Multithreaded implementation and performance of a modified artificial fish swarm algorithm for unconstrained optimization

Milan Tuba, Nebojsa Bacanin, and Nadezda Stanarevic

***Abstract*—**Artificial fish swarm (AFS) algorithm is one of the latest additions to the swarm intelligence family of metaheuristics for hard optimization problems. This paper presents multithreaded, object-oriented implementation of the modified artificial fish swarm algorithm for unconstrained optimization. Our proposed algorithm's modifications consist of reinforced exploration and different local search i.e. way to generate new candidate solution. Our software system implementation for unconstrained optimization problems was done in the .NET production environment using C# programming language with flexible GUI (Graphical User Interface) and it was successfully tested on five standard unconstrained benchmark problems using different sets of parameters.

***Keywords*—**Artificial fish swarm, Swarm intelligence, Optimization metaheuristics, Nature inspired algorithms, Unconstrained optimization.

I. Introduction

OPTIMIZATION is a process of finding the extreme value of a function in the domain of its definition (unconstrained optimization), possibly subject to various constraints on the variable values (constrained optimization). Numerous deterministic mathematical techniques are available to solve different types of optimization problems, continuous or combinatorial, both unconstrained and constrained, but these methods do not always produce satisfactory results. The development of methods for solving wide range of optimization problems has been conditioned by the size and complexity of the problems. The efficiency of such methods is measured by their ability to find acceptable results within a reasonable amount of time [1].

Fortunately, for solving complex problems (for example the Travelling salesman problem – TSP), usually we do not need to find the optimal solution, good quality suboptimal solution within reasonable computational time is satisfactory. Heuristic methods (or simply heuristics) have been devised to tackle such problems. These algorithms have relatively low computa-tional complexity (typically polynomial) and do not guarantee that the optimal solution will be retrieved. Heuristics improve algorithm's performance by shortening the execution time at the cost of accuracy. There are two types of heuristic methods [2]: constructive and local search heuristics. Constructive heuristics build solution to the problem in a step by step manner until the complete solution is generated. They start up with an "empty" solution and gradually build a complete solution. On the other side, when local search heuristics is used, complete solution is randomly chosen from the population of potential solutions, and then that solution is being incrementally improved during algorithm's execution. In its basic form, local search heuristics search neighborhood of the current solution and choose its best neighbor.

With the growing requirements for hard problem solving, metaheuristics have occupied attention in recent decades. Metaheuristics can be determined as the collection of algorithms' concepts which are used for defining general heuristic methods applicable on wide variety of problems [2]. Thus, metaheuristics search for a good heuristics for a particular problem.

The use of nature-inspired metaheuristics designed to solve optimization problems has become very popular. The corner stone of nature-inspired algorithms are the basic principles found in nature and social behavior. They are applicable to a number of problems, including scientific, industrial and commercial. Nature-inspired heuristic techniques should fulfill several requirements [3]:

- ability to handle different type of problems;
- ease of use with few control variables;
- good convergence mechanism to the global minimum in consecutive independent trials.

Nature-inspired metaheuristics consist of two families of algorithms [4]: evolutionary computation (EC) and swarm intelligence algorithms. Although both these groups of algorithms are used for solving optimization problems, they are complementary and can be joined together to create hybrid algorithms for hard-to-solve problems. The field of EC is often considered to comprise four major paradigms [4]: genetic

Manuscript received October 22, 2012. Revised April 29, 2013.

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 of Belgrade, Serbia, e-mail: tuba@ieee.org

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

N. Stanarevic is with the Faculty of Mathematics, University of Belgrade, Serbia, e-mail: srna@stanarevic.com

algorithms (GA), evolutionary programming (EP), evolution strategies (ES) and genetic programming (GP).

Swarm intelligence algorithms are focused on insect behavior and mimic insect's problem solution abilities. The key concept of the swarm intelligence lies in a simple set of rules that control each of the individuals which exhibit remarkable collective intelligence. Their advantage lies in the fact that they provide many near-optimal solutions in each iteration of the algorithm's execution and choose the best solution according to a given criteria. The emergent behavior of multiple unsophisticated agents interacting among themselves and with their environment leads to a functional strategy that is useful to achieve complex goals in an efficient manner [5]. Ants for instance are capable of finding the shortest path from their colony to food sources and back, while bees perform waggle dances to convey useful information on nectar sources to their hive mates. Swarm intelligence models have many desirable properties which were adopted from real systems and include feedback and adaptation to changing environments and multiple decentralized interactions among agents to work collaboratively as a group in completing complex tasks [5]. From the computational perspective, swarm metaheuristics are stochastic search algorithms enhanced with robust and efficient search process and diversity maintaining capability. With these characteristics, they successfully deal with the distributed and multimodal optimization problems. It also should be noted that the mechanism of forgetting is also adopted in swarm intelligence algorithms such that the solution space can be explored in a comprehensive manner. With this mechanism, the algorithms are able to avoid convergence to a local optimum solution and at the same time to discover a global optimum solution with a high probability [5].

Incorporating intelligent behavior of insects or animal groups such as flocks of birds, schools of fish, colonies of ants [6], [7], [8], [9] bees [10], [11], [12], cuckoos [13], [14], etc. [15] into computer algorithms are typical examples of a swarm intelligence algorithms.

The optimization algorithms which are inspired by intelligent behavior of school of fishes (Artificial fish swarm - AFS) are among the most recently introduced techniques. Several approaches have been proposed to model the specific intelligent behaviors of fish and since its invention the AFS algorithm has been successfully applied to many kinds of problems [16], [17], [18], [19], [20], [21]. According to the various applications mentioned above, AFS algorithm confirmed its good performance.

In this paper, we will present our implementation of modified AFS algorithm [20]. For the purpose of testing its robustness and performances, we developed software named modified AFS System (mAFSs) for solving unconstrained optimization problems in .NET environment using C# programming language. Our proposed modifications of the algorithm in [20], the software system, as well as testing results on standard benchmark functions for unconstrained problems will be in detail showed in this paper.

The organization of the remainder of the paper is as follows. Section 2 describes the original AFS paradigm. In Section 3 exhaustive discussion of modified AFS algorithm is represented. Section 4 shows in detail our software implementation of the modified AFS algorithm. In Section 5 we present the test results obtained by mAFSs application on five standard unconstrained benchmarks with various parameters. Conclusions and future work are contained in the final Section 6.

II. Fundaments of the AFS Algorithm

Mathematical models, which are adopted in the AFS algorithm, imitate the fish swarm series of behavior in nature and can be defined as follows [22]:

- random behavior;
- searching behavior;
- swarming behavior;
- chasing behavior;
- leaping behavior;

Random behavior is general fish behavior and it refers to random movement in the water for seeking food and other fish. *Searching* behavior is tightly connected with fish's vision or sense. When fish senses or visualizes areas with more food, it goes straight and quickly to those areas. Fish naturally organize themselves into the groups called swarm. This *swarming* behavior enables them to avoid risks and dangers and to ensure their survivor. *Chasing* behavior is correlated with the scenario when a fish or group of fish in the swarm discovers food, and other fish in the neighborhood also find that food dangling after it. *Leaping* behavior manifests when fish cannot find more food in a particular region. In this case, fish leaps to look for food in the other locations.

Movements of the artificial fish in the algorithm are implementations of the above listed mathematical models. Each fish is an independent artificial agent. The position of a fish in the search space is represented as a point, which can be for example a vector in multi-parameter function optimization problems (each vector component encodes one function's parameter). Collection of all points (potential solutions to the problem) forms the search space. Thus, AFS metaheuristics utilizes population of points to locate promising regions in the search space and potentially optimal solution.

Complying with the fact that AFS algorithm presented in this paper tackles global optimization problems, following notation will be employed: $x^i \in R^n$ is the *i*-th fish (point) in the population, x^{best} is the point with the least function value and f^{best} is the function value of point x^{best}. In this case, x^i is vector and $x_k^i \in R^n$ is the *k*-th ($k=1,\ldots,n$) component of vector x^i, m is the number of points in the population.

The next behavior of artificial fish depends on its current state and environmental state. Random behavior is expressed in the initialization phase of the algorithm. The crucial step in the AFS algorithms is a "visual scope". A basic biological behavior of any animal is to discover a region with more food, by its vision or sense. The visual scope of each point x^i is defined as the closed neighborhood of x^i with ray equal to a positive quantity called "visual" [20]. Depending on the current position of the individual in the population, marked as $x^i \in R^n$, three possible situations may occur [20]:

1. when the "visual scope" is empty ($np^i=0$, see (1)) and there are no other individuals in its neighborhood to follow, x^i individual moves randomly searching for a better region;
2. when the "visual scope" is crowded, the x^i individual has difficulty to follow any particular individual, and searches for a better region choosing randomly another location from the "visual scope";
3. when the "visual scope" is not crowded, the x^i individual can choose between two options: to swarm moving towards the central or to chase moving towards the best location within the "visual scope".

The condition that determines the crowd issue of x^i individual in the "visual scope" is given in (1):

$$\frac{np^i}{m} \leq \theta, \tag{1}$$

where $\theta \in (0, 1]$ is the crowd parameter, m is the number of individuals in the population and np^i is the number of individuals in the "visual scope".

When the "visual scope" is empty or when it is crowded, fish enters random or searching behavior phase respectively. In the first case, fish xi moves randomly seeking for a better area. In the second case, fish x^i chooses a random point within boundaries of its "visual scope" and a movement towards it is carried out if it improves current x^i location. Otherwise, fish employs random behavior like in the first case.

When "visual scope" is not crowded, fish x^i has option to swarm or to chase. The swarming behavior is characterized by a movement towards the central point in its "visual scope". The swarming behavior is progressive stage that is activated only if the central point has a better function value than the current xi, and it is defined as [20]:

$$s = \frac{\sum_{j \in I^i} x^j}{np^i}, \tag{2}$$

where I^i is the set of indices of the points inside the "visual scope" of point x^i, where $x^i \notin I^i$ and $I^i \in \{1,\ldots,m\}$. If the central point has worse function value than x^i, the searching behavior is activated.

The other option is that the point x^i follows the chasing behavior. The chasing behavior presents a movement towards the point x^{min} that has the least function value within the "visual scope". Again, like in the previous case, if function value of xmin, denoted as $f(x^{min})$ is worse than value of x^i, denoted as $f(x^i)$, searching behavior is employed.

The swarm and chase behavior can be considered as local search procedures. These two behaviors deal with the process of exploitation of previously discovered solutions. Leaping behavior solves the problem when the best objective function value (x^{best}) in the population does not change for a certain number of algorithm's iterations. In this scenario, there is a real possibility that the algorithm is trapped in suboptimal region (local optimum). Leaping behavior conducts the process of exploration by selecting random point (individual) from the population and helps the algorithm to leap out from suboptimal region of the search space. This exploration process empowers algorithm for obtaining better results in solving numerous problems.

III. Modified AFS Algorithm

In this Section we present modified AFS algorithm, along with our additional modifications, which were first introduced in [20]. We briefly describe algorithm's outline, as well as eight main implemented methods. This background is necessary for understanding inner working of our software system implementation described in the next Section.

Modification of AFS metaheuristics are directed towards:

- new implementation of random, searching and leaping fish behaviors;
- a greedy selection applied in the selection process;
- local search at the end of each iteration aiming to improve the best solution;
- a priority-based strategy which increase speed of fish movements.

Algorithm proposed in [20] has been forged for solving bound constrained optimization problems in aspect that feasibility is always satisfied during the algorithm's run. We slightly modified this algorithm by reinforcing exploration process with more frequent triggering of leaping behavior and by using different implementation of *Local* search around the best solution. Pseudo code of our algorithm is given below:

```
Start
   $t = 0$
   $x^i(t)\ (i = 1,\ldots,m) \leftarrow$ Initialize
  While (stopping criteria are not met) do
    For (each $x^i(t)$)
      If ("visual scope" is empty)
         $y^i(t) \leftarrow$ Random($x^i(t)$)
      else If ("visual scope" is crowded)
         $y^i(t) \leftarrow$ Search($x^i(t)$)
      else
         $y^i(t) \leftarrow$ best of Swarm($x^i(t)$) and Chase($x^i(t)$)
    End for
    $x^i(t+1)(i = 1,\ldots,m) \leftarrow$ Select($x^i(t), y^i(t)\ (i = 1,\ldots,m)$)
    If (leaping criteria is met or "stagnation" occurs)
      $x^{rand}(t+1) \leftarrow$ Leap($x^{rand}(t+1)$)
    $x^{best}(t+1) \leftarrow$ Local($x^{best}(t+1)$)
    $t = t + 1$
  End while
End
```

In the presented algorithm, t represents counter of iterations, while x^{rand} is used to denote a randomly selected point from the population of candidate solutions.

As we can see, the algorithm employs eight main methods: *Initialize*, *Random*, *Search*, *Swarm*, *Chase*, *Select*, *Leap* and *Local*. All mentioned methods and also other implementation details will be briefly described in the next few paragraphs.

Initialize method generates random initial population of candidate solutions (m points in the set of Ω). Each vector x^i is calculated using (3):

$$x^i_k = l_k + \alpha(u_k - l_k), \text{ where } k=1,2\ldots,n, \tag{3}$$

where u_k and l_k represent the upper and lower bounds of parameters respectively, and α is uniformly distributed random number in the range $[0,1]$ $(\alpha \sim U[0,1])$.

In the body of *Initialize* function, values of best and worst candidate solutions found in the population are also computed. If the objective is to minimize function, (4) is used:

$$f^{best} = \min(f(x^i), \quad i = 1, \ldots, m \text{ and } f^{worst} = \max(f(x^i), \quad i = 1, \ldots, m \tag{4}$$

Visual scope as a fixed "visual" value for all the population is defined as:

$$v = \delta \max(u_k - l_k), \text{ for } k=1,2\ldots,n \tag{5}$$

In (5), δ is positive visual parameter which is in initial AFS algorithm maintained fixed during iterative process. According to conducted experiments, slow reduction of δ hastens the convergence to the optimal solution [23]. Thus, this algorithm's implementation uses the following modification every s iterations [20]:

$$\delta = \max\{\delta^{min}, \pi\delta\}, \tag{6}$$

where $\pi\delta$ is in the range (0,1), and δ^{min} is sufficiently small positive constant.

Random method is triggered when visual scope is empty as well as in *Search* method when condition that x^{rand} is worse than x^i is satisfied. Details are shown in pseudo code below.

```
**For** (each component $x_k$)
  $\alpha_1 \sim U[0,1]; \alpha_2 \sim U[0,1]$
  **If** $(\alpha_1 > 0.5)$
    **If** $((u_k - x_k) > v)$
      $y_k = x_k + \alpha_2 v$
    **else**
      $y_k = x_k + \alpha_2 (u_k - xk)$
  **else**
    **If** $((x_k - l_k) > v)$
      $y_k = x_k - \alpha_2 v$
    **else**
      $y_k = x_k - \alpha_2 (x_k - l_k)$
**End for**
```

Search method is activated when the "visual scope" is crowded. A point inside "visual scope" (x^{rand}) is selected in a random manner, and the point x^i is moved towards x^{rand} if condition that $f(x^{rand})<f(x^i)$ is satisfied. If not, the point x^i is moved randomly (see Random method above). x^i is dislocated towards x^{rand} using direction $d^i = x^{rand} - x^i$ [20]. In the algorithm shown below, simple movement along a direction d is depicted.

```
$\alpha \sim U[0,1]$
**For** (each component $x_k$)
  **If** $(d_k > 0)$
    $y_k = x_k + \alpha \frac{dk}{||d||} (u_k - x_k)$
  **else**
    $y_k = x_k + \alpha \frac{dk}{||d||} (x_k - l_k)$
**End for**
```

When the "visual scope" of a point x^i is not crowded *Swarm* and *Chase* methods are invoked. In this scenario, the point may have two behaviors. The first one is correlated with movement towards the central point of "visual scope" (denoted as c). Direction of the movement is defined in *Swarm* method as $d^i = c - x^i$. xi is moved according to the algorithm shown in Search method if $f(c)<f(x^i)$. *Chase* method is related with a movement towards the point with the least function value x^{min} and it defines direction $d^i = x^{min} - x^i$.. x^i is moved according to the algorithm shown in *Search* method if x^{min} uplifts x^i. Otherwise, the method *Search* is triggered.

Code inside *Select* method is used to determine whether or not the foregoing selected trial point $y^i(t)$ should proceed as new i-th point position using form of greedy selection:

$$x^i(t+1) = \begin{cases} y^i(t), if(f(y^i(t)) < f(x^i(t)) \\ x^i(t), \text{ otherwise} \end{cases} \tag{7}$$

Leap method is used to maintain diversification in the population, or to help algorithm leap out the local and try to converge to the optimal solution (global minimum). In the first case, the population is enriched with new, random solution when leaping criteria is met (see (8)).

$$\frac{t}{\Psi} = i + 0 \tag{8}$$

where t is the iteration number, is leaping control parameter and i is integer. In the other words, when iteration number is divisible by the leaping control parameter without residuum, leaping behavior is triggered. Ψ is hard coded parameter of the algorithm.

If the algorithm stagnates for a certain number of iterations, it is possible that it has been trapped into a local minimum. In this case, leaping function is invoked every z iterations when the following expression holds:

$$|f^{best}(t)-f^{best}(t-z)|\leq\eta, \quad (9)$$

where η is a small positive tolerance and z defines the periodicity for testing the criterion [20].

Algorithm proposed in [20] calls leaping behavior only if the algorithm stagnates for a certain number of iterations. Our modified algorithm triggers leaping more frequently, and according to conducted tests slightly better results are achieved. Moreover, AFS algorithm described in [20] is special instance of modified one proposed in this paper when $\Psi > t$ parameter set holds. *Leap* behavior code is shown below.

```
For (each component $x_k^{rand}$)
  $\alpha_1 \sim U[0,1]; \alpha_2 \sim U[0,1]$
  If ($\alpha_1 > 0.5$)
    $x_k^{rand} = x_k^{rand} + \alpha_2(u_k - x_k^{rand})$
  else
    $x_k^{rand} = x_k^{rand} - \alpha_2(x_k^{rand} - l_k)$
End for
```

We completely modified *Local* method used in [20]. This method is used for gathering local information around the current best solution in population. In fact, this is random search employed on one component of x^{best}. After choosing random component of x^{best} (denoted as x_{rand}^{best}), and random solution from the population x^{rand}, the following expression is applied:

$$x_{rand}^{best} \;\; += \varphi 1 * (x_{rand}^{rand} - x_{rand}^{best}), \quad (10)$$

where $\varphi 1$ is random number between -0.1 and 0.1. If one component modification of x^{best} gives better result, enhanced current best solution is retained. Similar principle is also used in [24] in mutation operator implementation.

The algorithm is terminated when one of the following conditions is verified:

$$nfe > nfe^{max} or \,|\, f^{\,worst} - f^{\,best} \,| < \varepsilon \quad (11)$$

where *nfe* represents the counter for the number of objective function evaluations, nfe^{max} is the maximum number of function evaluations allowed, and ε is a small positive tolerance. The values fworst and f^{best} were previously defined in (4).

IV. Software Implementation

We have developed and tested our software implementation for modified AFS algorithm which we entitled mAFSs (modified AFS system). We coded software in object-oriented fashion and used multiple threads execution. With object-oriented design, software scalability is improved, and so, implementation of new programming logic for different optimization problems would take less time.

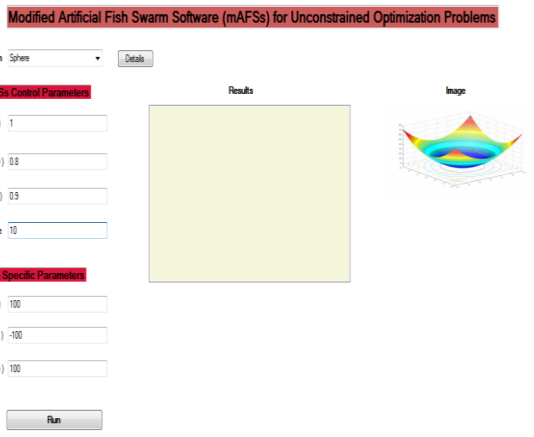
We used approach similar to global parallel EA where the only one population of candidate solution is employed. The only difference between this and our approach is in the level of parallelization. While global parallel EA explicitly parallelize fitness function computation and genetic operators' appliance, we implemented parallelization on the level of individual solutions. In our approach, each solution executes in its own thread. Similar software which parallelized algorithm's runs was proposed for the Artificial bee colony (ABC) algorithm in [25].

We developed software in C# using the.NET Framework 4.0. We chose C# as programming language because of its obvious advantages over C, C++ and JAVA.

We used many classes which are tightly connected. We wanted to make the adaptation process of our algorithm to new optimization problems easy, so we created abstract class *MAFSAbastract* which is later inherited by problem specific classes like in [25]. *MAFSHAbastract* has all above mentioned main methods (see Section 3). We also use *Boolean* methods called *CheckVisualScope* and *CheckStagnation* which check whether "visual scope" is empty or crowded and if stagnation occurs respectively. *CheckVisualScope* returns 0 if "visual scope" is empty and otherwise returns 1. Analogically is done in *CheckStagnation* function. The most important method in our algorithm is *Run* which encapsulates all other methods and enables multi-threaded functionality. Leaping criteria check is implemented as a simple line of code in this method. Pseudo-code for *Run* method is given below (for simplicity reasons, details about multi-threaded functionality are omitted):

```
Initialize
t=0
Repeat
  For ((each $x^i(t)$)
    If((CheckVisualScope=0))
      $y^i(t)$ = Random($x^i(t)$)
    else if ((CheckVisualScope==1))
      $y^i(t)$ = Search($x^i(t)$)
    else
      $y^i(t)$ = best of Swarm($x^i(t)$) and Chase($x^i(t)$)
  End for
  $x^i(t+1)(i = 1,...,m)$ = Select($x^i(t), y^i(t)$ ($i = 1,...,m$))
  If ((CheckStagnation==1) or ($t$ mod $\Psi$)==0)
    $x^{rand}(t+1)$ = Leap($x^{rand}(t+1)$)
  $x^{best}(t+1)$ = Local($x^{best}(t+1)$)
t=t+1
Until stopping criteria is met
```

Screenshot of basic Graphical user interface (GUI) of mAFSs can be seen in Fig. 1. From Fig.1 we can see that user can adjust multiple parameters of the modified AFS algorithm. Other parameters are hard coded into the software and cannot be changed by the user. For simplicity reasons, parameters are divided into two groups: mAFSs control and problem specific parameters.


Modified Artificial Fish Swarm Software (mAFSs) for Unconstrained Optimization Problems

Select function: Sphere

mAFSs Control Parameters

Visual Parameter (δ): 1

Crowded Parameter (θ): 0.8

Reduction Factor (πδ): 0.9

Runtime: 10

Problem Specific Parameters

No. of parameters (n): 100

Lower Bound (l): -100

Upper Bound (u): 100

Fig. 1 Screenshot of mAFSs GUI

Control parameters are:

- visual Parameter (δ) is positive visual parameter (δ >0);
- crowded Parameter (θ) is fraction of the fish that defines crowded situation;
- reduction Factor (πδ) if factor by which δ is reduced as iteration proceed;
- runtime defines number of algorithm's runs.

Problem specific parameters are:

- no. of parameters (*n*) is the number of parameters of the problem to be optimized;
- lower Bound (*l*) is lower bound of problem parameters;
- upper Bound (*u*) is upper bound of problem parameters.

V. Optimization Experiments and Benchmark Results

All tests have been performed on Intel Core2Duo T8300 2.4Ghz mobile processor with 4GB of RAM on Windows 7 Ultimate x64 Operating System in Visual Studio 2010 and .NET Framework 4.0 environment. Only operating system, Visual Studio system processes and mAFSs process have been executing during the tests.

The bound constraint optimization problems in this paper follow the form:

$$\textit{minimize } f(x) \textit{ subject to } x \in \Omega \tag{12}$$

where x is a continuous variable vector with domain $\Omega \subset R^n$, and $f(x) : \Omega \rightarrow R$ is a continuous real-valued function. The domain Ω is defined within upper and lower limits of each dimension [26].

For testing purposes, we used standard five unconstrained benchmark functions:

- Ackley;
- Griewank;
- Rastrigin;
- Rosenbrock;
- Sphere.

Parameter range and formulation of above enlisted benchmarks are given in Table I.

Table I. Benchmark functions summary

Function	Range	Formulation
Ackley	[-32,32]^n	$-20\exp(-0{,}2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) - \exp(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)) + 20 + e$
Griewank	[-600,600]^n	$\frac{1}{400}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos(\frac{x_i}{\sqrt{i}}) + 1$
Rastrigin	[-5.12,5.12]^n	$\sum_{i=1}^{n}[x_i^n - 10\cos(2\pi x_i) + 10]$
Rosenbrock	[-100,100]^n	$\sum_{i=1}^{n-1}[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$
Sphere	[-100,100]^n	$\sum_{i=1}^{n} x_i^2$

We ran two sets of test for each benchmark problem, first with 10 runs, and second with 30 runs, each starting from an independent population with a different random number seed. We wanted to see how runtime effects algorithm's performance.

Values for parameters which are not adjustable by the user (hard coded parameters) are set like in [23]. The number of fish (*m*) in the population depends on n (number of problem parameters), where *m*=10*n*. We used fixed values for nfe^{max} = 250000, ε = 10-4 and η=10-8. Leaping control parameter is set to 5 which mean that at in least each fifth iteration exploration is performed. Because of this parameter is added in our version of the algorithm, we wanted to see how it affects performance, so we run additional test with Ψ set to 3. These results will also be showed in this Section.

Values for parameters which can be controlled through software's GUI (see Fig.1) are shown in Table II and Table III. For all four benchmarks we used the same values for problem specific parameters.

For each benchmark, we show best, mean and standard deviation results. Tests for 10 and 30 runs are shown in Tables IV and V respectively.

Table II Control parameter values

Parameter	Value
Visual Parameter (δ)	1
Crowded Parameter (θ)	0.8
Reduction Factor (πδ)	0.9
Runtime	10/30

Table III Problem specific parameter values

Parameter	Value
No. of parameters (n)	100
Lower Bound (l)	-100
Upper Bound (u)	100

As we can see from Table IV and Table V, mAFSs obtains satisfying results for all presented benchmarks and can be compared with other algorithms and software systems like the one presented in [25].

All runs in conducted experiments were stopped with nfe^{max} = 250,000 as mentioned before. Comparative analysis of results with 10 and 30 runs (Table IV vs. Table V) lead to the conclusion that the performance of the algorithm is not affected by the number of runs. Almost all results are similar with very small digression which can be neglected. The only noticeable difference is observed in tests with *Griewank* and *Sphere* function. In the first case, bests obtained with 10 and 30 runs differ by the factor of 10^{-1}. In *Sphere* benchmarks, mean results differ by the same factor.

Table IV Optimization results for 10 runs

Function		Results
Ackley	Best Mean Stdev.	5.88E-4 0.005 0.003
Griewank	Best Mean Stdev.	1.13E-8 2.09E-6 2.15E-6
Rastrigin	Best Mean Stdev.	8.75E-4 8.31E-3 2.34E-3
Rosenbrock	Best Mean Stdev.	2.15E-2 0.018 0.077
Sphere	Best Mean Stdev.	2.15E-5 0.23E-3 6.88E-4

Table V Optimization results for 30 runs

Function		Results
Ackley	Best Mean Stdev.	1.18E-4 0.004 0.003
Griewank	Best Mean Stdev.	8.97E-9 0.13E-6 1.81E-6
Rastrigin	Best Mean Stdev.	3.62E-4 2.15E-3 1.15E-3
Rosenbrock	Best Mean Stdev.	1.02E-2 0.011 0.056
Sphere	Best Mean Stdev.	1.99E-5 8.77E-4 4.12E-4

As mentioned above, to measure the impact of newly added parameter on algorithm's performance, we conducted another test with 30 runs, but now is set to 3 as opposite to 5 like in previous tests with different runtime values. With the decrease of leaping control parameter, exploration power is elevated and at least, in each third iteration random search occurs.

By comparing Tables V and VI, where Ψ is set to 5 and 3 respectively, we conclude that this parameter does not have strong impact on the performance. It has slightly improved best, means, as well as standard deviation results in tests with all five benchmark functions.

Table VI Optimization results for 30 runs with Ψ set to 3

Function		Results
Ackley	Best Mean Stdev.	0.223E-4 0.004 0.003
Griewank	Best Mean Stdev.	4.24E-9 0.05E-6 0.99E-6
Rastrigin	Best Mean Stdev.	1.33E-4 0.75E-3 1.85E-3
Rosenbrock	Best Mean Stdev.	7.33E-1 0.009 0.031
Sphere	Best Mean Stdev.	0.85E-5 6.39E-4 4.25E-4

VI. Conclusion

In this paper, we presented our implementation of a modified AFS algorithm for solving unconstrained optimization problems. Our algorithm differs that the one proposed in [20]. We introduced additional parameter which is called leaping control parameter (Ψ) in order to maintain diversify in the population. Also, we modified local search procedure around the current best solution. Object-oriented design and appropriate GUI of presented software system allow for easy modifications and adjustments to different optimization problems. The performance of the modified AFS algorithm was tested on several well-known benchmark functions with different number of runs and different sets for Ψ. The algorithm has shown its potential to handle various unimodal and multimodal test functions. As a part of our future work, we are interested in exploring other benchmark and real life problems.

References

[1] Chiong R., Nature-Inspired Algorithms for Optimisation, Springer, 2009, p. 536.
[2] Michalewicz Z., Fogel B. D., How to solve it: Modern Heuristics 2nd edition, Springer-Verlag, 2004, p. 561.
[3] Storn R., Price K., Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces, Journal of Global Optimization, Vol. 11, Issue 4, 1997, pp. 341–359.
[4] Kennedy J., Eberhart, C. R., Swarm Intelligence, Morgan Kaufman, 2001, p. 512.

[5] Chee Peng L., Satchidananda D., Innovations in Swarm Intelligence, Springer, 2010, p. 256.

[6] Jovanovic R., Tuba M., An ant colony optimization algorithm with improved pheromone correction strategy for minimum weight vertex cover problem, Applied Soft Computing, Vol. 11, Issue 8, 2011, pp. 5360-5366.

[7] Tuba M., Jovanovic R.: An Analysis of Different Variations of Ant Colony Optimization to the Minimum Weight Vertex Cover Problem, WSEAS Transactions on Information Science and Applications, Volume 6, Issue 6, June 2009, pp. 936-945

[8] Jovanovic R., Tuba M.: Ant Colony Optimization Algorithm with Pheromone Correction Strategy for Minimum Connected Dominating Set Problem, Computer Science and Information Systems (ComSIS), Vol 10, No. 1, 2013, pp.133-149

[9] Tuba M., Jovanovic, R.: Improved Ant Colony Optimization Algorithm with Pheromone Correction Strategy for the Traveling Salesman Problem, International Journal of Computers, Communications & Control, Vol. 8, Issue 3, 2013, pp. 409-417

[10] Brajevic I., Tuba M., An upgraded artificial bee colony algorithm (ABC) for constrained optimization problems, Journal of Intelligent Manufacturing, 2012, available Springer Online First, DOI: 10.1007/s10845-011-0621-6, pp. 1-12.

[11] Tuba M., Bacanin N., Stanarevic N.: Adjusted artificial bee colony (ABC) algorithm for engineering problems, WSEAS Transaction on Computers, Volume 11, Issue 4, April 2012, pp. 111-120

[12] Subotic M., Tuba M., Stanarevic N.: Different approaches in parallelization of the artificial bee colony algorithm, International Journal of Mathematical Models and Methods in Applied Sciences, Vol. 5, Issue 4, 2011, pp. 755-762

[13] Tuba M., Subotic M, Stanarevic N.: Performance of a modified cuckoo search algorithm for unconstrained optimization problems, WSEAS Transactions on Systems, Volume 11, Issue 2, February 2012, pp. 62-74

[14] Tuba M., Jovanovic R., Brajevic I.: Parallelization of the Cuckoo Search Using CUDA Architecture, Proceedings of the 19th American Conference on Applied Mathematics, Cambridge, MA, USA, 2013, pp. 137-142

[15] Tuba M., Brajevic I., Jovanovic R.: Hybrid Seeker Optimization Algorithm for Global Optimization, Applied Mathematics and Information Sciences, Vol. 7, No. 3, 2013, pp. 867-875

[16] Farzi S., Efficient Job Scheduling in Grid Computing with Modified Artificial Fish Swarm Algorithm, International Journal of Computer Theory and Engineering, Vol. 1, No. 1, 2009, pp. 13-18.

[17] Rocha A. C., Fernandes E., Mutation-Based Artificial Fish Swarm Algorithm for Bound Constrained Global Optimization, Numerical Analysis and Applied Mathematics ICNAM, Article in Press, 2011, doi:10.1063/1.3636841, pp.751-754.

[18] Bing D., Wen D., Scheduling Arrival Aircrafts on Multi-runway Based on an Improved Artificial Fish Swarm Algorithm, ICCIS, Article in Press, 2010, doi: 10.1109/ICCIS.2010.338, pp. 499 – 502.

[19] Jiang M., Mastorakis N., Yuan D., Lagunas M. A., Image Segmentation with Improved Artificial Fish Swarm Algorithm, LNEE Vol 28, Springer, 2009, pp. 133-138.

[20] Rocha A. M. A. C., Fernandes E. M. G. P, Martins T. F. M. C, Novel Fish Swarm Heuristics for Bound Constrained Global Optimization Problems, Comp. Science and its App. – ICCSA 2011, Lecture Notes in Computer Science, Vol. 6784, 2011, doi: 10.1007/978-3-642-21931-3_16, pp. 185-199.

[21] Dong Z., Xiao W., Zhang X., Artificial Fish Swarm Algorithm-Assisted and Receive-Diversity Aided Multi-user Detection for MC-CDMA Systems, Computer and Information Science, Vol. 2, No. 4, 2009, pp. 75-80.

[22] Jiang M., Wang Y., Pfletschinger S., Lagunas M., Yuan D., Optimal multiuser detection with artificial fish swarm algorithm, Communications in Computer and Information Science, Vol. 2, Part 22, 2007, pp. 1084-1093.

[23] Fernandes E. M. G. P., Martins, T. F. M. C., Rocha, A. M. A. C, Fish swarm intelligent algorithm for bound constrained global optimization, CMMSE 2009, ISBN: 978-84-612-9727-6, pp. 461–472

[24] Bacanin N., Tuba M., Artificial Bee Colony (ABC) Algorithm for Constrained Optimization Improved with Genetic Operators, Studies in Informatics and Control, Vol. 22, No. 2, 2012, pp. 137-146.

[25] Bacanin N., Tuba M., Brajevic I., Performance of object-oriented software system for improved artificial bee colony optimization, International Journal of Mathematics and Computers in Simulation, Vol. 5, Issue 2, 2011, pp. 154-162.

[26] Ali M. M., Khompatraporn C., Zabinsky Z. B., A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems, Journal of Global Optimization, Vol. 31, 2005, pp. 635–672.

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. 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, IFNA. 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 a 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 author or coauthor of more than 10 research papers. His current research interest includes nature inspired metaheuristics with accent on swarm intelligence and global optimization.

Mr. Bacanin is also a research member of Project No. III-44006, Ministry of Science, Republic of Serbia.

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

Nadezda Stanarevic 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 She is the author and coauthor of more than 10 scientific papers. Her current research interest includes nature inspired metaheuristics.

Ms. Stanarevic participated in WSEAS conferences.