# Artificial Bee Colony (ABC) Algorithm for Constrained Optimization Improved with Genetic Operators

**Nebojsa Bacanin, Milan Tuba**

Megatrend University Belgrade, Faculty of Computer Science,
Bul. Umetnosti 29, 11070 N. Belgrade, Serbia
nbacanin@megatrend.edu.rs, tuba@ieee.org

**Abstract:** Artificial bee colony (ABC) is a relatively new swarm intelligence based metaheuristic. It was successfully applied to unconstrained optimization problems and later it was adjusted for constrained problems as well. In this paper we introduce modifications to the ABC algorithm for constrained optimization problems that improve performance of the algorithm. Modifications are based on genetic algorithm (GA) operators and are applied to the creation of new candidate solutions. We implemented our modified algorithm and tested it on 13 standard benchmark functions. The results were compared to the results of the latest (2011) Karaboga and Akay's ABC algorithm and other state-of-the-art algorithms where our modified algorithm showed improved performance considering best solutions and even more considering mean solutions.

**Keywords:** Artificial bee colony (ABC), Constrained optimization, Swarm intelligence, Nature inspired metaheuristics.

## 1. Introduction

### Optimization Problems

Optimization is one of the most applicable areas in mathematics and computer science since most real-life problems can be described as some kind of optimization problem. The types of mathematical relationships between the objective function, potential constraints and decision variables determine how difficult the particular problem is. Hard optimization problems can be combinatorial (discrete) or continuous (global optimization), where continuous problems can be constrained or unconstrained (bound constrained).

The nonlinear constrained optimization problem in the continuous space can be formulated as follows:

$$minimize\, f(x),\ x = (x_1, x_2, x_3, ..., x_n) \in R^n \quad (1)$$

where $x \in F \subseteq S$. The search space $S$ is an $n$-dimensional hyper-rectangular space in $R^n$ defined by lower and upper bounds for variables:

$$lb_i \le x_i \le ub_i,\ 1 \le i \le n \quad (2)$$

and the feasible region $F \subseteq S$ is defined by a set of $m$ linear or nonlinear constraints:

$$\begin{aligned} &g_j(x) \le 0,\ for\ j = 1,...,q \\ &h_j(x) = 0,\ for\ j = q+1,...,m \end{aligned} \quad (3)$$

where $q$ is the number of inequality constraints and $m$-$q$ is the number of equality constraints.

Most of the optimization algorithms start with random, unfeasible solutions in the initialization phase with expectation that after some number of iterations these solutions will reach the feasible area. However, equality constraints pose a difficult issue since their presence makes the feasible space very small compared to the entire search space. The equality constraints can be replaced by inequality constraints with some small violation limit $\varepsilon > 0$ [1]:

$$|h(x)| - \varepsilon \le 0 \quad (4)$$

The quality of results depends on the choice of the violation limit value $\varepsilon$. If it is selected too small, the algorithm may not find the feasible solutions, and if the tolerance $\varepsilon$ is too large the results may be far from the feasible region.

The promising approaches for handling equality constraints include dynamic, self-adaptive tolerance adjustment [2]. The process should start with a large violation value $\varepsilon$, which is gradually decreased through the cycles of the algorithm. A dynamic setting of the violation value $\varepsilon$ can be defined as follows:

$$\varepsilon(t+1) = \frac{\varepsilon(t)}{dec} \quad (5)$$

where $t$ is the current cycle and $dec>1$ is the decreasing rate value of each cycle.

### Metaheuristics

Many metaheuristic algorithms have been developed recently for solving optimization problems from both domains, numerical and combinatorial. They include population based, iterative based, stochastic, deterministic and other approaches.

One of the oldest metaheuristics for the global optimization problem is simulated annealing (SA) which is recognized as a generic probabilistic method. SA can be applied in many practical industrial problems, such as part type selection and operation allocation problem in flexible manufacturing system (FMS) [3].

Population based algorithms which are working with a set of solutions and iteratively trying to improve them were very successful recently. They can be divided into two groups: evolutionary algorithms (EA) and swarm intelligence.

Prominent among EA are genetic algorithms (GA). GA and other EA have been applied to a wide variety of different problems [4], [5].

Swarm intelligence based on the collective behavior of the social insect colonies and other animal societies has recently become an important research topic. The key concept of swarm intelligence lies in a simple set of rules that control each of the individuals which exhibit remarkable collective intelligence. The swarming concept can also be extended to human group decision process [6].

Particle swarm optimization (PSO) is a swarm intelligence algorithm which simulates social behavior of fish schooling or bird flocking. There are also other PSO approaches like interactive particle-swarm metaheuristic used for multi-objective optimization (MOO) [7].

Ant colony optimization (ACO) is a technique that is quite successful in solving many hard practical optimization problems. The foundation of the ACO is foraging behavior of real ants which are able to find the shortest paths between their nests and food sources due to the substance called pheromone. ACO has been applied to the minimum weight vertex cover problem [8], power distribution problems [9], and many others.

Several metaheuristics have been developed to simulate the specific intelligent behavior of honey bee swarms. Bee colony is a highly dynamical system which collects information from its surrounding and adopts its behavior accordingly. Artificial bee colony (ABC) algorithm is one of the latest representatives of the honey bee swarm algorithms. Originally, the ABC algorithm was proposed by Karaboga for finding global optimum over continuous space [10]. ABC was successfully applied to unconstrained [11] as well as to constrained function optimization problems [12]. Testing results show that the performance of ABC algorithm is comparable to other state-of-the-art algorithms for high dimensionality optimization [13]. ABC has recently become very active research area and many modifications [14] and enhancements [15] of the original algorithm were introduced.

## Our Improvement

The ultimate goal of any metaheuristic algorithm is to find the optimal feasible solution. To achieve this goal, appropriate balance between exploitation and exploration is required at each iteration of the algorithm. By studying the ABC algorithm, we noticed a deficiency during the solution search process. After significant number of iterations, when optimal solution is almost found, scout bees which perform the exploration process are not useful any more, just the opposite. This problem can be treated by better adjustment of exploration and exploitation balance [16], [5].

In order to improve the exploitation process at later stages of the algorithm, we adopted uniform crossover and mutation operators from GA during the replacement process of the exhausted food sources. We have found an appropriate empirical point where some scout bees (according to an appropriate probability parameter) are transformed into a new class of guided onlookers for strong exploitation. This new mechanism of replacing exhausted food sources performs intensive exploitation around current best solution using uniform crossover operator. After crossover, mutation operator takes place. Each function parameter is mutated with small probability thus preventing any variable to keep fixed value indefinitely.

In such a manner, by integrating GA with the ABC, we derived a modified ABC algorithm for constrained optimization improved with genetic operators and named it genetically inspired ABC algorithm (GI-ABC).

The rest of this paper is organized as follows. Section 2 explains the original ABC algorithm; Section 3 describes the principle of the crossover and mutation modifications and adjustment for the GI-ABC algorithm. In Section 4 an analysis of trade-offs between exploration and exploitation is performed first, using various parameter sets. After that, series of comparison experiments on the set of 13 well known *g* benchmark functions are performed to verify the effectiveness of our proposed approach over the latest Karaboga and Akay's [14] ABC algorithm and other state-of-the-art algorithms [17], [18], [19].

## 2. ABC Algorithm Overview

The artificial bee colony (ABC) algorithm was designed for numerical optimization problems, based on the foraging behavior of honey bees [10]. Since the performance of metaheuristic algorithms depend on the number and the choice of parameters, the main advantages of the ABC algorithm are derived from the fact that the algorithm uses only 3 control parameters: colony size, maximum cycle number and limit.

ABC algorithm employs three classes of artificial bees: employed bees, onlookers and scouts. Employed bee stays on a food source (candidate solution) and examines neighborhood. Onlookers are allocated to a food source based on the information which they gain from employed bees. If a food source does not improve for a certain number of cycles, scouts replace that food source with a new, random one.

The main difference between ABC and other swarm intelligence algorithms is based on the fact that the possible solutions are represented by the food sources, not the individuals in the population. The quality of the possible solution is presented as a fitness value that is calculated from the value of the objective function of the problem. Each solution $x_i$ $(i = 1, 2,.., SN)$ is a $D$-dimensional vector, where $SN$ denotes the size of the population. In the ABC algorithm onlookers and employed bees carry out the exploitation process in the search space, while the scouts control the exploration process.

Pseudo-code of the ABC algorithm for constrained optimization problems [12] is:

1. Initialize the population of solutions
2. Evaluate the population
3. cycle=1
4. repeat
5. Produce new solutions for the employed bees by using Equation (6) and evaluate them

$$\upsilon_{i,j} = \begin{cases} x_{i,j} + \phi_{i,j} * (x_{i,j} - x_{k,j}), & R_j < MR \\ x_{i,j} & otherwise \end{cases} \qquad (6)$$

   where $k$ is random number between 1 and $SN/2$ and different from $i$.

6. Apply selection process based on Deb's method [20].
7. Calculate the probability values $p_i$ for the solutions $x_i$, using fitness of the solutions and the constraint violations ($CV$) by

$$CV = \sum_{g_j > 0} g_j(x) + \sum_{q+1}^{m} h_j(x) \qquad (7)$$

$$pi = \begin{cases} \left( 0.5 + \left( \dfrac{fitness_i}{\sum_{i=1}^{SN} fitness_i} \right) * 0.5 \right) & if \ solution \ is \ feasible \\ \left( 1 - \dfrac{CV}{\sum_{i=1}^{SN} CV} \right) * 0.5 & if \ solution \ is \ infeasible \end{cases} \qquad (8)$$

8. For each onlooker bee, produce a new solution $v_{ij}$ by Equation (6) in the neighborhood of the solution selected depending on $p_i$ and evaluate it
9. Apply selection process between $\upsilon_i$ and $x_i$ based on Deb's method
10. Determine the abandoned solutions by using "limit" parameter for the scout; if they exist, replace them with new randomly produced solutions by

$$x_{ij} = lb_j + rand(0,1) * (ub_j - lb_j) \qquad (9)$$

11. Memorize the best solution achieved so far
12. cycle = cycle+1
13. until cycle = $MCN$

## 3. Genetically Inspired ABC

The main difference between the original ABC and our proposed genetically inspired GI-ABC algorithm considers the replacement of exhausted food sources. After a certain number of cycles, we assume that the ABC algorithm has found the proper part of the search space. Thus, there is no longer need for scout's random search which is replaced, with certain probability, with directed search among the best solutions found so far. A point in the algorithm's execution after which scout mechanism is replaced with guided onlookers which perform strong exploitation around current best point is called a *breakpoint* (abbr. *bp*). This process of replacement of food sources in later cycles uses crossover and mutation operators from genetic algorithms.

## GA Operators

In genetic algorithms, candidate solution is often referred to as chromosome. Chromosome can conceptually be divided into genes. Each gene represents a particular element of a candidate solution. Position of the gene in a chromosome is called locus. Different encoding schemes can be used for representing a chromosome, such as binary encoding, real-value encoding and tree encoding. After selection which selects chromosomes for reproduction according to their fitness, crossover and mutation operators take place.

Crossover consists of exchanging genes between two chromosomes (parents) and forming new chromosomes (offspring). Crossover occurs with some probability (crossover rate). If no crossover takes place, offspring is simply a copy of its parent.

There are three basic types of crossover: single-point, multi-point and uniform. In single-point crossover a random point is chosen within the range of the length of chromosome. Then, the genes to the right of randomly chosen point are swapped between two parents, generating two offspring. In multi-point crossover, crossover is performed at one or more points along the chromosome.

In the uniform crossover, for each gene position, the genes from two parents are swapped with a fixed, position independent probability *p*. Unlike single and multi-point crossover, this crossover method enables the parents to contribute to the offspring on the gene level, rather than on the segment level. Value of *p* has determinant influence on forming offspring. If the value of *p* is close to 0.5, then the gene exchange between two parents is frequent. In this case, the exploratory power of uniform crossover is high and the search process has global tendency. Oppositely, if the value of *p* is closer to 0 or 1, then the number of genes swapped between two parents is smaller and the search process is more locally directed.

Mutation is performed by choosing a gene at randomly chosen locus and replacing that gene with another one. Mutation operator maintains diversification in the population ensuring that no single gene position keeps fixed value during the algorithm's run.

## GA Operator Implementation for ABC

In the GI-ABC, GA operators are adopted in the process of replacement of the exhausted food sources. Candidate solution can be considered as a chromosome while genes represent function variables.

As mentioned above, after *breakpoint* iterations scouts are being replaced with guided onlookers with certain probability. This probability is a new parameter of the algorithm and it is called the replacement rate (abbr. *rr*). We empirically determined that the value 0.9 for *rr* gives best results. The parameter is not user adjustable.

If replacement occurs, the best fit and one random individual are chosen from the population as parents for recombination process. Recombination is performed using uniform crossover operator. According to conducted empirical experiments, we found that this crossover method with value of *p* set to 0.5 achieves best results for our purpose. In GI-ABC algorithm, after recombination of two parents, only one offspring is used.

After crossover, mutation operator takes place. Each gene (function parameter) is mutated with small probability according to Equation (10). Mutation probability rate (abbr. *mpr*) is also a control parameter of the algorithm that is not adjustable since algorithm is not very sensitive to its changes.

$$offsp[i] \mathrel{+}= \varphi 1 * (rndsol[i] - offsp[i]) \quad (10)$$

where *offsp* is child solution, *i* is the *i*-th gene and mutation process is applied to all genes for *i* between 1 and *D*, *rndsol* is one randomly selected solution, $\varphi 1$ is random number between -0.1 and 0.1. As can be seen from Equation (10), mutation mildly distracts guided onlooker from strong attraction towards best solution to some other, random direction.

In both, original ABC and the GI-ABC algorithm, employed and onlooker bees change one function parameter. They replace randomly chosen parameter with a new one, whose value is a combination of the current value and value of other randomly selected solution. However, mutation in guided onlooker phase changes all parameters but only by a relatively small value, again in the direction of a random solution, but only after strong bias towards current best solution has already been applied.

After algorithm has reached *bp*, guided onlooker for strong exploitation is triggered with replacement rate probability. Offspring is generated and the solution whose number of trials is exceeded is replaced with the offspring.

If replacement rate condition is not satisfied, the solution with which exhausted one is replaced is generated using Equation (9). Using this mechanism we ensure that the algorithm does not get stuck in the local optimum because in some cases, exploration is still performed.

When algorithm reaches the final stage of its execution (higher number of cycles), we introduce another breakpoint which we called second break point (abbr. *sbp*). We have empirically found that optimal setting is *sbp*=*bp**1.7. The difference between *bp* and *sbp* is in a way how the offspring solution is generated by guided onlooker. After the *sbp* number of cycles, guided onlooker combines two solutions with the highest fitness in the population. With assumption that almost optimum solution is found, this mechanism performs even stronger exploitation. After the creation, the offspring is exposed to mutation in the same way as described above.

Pseudo-code of the GI-ABC algorithm is given below:

```
1.  Initialize the population of solutions
2.  Evaluate the population
3.  If there is an equality constraint, then $\varepsilon = 1.0$
4.  cycle=1
5.  repeat
6.  Produce new solutions for the employed bees by using Equation (6) and evaluate them
7.  Apply selection process based on Deb's method.
8.  Calculate the probability values $p_i$ for the solutions $x_i$, using fitness of the solutions and the constraint violations ($CV$) by Equation (8) where $CV$ is defined by Equation (7).
9.  For each onlooker bee, produce a new solution $\upsilon_i$ by Equation (6) in the neighborhood of the solution selected depending on $p_i$ and evaluate it
10. Apply selection process between $\upsilon_i$ and $x_i$ based on Deb's method.
11. Determine the abandoned solutions by using "limit" parameter for the scout, if it exists, replace it with:
    a) A new randomly produced solution by Equation (9), if $cycle < bp$
    b) An offspring solution of the best and random solution if $bp <= cycle <= sbp$ and if $rr$ condition is satisfied, otherwise use procedure like in a).
    c) An offspring solution of two highest fitness solutions if $sbp < cycle$ and if $rr$ condition is satisfied, otherwise use procedure like in a).
12. Memorize the best solution achieved so far
13. If there is an equality constraint and $\varepsilon > 0.0001$, then reduce $\varepsilon$ according to Equation (5)
14. The constraints of food sources are recalculated using the new tolerance.
15. cycle = cycle+1
16. until cycle = MCN
```

With proper adjustment of *bp*, premature convergence can be avoided. If *bp* is set to lower value, the population of candidate solutions converges too early to a local minimum. In this situation, because of the loss of genetic variation, each candidate solution in population is almost identical.

Our modified GI-ABC algorithm employs five additional control parameters, but they all have fixed values determined empirically. These are *bp*, *rr*, *mpr*, *sbp* and *p* which are all preconfigured inside the algorithm and cannot be changed by the user. Original ABC is a special case of the GI-ABC, with *rr* set to 0 (or alternatively, *bp*>*MCN*). In this case scout is never replaced with guided onlooker and scout generates new solutions as in the original ABC algorithm according to Equation (9).

## 4. Experiments and Discussion

In this section we first present results that GI-ABC algorithm achieved with different parameter sets as a foundation for exploitation-exploration trade-off investigation. To evaluate the performance of the GI-ABC algorithm we used the set of 13 benchmark functions for constrained optimization [21].

All tests were performed on Intel® Core™ 2 Duo T8500 processor @4GHz with 4GB of RAM memory, Windows 7 x64 Ultimate 64 operating system and Visual Studio 2010 .NET 4.0 Framework.

The same basic parameter set as in Karaboga and Akay's ABC [14] algorithm was used for GI-ABC. The value of the modification rate (*MR*) is 0.8, colony size (*SN*) is 40, and the maximum cycle number (*MCN*) is 6000. So, the total number of objective function evaluations is 240,000. The value of limit is set to 150 (*MCN/SN*). For functions with equality constraints we used $\varepsilon=1$, and $dec=1.002$.

## Exploitation-Exploration Balance

Success of population based algorithm depends on establishing balance between exploitation and exploration which contradict each other. GI-ABC manages exploitation-exploration balance by adjusting *bp* and *rr* parameter values. By changing these parameters, optimum and mean results change accordingly. However, best values for these parameters were determined empirically and fixed in the algorithm. Parameter *mpr* occasionally has positive influence on both best and mean results when mutation helps algorithm to leave local minimum.

With decrease of bp or increase of rr, exploitation increases and exploration decreases and vice versa. For most test cases, with decrease of bp or increase of rr, best solution discovered within 30 runs gets better, while mean solution gets worse. This is because when algorithm in early cycles finds proper part of the search space (with little exploration effort), good solutions are combined, and consequently, better solutions are achieved. However, if the algorithm misses right part of the search space in early cycles, combining such low quality solutions leads towards worse results. For example, if in 5 of 30 runs algorithm early targets proper part of the search space, outstanding bests are discovered in these 5 runs, but, in other 25 runs, best solutions are far away from real optimum, which in overall leads to worse mean results. On the other side, by increasing bp and decreasing rr, best results get worse, while means get better. In this case, algorithm has more cycles to find right part of the search space, but has fewer cycles for combining good solutions if it enters the right part of the search space in early cycles.

We performed a set of experiments with varying GI-ABC specific control parameters (*bp, rr, mpr, sbp, p*) in order to investigate exploitation-exploration tradeoffs and impact on best and mean results. Parameter *sbp* depends on *bp*, and parameter *p* for uniform crossover was fixed and set to 0.5, as described in Section 3. We present here results of only few experiments, these with varying *bp*. Full spectrum of parameter sets was used for experiments, but most of them showed low sensitivity and significant independence of parameters, so it was easy to select and fix all other parameters except *bp*. By these empirical results we fixed *rr* to 0.9, *mpr* to 0.01 and we set *sbp* to 1.7**bp*. Here we present results for varying *bp* set to 500, 1000, 2000 and 3000.

All experiments were repeated 30 times, each starting from a random population with different seeds. As expected, each function is specific and behaves differently, but besides that, a general rule can be derived. We used the same random number seed for all four tests for the same function. Best and mean results with respective parameters are shown in Table 1.

As *bp* increases, exploitation power decreases in favor to exploration. This means that best results should get worse, while mean results should get better.

From the Table 1 it can be observed that for some functions best and mean results are equal to the function's optimum, for all tested values for *bp*. That means that these functions are easy for optimization by our algorithm. In this group of problems belong *g01, g03, g04, g06, g08, g11* and *g12*.

The most pronounced change in best and mean results can be seen in *g02* problem. Best is the same for *bp*=1,000 and *bp*=2,000, but it gets somewhat worse for *bp*=3,000. Means improve significantly from *bp*=1,000 to *bp*=3,000. For *bp*=500, both mean and best results are significantly worse than in all other tests. This is because within 500 cycles the algorithm cannot converge to the proper part of the search space.

In *g09* problem, known optimum is discovered no sooner than for *bp*=2,000. Thus, in this case, the algorithm cannot converge to the right part of the search space in smaller number of cycles. Mean values behave as expected.

**Table 1.** Best and mean results for GI-ABC with varying *bp*

| Problem | Optimum | | ***bp*=500** | ***bp*=1,000** | ***bp*=2,000** | ***bp*=3,000** |
|---|---|---|---|---|---|---|
| *g01* | | Best | -15.000 | -15.000 | -15.000 | -15.000 |
| | -15.000 | Mean | -15.000 | -15.000 | -15.000 | -15.000 |
| *g02* | | Best | -0.7946601 | -0.803618 | **-0.803618** | -0.803614 |
| | -0.803619 | Mean | -0.7202719 | -0.765693 | -0.798054 | **-0.800151** |
| *g03* | | Best | -1.000 | -1.000 | -1.000 | -1.000 |
| | -1.000 | Mean | -1.000 | -1.000 | -1.000 | -1.000 |
| *g04* | | Best | -30665.539 | -30665.539 | -30665.539 | -30665.539 |
| | -30665.539 | Mean | -30665.539 | -30665.539 | -30665.539 | -30665.539 |
| *g05* | | Best | 5126.497 | 5126.497 | 5126.497 | 5126.497 |
| | 5126.498 | Mean | 5258.619 | 5221.603 | 5209.028 | **5164.762** |
| *g06* | | Best | -6961.814 | -6961.814 | -6961.814 | -6961.814 |
| | -6961.814 | Mean | -6961.814 | -6961.814 | -6961.814 | -6961.814 |
| *g07* | | Best | 24.306 | 24.306 | 24.306 | 24.306 |
| | 24.306 | Mean | 24.513 | 24.481 | 24.421 | **24.395** |
| *g08* | | Best | -0.095825 | -0.095825 | -0.095825 | -0.095825 |
| | -0.095825 | Mean | -0.095825 | -0.095825 | -0.095825 | -0.095825 |
| *g09* | | Best | 680.631 | 680.631 | 680.630 | 680.630 |
| | 680.63 | Mean | 680.648 | 680.646 | 680.641 | **680.635** |
| *g10* | | Best | 7049.282 | 7049.282 | **7049.282** | 7049.285 |
| | 7049.25 | Mean | 7383.949 | 7282.871 | 7250.293 | **7192.397** |
| *g11* | | Best | 0.750 | 0.750 | 0.750 | 0.750 |
| | 0.75 | Mean | 0.750 | 0.750 | 0.750 | 0.750 |
| *g12* | | Best | -1.000 | -1.000 | -1.000 | -1.000 |
| | -1.000 | Mean | -1.000 | -1.000 | -1.000 | -1.000 |
| *g13* | | Best | 0.055 | 0.054 | 0.054 | 0.054 |
| | 0.053950 | Mean | 0.335 | 0.305 | 0.279 | **0.248** |

For *bp*=500, 1,000 and 2,000 in the *g10* problem, the algorithm obtains best which is very close to optimum. Unfortunately, with *bp* set too high (3,000), best gets worse. As anticipated, means get significantly better from *bp*=500 to *bp*=3,000. We conclude that in *g10* problem optimization, the algorithm early hits the proper part of the search space.

In *g13* test, the algorithm converges to the optimum solution in all tests except for *bp*=500. It seems that 500 cycles is not enough for matching the part of the search space where optimum solution resides.

GI-ABC achieves best results for *bp*=1,000 and for *bp*=2,000. However, the best balance between best and mean results (exploration-exploitation trade-off) is achieved in benchmarks for *bp*=3,000. In this case, with little sacrifice in bests, better means are obtained (Table 1, better solutions are bold). Because of this reasonable bests-means trade-off, we decided to select *bp*=3,000 and fix it in the algorithm. For this set of benchmark functions this looks like the most reasonable solution, however it can be a user adjustable parameter. Also, for more comprehensive set of benchmark functions it may be appropriate to fix *bp* parameter in the algorithm not as a constant, but function of the problem dimensionality.

## Comparison with the latest ABC and Other State-of-the-art Algorithms

As mentioned earlier, direct comparison is made between our modified GI-ABC and the latest Karaboga and Akay's [14] ABC algorithm, but for more comprehensive comparison, we added three other state-of-the-art algorithms: smart flight ABC (SF-ABC) by Mezura-Montes et al. [17], self-adaptive penalty function GA (SAPF-GA) by Tassema and Yen [18], and adaptive tradeoff model evolutionary strategy (ATMES) by Wang et al. [19]. Side by side best and mean results comparison is given in Table 2.In Tables 1 and 2 optimal values for easier comparison are from [14] (standard definitions are in [21]).

Based on experimental results shown in Table 2 we can see that considering best results, GI-ABC outperforms ABC algorithm for *g02, g07, g09, g10* and *g13* benchmark functions. The best result for *g05* reported in [14] is better than known optimum as a consequence of larger equality constraint violation. For *g01, g03, g04, g06, g08, g11* and *g12* problems, GI-ABC obtained optimal results, as well as ABC did. For all test functions, GI-ABC outperformed latest Karaboga and Akay's ABC algorithm [14] when mean results are considered.In comparison with other three algorithms, for both, best and mean results, only SF-ABC [17] and ATMES [19] reach better mean values for some functions. With respect to SF-ABC, GI-ABC showed substantially improved best results in *g02* and *g07* problems and maintained similar competitive results in other test functions. The largest result difference is observed in *g02* problem, where SF-ABC was affected by premature convergence. SF-ABC obtained better means than GI-ABC for *g05* and *g10* problems, where SF-ABC algorithm's directed search performs more efficiently than GI-ABC algorithm's guided onlooker. On the other side, GI-ABC is better in *g01, g02, g07, g09* and *g13* benchmarks. Both algorithms have means which are equal to optimums for easy problems *g03, g04, g06, g08, g11* and *g12*.

GI-ABC outperforms SAPF-GA [18] in all test functions if we compare both best and mean results.

GI-ABC obtains better both best and mean values than ATMES [19] in *g02* and *g10* problems. ATMES slightly outperforms GI-ABC only in *g05*, *g07*, *g09* and *g13* mean values. In *g07* test, ATMES obtains mean value which is very close to optimum, while mean in *g13* test is equivalent to optimum. For other benchmarks, these algorithms behave similarly.

It should be noted that the latest Karaboga and Akay's ABC algorithm [14] was favorable compared with other 9 state-of-the-art algorithms: Koziel and Michalewicz's homomorphous mappings (HM) [22], Runarsson and Yao's stochastic ranking (SR) [23], improved stochastic ranking (ISR) [24] and over-penalized approach (OPA) [24], adaptive segregational constraint handling evolutionary algorithm (ASCHEA) by Hamida

**Table 2.** Comparison of best and mean results for our GI-ABC, Karaboga and Akay's latest ABC and three other algorithms (SF-ABC, SAPF-GA and ATMES)

| **Problem** | **Optimum** | Best<br>Mean | **SF-ABC [17]** | **SAPF-GA [18]** | **ATMES [19]** | **ABC [14]** | **GI-ABC** |
|---|---|---|---|---|---|---|---|
| *g01* | | Best | -15.000 | -15.000 | -15.000 | -15.000 | -15.000 |
| | -15.000 | Mean | -14.13 | -14.552 | -15.000 | -15.000 | -15.000 |
| *g02* | | Best | -0.709034 | -0.803202 | -0.803388 | -0.803598 | **-0.803614** |
| | -0.803619 | Mean | -0.471210 | -0.755798 | -0.790148 | -0.792412 | **-0.800151** |
| *g03* | | Best | -1.000 | -1.000 | -1.000 | -1.000 | -1.000 |
| | -1.000 | Mean | -1.000 | -0.964 | -1.000 | -1.000 | -1.000 |
| *g04* | | Best | -30665.539 | -30665.401 | -30665.539 | -30665.539 | -30665.539 |
| | -30665.539 | Mean | -30665.539 | -30665.922 | -30665.539 | -30665.539 | -30665.539 |
| *g05* | | Best | 5126.497 | 5126.907 | 5126.498 | 5126.484* | 5126.497 |
| | 5126.498 | Mean | 5126.526 | 5214.232 | 5127.648 | 5185.714 | **5164.762** |
| *g06* | | Best | -6961.814 | -6961.046 | -6961.814 | -6961.814 | -6961.814 |
| | -6961.814 | Mean | -6961.814 | -6953.061 | -6961.814 | -6961.814 | -6961.814 |
| *g07* | | Best | 24.316 | 24.838 | 24.306 | 24.330 | **24.306** |
| | 24.306 | Mean | 24.657 | 27.328 | 24.316 | 24.473 | **24.395** |
| *g08* | | Best | -0.095825 | -0.095825 | -0.095825 | -0.095825 | -0.095825 |
| | -0.095825 | Mean | -0.095825 | -0.095635 | -0.095825 | -0.095825 | -0.095825 |
| *g09* | | Best | 680.630 | 680.773 | 680.630 | 680.634 | **680.630** |
| | 680.63 | Mean | 680.643 | 681.246 | 680.633 | 680.640 | **680.635** |
| *g10* | | Best | 7049.547 | 7069.981 | 7052.253 | 7053.904 | **7049.285** |
| | 7049.25 | Mean | 7116.934 | 7238.964 | 7250.437 | 7224.407 | **7192.397** |
| *g11* | | Best | 0.750 | 0.750 | 0.750 | 0.750 | 0.750 |
| | 0.75 | Mean | 0.750 | 0.751 | 0.750 | 0.750 | 0.750 |
| *g12* | | Best | -1.000 | -1.000 | -1.000 | -1.000 | -1.000 |
| | -1.000 | Mean | -1.000 | -1.000 | -1.000 | -1.000 | -1.000 |
| *g13* | | Best | 0.054 | 0.054 | 0.054 | 0.760 | **0.054** |
| | 0.053950 | Mean | 0.263 | 0.286 | 0.054 | 0.968 | **0.248** |

* Explained in the text

and Schoenaeur [25], genetic algorithm (GA) from [26], simple multimembered evolutional strategy (SMES) by Mezura-Montes and Coello Coello [26], particle swarm optimization (PSO) from [27], and differential evolution (DE). Most of the mentioned algorithms use larger number of function evaluations (350,000) and Karaboga and Akay concluded in [14] that ABC algorithm can efficiently be used for solving constrained optimization problems and that it is very competitive to the other state-of-the-art approaches. This conclusion holds even more for our proposed GI-ABC algorithm.

## 5. Conclusions

A modification to the ABC algorithm, named GI-ABC (genetically inspired artificial bee colony), is introduced in this paper. GI-ABC improves the performance of the ABC algorithm by applying uniform crossover and mutation operators from genetic algorithms.

By studying ABC algorithm we found that the number of scouts is too large for the late phase of algorithm's execution. Thus, we set a point where a large percentage of scouts are transformed into guided onlookers which are more exploitation intensive than ordinary onlookers. From time to time, scout is still triggered for preventing the algorithm from being stuck in local optimum.

GI-ABC was tested on 13 standard well-known benchmark optimization functions. From the comparative study GI-ABC has shown its potential to handle various constrained optimization problems. Our proposed algorithm was compared directly with the latest ABC algorithm by Karaboga and Akay [14] and three other state-of-the-art algorithms where it showed improved performance considering best results and even more significant robustness by outperforming ABC algorithm's mean results in all cases. Since Karaboga and Akay's in [14] favorably compare ABC with 9 other state-of-the-art algorithms, we conclude that our proposed GI-ABC algorithm is a very promising tool for constrained optimization.

## Acknowledgement

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

## REFERENCES

1. MEZURA-MONTES, E., **Constraint-Handling in Evolutionary Optimization**, Studies in Computational Intelligence, vol. 198, Springer-Verlag, 2009, p. 264.
2. MEZURA-MONTES, E., COELLO COELLO, A. C., **Constraint-Handling in Nature-Inspired Numerical Optimization: Past, Present and Future**, Swarm and Evolutionary Computation, vol. 1(4) 2011, pp. 173-194.
3. TIWARI, M. K., S. KUMAR, S. PRAKASH, S**olving Part-type Selection and Operation Allocation Problems in an FMS: An Approach using Constraints-based fast Simulated Annealing Algorithm**, IEEE Trans. on Systems, Man and Cybernetics Part A - Systems and Humans, vol. 36(6), 2006, pp. 1170-1184.
4. NICOARA, E. S., F. G. FILIP, N. PARASCHIV, **Simulation-based Optimization Using Genetic Algorithms for Multi-objective Flexible JSSP,** Studies in Informatics and Control, vol. 20(4), 2011, pp. 333-344.
5. GZARA, M., A.ESSABRI, **Balanced Explore-Exploit Clustering based Distributed Evolutionary Algorithm for Multi-objective Optimization**, Studies in Informatics and Control, vol. 20(2), 2011, pp. 97-106.
6. ZAMFIRESCU, C. B., F. G. FILIP, **Swarming Models for Facilitating Collaborative Decisions,** International Journal of Computers, Communications & Control, vol. V(1), 2011, pp. 125-137.
7. AGRAWAL, S., Y. DASHORA, M. K. TIWARI, **Interactive Particle Swarm: A Pareto-Adaptive Metaheuristic to Multiobjective Optimization**, IEEE Trans. on Systems, Man and Cybernetics Part A - Systems and Humans vol. 38(2), 2008, pp. 258-277.
8. JOVANOVIC, R., M. TUBA, **An Ant Colony Optimization Algorithm with Improved Pheromone Correction Strategy for the Minimum Weight Vertex Cover Problem**, Applied Soft Computing, vol. 11(8), 2011, pp. 5360-5366.
9. SECUI, C. D., S. DZITAC, G. V. BANDEA, **An ACO Algorithm for Optimal**

**Capacitor Banks Placement in Power Distribution Networks**, Studies in Informatics and Control, vol. 18(4), 2009, pp. 305-314.

10. KARABOGA, D., **An Idea Based on Honey Bee Swarm for Numerical Optimization**, Technical Report TR06, Computer Engineering, Department, Erciyes University, Turkey, 2005.

11. KARABOGA, D., B. BASTURK, **A Powerful and Efficient Algorithm for Numerical Function Optimization: Artificial Bee Colony (ABC) Algorithm**. Journal of Global Optimization, vol. 39(3), 2007, pp. 459-471.

12. KARABOGA, D., B. BASTUK, **Artificial Bee Colony (ABC) Optimization Algorithm for Solving Constrained Optimization Problems**, Proc. IFSA 2007, LNAI 4529, 2007, pp.789-798.

13. KARABOGA, D., BASTURK, B., **On the Performance of Artificial Bee Colony (ABC) Algorithm**, Applied Soft Computing, vol. 8 (1), 2007, pp. 687-697.

14. KARABOGA, D., B. AKAY, **A Modified ABC for Constrained Optimization Problems**, Applied soft computing, vol. 11(3), 2011, pp. 3021-3031.

15. BRAJEVIC, I., M. TUBA, **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, p. 12.

16. ZHU, G., S. KWONG,, **Gbest-guided Artificial Bee Colony Algorithm for Numerical Function Optimization**, Applied mathematics and computation, vol. 217(7), 2010, pp. 3166-3173.

17. MEZURA-MONTES, E., M. DAMIAN-ARAOZ, O. CETINA-DOMINGEZ, **Smart Flight and Dynamic Tolerances in the Artificial Bee Colony for Constrained Optimization**, Proc. IEEE Congress on Evolutionary Computation (CEC'2010), 2010, pp. 1-8.

18. TESSEMA, B., G. G. YEN, **A Self-adaptive Penalty Function based Algorithm for Constrained Optimization**, IEEE Cong. on Evolutionary Computation 2006 (CEC'2006), 2006, pp. 950-957.

19. WANG, Y., Z. CAI, Y. ZHOU, W. ZENG, **An Adaptive Tradeoff Model for Constrained Evolutionary Optimization**, IEEE Trans. on Evolutionary Computation, vol. 12(1), 2008, pp. 80-92.

20. DEB, K., **An Efficient Constraint-handling Method for Genetic Algorithms**, Computer Methods in Applied Mechanics and Engineering, vol. 186(2-4), 2000, pp. 311-338.

21. LIANG, J. J., T. P. RUNARSSON, E. MEZURA-MONTES, M. CLERK, P. N. SUGANTHAN, A. C.COELLO COELLO, K. DEB, **Problem Definitions and Evaluation Criteria for the CEC 2006 Special Session on Constrained Real-parameter Optimization,** Technical Report, 2006.

22. KOZIEL, S., Z. MICHALEWICZ, **Evolutionary Algorithms, Homomorphous Mappings, and Constrained Parameter Optimization**, Evolutionary Computation, vol. 7, issue 1, 1999, pp. 19-44.

23. RUNARSSON, T. P., X. YAO, **Stochastic Ranking for Constrained Evolutionary Optimization,** IEEE Trans. on Evolutionary Computation**, v**ol. 4(3), 2000, pp. 284-294.

24. RUNARSSON, T. P., X. YAO, **Search Biases in Constrained Evolutionary Optimization,** IEEE Trans. on Systems, Man and Cybernetics, vol. 35(2), 2005, pp.233-243.

25. HAMIDA, S. B., M. SCHOENAUER, **ASCHEA: New Results using Adaptive Segregational Constraint Handling**, Proc. of the 2002 Congress on Evolutionary Computation, 2002, pp. 884-889.

26. MEZURA-MONTES, E., A. C. COELLO COELLO, **A Simple Multimembered Evolution Strategy to Solve Constrained Optimization Problems**, IEEE Trans. on Evolutionary Computation, vol. 9(1), 2005, pp. 1-17.

27. ZAVALA, A. E. M., A. H. AGUIRRE, E. R. V. DIHARCE, **Constrained Optimization via Particle Evolutionary Swarm Algorithm (PESO)**, Proc. of the Conf. on Genetic and Evolutionary Computation GECCO 2005, 2005pp. 209-216.