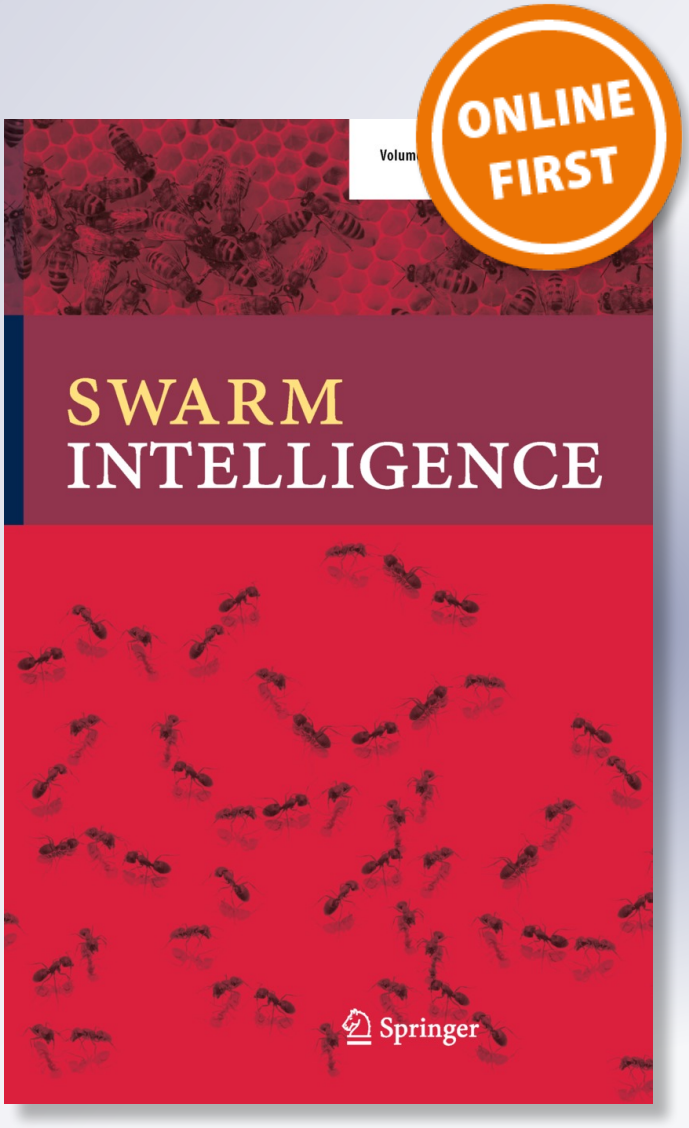
Artificial bee colonies for continuous optimization: Experimental analysis and improvements

Tianjun Liao, Doğan Aydın & Thomas Stützle

Swarm Intelligence

ISSN 1935-3812

Swarm Intell
DOI 10.1007/s11721-013-0088-5



Springer

Swarm Intell
DOI 10.1007/s11721-013-0088-5

Artificial bee colonies for continuous optimization: Experimental analysis and improvements

Tianjun Liao · Doğan Aydın · Thomas Stützle

Received: 28 September 2012 / Accepted: 13 September 2013

Abstract The artificial bee colony (ABC) algorithm is a recent class of swarm intelligence algorithms that is loosely inspired by the foraging behavior of honeybee swarms. It was introduced in 2005 using continuous optimization problems as an example application. Similar to what has happened with other swarm intelligence techniques, after the initial proposal, several researchers have studied variants of the original algorithm. Unfortunately, often these variants have been tested under different experimental conditions and different fine-tuning efforts for the algorithm parameters. In this article, we review various variants of the original ABC algorithm and experimentally study nine ABC algorithms under two settings: either using the original parameter settings as proposed by the authors, or using an automatic algorithm configuration tool using a same tuning effort for each algorithm. We also study the effect of adding local search to the ABC algorithms. Our experimental results show that local search can improve considerably the performance of several ABC variants and that it reduces strongly the performance differences between the studied ABC variants. We also show that the best ABC variants are competitive with recent state-of-the-art algorithms on the benchmark set we used, which establishes ABC algorithms as serious competitors in continuous optimization.

Keywords Artificial bee colony · Continuous optimization · Experimental study · Automatic algorithm configuration

Electronic supplementary material The online version of this article (doi:10.1007/s11721-013-0088-5) contains supplementary material, which is available to authorized users.

T. Liao · T. Stützle (✉)
IRIDIA, CoDE, Université Libre de Bruxelles (ULB), Brussels, Belgium
e-mail: stuetzle@ulb.ac.be

T. Liao
e-mail: tliao@ulb.ac.be

D. Aydın
Computer Engineering Dept., Dumlupinar University, 43000 Kütahya, Turkey
e-mail: dogan.aydin@dpu.edu.tr

Published online: 08 October 2013

1 Introduction

The artificial bee colony (ABC) algorithm is a recent representative of a number of swarm intelligence algorithms that are inspired by some type of behavior observed in real bee colonies (Karaboga and Akay 2009; Diwold et al. 2011b). It was introduced by Karaboga (2005), who applied it to continuous optimization problems. In the ABC algorithm, there are three types of (artificial) bees, namely employed bees, onlooker bees, and scout bees. Each employed bee is associated to a different solution of the optimization problem to be solved.[1] At each algorithm iteration, an employed bee explores the neighborhood of the solution to which it is associated. Onlooker bees also explore the neighborhood of solutions; however, differently from employed bees, they are not associated to one fixed solution, but they choose the solution they explore in each algorithm iteration probabilistically as a function of a solution quality. If the neighborhood of a solution has been explored unsuccessfully for a given number of times, a new solution in the search space is chosen uniformly at random by a scout bee, adding an exploration feature to the algorithm. The ABC algorithm can be seen as a population-based local search algorithm, where at each iteration the neighborhood of each of the solutions in the population is explored.

The original ABC algorithm obtained encouraging results on some standard benchmark problems, but, being an initial proposal, still a considerable performance gap with respect to state-of-the-art algorithms was observed. In particular, it was found to have relatively poor performance on composite and nonseparable functions and to have a slow convergence rate toward high-quality solutions (Akay and Karaboga 2012). Therefore, it is not surprising that in the years following its introduction, several modifications to the original ABC algorithm were proposed, with the goal of improving its performance.

In this article, we first review several variants of the original ABC algorithm. We have reimplemented these ABC variants under a same framework to allow their experimental study under identical experimental conditions. For the evaluation of the ABC algorithms, we have chosen a recent benchmark set of 19 continuous optimization problems whose dimension can be freely scaled; this benchmark set was proposed for the evaluation of evolutionary algorithms and metaheuristics on large-scale continuous optimization problems (Lozano et al. 2011). We evaluate the ABC algorithms under three experimental conditions. First, we apply each algorithm using default parameter settings taken from the literature. Second, we tune the parameters of each ABC variant with the automatic algorithm configuration tool *iterated F-race* (Balaprakash et al. 2007; Birattari et al. 2010; López-Ibáñez et al. 2011). The rationale for this step is to eliminate a possible bias due to an uneven tuning effort for the algorithm variants in the original papers. Third, we consider the integration of local search algorithms into the ABC variants to improve their performance. To do so, we again use an automatic algorithm configuration tool to avoid a manual redesign of these hybrid algorithms and to ensure a same tuning effort. Our experimental results show that the tuned ABC variants and the ABC variants that integrate local search perform often significantly better than the original ABC variants. In particular, the usage of local search diminishes the differences among the ABC variants. As a result, after the automatic tuning and the integration of the local search algorithms, also the ranking of the ABC variants changes.

[1] Note that, in analogy to the natural inspiration, in the original literature a food source corresponds to a solution. Here, we present the ABC algorithms using an optimization-oriented nomenclature rather than the original bee-inspired one. The analogy to real bee colonies is discussed in Karaboga (2005), Karaboga and Akay (2009) and Diwold et al. (2011b).

The article is structured as follows. In Sect. 2, we introduce the original ABC algorithm, the variants we study, and we give an overview of other ABC variants. Section 3 describes the experimental setup, while the experimental results are discussed in Sect. 4. In Sect. 5, we draw our conclusions. In the online supplementary material, we give details on the benchmark functions we used and additional tables with experimental results.

2 The artificial bee colony algorithm

2.1 The original ABC algorithm

In the ABC algorithm, a set of *SN* solutions is maintained, and at each iteration *SN* employed and *SN* onlooker bees explore the neighborhood of these solutions.

A high-level outline of the ABC algorithm is given in Algorithm 1. After the algorithm initialization, where *SN* initial solutions are generated, the ABC algorithm loops through three steps, which are the search steps done by the employed bees, the onlooker bees, and the scout bees in this order. Here, we apply ABC algorithms to continuous optimization problems, in which we are given a D-dimensional objective function $f : X \subseteq \mathbb{R}^D \to \mathbb{R}$ that is to be minimized, where X is the search space, and D is the dimension of the search space. We assume here box-constrained problems where for each solution vector $x_i = (x_{i1} x_{i2} \cdots x_{iD})$, we have that $x_{ij} \in [x_j^{\min}, x_j^{\max}]$, where $[x_j^{\min}, x_j^{\max}]$ is the interval of feasible values in dimension $j, 1 \leq j \leq D$. If nothing else is said in the following, we handle violations of these boundary constraints by setting a variable to the closest value on the bounds; that is, if $x_{ij} < x_j^{\min}$, then we set $x_{ij} = x_j^{\min}$, and if $x_{ij} > x_j^{\max}$, then we set $x_{ij} = x_j^{\max}$. Next, we describe the main steps of the ABC algorithm.

Initialization In the algorithm initialization, *SN* solutions are chosen uniformly at random in the search space. For each solution $x_i, 1 \leq i \leq SN$, this is done by generating the solution's coordinate as follows:

$$x_{ij} = x_j^{\min} + \varphi_{ij}\left(x_j^{\max} - x_j^{\min}\right), \tag{1}$$

where φ_{ij} is a number generated uniformly at random in $[0, 1]$. To each solution is associated an employed bee. This generation of values is done for each dimension $j \in \{1, \ldots, D\}$ independently.

Employed bees behavior In the employed bees step, each employed bee generates a new solution x_i' in a neighborhood of the solution x_i to which it is associated. This is done by modifying randomly one coordinate of x_i as follows. First, another solution x_k, $k \neq i$, $k \in \{1, \ldots, SN\}$, is chosen uniformly at random; then, x_{ij}' is set to

$$x_{ij}' = x_{ij} + \phi_{ij}(x_{ij} - x_{kj}), \quad i \neq k, \tag{2}$$

Algorithm 1 The pseudo-code of the artificial bee colony algorithm

```
initialization
while termination condition is not met do
   Employed bees step
   Onlookers bees step
   Scout bees step
end while
```

where the dimension j, $j \in \{1, \ldots, D\}$, is selected uniformly at random, and ϕ_{ij} is a random number chosen uniformly at random in $[-1, 1]$. If the new candidate solution x_i' is better than x_i, then x_i' replaces x_i.

Onlooker bees behavior Onlooker bees probabilistically select one of the *SN* solutions and then search the neighborhood of the chosen solution in the same way as employed bees do. Onlooker bees select a solution x_i with probability given by

$$p_i = \frac{fitness_i}{\sum_{n=1}^{SN} fitness_n}, \tag{3}$$

where $fitness_i$ is an evaluation function value assigned to x_i, which for minimization problems is defined as

$$fitness_i = \begin{cases} \frac{1}{1+f(x_i)}, & f(x_i) \geqslant 0, \\ 1 + |f(x_i)|, & f(x_i) < 0. \end{cases} \tag{4}$$

Since better quality solutions have a higher probability of being chosen, the onlooker bees prefer to examine the neighborhood of better quality solutions.

Scout bees behavior If a solution could not be improved by employed and onlooker bees for a given number of *limit* times, the solution reaches its visiting limit, and it is abandoned. In this case, a new solution is generated uniformly at random in the search space, replacing the abandoned solution. This so-called scout bee step increases the algorithm's exploration capabilities. In this paper, the value of *limit* is determined by the formula $lf \cdot SN \cdot D$, where *lf* is a real-valued parameter.

2.2 Variants of the artificial bee colony algorithm

In this paper, we examine experimentally the original ABC algorithm and eight variants of it, which have been proposed with the goal of improving its performance. These eight ABC variants are representative for the types of modifications that have been suggested in various journal papers or for which promising experimental results have been reported. In what follows, we shortly describe the main features of these variants.

Modified artificial bee colony (MABC) algorithm Akay and Karaboga (2012) proposed the modified ABC algorithm (MABC) to overcome the supposedly slow convergence speed of the original ABC algorithm. According to Akay and Karaboga (2012), the culprit of the slow convergence of the original ABC algorithm is the change of only one variable by an employed or onlooker bee. As an alternative, MABC modifies a larger number of variables at each employed and onlooker bee step. MABC introduces a parameter *MR* (modification rate) that gives the probability with which each variable x_{ij} of solution x_i is modified. This is implemented by the equation

$$x_{ij}' = \begin{cases} x_{ij} + \phi_{ij}(x_{ij} - x_{kj}) & \text{if } r_{ij} < MR, \\ x_{ij} & \text{otherwise}, \end{cases} \tag{5}$$

where r_{ij} is a random number that is drawn uniformly at random in $[0, 1]$. The value of *MR* has a strong impact on the search behavior: low values of *MR* favor exploitation, while large values of *MR* favor exploration.

A second modification concerns the magnitude ϕ_{ij} by which a variable is changed. In MABC, ϕ_{ij} is a random number drawn uniformly at random in the interval $[-SF, SF]$, where SF is a parameter called scaling factor. The original Eq. (2) of ABC is obtained by setting SF to one. Instead of using a fixed value for SF, Akay and Karaboga (2012) propose to adapt the value of SF at run-time using Rechenberg's 1/5 mutation rule (Rechenberg 1973). To do so, for every fixed number of iterations, the ratio of solution modifications that led to an improvement is computed. Depending on whether this ratio is smaller than, larger than, or equal to one fifth, the value of SF is multiplied by 0.85, divided by 0.85, or it remains the same.

Gbest-guided artificial bee colony (GbABC) algorithm The main idea of GbABC (Zhu and Kwong 2010; Diwold et al. 2011a) is to bias the modification of the reference solution by the best solution found so far. This solution x_{gbest} is called global-best in GbABC, and we adopt this naming in the following. Solution x_{gbest} is exploited by modifying Eq. (2) to

$$x'_{ij} = x_{ij} + \phi_{ij}(x_{ij} - x_{kj}) + \psi_{ij}(x_{\text{gbest}\,j} - x_{ij}), \quad i \neq k, \tag{6}$$

where $x_{\text{gbest}\,j}$ is the jth coordinate of the global-best solution, and ψ_{ij} is a random number drawn uniformly at random in $[0, C]$, where C is a constant that is set to one by Diwold et al. (2011a) or used as a parameter to be set by Zhu and Kwong (2010). This modification is inspired by the usage of the global-best solution to influence particles in particle swarm optimization; it is a straightforward modification that, as we will see later, results in significantly improved performance.

GbestDist-guided artificial bee colony (GbdABC) algorithm Diwold et al. (2011a) have proposed also a further variation of GbABC, where they change the selection of the solution x_k in Eq. (6). Let $d(x_i, x_k)$ be the Euclidean distance between two solutions x_i and x_k. In GbdABC, a solution x_k, $k \neq i$, is chosen with probability

$$p_k = \frac{\frac{1}{d(x_i, x_k)}}{\sum_{l=1, l\neq i}^{SN} \frac{1}{d(x_i, x_l)}}. \tag{7}$$

Thus, the closer a solution is to x_i, the higher is the probability of choosing it. The intuition behind this choice is that it is more probable to find a better solution by searching between two good solutions that are close to each other in the solution space (Diwold et al. 2011a).

Best-so-far selection artificial bee colony (BsfABC) algorithm As GbABC and GbdABC, BsfABC (Banharnsakun et al. 2011) exploits the best solution found so far (called best-so-far solution in Bsf-ABC) but uses it in a different way. The best solution is used to modify the onlooker bees step, thus leaving the employed bee step unchanged. The update is given by

$$x'_{id} = x_{ij} + \mathit{fitness}_{\text{gbest}}\big(\phi_{ij}(x_{ij} - x_{\text{gbest}\,j})\big), \quad i \neq k,\ d = 1, \ldots, D, \tag{8}$$

where j is a randomly selected dimension, and $\mathit{fitness}_{\text{gbest}}$ is the fitness value of the best solution found so far (see also Eq. (4)). BsfABC applies this position update not only in the randomly selected dimension j, but in all dimensions. This has the effect that the variable values of all dimensions of a candidate solution get closer to those of the best-so-far one.

BsfABC also modifies the scout bees step. Instead of choosing a random new solution, BsfABC randomly perturbs the current solution by using the equation

$$x'_{ij} = x_{ij} + x_{ij}\phi_{ij}\left(w_{\max} - \frac{itr}{itr_{\max}}(w_{\max} - w_{\min})\right), \quad j = 1, \ldots, D, \tag{9}$$

where $w_{\max}$ and $w_{\min}$ are control parameters that define the strength of the perturbation, itr is the number of algorithm iterations done so far, and $itr_{\max}$ is the maximum number of iterations. The effect of this equation is a decrease of the strength of the perturbation with an increasing number of iterations, making the algorithm more exploitative in later algorithm iterations.

Chaotic artificial bee colony (CABC) algorithm Alataş (2010) introduces two modifications to the original ABC algorithm. The first is to generate the initial solutions by using a chaotic map instead of a standard random number generator; this results in variant CABC_1. Alataş made tests with seven different chaotic maps. (In the later parameter tuning step, we consider the same seven proposed chaotic maps as parameters during the tuning process.) The second modification is to generate the solution of a scout bee by using a form of local search, resulting in variant CABC_2: if a solution cannot be improved for $limit/2$ trials, the algorithm searches for another $limit/2$ trials around the current solution by modifying one dimension and accepting a new solution if it improves over the current one. Finally, a third variant, CABC_3, which we use in this paper, combines the two modifications proposed for CABC_1 and CABC_2.

Improved artificial bee colony (ImpABC) algorithm ImpABC (Gao and Liu 2011) introduces three modifications to ABC. The first initializes the population using a chaotic random generator based on a logistic map. Once SN solutions are generated, for each solution, the value of each variable is mirrored at the center of the search range for this variable, resulting in SN new solutions; this process is called opposition-based population initialization. From the resulting $2 \cdot SN$ solutions the best SN solutions are kept.

The other two modifications concern the way how solutions are modified. Similar to MABC, Gao and Liu (2011) modify more than one variable. They introduce a parameter M, which gives the number of variables to be modified and explore values of M from 1 to D. They observe that the best value of M depends on the particular problem, but do not give a final recommendation on the setting of the parameter M. Therefore, as default, we adopt a setting analogous to the parameter MR of MABC, where MR is the probability that a variable is modified. This implies that for the same value of MR, the number of variables modified increases with D. The third modification is inspired by the search mechanisms used in differential evolution (DE) algorithms (Stern and Price 1997). ImpABC uses two new equations, called $ABC/best/1$ and $ABC/rand/1$, which are inspired by the $DE/best/1$ and $DE/rand/1$ scheme, respectively. The equation for $ABC/best/1$ is

$$x'_{ij} = x_{\text{gbest}\,j} + \phi_{ij}(x_{ij} - x_{r1\,j}), \tag{10}$$

and the search equation for $ABC/rand/1$ is

$$x'_{ij} = x_{r1\,j} + \phi_{ij}(x_{ij} - x_{r2\,j}), \tag{11}$$

where $r1$ and $r2$ are two random indices of solutions different from i, $x_{\text{gbest}\,j}$ is the value of the variable j of the best solution found so far, and j is a randomly chosen variable.

Equation (10) biases the search toward the global-best solution while Eq. (11) is more explorative. To balance the effects of the two equations, the authors suggest to apply Eq. (11) with probability p and Eq. (10) with probability $1 - p$. Based on some experiments, they propose to set p to 0.25.

In a more recent paper, Gao et al. (2012) propose essentially the same ideas but use yet another search equation, where variable j of a chosen solution x_i is updated similarly to what is done in Eq. (10) but considering two or four random solutions. Since they report that using four random solutions does not lead to better performance, we do not consider this variant here.

Rosenbrock artificial bee colony (RABC) algorithm RABC (Kang et al. 2011) proposes two main modifications to ABC. The first replaces Eq. (4) with the rank-based fitness adaptation method defined as

$$fitness_i = 2 - SP + \frac{2(SP-1)(r_i - 1)}{SN - 1}, \tag{12}$$

where $SP \in [1.0, 2.0]$ is a parameter called selection pressure, and r_i is the rank of solution x_i in the population.

The second modification is the integration of Rosenbrock's rotational direction method (RM) (Rosenbrock 1960), a local search technique, into ABC. RM is applied every n_c iterations to the global-best solution. For integrating RM into ABC, some adaptations have been done to RM. The most noteworthy from an algorithmic point of view is the usage of an adaptive initial step size in RM. To do so, the authors propose to use the best m ranked solutions and to compute

$$\delta_j = 0.1 \cdot \frac{\sum_{i=1}^{m} (\bar{x}_{i,j} - x_{\text{gbest},j})}{m}, \tag{13}$$

where δ_j is the step size in dimension j, and $\bar{x}_i$ is the ith best solution after ranking. The solution returned by RM replaces the middle ranked solution in the population, which, according to the authors, improves performance when compared to replacing the worst solution on multimodal problems. For a description of the other modifications to RM, we refer to Kang et al. (2011).

Incremental artificial bee colony (IABC) with local search (IABCLS) Aydın et al. (2012) have proposed an algorithm that integrates the incremental social learning (ISL) framework (Montes de Oca 2011) and local search procedures in ABC. The basic idea of ISL when applied to population-based optimization algorithms is to start with a small population and to add new solutions each g iterations (population increment), biased by members of the population (learning aspect). When the population is increased, a new employed bee's solution is generated as

$$x'_{\text{new}\,j} = x_{\text{new}\,j} + \varphi_{ij}(x_{\text{gbest}\,j} - x_{\text{new}\,j}), \quad j = 1, \ldots, D, \tag{14}$$

where $x_{\text{new}\,j}$ is generated according to Eq. (1), $x'_{\text{new}\,j}$ is the j-th coordinate biased by the global-best solution, and φ_{ij} is a parameter. Apart from adapting ISL to ABC, IABC uses two other modifications. The first is to bias the solution modification to the global-best solution. This is implemented by replacing Eq. (2) with

$$x'_{ij} = x_{ij} + \phi_{ij}(x_{\text{gbest}\,j} - x_{ij}). \tag{15}$$

The second biases the scout bees toward the global-best solution. This is done by initializing the scout bee using

$$x'_{ij} = x_{\text{gbest}\,j} + R_{factor}(x_{\text{gbest}\,j} - x_{\text{new}\,j}), \quad j = 1, \ldots, D, \tag{16}$$

where x'_i is a new solution replacing an abandoned solution, R_{factor} is a parameter that determines the bias toward the global-best solution x_{gbest}, and $x_{\text{new}\,j}$ is a variable value generated according to Eq. (1). Aydın et al. (2012) also hybridized IABC with either Powell's conjugate direction set method (Powell 1964) or Lin-Yu Tseng's Mtsls1 local search procedure (Tseng and Chen 2008), resulting in the IABCLS algorithm. This was done by invoking a local search procedure in every algorithm iteration starting from the global-best solution.

Other variants of the artificial bee colony algorithm Since the initial proposal of ABC, a large number of ABC variants for numerical optimization have been proposed. One common theme has been the hybridization of ABC algorithms with procedures taken from other techniques. Fister et al. (2012) proposed the memetic ABC algorithm, which is hybridized with two local search heuristics, the Nelder–Mead algorithm (NMA) and the random walk with direction exploitation (RWDE). Additionally, this algorithm takes inspiration from DE in the update equation. Yan et al. (2011) and Ming et al. (2010) hybridized ABC with a genetic algorithm. El-Abd (2011a) and Sharma et al. (2011) investigated the hybridization of ABC and particle swarm optimization, which essentially is done by modifying the update equations of ABC. Another hybrid approach was proposed by Zhong et al. (2011); they introduce, inspired by the chemotaxis behavior of bacterial foraging, a stronger local search type behavior in the update of the employed and onlooker bees. El-Abd (2011b) explored the use of opposition-based learning in ABC. Several other variants have focused on modifying the search equation of the original ABC algorithm. Abraham et al. (2012) applied, as others have done, a type of DE strategy into the search equation of ABC. Zou et al. (2010) have proposed the cooperative ABC algorithm, which selects the best solution found so far as the reference solution and mutates it. As chaotic ABC, Wu and Sh (2011) use a chaotic random number generator; however, they use it in the search equation instead of the initialization or the scout step. Alam et al. (2010) introduced an ABC algorithm with exponentially distributed mutations. Sharma and Pant (2012) proposed two new mechanisms for the movements of scout bees. The first method is based on a nonlinear interpolated path, while in the second one, scout bees follow a Gaussian movement. Rajasekhar et al. (2011) proposed an improved version of the ABC algorithm with mutation based on Levy probability distributions. The global ABC algorithm, which was introduced by Guo et al. (2011), proposed three modified search equations by using global and individual best positions. To balance between exploration and exploitation, a diversity strategy was used by Lee and Cai (2011).

3 Experimental setup

We have reimplemented the original ABC algorithm and the eight variants we have presented in detail in Sect. 2.2. Thus, in our experimental analysis, we will compare a total of nine algorithms. For convenience, we repeat the abbreviation, the name, and the main reference to each of the nine algorithms in Table 1.

We compare these algorithms in three settings. The first setting uses the default parameter settings that were proposed in the original publications. In a few cases, not all the default

Table 1 Abbreviation, name, and main reference for the nine ABC algorithms compared in our study

ABC	Original ABC	Karaboga and Basturk (2007)
GbABC	Gbest-guided ABC	Zhu and Kwong (2010) Diwold et al. (2011a)
GbdABC	GbestDist-guided ABC	Diwold et al. (2011a)
BsfABC	Best-so-far selection ABC	Banharnsakun et al. (2011)
MABC	Modified ABC	Akay and Karaboga (2012)
ImpABC	Improved ABC	Gao and Liu (2011)
CABC	Chaotic ABC, version 3	Alataş (2010)
RABC	Rosenbrock ABC	Kang et al. (2011)
IABC	Incremental ABC	Aydın et al. (2012)

parameter settings have been specified unambiguously or several values have been tested without giving a final recommendation. In these cases, we have made an educated guess of a reasonable setting or used values that, from the reported experimental results, seemed to be appropriate.

The second setting considers the usage of an offline automatic algorithm configuration tool. By the usage of an offline tool for parameter tuning and a same tuning effort for all algorithms we intend (i) to make an unbiased comparison between the ABC algorithms, independent of the tuning effort spent by the original authors and (ii) to check whether and by how much the performance of ABC algorithms can be improved without changing any detail of their algorithmic structure.

In the third setting, we hybridize the ABC algorithms with local search procedures that are used to refine some of the generated solutions. This is done because there is strong evidence in the literature that many population-based metaheuristics for continuous optimization can improve their performance by exploiting local search algorithms. Examples include the MOS-DE algorithm, which was the best performing algorithm in a recent special issue for the SOCO benchmark functions (LaTorre et al. 2011), memetic algorithms (Molina et al. 2010), or hybrid ant colony optimization algorithms (Liao et al. 2011). Also, few ABC algorithms such as IABC and RABC have made use of additional local search procedures. For defining appropriate parameter settings, we retune the parameters of the resulting hybrid ABC algorithms with the algorithm configuration tool. This retuning is necessary as there may be an interaction between the algorithm parameters and the usage of a local search.

3.1 Benchmark set

All experiments were performed on the benchmark functions that were proposed for a special issue of the Soft Computing journal on the scalability of evolutionary algorithms and other metaheuristics for large-scale continuous optimization problems (Lozano et al. 2011). This benchmark set, to which we refer as SOCO, contains 19 benchmark functions comprising widely used functions such as Rastrigin, Griewank, or Ackley, as well as hybrid functions composed of two basic functions (Herrera et al. 2010). A detailed description of the benchmark function set is given in the online supplementary material in Table 1. All functions are freely scalable, and we conducted our experiments using functions of dimension $D \in \{10, 50, 100, 500\}$ to examine the scaling behavior of ABC algorithms. All algorithms were run 25 times on each SOCO function, each algorithm being run for a maximum of $5000 \cdot D$ function evaluations, as proposed in the original protocol of the SOCO benchmark set. Error values lower than 10^{-14} are approximated to 10^{-14}, where the error value is defined

as $f(x) - f(x^*)$ with x being a solution and x^* being an optimal solution. (10^{-14} is referred to as an optimum threshold.) For statistical analyses, we used the nonparametric Wilcoxon test at the significance level $\alpha = 0.05$ and Holm's multiple test corrections if more than two algorithms are compared.

3.2 Local search

As the local search algorithms to be hybridized with the ABC algorithms, we have considered Powell's conjugate direction set method (Powell 1964) and the Mtsls1 local search algorithm (Tseng and Chen 2008). The reason for selecting these two methods is that they are relatively simple to implement, they are high performing, and in our own previous research, they were found to be useful for obtaining effective hybrid algorithms (Liao et al. 2011; Aydın et al. 2012; Montes de Oca et al. 2011). For the hybridization of ABC algorithms with local search, we follow our own earlier experience and use the following main design decisions:

- The local search procedure is applied using the global-best solution as starting solution and replaces it if the local search finds a better solution. If $Failure_{\max}$ successive local search applications fail to improve the global-best solution, local search is applied to a randomly chosen solution different from the global-best one.
- There are two options for determining the step size parameter of the local search: adaptive or fixed step size. In an adaptive step size, the maximum norm of the difference vector between a randomly selected solution in the population and the global-best solution is used. For the second option, half the length of the search range is always selected as step size.
- The number of iterations and other parameters related to the local search procedures are determined by using an automatic algorithm configuration tool.

It is important to note that there is a trade-off between global and local search. If the effect of the local search is too strong, the impact of the ABC search mechanism may be considered insignificant after hybridization. To check for this possibility, we have compared the hybrid ABC algorithm to a random-restart local search algorithm (RLS), where the starting points for the local search are generated uniformly at random in the search space. For RLS, we also automatically tuned the local search parameters with a same tuning effort as spent for the hybrid ABC algorithms (see Table 4 for the tuned parameter settings).

3.3 Tuner setup and parameter settings

We used the iterated F-race procedure (Balaprakash et al. 2007; Birattari et al. 2010) implemented in the irace package (López-Ibáñez et al. 2011) as the offline automatic algorithm configuration tool. Iterated F-race iteratively applies F-race (Birattari et al. 2002) to a set of configurations that are generated using a probabilistic model. F-race is a racing method for selecting the best from a set of configurations. At each step of F-race, all surviving configurations are evaluated on one benchmark function. The obtained error values are then ranked, and the F-test checks after each step whether a significant difference arises between the configurations. If yes, the worst configurations are eliminated using Friedman post-tests. F-race stops when only a single (or very few) configurations remain in the race. Then iterated F-race samples a new set of configurations around the best configurations found so far and runs a new F-race. Iterated F-race terminates after a maximum number of algorithm runs.

Iterated F-race allows one to tune real-valued, integer, ordinal, and categorical parameters. The set of the parameters of the ABC algorithms, their ranges, and types are listed in

Table 2 Summary of ABC algorithm parameters. Each parameter is given the type, which can be real-valued (real), integer (int), or categorical (cat); the range that is considered in the tuning; and it is specified in which ABC algorithms the respective parameters occur

Parameter	Type	Range	Algorithm
General parameters			
SN	int	[5, 100]	all ABC algorithms
lf	real	[0, 3]	all ABC algorithms
ABC algorithm specific parameters			
p	real	[0, 1]	ImpABC
MR	real	[0, 1]	MABC, ImpABC
chaoticMap	cat	7 maps	CABC
C	real	[0, 4]	GbABC,GBdABC
SF	real	[0, 1]	MABC
$adaptSF$	cat	Yes/No	MABC
$w_{\min}$	real	[0, 0.5]	BsfABC
$w_{\max}$	real	[0.5, 1]	BsfABC
$SN_{\max}$	int	[10, 100]	IABC
R_{factor}	real	$10^{[-14,0]}$	IABC
g	int	[1, 20]	IABC
Local search related parameters			
LS	cat	Powell, Mtsls1	all ABC algorithms
ls_{itr}	int	[1, 100]	all ABC algorithms
$Failure_{\max}$	int	[1, 20]	all ABC algorithms
SP	real	[1, 2]	RABC
RM_{itr}	int	[1, 100]	RABC
m	int	[1, 100]	RABC

Table 2. In each tuning, at most 5 000 runs of an ABC algorithm are executed. As training instances, we use the 19 SOCO benchmark functions of dimension 10, which are seen by F-race in a random order. If all 19 benchmark functions have been tested, we reuse the benchmark functions in the same random order. The default parameter settings and the parameter settings that are obtained by the tuning are summarized in Table 3 for the parameters related to the ABC algorithms and in Table 4 for the parameters related to the local search.

As iterated F-race uses ranking, the magnitude of differences between configurations is not considered. An alternative would be to use t-race, which is a racing method implemented in the irace package (López-Ibáñez et al. 2011) that makes use of Student's t-test. The t-race procedure would tend to search for configurations with best mean performance. The biases introduced by using ranking or averaging in tuning have also been examined by Smit and Eiben (2010). Given the possible bias incurred by ranking, we evaluate here the ABC variants according to their median performance. Information on the full distribution of the algorithms results will be given in the form of solution quality distributions and run-length distributions on the article's supplementary pages (Liao et al. 2013).

Table 3 Summary of the tuned ABC algorithm parameter settings. For each ABC algorithm are given its default, tuned, and tuned with local search parameter settings in a three field notation: *default/tuned/tuned+LS*. The parameter settings of the local search used by the ABC variants are given in Table 4. For RABC, the parameter settings related to the Rosenbrock local search algorithm are given here

Algorithm	*SN*	*lf*	Other parameters	
ABC	62/8/37	1.0/2.734/2.982	–	
GbABC	15/12/40	1.0/1.12/1.171	C	1.0/1.507/2.069
BsfABC	100/6/64	0.1/2.164/1.962	$w_{\min}$	0.2/0.3327/0.2454
			$w_{\max}$	1.0/0.725/0.5843
MABC	10/11/25	1.0/1.978/0.2818	MR	0.4/0.774/0.2743
			SF	1.0/0.9707/0.6737
			$adaptSF$	No/No/No
ImpABC	25/28/35	1.0/1.62/1.58	MR	1.0/0.4108/0.7269
			p	0.25/0.4686/0.3837
CABC	10/17/54	1.0/2.819/0.4039	chaoticMap	1/3/7
GbdABC	15/11/84	1.0/2.031/2.03	C	1.0/2.176/1.469
RABC	25/10/37	1.0/2.089/1.023	SP	1.5/1.864/1.618
			RM_{itr}	15/17/34
			m	5/2/64
IABC	5/6/6	1.0/2.272/0.0619	$SN_{\max}$	50/12/93
			R_{factor}	−1/−3.47/−3.868
			g	1/12/5

Table 4 Summary of the parameter settings related to the local search for each ABC algorithm and for a random restart local search algorithm (RLS). In the automatic tuning process, for all ABC algorithms and RLS the local search method used was Mtsls1. In the table we therefore give only the numerical parameter settings relevant to each algorithm. For RABC, the parameter settings related to the Rosenbrock local search algorithm are given in Table 3

Parameter	ABC	GbABC	BsfABC	MABC	ImpABC
LS_{itr}	76	81	76	61	4
$Failure_{\max}$	1	5	1	20	9
Parameter	CABC	GbdABC	RABC	IABC	RLS
LS_{itr}	66	59	88	85	56
$Failure_{\max}$	11	9	2	12	–

4 Experimental results and analysis

4.1 Main comparison

First, we summarize the overall results of our comparison in Figs. 1 to 3. Figures 1 and 2 give box-plots that show the distribution of the median error values observed for each studied ABC variant on the 19 benchmark functions. The number on top of each box indicates for how many functions the achieved median error value is below the optimum threshold. A "+" ("−") symbol on the top of a bar indicates that an ABC variant is performing significantly

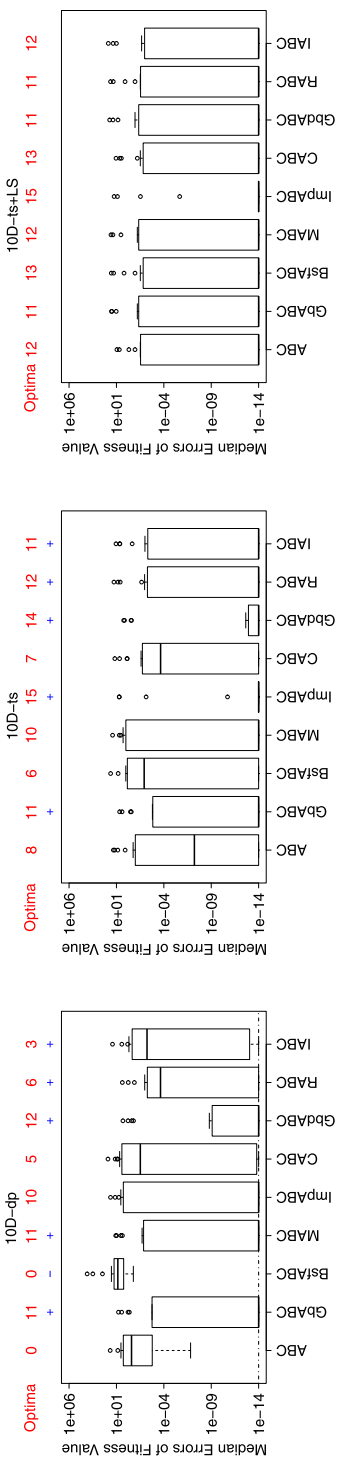


Fig. 1 Box-plots of the distribution of the median solution quality reached by each of the studied ABC variants on the 19 SOCO benchmark problems. The box-plots on the *top row* are for functions of dimension 10; the box-plots on the *bottom row* are for functions of dimension 50. From left to right, the box-plots are for the ABC algorithms with default parameter settings (−dp), with tuned parameter settings and local search (−ts + LS). The number on *top* of each box gives the number of functions on which the median error found was below the optimum threshold of 10^{-14}. A "+" or "−" symbol on *top* of a box-plot denotes that the algorithm performed significantly better or worse, respectively, than the original ABC algorithm (*first box*). For the statistical testing, we used a Wilcoxon's test at the 0.05 level with Holm's correction for multiple testing

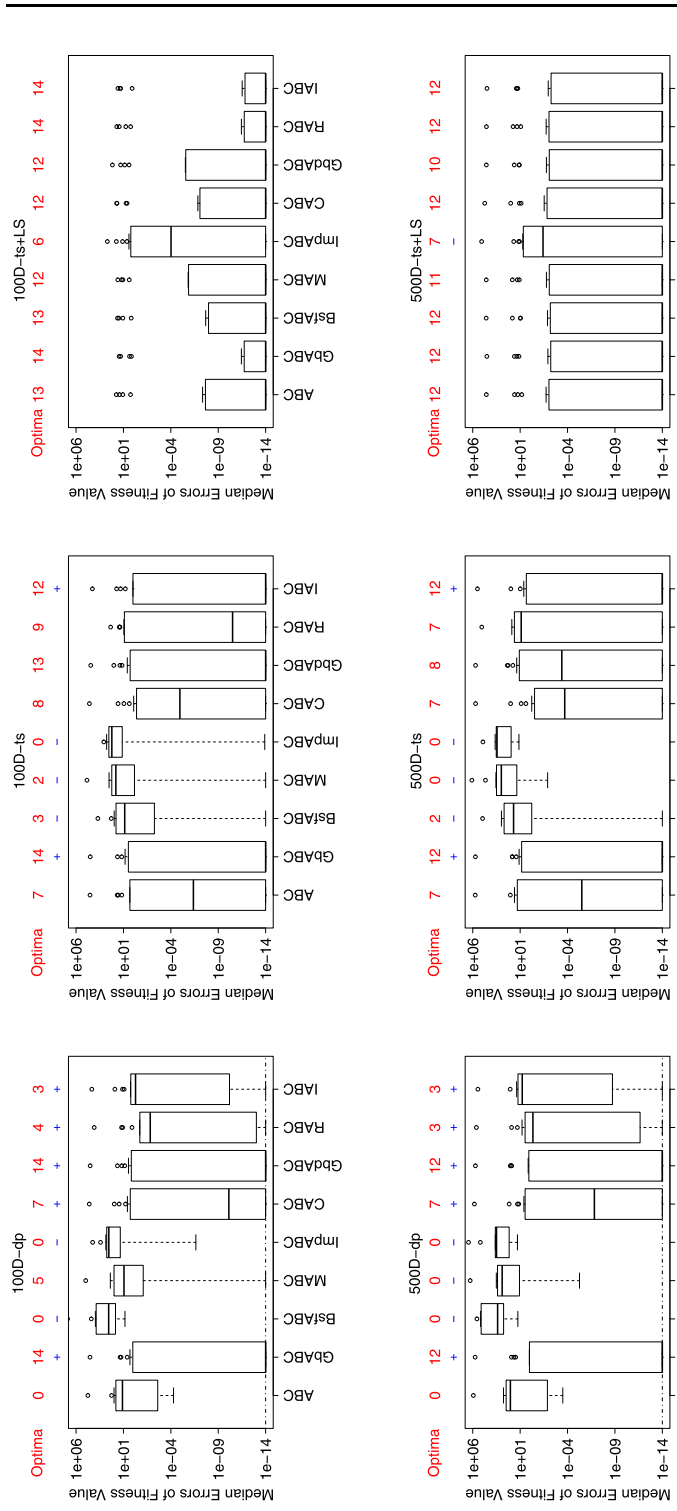


Fig. 2 Box-plots of the distribution of the median solution quality reached by each of the studied ABC variants on the 19 SOCO benchmark problems. The box-plots on the *top row* are for functions of dimension 100; the box-plots on the *bottom row* are for functions of dimension 500. From left to right, the box-plots are for the ABC algorithms with default parameter settings (−dp), with tuned parameter settings and local search (−ts + LS). The number on *top* of each box gives the number of functions on which the median error found was below the optimum threshold of 10^{-14}. A "+" or "−" symbol on *top* of a box-plot denotes that the algorithm performed significantly better or worse, respectively, than the original ABC algorithm (*first box*). For the statistical testing, we used a Wilcoxon's test at the 0.05 level with Holm's correction for multiple testing

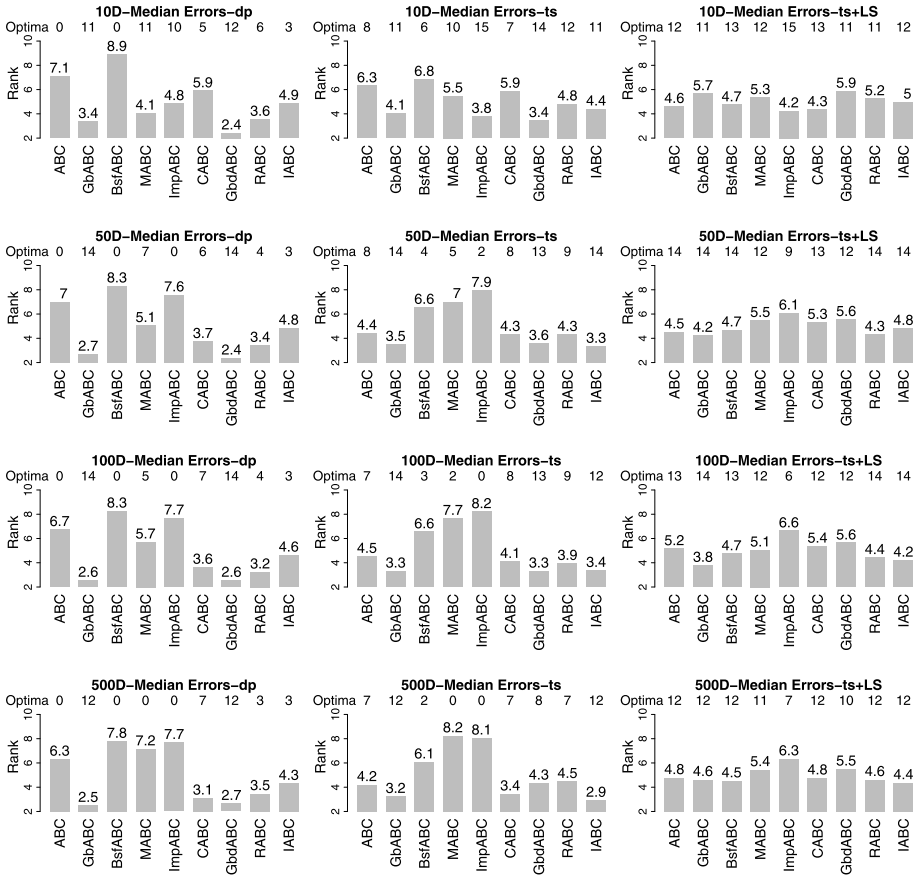


Fig. 3 Performance comparison based on the average rank over 19 functions. The ranking was computed using the median error value of each algorithm on the 19 SOCO benchmark functions

better (worse) than the original ABC algorithm. The detailed numerical data on which these box-plots are based are given in Tables 2 to 5 in the online supplementary material; further data are available at Liao et al. (2013). Additionally, we give in Fig. 3 the ranking of the variants across the various dimensions (from top to bottom, the results are given for 10, 50, 100, and 500 dimensions) for default parameters (left column), tuned parameter settings (middle column), and for the tuned ABC variants with local search (right column). To obtain these ranks, the median errors of all variants on a same function were ranked from best (rank 1) to worst (rank 9); Fig. 3 gives the so-obtained average ranks for each algorithm.

Considering the box-plots for the default parameter settings in dimension 10 (top left plot in Fig. 1), we can observe that all variants except BsfABC reach typically lower median errors that the original ABC algorithm, which is confirmed by the average ranks (top left plot in Fig. 3). (The reasons for the poor performance of BsfABC are given in the next section.) The relatively poor performance of the original ABC algorithm and BsfABC can also be noted by the fact that for none of the 19 benchmark functions, the median solution quality obtained reaches the optimum threshold. The overall best performing ABC variants in this setting and dimension are GbABC and GbdABC.

The conclusions change when comparing the results obtained by the tuned parameter settings for dimension 10 (middle plots on top in Figs. 1 and 3). In this case, the potential advantage of the ABC variants over the original ABC is reduced, and the original ABC reaches a ranking similar to that of MABC, BsfABC, and CABC. Figure 3 confirms the more similar performance among the various ABC algorithms as the range of the average ranks is much reduced when compared to the rankings obtained for default parameter settings. When moving from the default to tuned parameter settings, the improvement in performance varies strongly among the various ABC algorithms. In particular, the original ABC algorithm, BsfABC, RABC, and IABC profit strongly from the tuning. This can be noted in the large number of additional functions in which they reach the optimum threshold after tuning (for example, for the original ABC on eight functions, the optimum threshold is reached after tuning, while it was never reached by default parameter settings) and by the fact that either on all or on the large majority of the functions the tuned versions improve upon the default versions (specific data that support this statement are given in Table 2 of the supplementary material). Nevertheless, on the functions of dimension 10 still five ABC variants reach statistically significantly better results than the original ABC algorithm. However, once local search is added to each of the ABC variants, none of the variants reaches a statistically significantly improved performance over the original ABC: the introduction of an effective local search procedure reduces the gaps among the ABC variants, and none is a clear winner.

Let us consider now the dependence of the relative performance of the ABC variants when moving from the ten-dimensional problems up to the 500-dimensional ones. The most noteworthy result is the poor scaling behavior of MABC and ImpABC, which for high-dimensional problems are among the worst ranked ABC algorithms. Considering default parameter settings, ImpABC ranks worse than the original ABC for dimensions 50, 100, and 500, and MABC ranks worse than the original ABC for dimension 500; often, the observed differences are also statistically significant. Taking into account the differences in the absolute values as indicated by the box-plots and the numerical results in the online supplementary material, the differences with other ABC algorithms are substantial. The same poor scaling behavior for ImpABC and MABC is observed for the tuned parameter settings. In Sect. 4.2.5, we identify the reason for this behavior, and we give the remedy for ImpABC and MABC. GbABC and GbdABC remain among the best performing ABC algorithms also for higher-dimensional problems, showing that the emphasis on the global-best solution is important for a good scaling behavior. For the 500-dimensional problems, IABC becomes the best ranking one after tuning and the addition of local search; in fact, it is one of the ABC algorithms that most profits from tuning. Finally, considering the ABC variants with local search, their performance levels remain comparable, and no statistically significant differences in favor of any of the hybrid ABC variants over the original ABC with local search are detected.

The results of our comparison of ABC variants can be summarized as follows.

- Tuning has a major impact on the comparison results. In fact, on the higher-dimensional benchmark problems, several ABC algorithm variants do not give a statistically significant improvement over the original ABC algorithm once one considers tuned parameter settings.
- The introduction of a local search smoothes the differences between the different ABC variants, and poorly performing ones without local search become very competitive with the best ABC variants.
- Some of the variants such as BsfABC perform surprisingly poorly when compared to the original papers. Some ABC variants appear to have poor scaling behavior.

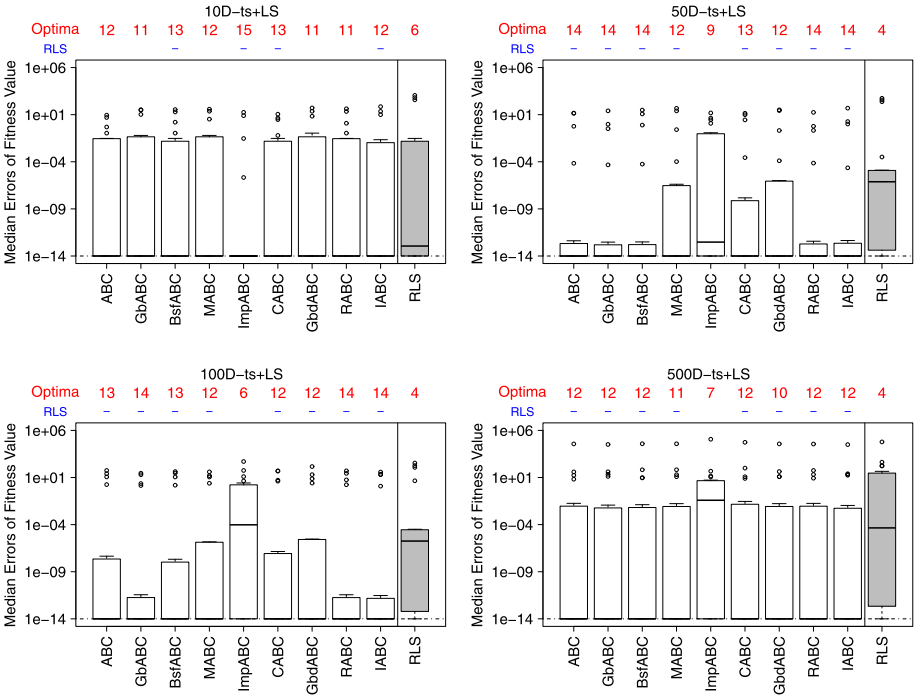


Fig. 4 Performance comparisons with RLS. The "−" symbol on top of a box-plot denotes a significant difference detected by a Wilcoxon's test at the 0.05 level (using Holm's correction) to RLS

In the following section, we revisit some of the above made conclusions and provide a more detailed analysis of several ABC variants.

4.2 Discussion and further analysis

4.2.1 Improvement over random restart local search

The best performing ABC algorithms typically include local search, as shown in the previous section. Hence, a first question that arises is whether the ABC algorithms contribute significantly to performance or whether the local search alone is enough to reach a same performance level. We explore this question by comparing the hybrid ABC algorithms to the tuned RLS. As we can see in Fig. 4, RLS gives typically worse results across all dimensions of the benchmark functions. In fact, the local search alone is able to find solutions better then the optimum threshold only on few functions across all dimensions (f_1, f_4, f_5, and f_{10}). Differently, hybrid ABC algorithms do find such solutions often for more than ten functions. Except for a few ABC variants, the observed differences are statistically significant, as indicated by the "−" symbol on top of a box. Therefore, we may conclude that there is an overall synergetic effect between the ABC algorithms and the local search.

4.2.2 Original ABC algorithm

The impact of parameter settings on the performance of ABC was studied already in earlier papers (Karaboga and Basturk 2008; Akay and Karaboga 2009; Diwold et al.

2011a). Conclusions were, for example, that a too low value of the limit parameter leads to poor performance, that ABC's performance is relatively robust given the limit value, as determined through parameter *lf*, is large enough (Akay and Karaboga 2009; Diwold et al. 2011a), and that the population size does not need to be fine-tuned in order to obtain good results (Akay and Karaboga 2009). Our results indicate, however, that even the original ABC algorithm can profit substantially from a further fine-tuning of its parameters to a point where it reaches a performance that is similar to various of its extensions that have been proposed as improvements. This result also illustrates the danger of comparing to a default version of ABC, and we recommend that in future papers on potential improvements over ABC, the original version and the modified version are automatically tuned to avoid biases by the designer of a particular extension.

4.2.3 Global-best and global-best distance ABC

The GbABC and GbdABC algorithms did not profit much or at all from the additional parameter tuning. Summary data that confirm this statement can be found in the data on win, draw, loose at the bottom of Tables 2 to 5 in the online supplementary material; these data give the number of functions that obtain better, same or worse median quality after tuning or the additional introduction of local search. The data indicate that either the GbABC design makes them rather robust w.r.t. modified parameter settings or that the original authors have already fine-tuned the parameter settings. In fact, for GbABC, the tuned parameter settings are similar to the default settings, giving some evidence for the latter. Interestingly, for GbABC and GbdABC, the additional local search does actually slightly (though not significantly) worsen performance. This indicates that appropriately designed and tuned ABC algorithms can reach very high performance on the SOCO benchmark set without an additional local search.

4.2.4 Best-so-far ABC

The poor performance of BsfABC we observed is in apparent contradiction to the excellent results reported in the original paper by Banharnsakun et al. (2011). However, these differences can be explained by particularities in the design of BsfABC. As mentioned in Sect. 2.2, BsfABC applies a solution update to each dimension such that the variables in all dimensions get closer to each other. This induces a strong bias toward solving well problems where the optimum has in all dimensions the same variable values. This is the case for the benchmark problems used in Banharnsakun et al. (2011), which have the optimum in $x = (0, \ldots, 0)$, while the benchmark problems in the SOCO set have their optimum randomly shifted within the search range (the shift is independently done for each dimension).

In Fig. 5, we show the development of the median error value over the number of function evaluations for the shifted and unshifted versions of two SOCO benchmark functions (Sphere, f_1, and Rastrigin, f_4, respectively). As expected, whether the optimum of a function is randomly shifted in the search range or not has a huge impact on the performance of BsfABC, which is only effective for problems where the optimum solutions have a same variable value in each dimension. The same experiments with other ABC variants did not show a significant influence of an optimum shift on performance. After tuning, the performance of BsfABC is strongly improved, which probably is due to the larger setting of *lf* and the smaller population size.

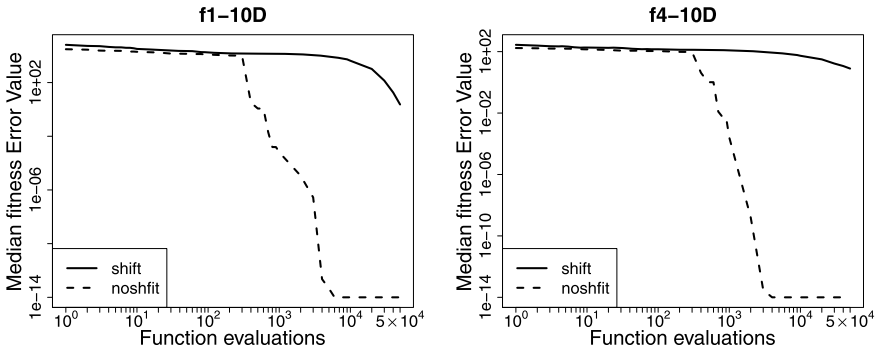


Fig. 5 Development of the median solution value over time for BsfABC on the ten-dimensional SOCO functions f_1 (Sphere, *left plot*) and f_4 (Rastrigin, *right plot*) for the shifted and unshifted versions. The unshifted version has the optimum at the point $(0, \ldots, 0)$

4.2.5 Modified and improved ABC

For functions of ten dimensions, MABC and ImpABC show good results; in fact, on ten-dimensional functions, the tuned version of ImpABC and tuned ImpABC with local search is the second best and best ranking ABC algorithm, respectively (see Fig. 3). In addition, for 10 dimensions, MABC appears to be competitive with the other ABC algorithms in its tuned version and tuned with local search (see Fig. 1). However, the performance of ImpABC and MABC declines quickly with increasing dimensionality. For example, the tuned version of ImpABC is the worst ranking ABC algorithm for 50- and 100-dimensional problems and the only ABC algorithm whose median result is not below the optimum threshold of any function for 100-dimensional problems. Similarly, the performance of MABC and ImpABC with local search declines with high-dimensional functions, though less due to the mitigating effect of the local search. As the main reason for this poor scaling behavior, we identified the choice of increasing the number of variables that are modified with the dimension of the functions.[2] In fact, if only a small, constant number of variables is modified, MABC and ImpABC become competitive with the other ABC algorithms (see also the supplementary pages Liao et al. 2013). This effect is also illustrated in Fig. 6 for MABC and in Fig. 7 for ImpABC. In each of these plots, we compare the median error values of MABC and ImpABC using default parameter settings except that the number of variables to be changed is set to eight for MABC and to four for ImpABC, as suggested by the tuning on the ten-dimensional problems. We refer to the latter variants as MABC-new and ImpABC-new.

4.2.6 Chaotic ABC

Tuning improves to some extent the performance of CABC when compared to default parameter settings, which can best be observed in the detailed results in Tables 2 to 5 in the online supplementary material and in the larger number of functions for which the optimum threshold is reached. However, the original ABC benefits even more from tuning so

[2]Note that the wrong scaling choice for the number of variables to be modified can be seen as an artifact that is introduced by tuning on training problems of only one single dimension and by the choice of defining the parameter *MR* as a factor. To avoid this artifact, more advanced possibilities for tuning the scaling behavior of parameters would have to be considered.

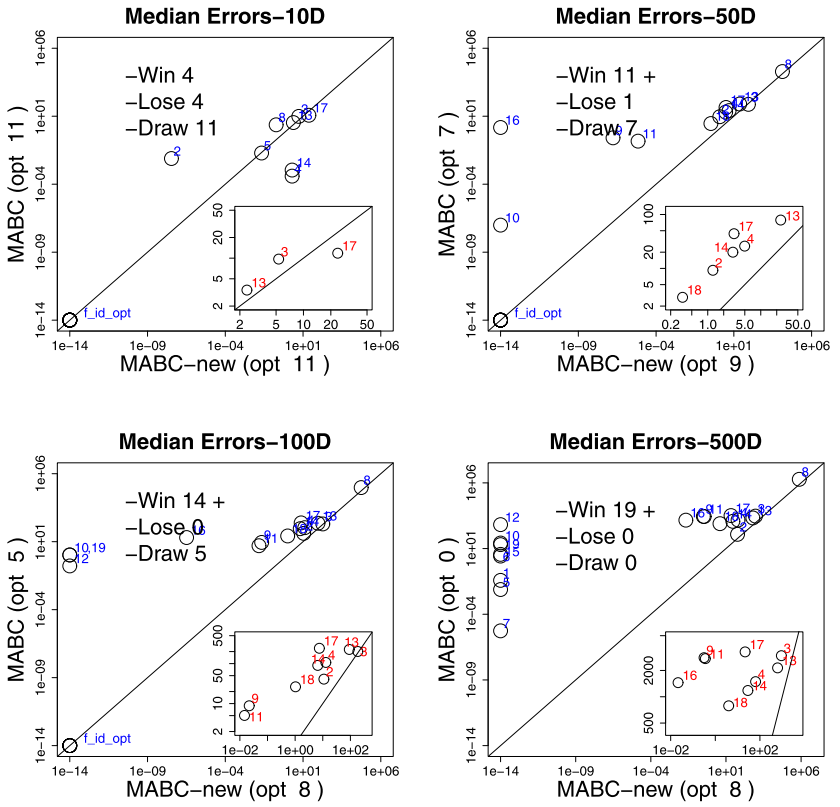


Fig. 6 Correlation plots of MABC and MABC-new on dimensions 10, 50, 100, and 500. Each point has as x- and y-coordinates the median error values obtained by MABC and MABC-new, respectively. A *point on the upper triangle* delimited by the diagonal indicates better performance for the algorithm on the x-axis; a *point on the lower right triangle* indicates better performance for the algorithm on the y-axis. If a point has a number associated, this number indicates the index of the SOCO benchmark function in Table 1 in the online supplementary material. The *inserts* in each plot enlarge the part close to the diagonal in which multiple points are clustered. The numbers related to win, draw, lose indicate how often the median error is lower for the algorithm on the x-axis than for the one on the y-axis. A $+$ symbol indicates the cases in which there is a statistically significant difference at the 0.05 α-level between the algorithms. For each algorithm, we give in parentheses (opt #) the number of times the algorithm obtained a median error lower than the optimum threshold

that once tuned, CABC does not show anymore a significant improvement over the original ABC version; this puts in doubt the real importance of the proposed modifications.

4.2.7 Rosenbrock ABC

RABC is the only ABC algorithm in the comparison that in its original form makes use of a local search procedure. While on dimension 10 it obtains on several functions better median performance than the tuned original ABC algorithm (better on nine and worse on two), for larger dimensions, it is roughly on par with the original ABC, despite making use of the Rosenbrock rotational direction method (RM). One reason may be that RM is not a very effective local search method for the benchmark problems tested here. In fact,

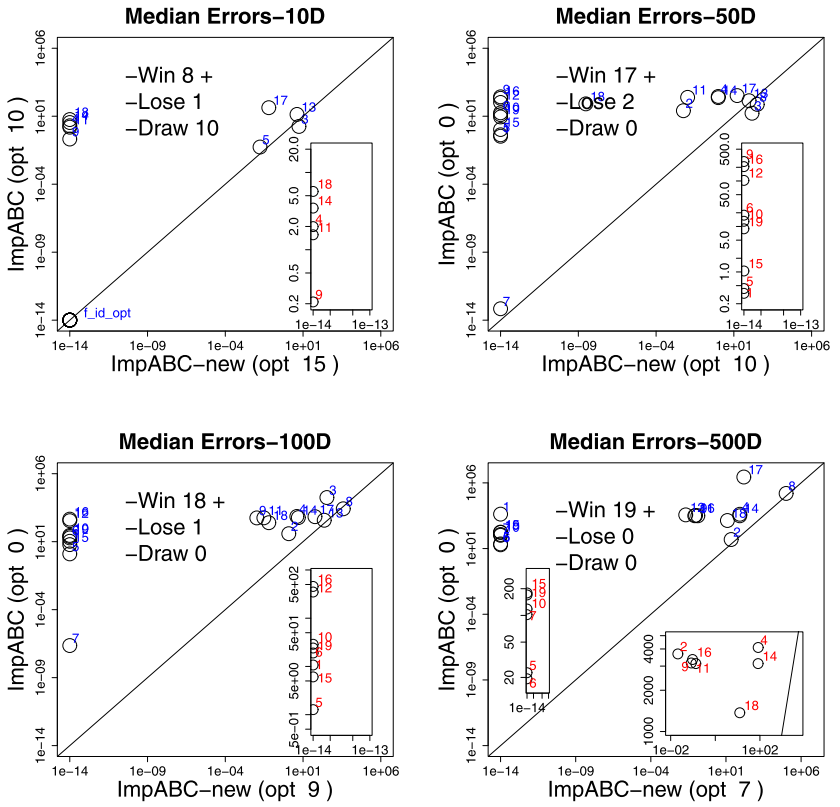


Fig. 7 Correlation plots of ImpABC and ImpABC-new on dimensions 10, 50, 100, and 500 respectively. For an explanation of the plots, see caption of Fig. 6

after adding the Mtsls1 local search, which is executed in addition to the usage of RM in RABC, the resulting hybrid RABC improves strongly its performance; this indicates that this conjecture is true.

We can also test the effect RM has in RABC by removing it. Once RM is removed from RABC, the only difference from the resulting ABC algorithm to the original ABC algorithm is the usage of the rank-based probabilistic selection of a solution by the onlooker bees instead of the fitness-based one through Eq. (4). The impact of this choice on ABC performance can be observed from the plots in Fig. 8, where the parameter settings correspond to the tuned settings of the original ABC algorithm. In fact, replacing the fitness-based selection by the rank-based probabilistic selection of solutions leads for all dimensions to improved performance (being statistically significant in dimensions 10 and 100). Additionally, a rank-based selection has the advantage of being invariant to a monotonous scaling of the objective function.

4.2.8 Incremental ABC

Similarly to other ABC algorithms, IABC profits strongly from the additional tuning and the addition of the local search algorithm. In fact, the tuned version of IABC and the version with local search reach better performance than the default version on the majority of the

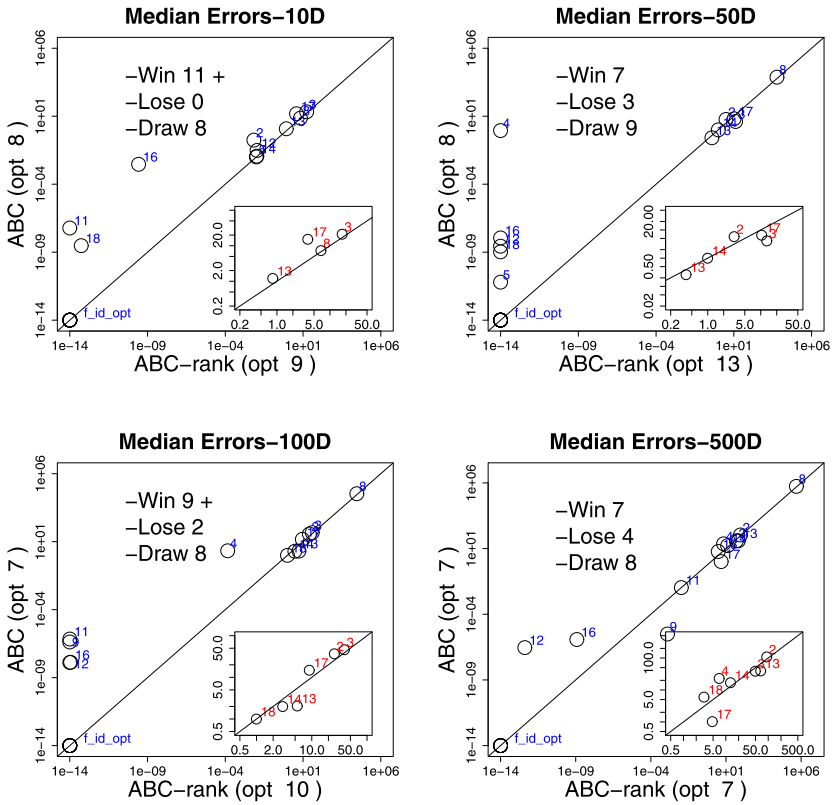


Fig. 8 Correlation plots of ABC and ABC with a rank-based selection of solutions by onlooker bees (ABC-rank) on dimensions 10, 50, 100, and 500, respectively. The parameter settings used are the tuned settings for ABC. For an explanation of the plots, see caption of Fig. 6

benchmark functions, as can be seen in the data on win, draw, loose at the bottom of Tables 2 to 5 in the online supplementary material. The strong improvement after tuning is probably due to the lower limit on the maximum population size in IABC and the slower increase of the population size when compared to the default parameter settings.

4.2.9 Solution behavior of the ABC variants

Here, we first examine the impact of parameter tuning and the usage of an additional local search have on the development of the median error over computation time measured by the number of function evaluations; these plots are also called SQT plots (Hoos and Stützle 2005). In Fig. 9, there are given representative SQT plots for the original ABC algorithm, GbABC, and IABC on three functions of dimension 50 (functions f_1, f_3, and f_{13} from the SOCO benchmark set, that is, Sphere, Rosenbrock, and a hybrid function). In these plots, we can observe that the tuning improves strongly the convergence behavior of the original ABC algorithm and IABC. However, GbABC is almost not affected by the tuning—this can be noticed, in part, by the overlapping curves for the default and tuned versions. Additional local search may further speed up convergence toward high-quality solutions as noticed by the fact that these curves are typically left-most in the plots. However, the additional local

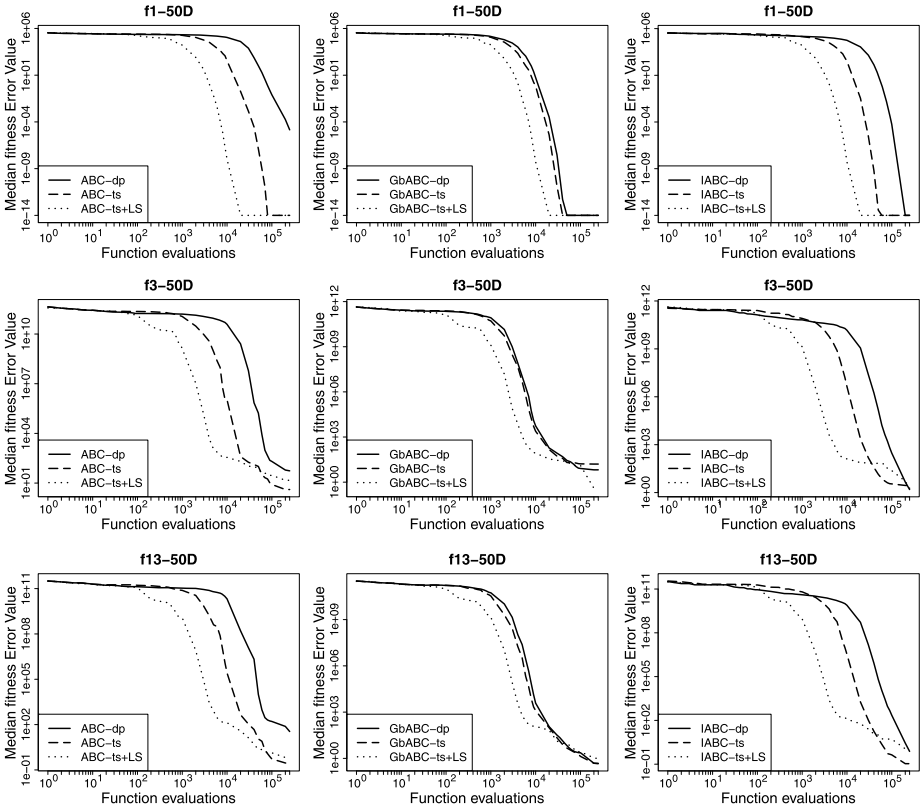


Fig. 9 SQT curves for the original ABC (*left column*), GbABC (*middle column*), and IABC (*right column*) for functions f_1 (Sphere, *top*), f_3 (Rosenbrock, *middle*) and f_{13} (a hybrid function, *bottom*)

search does not always improve the final solution quality as can be noticed, for example, on the plots for function f_{13} for IABC.

In addition to SQT plots, we have generated run-length distributions (RLDs) for reaching the optimum threshold (or other very high-quality solutions) and solution-quality distributions (SQDs) (Hoos and Stützle 2005). In Fig. 10, there are given RLDs for the same three algorithms with tuned parameter settings and hybridized with local search on benchmark functions f_5 and f_6. The RLDs on benchmark function f_5 indicate that the ABC variants suffer in part from severe stagnation behavior. The most extreme case is for IABC with local search: after less than 20000 function evaluations, it finds in almost 50 % of the runs a solution better than the optimum threshold, but in the remaining, much longer runs, it hardly finds additional optima. The RLDs on benchmark function f_5 are also interesting because they show how the ranking of the variants changes depending on whether local search is used or not: on f_5 the original ABC with local search performs better than the other two algorithms with local search, while without local search IABC and GbABC are clearly better than the original ABC algorithm. The shape of the RLDs on f_6 are more representative for the RLDs that are observed on the majority of the functions: with local search, many ABC variants have rather similar performance, while without local search, differences are still apparent. Typical is also that for many functions, the ABC variants do not show stagnation behavior (see supplementary pages (Liao et al. 2013)).

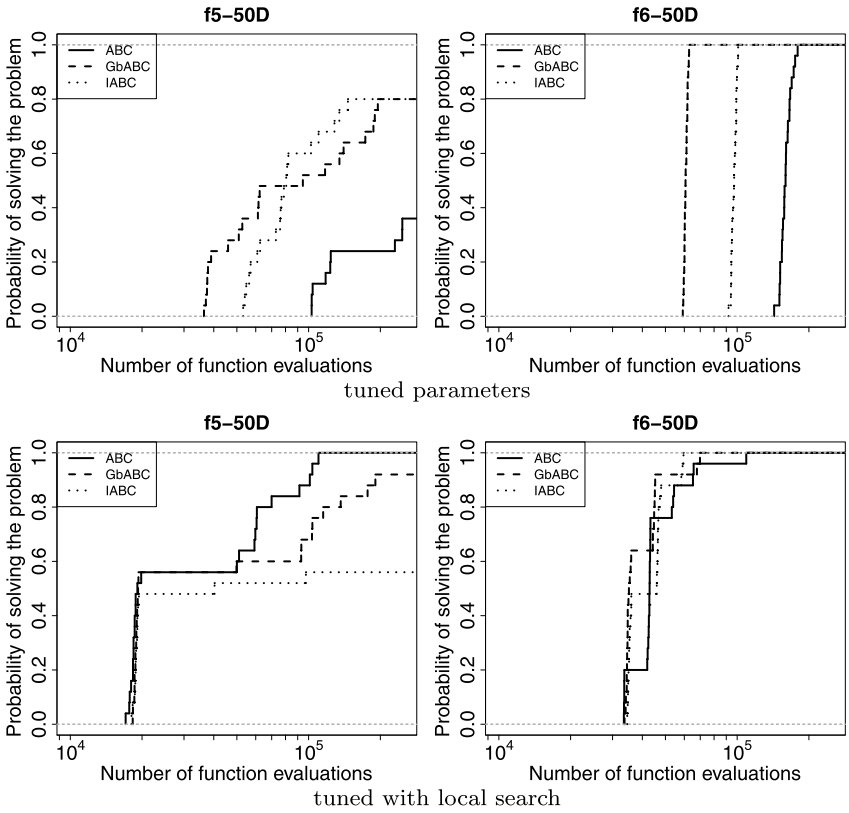


Fig. 10 RLDs for the original ABC, GbABC, and IABC for functions f_5 (*left column*) and f_6 (*right column*) using tuned parameter settings (*upper row*) and in combination with local search (*bottom row*)

Finally, Fig. 11 gives two SQD curves for benchmark function f_5, which give the distribution of the solution quality at the maximum number of function evaluations. These SQD curves confirm the stagnation behavior showing that some runs are trapped in suboptimal solutions, an effect that is most visible in the SQD curves for ABC variants with local search. In such cases, we conjecture that algorithm performance could be improved by either changing the choice of which solutions undergo improvement by local search or some partial algorithm restart.

4.3 Comparison with SOCO special issue contributors

We now compare one of the best performing ABC variants with local search to the algorithms that have been contributed to the Soft Computing special issue (Lozano et al. 2011). In our experiments, we used the same experimental setup as proposed for this special issue; hence, such a comparison is fair. The results of 13 algorithms have been published in the special issue. In addition, three reference algorithms were included in the comparison, which are a differential evolution algorithm (Stern and Price 1997), the real-coded CHC algorithm (Eshelman and Schaffer 1993), and G-CMA-ES (Auger and Hansen 2005). In particular, G-CMA-ES was the best performing algorithm in the special session on real function optimization of the 2005 IEEE Congress on Evolutionary Computation (CEC'05),

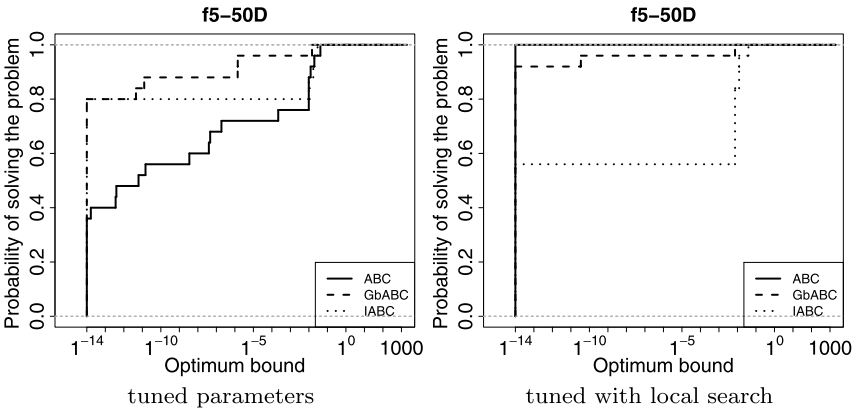


Fig. 11 SQD curves for the original ABC, GbABC, and IABC for function f_5 using tuned parameter settings (*left*) and in combination with local search (*right*)

and the MA-SSW algorithm (Molina et al. 2011), one of the 13 algorithms published in the SOCO special issue, was the best performing algorithm at the CEC 2010 special session on large-scale global optimization. Hence, the algorithms to which we compare can be considered representatives of the state-of-the-art in real parameter optimization.

For this comparison, we have chosen GbABC with local search as it is one of the best ABC algorithms across various dimensions and a rather straightforward ABC variant. We have computed for all the algorithms the distribution of the median error values found from the publicly available results tables, and the box-plots in Fig. 12 compare these distributions across several dimensions. GbABC reaches a median performance that is competitive to the best continuous optimizers from the competition; for example, it reaches for 50 and 100 dimensions a median error that is below the optimum threshold for 14 functions, the same as the overall best performing algorithm in the benchmark competition, MOS-DE (LaTorre et al. 2011). A further statistical analysis (using Wilcoxon's test with Holm's correction for multiple comparisons) shows that GbABC is statistically significantly better at the 0.05 significance level than the reference algorithms DE, CHC, and G-CMA-ES, as well as some of the contributors such as SOUPDE, GODE, MA-SSW, RPSO-vm, EvoPROpt, EM323, and VXQR1 for dimensions 100 and 500. Hence, when considering median performance, GbABC (and actually also several other ABC algorithms such as IABC, which have very similar performance to GbABC) is competitive with state-of-the-art algorithms for large-scale continuous optimization problems.

The good median performance is in part due to the fact that the tuning is done by a rank-based method, in our case F-race. If the algorithms are compared based on mean error values, GbABC is still among the best performing ones, but when compared to, for example, MOS-DE, it performs worse on more functions than when the comparison is based on median error values. (A comparison of GbABC to the other algorithms from the SOCO special issue based on mean error values is available from Liao et al. 2013.) The main reason for the worse mean performance is due to the stagnation of the algorithm in some runs on some of the functions. It would be interesting to retune the algorithms based on, for example, t-race (Birattari 2009; López-Ibáñez et al. 2011), which implicitly favors algorithms having better average behavior.

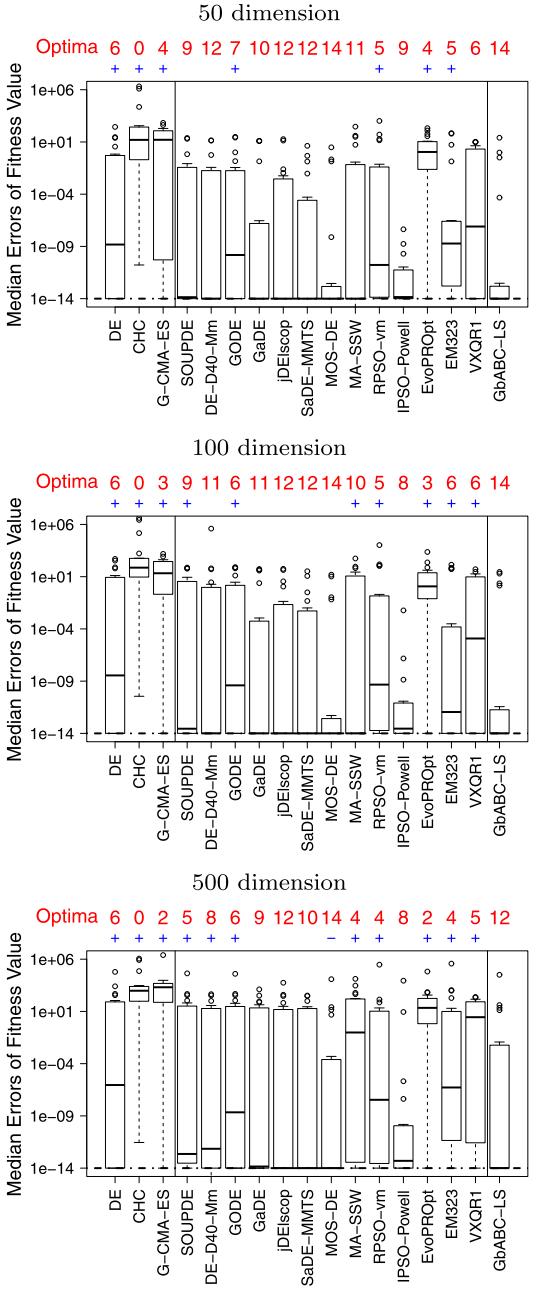


Fig. 12 Comparison of tuned GbABC with local search to the algorithms from the SOCO benchmark competition. A "+" or "−" symbol on top of a box-plot denotes that GbABC performed significantly better or worse, respectively, than the indicated algorithm. For the statistical testing, we used a Wilcoxon's test at the 0.05 level with Holm's correction for multiple testing

5 Conclusions

In this article, we have reviewed ABC algorithms for continuous optimization, and we have experimentally studied many of the proposed ABC variants using a recent benchmark function set for large-scale continuous optimization. Our experimental study is based on reimple-

mentations of nine ABC algorithms. The ABC variants were compared using three setups: first, using their default parameter settings; second, using an automatic algorithm configuration tool to determine parameter settings; and third, by combining the ABC algorithms with local search and retuning their parameter settings. The usage of an automatic algorithm configuration tool allowed us (i) to improve significantly the default algorithm parameter settings for most of the ABC variants; (ii) to avoid the bias in results due to differences in the computational effort spent for tuning the different algorithm variants; and (iii) to help in the design of hybrid ABC algorithms that include an effective local search for continuous optimization.

The main results of our experimental analysis and comparison are the following. First, when considering default parameter settings and low-dimensional benchmark problems, most of the ABC variants improve over the original ABC algorithm.[3] However, this conclusion changes as we move to tuned parameter settings and to the ABC variants with local search or as we move to higher-dimensional problems. Considering tuned parameter settings, several of the proposed ABC variants do not result anymore in significantly improved performance (especially in higher-dimensional problems). The situation gets even "worse" if we move to the ABC variants that include an effective local search algorithm. In fact, no statistically significant differences are detected between the ABC variants and the original ABC algorithm once these algorithms are hybridized with local search. This result is due to the fact that an effective local search smoothes the performance differences between algorithms without local search. In fact, we could claim that, as a side-product of our experimental analysis, we have derived a number of new state-of-the-art ABC algorithms: in fact, most of the high-performing, hybrid ABC algorithms we examine have never been considered before. However, the particular scheme we used of adding local search to ABC algorithms has been examined before in other contexts (Montes de Oca et al. 2011; Aydın et al. 2012; Liao et al. 2011), and the resulting high performance is rather an indication that this particular hybridization scheme is promising. A surprising result is that some of the ABC variants that we examined do not profit from the additional local search phase and reach very high performance without it. This is the case for GbABC and GbdABC, and it suggests that ABC algorithms, if appropriately configured and tuned, combine well a global search behavior with a strong local search behavior.

We also compared the median error values obtained by one of the best performing ABC algorithms, the hybrid GbABC, to the results of a recent benchmarking effort for high-dimensional function optimization. This comparison shows that the best performing ABC variants we examined are competitive with state-of-the-art algorithms for continuous function optimization on the benchmark set we used.

There are a number of directions in which our work could be extended. A first direction is to analyze ABC algorithms also on other benchmark sets; in fact, depending on the properties on the benchmark functions, the relative performance of algorithms may change.[4] Another direction would be to reconsider the integration of local search algorithms into ABC algorithms and try to elaborate a more refined and potentially more performing scheme for integration. The fact that our hybrid algorithms have shown on some functions stagnation behavior is an indication that this direction is promising. Still another direction

[3] An exception is the best-so-far ABC algorithm (Banharnsakun et al. 2011) for which we have observed poor performance. In Sect. 4.2.4, we have shown that this poor behavior is due to specific choices in the algorithm design.

[4] Possible options might be the Black-Box Optimization Benchmarking (BBOB) suite or the benchmark set from the CEC 2005 and 2013 special sessions on real-parameter optimization.

would be to include other ABC variants in our experimental comparison. However, we believe that more interesting would be the elaboration of a flexible algorithm framework for ABC algorithms, where the various proposed modifications to ABC are implemented as algorithm components that may be combined. Automatic algorithm configuration could then be applied to such a framework to obtain even more performing ABC variants automatically. Such an approach has been explored previously (KhudaBukhsh et al. 2009; López-Ibáñez and Stützle 2012), and the high performance of our initial effort on configuring hybrid ABC algorithms with local search just confirms that such an approach is very promising.

Acknowledgements The research leading to the results presented in this paper has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007–2013)/ERC grant agreement No. 246939. This work was also supported by the META-X project, an *Action de Recherche Concertée* funded by the Scientific Research Directorate of the French Community of Belgium. Thomas Stützle acknowledges support from the Belgian F.R.S.-FNRS, of which he is a Research Associate. Tianjun Liao acknowledges a fellowship from the China Scholarship Council. We also acknowledge the detailed comments by the editor and the referees that helped to improve considerably the paper.

References

Abraham, A., Jatoth, R. K., & Rajasekhar, A. (2012). Hybrid differential artificial bee colony algorithm. *Journal of Computational and Theoretical Nanoscience*, *9*(2), 249–257.

Akay, B., & Karaboga, D. (2009). Parameter tuning for the artificial bee colony algorithm. In N. T. Nguyen et al. (Eds.), *LNCS: Vol. 5796. Proceedings of the international conference on computational collective intelligence* (pp. 608–619). Berlin: Springer.

Akay, B., & Karaboga, D. (2012). A modified artificial bee colony algorithm for real-parameter optimization. *Information Sciences*, *192*, 120–142.

Alam, M. S., Ul Kabir, M. W., & Islam, M. M. (2010). Self-adaptation of mutation step size in artificial bee colony algorithm for continuous function optimization. In *Proceedings of international conference on computer and information technology* (pp. 69–74). Piscataway: IEEE Press.

Alataş, B. (2010). Chaotic bee colony algorithms for global numerical optimization. *Expert Systems with Applications*, *37*(8), 5682–5687.

Auger, A., & Hansen, N. (2005). A restart CMA evolution strategy with increasing population size. In *Proceedings of IEEE Congress on evolutionary computation* (pp. 1769–1776). Piscataway: IEEE Press.

Aydın, D., Liao, T., Montes de Oca, M. A., & Stützle, T. (2012). Improving performance via population growth and local search: the case of the artificial bee colony algorithm. In J. K. Hao et al. (Eds.), *LNCS: Vol. 7401. Proceedings of the international conference on artificial evolution* (pp. 85–96). Berlin: Springer.

Balaprakash, P., Birattari, M., & Stützle, T. (2007). Improvement strategies for the F-race algorithm: sampling design and iterative refinement. In T. Bartz-Beielstein et al. (Eds.), *Hybrid metaheuristics, LNCS* (Vol. 4771, pp. 108–122). Berlin: Springer.

Banharnsakun, A., Achalakul, T., & Sirinaovakul, B. (2011). The best-so-far selection in artificial bee colony algorithm. *Applied Soft Computing*, *11*(2), 2888–2901.

Birattari, M. (2009). *Tuning metaheuristics: a machine learning perspective. Studies in computational intelligence: Vol. 197*. Berlin: Springer.

Birattari, M., Stützle, T., Paquete, L., & Varrentrapp, K. (2002). A racing algorithm for configuring metaheuristics. In *Proceedings of the genetic and evolutionary computation conference* (pp. 11–18). San Francisco: Morgan Kaufmann.

Birattari, M., Yuan, Z., Balaprakash, P., & Stützle, T. (2010). F-race and iterated f-race: an overview. In T. Bartz-Beielstein et al. (Eds.), *Experimental methods for the analysis of optimization algorithms* (pp. 311–336). Berlin: Springer.

Diwold, K., Aderhold, A., Scheidler, A., & Middendorf, M. (2011a). Performance evaluation of artificial bee colony optimization and new selection schemes. *Memetic Computing*, *3*(3), 149–162.

Diwold, K., Beekman, M., & Middendorf, M. (2011b). Honeybee optimisation-an overview and a new bee inspired optimisation scheme. In B. K. Panigrahi et al. (Eds.), *Handbook of swarm intelligence-concepts, principles and application, adaptation, learning, and optimization* (Vol. 8, pp. 295–328). Berlin: Springer.

El-Abd, M. (2011a). A hybrid ABC-SPSO algorithm for continuous function optimization. In *Proceedings of IEEE symposium on swarm intelligence* (pp. 1–6). Piscataway: IEEE Press.

El-Abd, M. (2011b). Opposition-based artificial bee colony algorithm. In *Proceedings of the genetic and evolutionary computation conference* (pp. 109–116). New York: ACM.

Eshelman, L. J., & Schaffer, J. D. (1993). Real-coded genetic algorithms and interval-schemata. In D. L. Whitley (Ed.), *Foundation of genetic algorithms 2* (pp. 187–202). San Mateo: Morgan Kaufmann.

Fister, I., Fister, I. Jr., Brest, J., & Zumer, V. (2012). Memetic artificial bee colony algorithm for large-scale global optimization. In *Proceedings of IEEE Congress on evolutionary computation* (pp. 1–8). Piscataway: IEEE Press.

Gao, W., & Liu, S. (2011). Improved artificial bee colony algorithm for global optimization. *Information Processing Letters*, *111*(17), 871–882.

Gao, W., Liu, S., & Huang, L. (2012). A global best artificial bee colony algorithm for global optimization. *Journal of Computational and Applied Mathematics*, *236*(11), 2741–2753.

Guo, P., Cheng, W., & Liang, J. (2011). Global artificial bee colony search algorithm for numerical function optimization. In *Proceedings of international conference on natural computation* (Vol. 3, pp. 1280–1283). Piscataway: IEEE Press.

Herrera, F., Lozano, M., & Molina, D. (2010). Test suite for the special issue of Soft Computing on scalability of evolutionary algorithms and other metaheuristics for large scale continuous optimization problems. http://sci2s.ugr.es/eamhco/.

Hoos, H. H., & Stützle, T. (2005). *Stochastic local search—foundations and applications*. San Francisco: Morgan Kaufmann.

Kang, F., Li, J., & Ma, Z. (2011). Rosenbrock artificial bee colony algorithm for accurate global optimization of numerical functions. *Information Sciences*, *181*(16), 3508–3531.

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

Karaboga, D., & Akay, B. (2009). A survey: algorithms simulating bee swarm intelligence. *Artificial Intelligence Review*, *31*(1–4), 61–85.

Karaboga, D., & Basturk, B. (2007). A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *Journal of Global Optimization*, *39*(3), 459–471.

Karaboga, D., & Basturk, B. (2008). On the performance of artificial bee colony (ABC) algorithm. *Applied Soft Computing*, *8*(1), 687–697.

KhudaBukhsh, A. R., Xu, L., Hoos, H. H., & Leyton-Brown, K. (2009). SATenstein: automatically building local search SAT solvers from components. In *Proceedings of international joint conferences on artificial intelligence* (pp. 517–524). Menlo Park: AAAI Press.

LaTorre, A., Muelas, S., & Peña, J. M. (2011). A MOS-based dynamic memetic differential evolution algorithm for continuous optimization: a scalability test. *Soft Computing*, *15*(11), 2187–2199.

Lee, W. P., & Cai, W. T. (2011). A novel artificial bee colony algorithm with diversity strategy. In *Proceedings of international conference on natural computation* (Vol. 3, pp. 1441–1444). Piscataway: IEEE Press.

Liao, T., Montes de Oca, M. A., Aydin, D., Stützle, T., & Dorigo, M. (2011). An incremental ant colony algorithm with local search for continuous optimization. In *Proceedings of the genetic and evolutionary computation conference* (pp. 125–132). New York: ACM.

Liao, T., Aydın, D., & Stützle, T. (2013). Artificial bee colonies for continuous optimization: Experimental analysis and improvements. http://iridia.ulb.ac.be/supp/IridiaSupp2013-002.

López-Ibáñez, M., & Stützle, T. (2012). The automatic design of multi-objective ant colony optimization algorithms. *IEEE Transactions on Evolutionary Computation*, *16*(6), 861–875.

López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., & Birattari, M. (2011). *The irace package, iterated race for automatic algorithm configuration* (Technical Report TR/IRIDIA/2011-004). IRIDIA, Université Libre de Bruxelles, Belgium.

Lozano, M., Molina, D., & Herrera, F. (2011). Editorial scalability of evolutionary algorithms and other metaheuristics for large-scale continuous optimization problems. *Soft Computing*, *15*(11), 2085–2087.

Ming, H., Baohui, J., & Xu, L. (2010). An improved bee evolutionary genetic algorithm. In *Proceedings of IEEE international conference on intelligent computing and intelligent systems* (Vol. 1, pp. 372–374). Piscataway: IEEE Press.

Molina, D., Lozano, M., García-Martínez, C., & Herrera, F. (2010). Memetic algorithms for continuous optimisation based on local search chains. *Evolutionary Computation*, *18*(1), 27–63.

Molina, D., Lozano, M., Sánchez, A., & Herrera, F. (2011). Memetic algorithms based on local search chains for large scale continuous optimisation problems: MA-SSW-Chains. *Soft Computing*, *15*(11), 2201–2220.

Montes de Oca, M. A. (2011). *Incremental social learning in swarm intelligence systems*. Ph.D. Thesis, Université Libre de Bruxelles, Brussels, Belgium.

Montes de Oca, M. A., Aydin, D., & Stützle, T. (2011). An incremental particle swarm for large-scale continuous optimization problems: an example of tuning-in-the-loop (re)design of optimization algorithms. *Soft Computing*, *15*(11), 2233–2255.

Powell, M. J. D. (1964). An efficient method for finding the minimum of a function of several variables without calculating derivatives. *Computer Journal*, *7*(2), 155–162.

Rajasekhar, A., Abraham, A., & Pant, M. (2011). Levy mutated artificial bee colony algorithm for global optimization. In *Proceedings of IEEE international conference on systems, man, and cybernetics* (pp. 655–662). Piscataway: IEEE Press.

Rechenberg, I. (1973). *Evolutionsstrategie: optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Stuttgart: Frommann-Holzboog.

Rosenbrock, H. H. (1960). An automatic method for finding the greatest or least value of a function. *Computer Journal*, *3*(3), 175–184.

Sharma, T., & Pant, M. (2012). Enhancing scout bee movements in artificial bee colony algorithm. In *Proceedings of the international conference on soft computing for problem solving, advances in intelligent and soft computing* (Vol. 130, pp. 601–610). India: Springer.

Sharma, T., Pant, M., & Bhardwaj, T. (2011). PSO ingrained artificial bee colony algorithm for solving continuous optimization problems. In *Proceedings of international conference on computer applications and industrial electronics* (pp. 108–112). Piscataway: IEEE Press.

Smit, S. K., & Eiben, A. E. (2010). Beating the 'world champion' evolutionary algorithm via REVAC tuning. In *Proceedings of IEEE Congress on evolutionary computation* (pp. 1–8). Piscataway: IEEE Press.

Stern, R., & Price, K. (1997). Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, *11*(4), 341–359.

Tseng, L. Y., & Chen, C. (2008). Multiple trajectory search for large scale global optimization. In *Proceedings of IEEE Congress on evolutionary computation* (pp. 3052–3059). Piscataway: IEEE Press.

Wu, B., & Sh, F. (2011). Improved artificial bee colony algorithm with chaos. In Y. Yu et al. (Eds.), *Communications in computer and information science: Vol. 158. Computer science for environmental engineering and EcoInformatics* (pp. 51–56). Berlin: Springer.

Yan, X., Zhu, Y., & Zou, W. (2011). A hybrid artificial bee colony algorithm for numerical function optimization. In *Proceedings of international conference on hybrid intelligent systems* (pp. 127–132). Piscataway: IEEE Press.

Zhong, Y., Lin, J., Ning, J., & Lin, X. (2011). Hybrid artificial bee colony algorithm with chemotaxis behavior of bacterial foraging optimization algorithm. In *Proceedings of international conference on natural computation* (Vol. 2, pp. 1171–1174). Piscataway: IEEE Press.

Zhu, G., & Kwong, S. (2010). Gbest-guided artificial bee colony algorithm for numerical function optimization. *Applied Mathematics and Computation*, *217*(7), 3166–3173.

Zou, W., Zhu, Y., Chen, H., & Zhu, Z. (2010). Cooperative approaches to artificial bee colony algorithm. In *Proceedings of international conference on computer application and system modeling* (Vol. 9, pp. 44–48). Piscataway: IEEE Press.