An Object-Oriented Software Implementation of a Modified Artificial Bee Colony (ABC) Algorithm

Nebojsa BACANIN
Faculty of Computer Science
University Megatrend Belgrade
Bulevar umetnosti 29
SERBIA
nbacanin@megatrend.edu.rs

Milan TUBA
Faculty of Computer Science
University Megatrend Belgrade
Bulevar umetnosti 29
SERBIA
tubamilan@ptt.rs

Ivona BRAJEVIC
Faculty of Mathematics
University of Belgrade
Studentski trg 16
SERBIA
ivona.brajevic@googlemail.com

Abstract: - This paper describes an object-oriented software system for continuous optimization by a modified artificial bee colony (ABC) metaheuristic. Karaboga's ABC algorithm was successfully used on many optimization problems and there is also a corresponding program in C. We implemented a modified version in C# which is easier for maintenance since it is object-oriented and which uses threads and significantly increases execution speed on multicore processors. The application includes flexible GUI (graphical user interface) and it was successfully tested on standard benchmark problems and one additional problem.

Key-Words: - Artificial Bee Colony, Optimization Metaheuristic, Software System, Swarm Intelligence

1 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]. 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 behaviour, 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 [2]. Even though there is no centralized component that controls the behaviour of individuals, local interactions between all individuals often lead to the emergence of global behaviour. These characteristics of swarms inspired huge number of researchers to implement such behaviour 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. The inspiring source of ACO was the foraging behaviour of real ants which enables them to find shortest paths between food sources and their nests. While working from their nests to food source, ants deposit a substance called pheromone. Paths that contain more pheromone concentrations are chosen with higher probability by ants than those that contain lower pheromone concentrations.

Particle swarm optimization (PSO) algorithm is another example of swarm intelligence algorithms. PSO simulates social behaviour 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 operator while retaining the workings of PSO.

Several metaheuristics have been proposed to model the specific intelligent behaviour of honey

bee swarms [3], [4], [5]. Bee colony is a dynamical system which gathers information from the environment and adjusts its behaviour 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 [3]. 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 [3]. Another approach inspired by the behaviour of real bees is Bees Swarm Optimization (BSO) which is adapted for solving maximum weighted satisfiability (max-sat) problem.

In this paper, we present a modification of the *Artificial Bee Colony* (ABC) algorithm proposed by *Karaboga* and *Bastuk* [6]. We developed our ABC software for solving combinatorial and numeric optimization problems in C# programming language.

2 ABC algorithm

The ABC algorithm is relatively new population based meta-heuristic approach firstly proposed by *Karaboga* [7], and lately developed by the *Karaboga* and *Bastuk* [6], [8], [9]. 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 [8]. 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 [6]:

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) [10]. In each iteration, each employed bee discovers a food source in its neighbourhood 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:

$$\frac{fi}{\sum_{j=1}^{m} fj} \quad (1)$$

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 neighbourhood 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 bee becomes a scout. 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 neighbourhood of a certain food source. Neighbourhood 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 [10]. 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 neighbourhood 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 xi` are the same as in the solution $xi$, 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} parameter in xi`

solution is computed using the following expression:

$$x_{ij}`=x_{ij}+u(x_{ij}-x_{kj}) \quad (2)$$

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

From the Equation (2) 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.

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.

3 ABCapp Software

We`ve developed our software for ABC algorithm called ABCapp. We could use existing *Karaboga*`s software [6], [8], [9], but we chose to develop a new version because we wanted to implement few improvements. Firstly, in order to make algorithm execute faster, we used multiple threads. Each algorithm`s run executes within a different thread, so it runs much faster. Each thread puts best result in an array length of 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 4.1.

Secondly, our software is object-oriented. With object-oriented concept, software scalability and maintenance is much easier. 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 prefer 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.
- 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.

So, using previously described environment makes our code more robust, errorless and performance is much better.

Implemented ABC algorithm employs four different selection processes:

(1) a global selection process used by the onlooker bees for discovering promising regions
(2) a 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) a 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) a random selection process carried out by scouts [8].

There are a 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, init, initial, SendEmployedBees,CalculateProbabilities,SendOnlookerBees, SendScoutBees, run*. These methods, which will be briefly described, form the basis of ABC metaheuristics and they are similar to those used by *Karaboga* and *Bastuk* in their software [6], [8], [9]. *CalculateFitness* calculates the fitness of a solution. *MemorizeBestSource* memorizes best solution found so far. *Init* function initializes varbiales 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 proportioal 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 given in Figure 1.

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

Fig.1: Pseudocode for Run method

Screenshot of basic *Graphical user interface* (GUI) of ABCapp can be seen in Figure 2. As we can see from the Fig.2, 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.

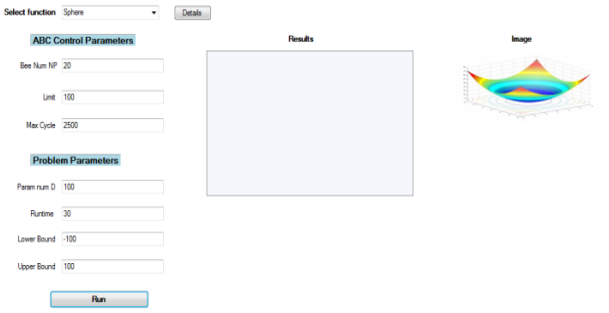

Fig. 2: Screenshot of ABCapp GUI

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.3 and Fig.4).

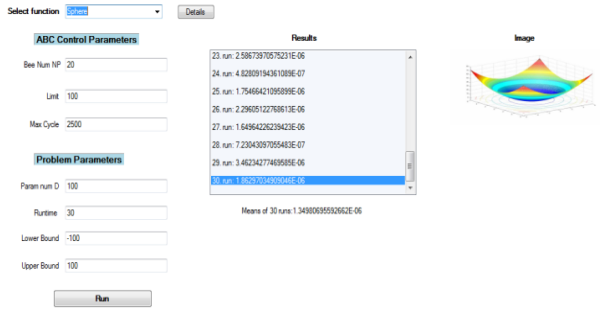

Fig. 3: Results for Sphere function

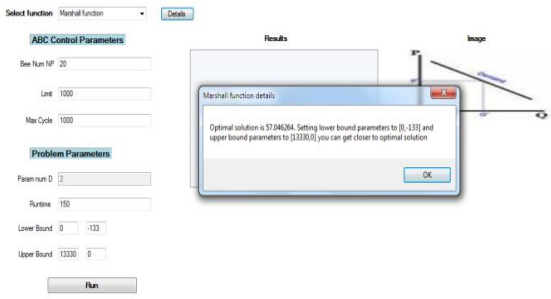

Fig. 4: Additional information about selected function

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:

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

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 of Figure 5.

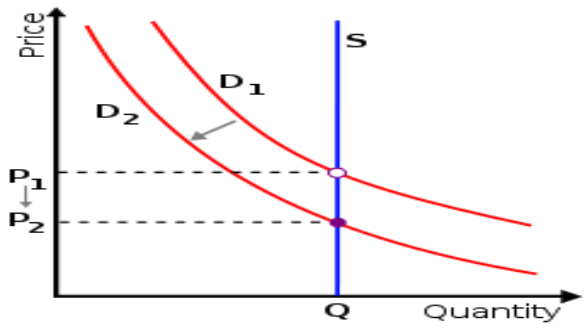

Fig 5: Marshallian demand function

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 (p_i, q_i), 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.

4 Tests and Results

For test purposes, we created test application in c# without multiple threads, like *Karaboga`s* and *Bastuk`s* software in C programming language [6], [8], [9]. 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. Tests were done on Intel Core2Duo T8300 mobile processor with 4GB of RAM on Windows 7 Operating System in Visual Studio 2008.

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]

Table 1. Parameter values for benchmark functions.

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

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

Table 2: Speed Test results

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 4, we show results of optimization tests between single thread and multiple threaded ABCapp.

Function	Single thread	One run - one thread
Sphere	4,66985 E -06	1,84142 E -06
Rosenbrock	192,21429	171,92439
Griewank	2,36781 E -09	8,044651 E -10
Rastrigin	11,10223	9,10561
Schwefel	39632,89250	33665,88734
Marshallian	57,34713	57,10026

Table 3: Results for function optimization

From Table 4 can be seen that ABCapp gives noticeable better results than ordinary ABC software.

5 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.

Acknowledgment: This research is supported by Project 144007, Ministry of Science, Republic of Serbia.

References:

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] D. Karaboga, B. Basturk, *Artificial bee colony (ABC) optimization algorithm for solving constrained optimization problems*, Lecture Notes in Artificial Intelligence 4529, Springer - Verlag, Berlin, pp. 789–798, 2007.

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

[8] 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.

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

[10] 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.