization results for all test functions. Better results are printed bold. Mean value, standard deviation and standard error for mean values (SEM) were compared. Mean values were used to illustrate the ability of an algorithm to reach quality results while standard deviation and standard error of mean values showed consistency and robustness of an algorithm.

Quality of the obtained results will be analysed first. For 12 test functions MS-ABC always obtained better solution. Out of these 12 functions, MS-ABC reaches optimum for 2 functions; Langerman5 function and Zakharov function. For 10 functions neither of algorithms managed to reach optimum solution, but MS-ABC showed better results than ABC. This illustrates the ability of MS-ABC to outperform ABC regardless of the type of objective function. Functions 9, 12, 37, 46 and 49 especially showed the supremacy compared to the ABC.

If standard deviation and standard error of means are analysed, we can see that MS-ABC is more consistent and robust than the original ABC. MS-ABC reaches better values for 10 functions, while ABC got better results for Rosenbrock and Langerman10 functions.

Rosenbrock function is very specific with lots of local minima and it is easy to be trapped in one of them. MS-ABC shows greater ability to find optimum solution, but not every time. For Langerman10 function, MS-ABC shows much better performance regarding the quality of results, and a poorer performance regarding standard deviation and standard error of means.

The last column of the Table 3 presents results of an additional experiment. Instead of comparing ABC and MS-ABC algorithms by the number of function evaluation calls, they were compared by the time needed for algorithm completion. We wanted to test what quality of results MS-ABC can obtain with the same amount of time that ABC needed for

Table 3. Results

No	Objective function	Stat	Opt	ABC	MSABC	MSABC same execution times
5	Quatric	Mean	0.000	0.0300166	**0.0179069**	**0.0049854**
		St dev		0.0048660	**0.0042061**	**0.0013943**
		SEM		0.0008880	**0.0007679**	**0.0002546**
9	Colville	Mean	0.000	0.0929674	**0.0105299**	**0.0048765**
		St dev		0.0662770	**0.0143852**	**0.0054948**
		SEM		0.0121000	**0.0026264**	**0.0010032**
12	Zakharov	Mean	0.000	0.0002476	**0.0000000**	**0.0000000**
		St dev		0.0001830	**0.0000000**	**0.0000000**
		SEM		0.0000334	**0.0000000**	**0.0000000**
13	Powell	Mean	0.000	0.0031344	**0.0020674**	**0.0007320**
		St dev		0.0005030	**0.0003912**	**0.0000862**
		SEM		0.0000918	**0.0000714**	**0.0000157**
16	Rosenbrock	Mean	0.000	0.0887707	**0.0678281**	**0.0079711**
		St dev		**0.0773900**	0.1134585	**0.0137600**
		SEM		**0.0141290**	0.0207146	**0.0025122**
33	Kowalik	Mean	0.000	0.0004266	**0.0003728**	**0.0003099**
		St dev		0.0000604	**0.0000589**	**0.0000065**
		SEM		0.0000110	**0.0000108**	**0.0000012**
37	Perm	Mean	0.000	0.0411052	**0.0059271**	**0.0030572**
		St dev		0.0230560	**0.0059858**	**0.0024867**
		SEM		0.0042090	**0.0010928**	**0.0004540**
38	PowerSum	Mean	0.000	0.0029468	**0.0013247**	**0.0003943**
		St dev		0.0022890	**0.0013684**	**0.0004682**
		SEM		0.0004180	**0.0002498**	**0.0000855**
46	Langerman5	Mean	-1.500	-0.9381500	**-1.4999438**	**-1.4999438**
		St dev		0.0002080	**0.0000000**	**0.0000000**
		SEM		0.0000380	**0.0000000**	**0.0000000**
47	Langerman10	Mean	NA	-0.4460925	**-0.8568220**	**-0.9424212**
		St dev		**0.1339580**	0.1768402	0.2537003
		SEM		**0.0244570**	0.0322865	0.0463191
49	FletcherPowell5	Mean	0.000	0.1735495	**0.0013390**	**0.0000151**
		St dev		0.0681750	**0.0042765**	**0.0000439**
		SEM		0.0124470	**0.0007808**	**0.0000080**
50	FletcherPowell10	Mean	0.000	8.2334401	**3.4628974**	**0.1297846**
		St dev		8.0927420	**4.0805144**	**0.4503950**
		SEM		1.4775260	**0.7449966**	**0.0822305**

finishing algorithm execution. Number of swarms was set to 4, *FEC* to 4,000, *ECR* to 200 and elitism factor to 1. Algorithm was performing function optimization until time for serial version of the algorithm was reached. Results are presented in the last column of the Table 3. Quality of the results was further increased by running MS-ABC for as long as ABC needed for finishing and the difference between quality of results reached by MS-ABC and ABC is even more obvious. FletcherPowell5 and FletcherPowell10 show the most significant improvement of results. Consistency and robustness increase even more. In terms of standard deviation and standard error of means, MS-ABC outperforms the original ABC significantly in every case except Langerman10 function.

7. Parallel Performance Evaluation

Parallelization of an algorithm is usually judged not only by the quality of results but by terms like speedup and parallel efficiency. Speedup shows ratio between serial and parallel implementations of an algorithm:

$$Speedup = t_1 / t_p \quad (3)$$

where t_1 is the execution time of the serial version of an algorithm and t_p is the execution time of parallel algorithm using p processors/ cores. Since performance of non-deterministic algorithms was analysed here, it was not enough to run algorithm just once and to measure execution time. Instead, we measured execution time for finishing 30 runs of the algorithm. Parallel efficiency shows the usage of available cores/processors. Eq. (4) shows the expression for calculating parallel efficiency:

$$Parallel\ Efficiency = Speedup / p \quad (4)$$

where p is the number of processors/cores. The CPU we used for testing had 4 physical and 8 logical cores, but since four swarms were used, we disabled HyperThreading, and we consider that p is equal to 4. Results presented in Tables 4 and 5 show that MS-ABC has a very good parallel efficiency, and significant speedup can be reached, although it is a sub-linear speed up.

We can see that speedup is 3.36 on average and it varies from 2.62 to 3.90. Parallel efficiency has a range from 0.65 to 0.97, and average parallel efficiency is 0.84. Although the goal of this paper was not to decrease execution times of the algorithm, MS-ABC shows very good speedup and parallel efficiency. Speedups and parallel efficiencies for functions which are simpler for calculation and/or functions with fewer numbers of parameters have lower values. This is due to the fact that greatest speed improvements lie with parallelization of function evaluation.

Table 4. Execution times

	ABC Exe. time (s)	MSABC Exe. time (s)	Speedup	Parallel efficiency
5	71.19	18.50	3.85	0.96
9	3.45	1.14	3.03	0.76
12	4.81	1.29	3.74	0.93
13	31.55	8.20	3.85	0.96
16	15.63	4.39	3.56	0.89
33	4.44	1.41	3.15	0.79
37	37.06	9.74	3.80	0.95
38	22.29	5.98	3.73	0.93
46	86.79	22.36	3.88	0.97
47	114.10	31.67	3.60	0.90
49	53.71	15.18	3.54	0.88
50	183.85	49.14	3.74	0.94

Table 5. Speedup and parallel efficiency

Value	Speedup	Parallel efficiency
Average	3.36	0.84
Min	2.62	0.65
Max	3.90	0.97

8. Conclusion

In this study we presented modification and parallelization of the ABC algorithm and application to unconstrained numerical functions. We used multiple population based model where whole colony is divided into four sub-colonies that were running serial version of the ABC with different random seeds. Those sub-colonies were called swarms. Results were exchanged between swarms periodically. We also introduced elitism, a well-known concept from genetic algorithms. By using elitism, the loss of good solutions was prevented. In our approach the count of unsuccessful modifications for the best solution in each sub-colony was decreased by one at the end of each cycle. The best

solutions were never replaced by randomly chosen ones in the scout bee phase.

The proposed model was compared against Karaboga and Akay's original ABC algorithm [16] and tested on extensive set of 50 benchmark functions, but results for only 12 functions where there are differences are reported. 30 independent runs were conducted with different random seeds. We compared mean values, standard deviation and standard error of means. Our proposed MS-ABC always reached better results for means, while original ABC twice obtained better results for standard deviation and standard error of means. MS-ABC reached two more know optimums than the original ABC. With average speedup of 3.36 and average parallel efficiency of 0.84, MS-ABC reduced execution times significantly. Karaboga and Akay in [16] favourably compared the original ABC algorithm to genetic algorithm, particle swarm optimization algorithm, differential evolution algorithm and evolution strategies. Since our proposed MS-ABC algorithm improves results of the original ABC it is also indirectly favourably compared to the mentioned state-of-the-art nondeterministic optimization metaheuristics. Future work will extend parallelization method to constrained functions and try different parallelization strategies adjusted for CUDA implementation on GPU.

Acknowledgement

This research is supported by Ministry of Education and Science of Republic of Serbia, Grant No. III-44006

REFERENCES

1. DORIGO, M., V. MANIEZZO, A. COLORNI, **Ant System: Optimization by a Colony of Cooperating Agents**, IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, vol. 26, no. 1, 1996, pp. 29-41.
2. JOVANOVIC, R., M. TUBA, **An Ant Colony Optimization Algorithm with Improved Pheromone Correction Strategy for the Minimum Weight Vertex Cover Problem**, Applied Soft Computing, vol. 11, no. 8, 2011, pp. 5360-5366.
3. JOVANOVIC, R., M. TUBA, **Ant Colony Optimization Algorithm with Pheromone Correction Strategy for the Minimum Connected Dominating Set Problem**, Computer Science and Information Systems, vol. 10, no. 1, 2013, pp. 133-149.
4. TUBA, M., R. JOVANOVIC, **Improved Ant Colony Optimization Algorithm with Pheromone Correction Strategy for the Traveling Salesman Problem**, International Journal of Computers, Communications and Control, vol. 8, no. 3, 2013, pp. 477-485.
5. KARABOGA, D., **An Idea Based on Honey Bee Swarm for Numerical Optimization**, Erciyes University, Kayseri, Turkey, Technical Report-TR06, 2005, p. 10.
6. BRAJEVIC, I., M. TUBA, **An Upgraded Artificial Bee Colony (ABC) Algorithm for Constrained Optimization Problems**, Journal of Intelligent Manufacturing, vol. 24, no. 4, 2013, pp. 729-740.
7. BACANIN, N., M. TUBA, **Artificial Bee Colony (ABC) Algorithm for Constrained Optimization Improved with Genetic Operators**, Studies in Informatics and Control, vol. 21, no. 2, 2012, pp. 137-146.
8. YANG, X.-S., **Firefly Algorithms for Multimodal Optimization**, Proceedings of the 5th international conference on Stochastic algorithms: foundations and applications, 2009, pp. 169-178.
9. TUBA, M., N. BACANIN, B. PELEVIC, **Framework for Constrained Portfolio Selection by the Firefly Algorithm**, International Journal of Mathematical Models and Methods in Applied Sciences, vol. 7, no. 10, 2013, pp. 888-896.
10. DAI, C., W. CHEN, Y. SONG, Y. ZHU, **Seeker Optimization Algorithm: A Novel Stochastic Search Algorithm for Global Numerical Optimization**, Journal of Systems Engineering and Electronics, vol. 21, no. 2, 2010, pp. 300-311.
11. TUBA, M., I. BRAJEVIC, R. JOVANOVIC, **Hybrid Seeker Optimization Algorithm for Global Optimization**, Applied Mathematics and Information Sciences, vol. 7, no. 3, 2013, pp. 867-875.

12. DEMMEL, J., **Optimization of sparse matrix–vector multiplication on emerging multicore platforms**, Parallel Computing, vol. 35, no. 3, 2009, pp. 178-194.

13. PEDEMONTE, M., S. NESMACHNOW, H. CANCELA, **A Survey on Parallel Ant Colony Optimization**, Applied Soft Comp., vol. 11(8), 2011, pp. 5181-5197.

14. NICOARA, E. S., F. G. FILIP, N. PARASCHIV, **Simulation-based Optimization using Genetic Algorithms for Multi-objective Flexible JSSP**, Studies in Informatics and Control, vol. 20, no. 4, 2011, pp. 333-344.

15. SECUI, C. D., S. DZITAC, G. V. BENDEA, I. DZITAC, **An ACO Algorithm for Optimal Capacitor Banks Placement in Power Distribution Networks**, Studies in Informatics and Control, vol. 18, no. 4, 2009, pp. 305-314.

16. KARABOGA, D., B. AKAY, **A Comparative Study of Artificial Bee Colony Algorithm**, Applied Mathematics and Computation, vol. 214, no. 1, 2009, pp. 108-132.

17. KALIVARAPU, V., E. WINER, **Asynchronous Parallelization of Particle Swarm Optimization through Digital Pheromone Sharing**, Structural and Multidisciplinary Optimization, vol. 39, no. 3, 2009, pp. 263-281.

18. FARMAHINI-FARAHANI, A., S. VAKILI, S. M. FAKHRAIE, S. SAFARI, C. LUCAS, **Parallel Scalable Hardware Implementation of Asynchronous Discrete Particle Swarm Optimization**, Eng. Applications of Artificial Intelligence, vol. 23, no. 2, 2010, pp. 177-187.

19. TU, K.-Y., LIANG Z.-C., **Parallel Computation Models of Particle Swarm Optimization Implemented by Multiple Threads**, Expert Systems with Applications, vol. 38, no. 5, 2011, pp. 5858-5866.

20. PARPINELLI, R., C. BENITEZ, H. LOPES, **Parallel Approaches for the Artificial Bee Colony Algorithm**, Handbook of Swarm Intelligence, Springer series Adaptation, Learning, and Optimization, vol. 8, 2010, pp. 329-345.

21. BASTURK, A., R. AKAY, **Parallel Implementation of Synchronous Type Artificial Bee Colony Algorithm for Global Optimization**, Journal of Optimization Theory and Applications, vol. 155, no. 3, 2012, pp. 1095-1104.

22. VARGAS BENÍTEZ, C., H. LOPES, **Parallel Artificial Bee Colony Algorithm Approaches for Protein Structure Prediction Using the 3DHP-SC Model**, Intelligent Distributed Computing IV, vol. 315, 2010, pp. 255-264.

23. RUHAI, L., **Parallelized Artificial Bee Colony with Ripple-communication Strategy**, International Conference on Genetic and Evolutionary Computing, 2010, pp. 350-353.

24. CHRISTOPHER COLUMBUS, C., SIMON S. P., **Profit based Unit Commitment: A Parallel ABC Approach using a Workstation Cluster**, Computers and Electrical Engineering, vol. 38, no. 3, 2012, pp. 724-745.

25. BASTURK, A., AKAY, R., **Performance Analysis of the Coarse-Grained Parallel Model of the Artificial Bee Colony Algorithm**, Information Sciences, vol. 253, 2013, pp. 34-55.