Artificial Fish Swarm Algorithm for Unconstrained Optimization Problems

Nebojsa BACANIN, Milan TUBA, Nadezda STANAREVIC
Faculty of Computer Science
University Megatrend Belgrade
Bulevar umetnosti 29
SERBIA
nbacanin@megatrend.edu.rs, tuba@ieee.org, srna@stanarevic.com

Abstract: This paper presents an object-oriented software system that implements a modified artificial fish swarm (AFS) algorithm based on a particular intelligent behavior of schools of fish. We outline our implementation of the algorithm for unconstrained optimization problems. The application was implemented in C# with flexible GUI (Graphical User Interface) and it was successfully tested on four standard unconstrained benchmark problems.

Key-Words: - Artificial fish swarm, Swarm intelligence, Metaheuristic optimization, Nature inspired algorithms, Unconstrained optimization

1 Introduction

Optimization is as a process of finding the extreme value of a function in a domain of definition, subject to various constraints on the variable values. Different mathematical programming algorithms offer a set of techniques to solve different type of optimization problems like numerical, discrete or combinatorial optimization problems, but these methods do not always produce satisfactory results. The development of methods for solving wild range of optimization problems has been conditioned by the size and complexity of the problems. The efficiency of suitable solutions is measured by their ability to find acceptable results within a reasonable amount of time [1]. The use of nature-inspired heuristics designed to solve optimization problems has become very popular. These algorithms are focused on insect behavior and mimic insect's problem solution abilities. Collective intelligent behavior of insects or animal groups such as flocks of birds, schools of fish, colonies of ants or bees are typical examples of a swarm system [2], [3]. Their advantage lies in the fact that they provide many near-optimal solutions in each iteration in the algorithm and choose the best solution according to given criteria.

Nature-inspired heuristics 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.

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 fishes and since its invention AFS algorithm has been successfully applied to many kinds of problems [4], [5], [6], [7], [8], [9]. 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 [8]. In order to test its robustness and performances, we developed software named mAFSs, for solving unconstrained optimization problems in C# programming language. This software will be in detail presented in this paper as well as testing results on standard benchmark functions for unconstrained problems.

The organization of the remaining of the paper is as follows. Section 2 details the original AFS paradigm. In Section 3 exhaustive discussion of modified AFS algorithm is represented. Section 4 shows in detail our software implementation of modified AFS algorithm. In Section 5 we present the test results obtained by mAFSs application on four standard unconstrained benchmarks.

This research is supported by Ministry of Science, Republic of Serbia, Project No. 44006

Conclusions and future work are contained in the final Section 6.

2 Basics of the AFS algorithm

Mathematical models imitate the fish swarm series of behavior in nature which can be defined as [8]:

1. Random behavior
2. Searching behavior
3. Swarming behavior
4. Chasing behavior
5. Leaping behavior

The next behavior of artificial fish depends on its current state and environmental state. Random behavior can be presented as 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 vision or sense. Depending on the current position of the individual in the population, marked as $x^i \in R^n$, three possible situations may occur [8]:

1. When the "visual scope" is empty, 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 option: to swarm moving towards the central or to chase moving towards the best location.

The condition that determines the crowd issue of x^i individual in the 'visual scope' is given in Equatation 1:

$$\frac{np^i}{m} \leq \theta \qquad (1)$$

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". In the searching behavior phase, an individual is randomly chosen in the "visual scope" of x^i and a movement towards it is carried out if it improves current x^i location. Otherwise, the x^i individual moves randomly. The swarming behavior is characterized by a movement towards the central point in the "visual scope" of x^i. The swarming behavior is progressive stage that is activated only if the central point has a better function value than the current x^i. Otherwise, the point x^i follows the searching behavior. The chasing behavior presents a movement towards the point that has the last function value, x^{min}. The swarm and chase behavior can be considered as local search. Leaping behavior solves the problem when the best objective function value in the population does not change for a certain number of iterations. In this case the algorithm selects random individual from the population. This process empowers algorithm for obtaining better results in solving numerous problems.

3 Outline of modified AFS algorithm

In this Section we present modified AFS algorithm which was introduced in [8]. We briefly delineate algorithm`s outline, as well as eight main implemented methods. This background is necessary for understanding inner working of our software implementation described in the next Section.

Algorithm proposed in [8] has been forged for solving bound constrained optimization problems in aspect that feasibility is always satisfied during the algorithm`s run. Pseudo code of algorithm is given below [8]:

```
Start
  $t = 0$
  $x^i(t)$ $(i = 1, . . .,m)$ ← Initialize
  While (stopping criteria are not met) do
    For (each $x^i(t)$)
      If ("visual scope" is empty)
        $y^i(t)$ ← Random($x^i(t)$)
      else If ("visual scope" is crowded)
        $y^i(t)$ ← 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 ("stagnation" occurs)
      $x^{rand}(t + 1)$ ← Leap($x^{rand}(t + 1)$)
    $x^{best}(t + 1)$ ← Local($x^{best}(t + 1)$)
    $t = t + 1$
  End while
End
```

In 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 Equitation 2:

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

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, candidate solution`s best and worst function values found in the population are also computed. For this computation, Expression 3 is used:

$$f_{\text{best}} = \min (f(x^i), i = 1, \ldots, m) \text{ and } f_{\text{worst}} = \max (f(x^i), i = 1, \ldots, m) \tag{3}$$

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

$$v = \delta \max (u_k - l_k), \text{ for } k=1,2\ldots,n \tag{4}$$

In Eq.4, 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 [10]. Thus, this algorithm`s implementation uses the following modification every s iterations [8]:

$$\delta = \max\{\delta_{\min}, \pi\delta\} \tag{5}$$

where $\pi\delta$ is in the range (0,1) and $\delta_{\min}$ 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 met. Details are shown below [8]:

```
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 $d^i = x^{rand} - x^i$ [8]. 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” (c). This is denoted as swarming behavior. Direction of the movement is defined in *Swarm* method as $d^i = c - x^i$. x^i 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 *ith* point position using form of greedy selection:

$$x^i(t+1) = \begin{cases} y^i(t), & if \;\; (f(y^i(t)) < f(x^i(t)) \\ x^i(t), & \text{otherwise} \end{cases} \tag{6}$$

If the algorithm stagnates for a certain number of iterations, it is possible that it has fallen into a local minimum. In this scenario, *Leap* method is used to help algorithm leap out the local and try to converge to the optimal solution (global minimum). This function is invoked every z iterations when the following expression holds:

$$|f\text{best}(t) - f\text{best}(t-z)| \leq \eta \tag{7}$$

where η is a small positive tolerance and z defines the periodicity for testing the criterion [8].

Local method is used for gathering the local information around the best current solution in population. In fact, this is primitive random line search employed component by component to x^{best}. This straightforward local search is used to improve accuracy with reduced computational cost although more sophisticated procedure could be used like in [11].

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

$$nfe > nfe_{max} \text{ or } |f_{worst} - f_{best}| < \varepsilon \quad (8)$$

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 f_{worst} and f_{best} were previously defined in Eq. 3.

4 mAFSs implementation

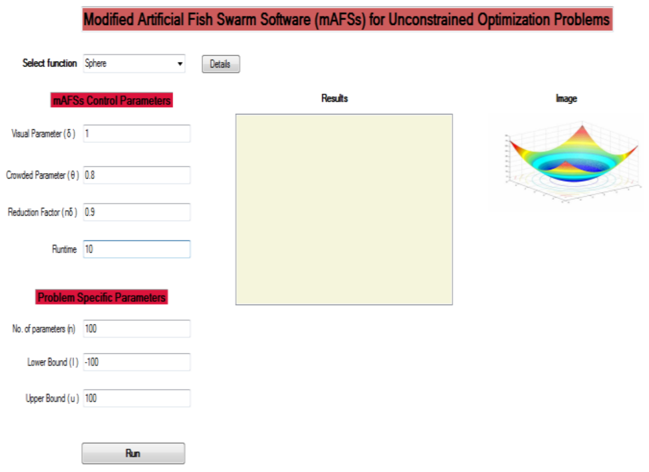
We have developed and tested our software implementation for modified AFS algorithm which we called mAFSs (modified AFS software). We coded software in object-oriented fashion and used multiple threads in its execution. With object-oriented design, software scalability is improved, and so, implementation of new programming logic for different optimization problems would take less time. Each algorithm`s run executes within a different thread, so it runs much faster. Similar software was proposed for Artificial bee colony (ABC) algorithm in [12].

We developed software in C# using the newest .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 *MAFSAbstract* which is later inherited by problem specific classes like in [12]. *MAFSHAbstract* 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. Pseudo-code for *Run* method is given below (for simplicity reasons, details about multi-threaded functionality are omitted):

```
Initialize
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))
    $x^{rand}(t+1)$ = Leap($x^{rand}(t+1)$)
  $x^{best}(t+1)$ = Local($x^{best}(t+1)$)
Until stopping criteria is met
```

Screenshot of basic *Graphical user interface* (GUI) of mAFSs can be seen in Figure 1. From Fig.1 we can see that user can adjust multiple parameters of modified AFS algorithm. Other parameters are hard coded into the software and cannot be changed by the user. For simplicity, 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 ($\delta > 0$).
- *Crowded Parameter* (θ) is fraction of the fish that defines crowded situation.
- *Reduction Factor* ($\pi\delta$) 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.

5 Optimization experiments and bencmark results

All tests have been run on Intel Core2Duo T8300 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 and Visual Studio core processes and mAFSs process have been executing during the tests.

The unconstrained optimization problems in this paper follow the form:

$$minimize\ f(x)\ subject\ to\ x \in \Omega \qquad (9)$$

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 [13].

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

- Sphere
- Rosenbrock
- Griewank
- Rastrigin

We ran two sets of test for each benchmark problem, first with 10 runs, and second with 30 runs, each starting from a random population with a different 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 [10]. The number of fish (m) in the population depends on n (number of problem parameters), where $m=10n$. We used fixed values for $nfe_{\max}$ = 250000, $\varepsilon = 10^{-4}$ and $\eta=10^{-8}$.

Values for parameters which can be controlled through software`s GUI (see Fig.1) are shown in Table 1 and Table 2. As a remark we cite that for all four benchmarks we used the same values for problem specific parameters.

Parameter	Value
Visual Parameter (δ)	1
Crowded Parameter (ϑ)	0.8
Reduction Factor ($\pi\delta$)	0.9
Runtime	10/30

Table 1: Control parameter values

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

Table 2: Problem specific parameter values

For each benchmark, we show best, mean and standard deviation results. Results for 10 and 30 runs are shown in Tables 3 and 4 respectively.

Function		Results
Sphere	Best	2.15E-5
	Mean	0.23E-3
	Stdev.	7.08E-4
Rosebrock	Best	3.40E-2
	Mean	0.029
	Stdev.	0.097
Griewank	Best	5.13E-8
	Mean	4.02E-6
	Stdev.	3.49E-6
Rastrigin	Best	9.95E-4
	Mean	8.31E-3
	Stdev.	2.64E-3

Table 3: Optimization results for 10 runs

As we can see from Table 3 and Table 4, mAFSs obtains satisfying results for all presented benchmarks and can be compared with other algorithms and software systems like [12].

All runs in conducted experiments were stopped with $nfe_{\max}$ = 250000 as mentioned before. Comparative analysis of results with 10 and 30 runs (Table 3 vs. Table 4) lead to conclusion that the performance of the algorithm is not affected by the number of runs. All results are similar with very small digression which can be neglected.

Function		Results
Sphere	Best	6.99E-5
	Mean	1.15E-3
	Stdev.	5.02E-4
Rosebrock	Best	1.02E-2
	Mean	0.015
	Stdev.	0.085
Griewank	Best	2.67E-8
	Mean	2.82E-6
	Stdev.	2.01E-6
Rastrigin	Best	7.62E-4
	Mean	5.29E-3
	Stdev.	1.04E-3

Table 4: Optimization results for 30 runs

6 Conclusions

We present our implementation of AFS algorithm for solving unconstrained optimization problems. Object-oriented design and appropriate GUI allow for easy modifications and applications to different optimization problems. The performance of the modified AFS algorithm was tested on several well-known benchmark functions. 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 [14].

References:

[1] Chiong R., *Nature-Inspired Algorithms for Optimisation*, Springer, 2009, ISBN 978-3-642-00266-3, p. 536.

[2] Bonabeau E., Dorigo M., Theraulaz G., *Swarm Intelligence: From Natural to Artificial Systems*, Oxford University Press Inc., 1999., NY, p. 233.

[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] 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.

[5] Rocha A. C., Fernandes E., *Mutation-Based Artificial Fish Swarm Algorithm for Bound Constrained Global Optimization*, Numerical Analysis and Applied Mathematics ICNAM 2011Vol. 1389, Article in Press, 2011, doi:10.1063/1.3636841, pp.751-754.

[6] 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.

[7] Jiang M., Mastorakis N., Lagunas D., M. A., *Multi-threshold Image Segmentation with Improved Artificial Fish Swarm Algorithm Block-coding and Antenna Selection*, in Proc. of ECC, Article in Press, 2007, doi: 10.1007/978-0-387-85437-3, pp. 123-129.

[8] 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, Volume 6784, 2011, doi: 10.1007/978-3-642-21931-3_16, pp. 185-199.

[9] 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.

[10]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

[11] Rocha A.M.A.C., Fernandes E.M.G.P., *Hybridizing the electromagnetism like algorithm with descent search for solving engineering design problems*. International Journal of Computer Mathematics, 2009, pp. 1932–1946.

[12] Bacanin N., Tuba M., Brajevic I., *An Object-Oriented Software Implementation of a Modified Artificial Bee Colony (ABC) Algorithm,* Proc. of the 11th WSEAS int. conference on neural networks and 11th WSEAS int. conference on e.c. computing and 11th WSEAS int. conference on Fuzzy systems, 2010, ISBN: 978-960-474-195-3, pp. 179-184.

[13] 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, 2005

[14] Mahmood M. Nesheli, Othman C., Arash Moradkhani R., *Optimization of Traffic Signal Coordination System on Congestion: A Case Study*, WSEAS Transactions on Advances in Engineering Education, Vol. 6, Issue 7, 2009, pp. 203-212.