Performance of object-oriented software system for improved artificial bee colony optimization

Nebojsa Bacanin, Milan Tuba, and Ivona Brajevic

***Abstract*—** Artificial bee colony (ABC) metaheuristic algorithm introduced by Karaboga was successfully used on many continuous optimization problems. There is also a corresponding program written in C. This article describes an object-oriented software system for improved artificial bee colony algorithm written in C# with corresponding flexible graphical user interface (GUI). Since this implementation is object-oriented it is easier for maintenance and it uses threads which significantly increases execution speed on multicore processors. The application was successfully tested on standard benchmark problems.

***Keywords*—**Artificial bee colony, Optimization, Software system, Swarm intelligence, Nature inspired metaheuristic algorithms

I. Introduction

A HUGE number of practical problems in industry and business are in the class of intractable combinatorial (discrete) or numerical (continuous or mixed) optimization problems. There are many traditional methods for continuous optimization and many heuristics for discrete problems.

Several modern metaheuristic algorithms (typically high-level strategies which guide an underlying subordinate heuristic to efficiently produce high quality solutions and increase their performance) that apply to both domains have been developed for solving such problems [1], [2]. They include population based, iterative based, stochastic, deterministic and other approaches.

The algorithm that is working with a set of solutions and trying to improve them is called population based. Population based algorithms can be classified by the nature of phenomenon simulated by the algorithm into two groups: evolutionary algorithms (EA) and swarm intelligence based algorithms.

Research branch that models the population of interacting agents is swarm intelligence. Flocking of birds and schooling of fish, ant colonies, bee's behavior, immune systems are few examples of swarm systems. Swarm intelligence systems are typically made up of a population of self-organized individuals interacting locally with one another and with their environment [3]. Even though there is no centralized component that controls the behavior of individuals, local interactions between all individuals often lead to the emergence of global behavior. These characteristics of swarms inspired huge number of researchers to implement such behavior in computer software for optimization problems.

A lot of swarm intelligence algorithms have been developed. For example, Ant Colony Optimization (ACO) is a technique that is quite successful in solving many combinatorial optimization problems [4]. The inspiring source of ACO was the foraging behavior of real ants which enables them to find shortest paths between food sources and their nests. Each ant moves along the path until it reaches intersection, where it decides which path to take. In the beginning, when ants chose next path, it seems as a random choice, but after some time, the majority of them are using optimal path. While walking from their nests to food source, ants deposit a substance called pheromone. Pheromone is a collective memory for ants in the colony and because of this, ant colony has the ability of reconnecting a broken line after a sudden appearance of an unexpected obstacle that has interrupted initial path. Paths that contain more pheromone concentrations are chosen with higher probability by ants than those that contain lower pheromone concentrations. As time passes, pheromone trail evaporates. So, a shorter path will have more pheromone than longer paths, because it will have less time to evaporate before new pheromone is disposed by ants. The pheromone trail is maintained using two types of updates: local and global. Global update is used to assure that better paths persist. Local update is used to avoid using suboptimal path by majority of ants, and it emulates pheromone evaporation.

Particle swarm optimization (PSO) algorithm is another example of swarm intelligence algorithms [5]. PSO simulates social behavior of bird flocking or fish schooling. PSO is a stochastic optimization technique which is well adapted to the optimization of nonlinear functions in multidimensional space and it has been applied to several real-world problems. Improved version of the PSO algorithm is Particle swarm inspired evolutionary algorithm (PS-EA) which is a hybrid model of EA and PSO. PS-EA incorporates PSO with heuristics of EA in the population generator and mutation

Manuscript received February 10, 2011.

The research was supported by the Ministry of Science, Republic of Serbia, Project No. III 44006

M. Tuba is with the Faculty of Computer Science, Megatrend University, Belgrade, Serbia, e-mail: tuba@ieee.org

N. Bacanin is with the Faculty of Computer Science, Megatrend University, Belgrade, Serbia, e-mail: nbacanin@megatrend.edu.rs

I. Brajevic is with the Faculty of Mathematics, University of Belgrade, Serbia, e-mail: ivona.brajevic@googlemail.com

operator while retaining the workings of PSO.

Several metaheuristics have been proposed to model the specific intelligent behavior of honey bee swarms [6], [7], [8]. Bee colony is a dynamical system which gathers information from the environment and adjusts its behavior in accordance to it. The bee swarm intelligence was used in the development of artificial systems aimed at solving complex problems in traffic and transportation [6]. That algorithm is called Bee Colony Optimization meta-heuristic (BCO), which is used for solving deterministic combinatorial problems, as well as combinatorial problems characterized by uncertainty. Another approach inspired by the behavior of real bees is Bees Swarm Optimization (BSO) which is adapted for solving maximum weighted satisfiability (max-sat) problem. Bee inspired algorithms were successfully applied to different practical problems [9], [10], [11], [12].

In this paper, we present a modification of the artificial bee colony (ABC) algorithm proposed by Karaboga and Basturk [13]. We developed our ABC software for solving combinatorial and numeric optimization problems in C# programming language.

II. Bees behavior in nature

For fully apprehension of ABC algorithm it is crucial to understand real behavior of honey bees, because this algorithm is inspired by this natural behavior.

A bee colony is a swarm whose individual agents are bees. Each bee as the low-level component works through a swarm at the global level to form a system. Thus, the system global behavior depends on local interactions and coordination between individuals which leads to an organized teamwork system. This system is characterized by the interacting collective behavior through labor division, distributed simultaneous task performance, specialized individuals, and self-organization. The exchange of information among bees results in a system`s collective knowledge.

Bees` energy source is nectar and two kinds of worker bees are responsible for it: scout and forager bees. Two main processes in a honey bee colony associated with food are exploration and exploitation. While scouts carry out the exploration, foragers control the exploitation. Thus, the increase in the number of scouts encourages the exploration process, the increase of foragers encourages the exploitation processes, and vice versa. Studying the foraging behavior leads to optimal foraging theory that directs activities towards achieving goals. This theory states that organisms forage in such a way as to maximize their intake energy per unit time. In other words, swarm of bees is trying to find and capture the food that contains the most energy by expending the least possible amount of time in real variables.

In foraging process, there are two forms of behavior patterns. First pattern refers to scout and second refers to forager bee. Scouts fly around the hive and search for a food source. When they find source (nectar or pollen), they fly back to the colony and communicate with other bees by flying on a particular region in the comb.

Scout bee behavior pattern can be briefly depicted as a set of the following activities:

- The scout bee flies from its colony, randomly seeking for food sources.
- When it finishes a full trip, it returns to its colony, and announces its presence to other bees by wing vibrations. These vibrations mean that it has a message to communicate.
- If scout bee performs a circular dance, it means that a bee has found a nearby source of nectar or pollen. The nearby bees follow the scout and smell it for the identity of the flowers. They listen to the intensity of scout`s wing vibrations to indicate the value of the food source.
- If food source is close, no directions are given. But, if a source is far away, precise directions are given.
- The abstract convention that the scout makes is that the up position on the comb is the position of the sun. Because bees can see polarized light, they can tell sun position without actually seeing the sun. The scout dances in a precise angle from the vertical. This equals to the horizontal angle of the sun with reference to the colony exit with the location of the food source.
- Besides all this information, the scout bee must also show the others how far away the flower source is. This is done by waggling the abdomen from side to side. Slower the waggling, the further away is the food source.

Forager bees react to the scout bees` show, which is described above. This reaction can be summarized through the following steps:

- The bees in the colony with great attention follow the scout to learn food source directions, and also smell the fragrance of the flower on scout bee, so they can find it when they arrive at the source location.
- Because the sun is moving in the sky, the bees should use an accurate clock sense to adjust for the changing sun position with reference to the food source and the colony exit.

When an acceptable food source is found, the forager takes a load of nectar from that source and return to the colony to unload the nectar and store it. Foraging, as any other activity requires energy, and, therefore, honey bees must evaluate where, what, and how long to forage taking into account the economics of energy consumption and the net gain of food to the colony. Generally bees fly only as far as necessary to secure an acceptable food source from which there is a net gain. Thus, these are the factors that influence foraging behavior of honey bees. The net rate of energy intake is defined as the energy gained while foraging minus the energy spent on foraging, divided by time spent foraging.

III. ABC ALGORITHM

The ABC algorithm is relatively new population based meta-heuristic approach firstly proposed by Karaboga [13], and lately developed by the Karaboga and Basturk [14], [15], [16] and extended to combinatorial problems [17].

In ABC algorithm, possible solution of the problem is represented by the food source. Quality of solution is indicating by the amount of nectar amount of a particular food source.

In ABC algorithm, there are three types of artificial bees (agents): employed, onlookers and scouts [14]. Half of the colony are employed bees. The relation between employed bee and the food source is one-to-one, that means that there is only one employed bee per each food source. If a food source becomes abandoned, mapped employed bee to that food source becomes a scout, and as soon as it finds a new food source, it again becomes employed. Main steps of the algorithm are given below [13]:

```
Initialize.
Repeat
  Place the employed bees on the food
  sources in the memory;
  Place the onlooker bees on the food
  sources in the memory;
  Send the scouts to the search area for
  discovering new food sources.
Until (requirements are met).
```

ABC algorithm, as an iterative algorithm, starts by associating each employed bee with randomly generated food source (solution). In each iteration, each employed bee discovers a food source in its neighborhood and evaluates its nectar amount (fitness). If fitness of new food source is better than the fitness of the old one, employed bee moves to the new source, otherwise it retains the old one. After completing this process, employed bees share food source fitness information with the onlookers. Onlookers select a food source (i) with a probability that is proportional to the fitness of the food source, using the following expression:

$$\frac{f_i}{\sum_{j=1}^{m} f_j} \quad (1)$$

where f_i is the fitness of the solution i, and m is the total number of food sources. From the expression, it is obvious that good food sources will get more onlookers than the bad ones. When all onlookers finished food source selection process, each of them search for the food source in the neighborhood of his chosen food source and computes its fitness. The best among all of this food sources will be the new location of the food source i. In ABC algorithm, at each cycle at most one scout goes outside for searching a new food source and the number of employed and onlooker bees were equal.

In this algorithm, there is also a trial parameter. If a solution (food source) does not improve for a predetermined number of iterations which is a trial value, then that food source is abandoned by its associated employed bee, and the bee becomes a scout.

The scout bee mechanism replaces a solution that has not been improved in a determined number of cycles for a new solution randomly calculated, with uniform distribution, within the search space. This mechanism is described in Equation (2):

$$x_{i,j} = rand(L_J, U_J) \quad (2)$$

where L_J is the lower limit of variable j, U_J is the upper limit of variable j and $rand(L_J, U_J)$ is a random real number (with uniform distribution) within the range $[L_J, U_J]$.

Previously described scout mechanism is satisfying for unconstrained functions optimization, but when performing optimization of constrained functions, for some of them it does not generate acceptable results. In such cases, it needs to be modified.

After the new location of each food source is determined, another iteration of ABC algorithm begins. The whole process is repeated until the termination condition is met.

Particularly interesting is the process of determining food source in the neighborhood of a certain food source. Neighborhood food source has being generated by altering the value of one randomly chosen solution parameter and keeping other parameters unchanged. This can be done by adding to the chosen parameter the product of a uniform variable in [-1,1] and the difference in values of this parameter for this food source and some other randomly chosen food source. Let us notate the solution x_i, and let us suppose that the solution x_i has d parameters with values $x_{i1}, x_{i2} .. .x_{id}$, etc. In order to find a solution x_0 in the neighborhood of x_i, a solution parameter j, and another solution x_k are selected on random basis. Except for the value of the chosen parameter j, all other parameter values of x_i' are the same as in the solution x_i, for example, $x_i'=(x_{i1}, x_{i2},.... x_{i(j-1)}, x_{ij}, x_{i(j+1)...} x_{id})$. The value of x_{ij} (let us denote x_{ij} as v_{ij} to make better distinction between old and new parameter value) parameter in x_i' solution is computed using the following expression:

$$v_{ij}' = x_{ij} + u(x_{ij}-x_{kj}) \quad (3)$$

where u is a uniform variable in [-1,1].

From the Equation (3) we can see that if the difference between the parameters of the x_{ij} and x_{kf} decreases, the perturbation on the position x_{ij} decreases too. Thus, as the search approaches to the optimum solution in the search space, the step length is adaptively reduced.

In Fig. 1, we can see a graphical representation of the Equation (3). Fig. 1 (a) shows that the vector generated by the difference between x_i and x_k defines a search direction. Subsequently the candidate solution is generated by Equation (3) in Fig. 1 (b). It can be noticed that v and v' were generated using the same value of u but with opposite sign. This is possible because it is allowed by interval [-1, 1] for u.

If a parameter produced by this operation exceeds its predetermined limit, the parameter can be set to an acceptable value. In this work, the value of the parameter exceeding its limit is set to its limit value.

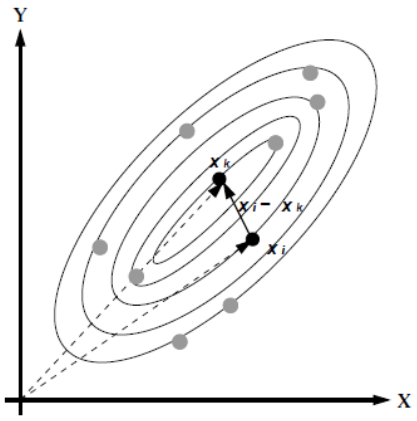

Fig.1 (a): Result vector from $x_i - x_k$.

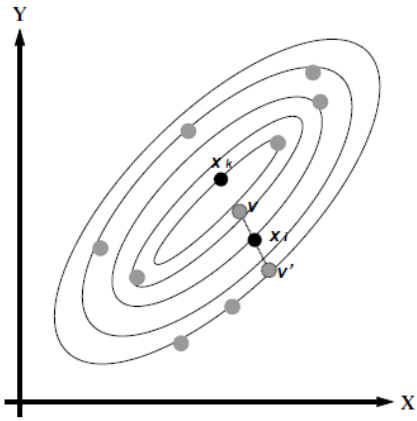

Fig.1 (b): Two possible candidate solutions generated using Equation (3) and the same value of u with opposite sign

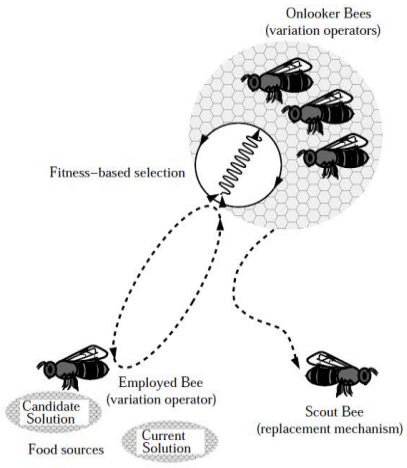

Fig. 2: Graphical representation of the elements of ABC algorithm

In Fig. 2, we show a visual representation of the ABC algorithm compared with the elements of foraging behavior of honey bees (Section 2).

IV. ABCapp Software

We have developed our software for ABC algorithm called ABCapp. We could use existing Karaboga`s software [13], [14], [15], but we chose to develop a new version because we wanted to implement few improvements.

First, in order to make algorithm execute faster, we used multiple threads. Thread is the smallest unit of processing that can be scheduled by an operating system. Each algorithm`s run executes within a different thread, so it runs much faster. Each thread puts best result in an array with number of elements equal to number of runs. Then, we calculate mean result according to the values stored in this array. Threads do not make conflicts with each other, they execute independently. We noticed great performance increase when we run our software on multiple core processors because each thread execute on different core in parallel way. Speed test will be presented in Section 5.

Second, our software is object-oriented. With object-oriented concept, software scalability and maintenance is much easier. Software consists of different components which can be easily replaced. So, if we want to implement new logic for different optimization problems, it will take substantially less time.

We chose to develop ABCapp in C# because of its many advantages over C, C++ and Java. We preferred C# over C even though C is faster. With C# we could gain more control over ABC algorithm execution. Some of C# advantages which made us chose this programming language are:

- Usually it is much more efficient than Java and runs faster.
- CIL (Common (.NET) Intermediate Language) is a standard language, while java byte codes are not.
- It has more primitive types (value types), including unsigned numeric types.
- Indexers let you access objects as if they were arrays.
- Conditional compilation.
- Simplified multithreading.
- Operator overloading. It can make development a bit trickier but they are optional and sometimes very useful.
- Bounds checking (for more than just buffer overflow).
- Partial Classes (C# 2.0 and later).
- Anonymous variables (C# 3.0 and later).
- Better memory management.
- Better exceptions handling.
- Limited use of pointers if you really need them, as when calling unmanaged (native) libraries which does not run on top of the virtual machine (CLR).
- More clean events management using delegates.

We developed our software using Visual Studio 2008 environment and .NET Framework 3.5. A framework is a special kind of software library that is similar to an application program interface (API) in the class of packages that make

possible faster development of applications. Two main components of .NET Framework are Common Language Runtime (CLR) and Class Library. CLR is the .NET runtime environment responsible for program execution management and for providing container services—debugging, exception management, memory management, profiling, and security. The CLR provides a managed environment for code execution, which makes code more secure by protecting the code from doing things such as illegal memory access operations, manages memory for the program and adds additional runtime support not available in native programs, like garbage collection. The .NET class libraries are pre-written classes that provide a rich assortment of pre-defined code. The programmer specifies which classes are being used and furnishes data that instantiate s each class as an object that can be called when the program is executed. Access to and use of a class library greatly simplifies the job of the programmer since standard, pretested code is available that the programmer doesn't have to write.

The use of previously described environment makes our code more robust, errorless and performance is much better.

Implemented ABC algorithm employs four different selection processes:

1. global selection process used by the onlooker bees for discovering promising regions
2. local selection process carried out in a region by the artificial employed bees and the onlookers depending on local information (in case of real bees,
3. local selection process called greedy selection process carried out by all bees in that if the nectar amount of the candidate source is better than that of the present one, the bee forgets the present one and memorizes the candidate source. Otherwise, the bee keeps the present one in the memory.
4. random selection process carried out by scouts.

There are large number of connections between classes in our program. ABC algorithm cannot be used in its basic form for all function optimization problems. So, we created abstract class *BeesAbstract* which is inherited by problem specific classes. *BeesAbstract* has the following methods: *CalculateFitness, MemorizeBestSource, SendEmployedBees, CalculateProbabilities, SendOnlookerBees, SendScoutBees, init, initial, run*. These methods, which will be briefly described, form the basis of ABC metaheuristics and they are similar to those used by Karaboga and Bastruk in their software [13], [14], [15]. *CalculateFitness* calculates the fitness of a solution. *MemorizeBestSource* memorizes best solution found so far. *Init* function initializes variables and counters of the food sources (solutions). Variables are initialized within ranged defined by the user. *Initial* initializes food sources (solutions) at the beginning of the process. *SendEmployedBees* executes employed bee phase. *CalculateProbabilities* calculate probabilities which are important because a food source is chosen with the probability which is proportional to its quality. *SendOnlookerBees* and *SendScoutBees* executes onlooker and scout bee phase respectively. *Run* is specific method used for implementing multiple thread functionality into our software. In the *Run* method, previously described functions are being executed. Pseudo-code for *Run* method is:

```
Initialize
MemorizeBestSource
Repeat
   SendEmployedBees
   CalculateProbabilities
   SendOnlookerBees
   MemorizeBestSource
   SendScoutBees
Until max iterations are met
```

Screenshot of basic graphical user interface (GUI) of ABCapp can be seen in Fig. 3. As we can see from the Fig. 3, user can adjust multiple parameters for ABC algorithm. Parameters are divided into two groups: ABC control parameters and problem specific parameters.

Control parameters are:

- Bee Num NP is number of bees in the colony (employed bees plus onlooker bees).
- Limit controls the number of trials to improve certain food source. If a food source could not be improved within defined number of trial, it is abandoned by its employed bee.
- Max Cycle defines the number of cycles for foraging. This is a stopping criterion.

Problem specific parameters are:

- Param Num D is the number of parameters of the problem to be optimized.
- Runtime defines the number of times to run the algorithm.
- Lower bound is lower bound of problem parameters.
- Upper bound is upper bound of problem parameters.

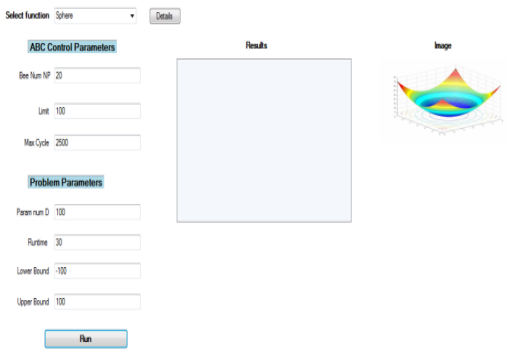

Fig. 3: Screenshot of ABCapp GUI

In the results text area, we can see results for each algorithm`s run, and below, mean results of all runs is shown. Button details give us additional information about the function to be optimized (Fig.5 and Fig.6).

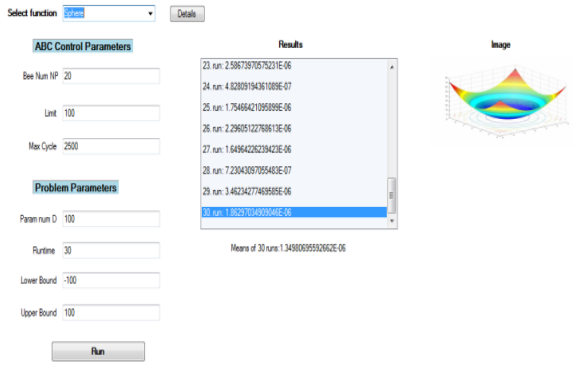

Fig. 4: Results for Sphere function

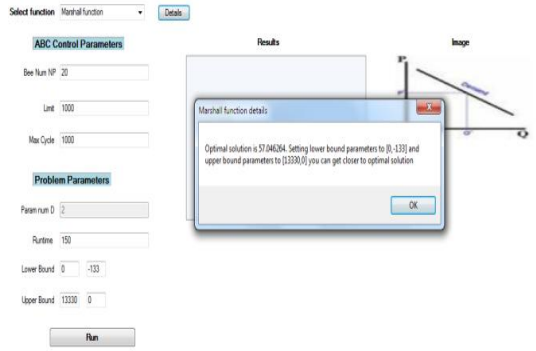

Fig. 5: Additional information about selected function

We used six benchmark functions:

Sphere
Rosenbrock
Griewank
Rastrigin
Schwefel
Marshallian demand function

Sphere function`s value is 0 at its global minimum is (0,0,…,0). Definition:

$$f(x)=\sum_{i=1}^{n} X_i^2$$

Rosenbrock function has a value 0 at its global minimum is (1,1,…,1). Definition:

$$f(x)=\sum_{i=1}^{n}[100(x_i^2-x_{i+1})^2+(x_i-1)^2]$$

Griewank`s value is 0, and its global minimum is (0,0,…,0). Definition:

$$f(x)=\sum_{i=1}^{n}\frac{x_i^2}{4000}-\prod_{i=1}^{n}\cos(x_i/\sqrt{i})+1$$

Rastrigin has value 0, and global minimum (0,0,…,0) . Definition:

$$f(x)=10n+\sum_{i=1}^{n}(x_i^2-10\cos(2\pi x_i))$$

Fifth function is Schwefel whose value is 0 at its global minimum (420.9867,420.9867,…, 420.9867). Definition:

$$f(x)=418.9829n-\sum_{i=1}^{n}(x_i\sin\sqrt{|x_i|})$$

Sixth benchmark function is Marshallian demand function. This function specifies what the consumer would buy in each price and wealth situation, assuming it perfectly solves the utility maximization problem. Function is shown on Figure 7.

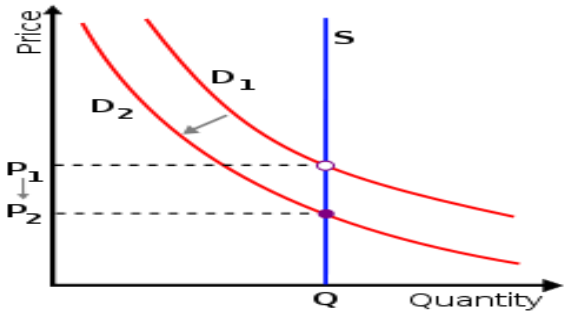

Fig 6: Marshallian demand function

The chosen mathematical model of Marshallian demand function must be the most appropriate approximation of expressed dependency between price flow and demand for observed product. Wide range of mathematical functions can be used for the model, such as: linear, quadratic, exponential, etc. We used $q=ap^b$, where p is price, and q is demand. Parameters a and b should be designated according to previously known pairs , where $i=1,2,...n$. Parameters $a,b,c,...$ are for the most part calculated by using method of least squares.

In this example, we are using function:

$$F(a,b)=\sum_{i=1}^{n}(q_i-ap_i^b)^2 \ ,$$

where we are trying to calculate parameters a and b using ABC algorithm in order to minimize the function value. The best result achieved using method of least squares is 57,046264. The results gained with ABC algorithm are shown in Table 3.

V. Tests and Results

For test purposes, we created test application in C# without multiple threads, like Karaboga`s and Basturk`s software in C programming language [13], [14], [15]. We ran two types of tests. First, we ran speed test, where we compare single thread application to multiple threaded ABCapp (as described in Section 3). Second, we ran optimization tests. For all benchmark functions we set the parameters as shown in Table 1, second column and for the Marshal function the third column. GUI screenshots of parameter sets in ABCapp for benchmark and Marshal function are shown in Fig. 7 and 8, respectively. Tests were done on Intel Core2Duo T8300 mobile processor with 4GB of RAM on Windows 7 Operating System in Visual Studio 2008 environment.

TABLE I
PARAMETER VALUES FOR BENCHMARK FUNCTIONS

Parameter	Bench	Marsh.
Bee Num NP	20	20
Limit	100	1000
Max Cycle	2500	1000
Param Num D	100	2
Runtime	30	150
Lower bound	-100	[0,-133]
Upper bound	100	[13330,0]

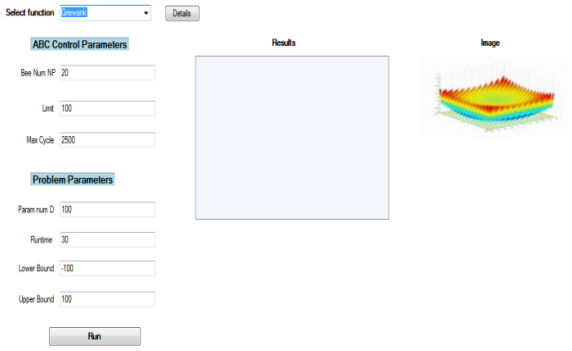

Fig. 7: Parameter set for benchmark functions (ABCapp GUI screenshot)

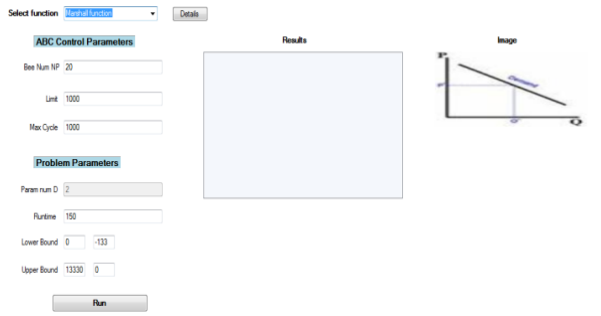

Fig. 8: Parameter set for Marshal function (ABCapp GUI screenshot)

In Table 2, we show results of speed tests (in seconds) between single thread and multiple threads ABC software.

TABLE II
SPEED TEST RESULTS

Function	Single thread	One run - one thread
Sphere	18	15,9
Rosenbrock	32,2	16,1
Griewank	31	12,8
Rastrigin	33	11,9
Schwefel	39	13,6
Marshall	28	10,5

From Table 2, we can see that ABCapp is substantially faster than ordinary C# application. So, when each run executes within different thread, great performance gain is achieved.

In Table 3, we show results of optimization tests for single thread and multiple threaded ABCapp.

TABLE III
RESULTS FOR FUNCTION OPTIMIZATION

Function	Single thread	One run - one thread
Sphere	4,66985 E -06	1,84142 E -06
Rosenbrock	0,095	0,088
Griewank	2,36781 E -09	8,044651 E -10
Rastrigin	2,43016 E -08	1,73152 E - 09
Schwefel	764.972	640.949
Marshallian	57,34713	57,10026

Table 3 shows that ABCapp gives noticeable better results than ordinary ABC software.

We also wanted to see results on both, speed and optimization tests if we slightly change initial set of parameters presented in Table 1. We changed ABC control parameters only, not problem specific parameters (see Section 3). For this test, we used only benchmark functions, Marshal function was omitted. Modified parameters are shown in Table 4. GUI screenshots of modified parameter in ABCapp for benchmark functions are depicted in Fig. 9.

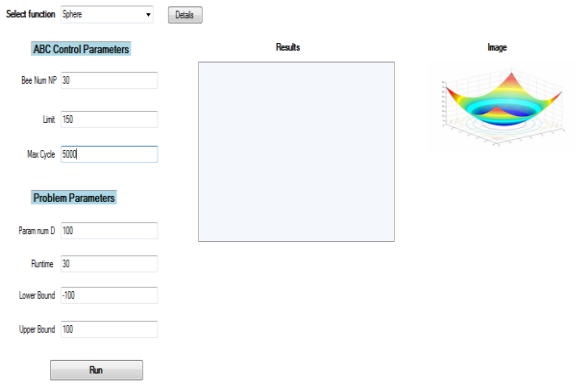

Fig. 9: New parameter set for benchmark functions (ABCapp GUI screenshot)

As it can see from Table 4, we changed *BeeNumNP* from 20 to 30. This means that we increased number of bees (onlooker plus employee) in the colony. Logical consequence of this modification should be better results, but slower program execution. Secondly, we modified *MaxCycle* parameter from 2500 to 5000. By increasing number of cycles for foraging again better results should be obtained, but at the cost of slower execution. Finally, we modified *Limit* parameter to 150. This means that certain food source has more space for improvement before it is abandoned by its bee.

TABLE IV
NEW SET OF PARAMETER VALUES FOR BENCHMARK FUNCTIONS

Parameter	Value
Bee Num NP	30
Limit	150
Max Cycle	5000
Param Num D	100
Runtime	30
Lower bound	-100
Upper bound	100

Results of speed and optimization tests with new parameter set are presented in Tables 5 and 6, respectively.

TABLE V
SPEED TEST RESULTS WITH NEW PARAMETER SET

Function	Single thread	One run - one thread
Sphere	24	17,3
Rosenbrock	50,2	25,6
Griewank	43	15,3
Rastrigin	41	14,6
Schwefel	58	18,1

TABLE VI
RESULTS FOR FUNCTION OPTIMIZATION TEST WITH NEW PARAMETER SET

Function	Single thread	One run - one thread
Sphere	4,25613 E -15	6,03337 E -16
Rosenbrock	0.073	0.069
Griewank	1,11318 E -14	5,99524 E -16
Rastrigin	1,95012 E -11	6,28115 E - 12
Schwefel	466.162	290.953

Table 5 shows again that ABCapp is much faster than ordinary C# application, just like it was with previous parameter set (see Table 2). With new parameters, speed difference between multiple threads and single thread application is greater and in this case, ABCapp remarkably outperforms ordinary single thread ABC application.

According to the results presented in Table 6, ABCapp gains far better results than ordinary ABC application. So, with changing the ABC control parameters, results gap between ordinary software and ABCapp is even greater. This gap is the most pronounced in the case of Griewank function.

Interesting comparison can be made if we compare results for benchmark functions with initial parameter values to the results for benchmark functions with new parameter values (Table 3 vs. Table 6). Both, on single thread and on one run-one thread application, results on benchmark function optimization are far better with new parameters than with the old ones. Let us take for example Griewank function. With initial parameter values, the result achieved using ABCapp software was 8,044651 E -10, while with new parameters result was 5,995243 E -16. As we can see, there is great value improvement. The most remarkable improvement is produced on Sphere function. The difference between 1,84142 E -06 (initial parameter set) and 5,13900 E -16 (new parameter set) is very significant. Even with these parameters results for Schwefel function are worse than others but compatible with discussion in [17].

Finally, we ran both tests on the same hardware platform, but in the different environment. Thus, we used the same Intel Core2Duo T8300 mobile processor with 4GB of RAM on Windows 7 Ultimate Operating System, but this time we tested our software on Visual Studio 2010 and .NET Framework 4.0. We converted both applications (ABCapp and standard ABC software) to comply with the new environment. For testing purposes, we used new set of parameters (see Table 4), just like in the previous test. Speed test results on Visual Studio 2010 are shown in Table 7, while optimization test results can be seen in Table 8.

TABLE VII
SPEED TEST RESULTS ON VISUAL STUDIO 2010

Function	Single thread	One run - one thread
Sphere	21	16,7
Rosenbrock	48,9	23,2
Griewank	41,5	14,1
Rastrigin	39,3	12,1
Schwefel	55,8	17,7

Speed test show that execution speed is slightly better on Visual Studio 2010 than it was on Visual Studio 2008. This means that new environment is better optimized.

TABLE VIII
FUNCTION OPTIMIZATION RESULTS ON VISUAL STUDIO 2010

Function	Single thread	One run - one thread
Sphere	3,22219 E -15	5,13900 E -16
Rosenbrock	0.071	0.066
Griewank	0,92881 E -14	6,10024 E -16
Rastrigin	0,75221 E -11	2,87902 E -12
Schwefel	409.265	194.403

If we compare results in Table 6 and Table 8 (function optimization results on Visual Studio 2008 and Visual studio 2010), it can be seen that results in those tables are approximately the same, differences are due to different random seeds. So, the conclusion is that a different environment does not have impact on the optimization results.

VI. CONCLUSION

We implemented and tested a software system in C# for optimization problems based on a modification of Karaboga's ABC algorithm and corresponding software. Object-oriented

design and appropriate GUI allow for easy modifications and applications to different optimization problems. Performance was tested and proved to be superior to existing software since use of threads better utilizes multicore processors. Benchmark problems that are used in the literature were tested and system is ready to be applied to new problems.

REFERENCES

[1] Johann Dréo, Patrick Siarry, Alain Pétrowski and Eric Taillard, Metaheuristics for Hard Optimization, Springer Berlin Heidelberg, pp. 1-19, 2006.

[2] T. Y. Chen,Y. L. Cheng: Global optimization using hybrid approach, WSEAS Transactions on Mathematics, Vol.7 ,2008 , pp. 254-262.

[3] Saif Mahmood Saab , Dr. Nidhal Kamel Taha El-Omari, Dr. Hussein H. Owaied, Developing optimization algorithm using artificial bee colony system, UbiCC Journal, Volume 4, No 5, pp. 391-396, December 2009.

[4] N. Buniyamin, N. Sariff, W. A. J. Wan Ngah, Z. Mohamad, Robot Global Path Planning Overview and a Variation of Ant Colony System Algorithm, International Journal of Mathematics and Computers in Simulation, Vol. 5, Issue 1, 2011, pp. 9-16.

[5] Milan Rapaic, Zeljko Kanovic, Zoran Jelicic, A theoretical and empirical analysis of convergence related particle swarm optimization, WSEAS Transactions on Systems and Control, Vol. 4, Issue 11, Nov 2009, pp. 541-550.

[6] Teodorovic, D., Dell'Orco M., Bee colony optimization—a cooperative learning approach to complex transportation problems, Advanced OR and Al. Methods in Transportation, pp. 51-60, 2005.

[7] Drias, H., Sadeg, S., Yahi, S, Cooperative bees swarm for solving the maximum weighted satisfiability problem, LNCS, Volume 3512/2005, Springer, Berlin, pp. 318 - 325, 2005.

[8] Benatchba, K., Admane, L., Koudil, M, Using bees to solve a data-mining problem expressed as a max-sat one, LNCS, Volume 3562/2005, Springer, Berlin, pp. 212-220, 2005.

[9] L. Jiann-Horng, H. Li-Ren: Chaotic bee swarm optimization algorithm for path planning of mobile robots, Proceedings of the 10th WSEAS international conference on evolutionary computing, Prague, Czech Republic, 2009, pp. 84-89.

[10] L. Jiann-Horng, L. Meei-Ru, H. Li-Ren: A novel bee swarm optimization algorithm with chaotic sequence and psychology model of emotion, Proceedings of the 9th WSEAS International Conference on Systems Theory and Scientific Computation table of contents, Moscow, Russia, 2009, pp. 87-92.

[11] Tricia Rambharose and Alexander Nikov, Computational intelligence-based personalization of interactive web systems, WSEAS Transactions on Information Science and Applications, Vol. 7, Issue 4, Apr 2010, pp. 484-497.

[12] R. Mohamad Idris, A. Khairuddin and M.W. Mustafa, Optimal Allocation of FACTS Devices in Deregulated Electricity Market Using Bees Algorithm, WSEAS Transactions on Power Systems, Vol. 5, Issue 2, Apr 2010, pp. 108-119.

[13] D. Karaboga, An idea based on honey bee swarm for numerical optimization, Technical Report TR06, Computer Engineering, Department, Erciyes University, Turkey, 2005.

[14] Dervis Karaboga, Bahriye Basturk, A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm, Springer Science and Business Media, pp. 459-471, 2007.

[15] D. Karaboga, B. Basturk, On the performance of artificial bee colony (ABC) algorithm, Applied Soft Computing 8, pp. 687-697, 2008.

[16] Dervis Karaboga, Bahriye Basturk: A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm, Journal of Global Optimization (2007) 39, pp. 459-471

[17] Alok Singh, An artificial bee colony algorithm for the leaf-constrained minimum spanning tree problem, Applied Soft Computing, Vol. 9, Issue 2, 2009, pp. 625-631.

Milan Tuba received B.S. in mathematics, M.S. in mathematics, M.S. in computer Science, M.Ph. in computer science, Ph.D. in computer science from University of Belgrade and New York University.

From 1983 to 1994 he was in the U.S.A. first as a graduate student and teaching and research assistant at Vanderbilt University in Nashville and Courant Institute of Mathematical Sciences, New York University and later as an assistant professor of electrical engineering at Cooper Union Graduate School of Engineering, New York. During that time he was the founder and director of Microprocessor Lab and VLSI Lab, leader of scientific projects and supervisor of many theses. From 1994 he was associate professor of computer science and Director of Computer Center at University of Belgrade, Faculty of Mathematics, and from 2004 also a Professor of Computer Science and Dean of the College of Computer Science, Megatrend University Belgrade. He was teaching more than 20 graduate and undergraduate courses, from VLSI design and Computer architecture to Computer networks, Operating systems, Image processing, Calculus and Queuing theory. His research interest includes mathematical, queuing theory and heuristic optimizations applied to computer networks, image processing and combinatorial problems. He is the author of more than 100 scientific papers and a monograph. He is coeditor or member of the editorial board or scientific committee of number of scientific journals and conferences.

Prof. Tuba is member of the ACM since 1983, IEEE 1984, New York Academy of Sciences 1987, AMS 1995, SIAM 2009. He participated in many WSEAS Conferences with plenary lectures and articles in Proceedings and Transactions.

Nebojsa Bacanin received B.S. and M.S. in economics and computer science in 2006 and 2008 from Megatrend University of Belgrade and also M.S. in computer science in 2008 from University of Belgrade

He is currently Ph.D. student at Faculty of Mathematics, Computer science department, University of Belgrade and works as teaching assistant at Faculty of Computer Science, Megatrend University of Belgrade. He is the coauthor of two papers. His current research interest includes nature inspired metaheuristics.

Mr. Bacanin participated in WSEAS conferences**.**

Ivona Brajevic received B.S. in mathematics in 2006 and M.S. in mathematics in 2008 from University of Belgrade, Faculty of Mathematics.

She is currently Ph.D. student at Faculty of Mathematics, Computer science department, University of Belgrade and works as teaching assistant at College of Business, Economy and Entrepreneurship in Belgrade. She is the coauthor of two papers. Her current research interest includes nature inspired metaheuristics.

Ms. Brajevic participated in WSEAS conferences.