A modified bees algorithm and a statistics-based method for tuning its parameters

Q T Pham, D T Pham and M Castellani

Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering 2012 226: 287 originally published online 13 October 2011
DOI: 10.1177/0959651811422759

A modified bees algorithm and a statistics-based method for tuning its parameters

Q T Pham[1], **D T Pham**[2,3]*, and **M Castellani**[4]

[1]University of New South Wales, Sydney, Australia

[2]School of Mechanical Engineering, University of Birmingham, Birmingham, UK;

[3]Department of Information Systems, King Saud University, Riyadh, Saudi Arabia

[4]Department of Biology, University of Bergen, Bergen, Norway

The manuscript was received on 15 January 2011 and was accepted after revision for publication on 17 August 2011.

DOI: 10.1177/0959651811422759

Abstract: This paper presents a novel version of the bees algorithm. This version is characterized by an extended set of search operators, and a mechanism that protects the most recently generated solutions from competition with more evolved individuals. Compared to the standard implementation of the bees algorithm, the new procedure requires the selection of an additional set of parameters. A new statistical method is proposed to tune these extra parameters. The proposed tuning method was used to determine a unique set of learning parameters for the modified bees algorithm on eight popular function optimization benchmarks. When tested against the standard bees algorithm and two other well-known optimization procedures, the new algorithm attained top performances on nearly all the benchmarks. The experimental results also proved that, tested on a search space much larger than that where it was tuned, the modified bees algorithm still outperformed the standard method, and the degradation of the performance of the two algorithms was comparable. These results prove the effectiveness of the modified bees algorithm, and show that the proposed tuning procedure is a valuable alternative to the complex and subjective trial-and-error methods that are often used.

Keywords: bees algorithm, swarm intelligence, evolutionary algorithms, parameter tuning, factorial experiment, optimization.

1 INTRODUCTION

The globalization and development of science, technology, and economic networks have created large and complex systems which are difficult to model and optimize. At the same time, advances in the understanding of biological systems have revealed that nature is able to solve many complex optimization tasks, such as the evolution of species and animal foraging behaviours, through the self-organization of large and decentralized ensembles of individuals.

Nature's strategies have inspired a number of very effective engineering approaches based on the collective problem-solving capabilities of populations of elementary agents. Evolutionary algorithms (EAs) [**1**] are a class of search methods that mimic the mechanisms of natural evolution to solve hard optimization problems. Swarm intelligence (SI) [**2**] models the behaviour of social insects like ants [**3**] and bees [**4**] to control, model, and optimize highly complex distributed systems.

The behaviour of EAs and genetic algorithms (GAs) is determined by a number of system parameters which need to be tuned. These parameters include the population size, the rate of application of the search operators, the population replacement rate, and so on. In more complex evolutionary and SI algorithms, the use of extra features entails the optimization of

**Corresponding author: School of Mechanical Engineering, University of Birmingham, Birmingham B15 2TT, UK.*
email: d.t.pham@birmingham.ac.uk

additional factors. In some cases, the total number of adjustable parameters can be quite substantial. The 'No Free Lunch' Theorem [**5**] postulates that there is no universal setting that guarantees top performance on all possible optimization problems. Nevertheless, on restricted classes of problems some settings work better than others. That is, optimization tasks that share certain characteristic patterns are more likely to benefit from similar search approaches. The importance of tailoring the problem solver to the specific domain has brought some authors to regard parameter selection as one of the key factors in building a successful application [**6**].

The tuning of an algorithm is itself an optimization task which can be considered a multi-objective optimization problem, since speed and robustness are often contradictory requirements. Most designers carry out at least some manual tuning [**7**]. Grefenstette [**8**] used a meta-optimization method based on two cascaded evolutionary processes. In this case, evolutionary search was used to optimize the performance (i.e. the parameters) of a population of EAs. Each parameter setting was evaluated on the results of a full EA run. Owing to the slowness of evolutionary procedures, this meta-optimization method is very time consuming. Moreover, due to the statistical nature of stochastic search methods, the evaluation of the solutions is affected by a considerable noise problem.

Adaptive tuning is a popular and effective approach. It was used to adapt the mutation step size in evolutionary strategies by Rechenberg [**9**] and first applied to genetic algorithms by Davis [**10**]. A recent implementation employs statistical outlier detection to estimate operator capabilities and adjust operator frequencies accordingly [**11**]. Another kind of adaptive tuning takes the form of competitive evolution between separate but interacting populations with different parameter settings. This approach was first proposed by Pham [**12**].

Adaptive tuning is conceptually attractive to EA practitioners. However, it requires some computational effort to be spent on the adaptation process. On a more fundamental level, adaptive tuning is a hill climbing procedure (with some stochasticity arising from noise). Particularly in the case that the search is started far away from the global optimum, the algorithm is likely to converge to sub-optimal solutions, or be led astray by noisy fitness evaluations. For the above reasons, it is important to have at least a rough idea of where to start the adaptive tuning procedure. For certain classes of problems, it may also be possible to determine a common tuning based on a number of tests. Pham [**13**] proposed a statistics-based tuning method involving carrying out a two-level factorial experiment. With this method, the operator probabilities take either a low (usually zero) or high value in each test. The effect of each operator can then be quantified and the results used to tune the algorithm.

In this paper, the optimization of an improved version of the bees algorithm [**4**] is discussed. The bees algorithm is a recently developed SI method that loosely mimics the foraging behaviour of honey bees. The algorithm does not make assumptions about the nature of the solution space, such as continuity or differentiability, and requires little information about the problem domain beyond a criterion to evaluate the performance of the solutions. Pham and Castellani [**14**] proved that the bees algorithm is particularly effective at searching accurately narrow fitness valleys and holes, and optimizing highly multimodal functions. The main shortcoming of the bees algorithm is the need to tune a number of system parameters. The bees algorithm has been successfully applied to various engineering problems, such as pattern classifier training [**15**], manufacturing cell formation [**16**], mechanical design [**17**], machine shop scheduling [**18**], control system tuning [**19**], and digital filter design [**20**].

Some modifications to the bees algorithm are first introduced. A novel method is then described that was used to perform a first rough tuning of the learning parameters of the modified bees algorithm. The validity of the proposed procedures is demonstrated on a set of analytical benchmark problems commonly used in evolutionary optimization. Section 2 outlines the standard bees algorithm. The modified version is presented in section 3. Section 4 discusses the tuning of the learning parameters. Section 5 gives preliminary results concerning the effect of the setting of various (combinations of) parameters on the performance of the algorithm. Section 6 presents the proposed tuning procedure. Section 7 compares the performance of the tuned algorithm to that of the standard bees algorithm and two other well-known optimization methods. Section 8 concludes the paper and gives suggestions for further work.

2 THE STANDARD BEES ALGORITHM

This section outlines the standard bees algorithm and its biological motivation. For a more detailed description, the reader is referred to Pham and Castellani [**14**].

2.1 Bees foraging process in nature

In a bee colony, part of the population randomly explores the fields surrounding the hive in search of

food [**21**]. When a scout finds a flower patch rich in nectar or pollen, it takes a load of the content and heads back to the nest to deposit it. At the nest, the scout communicates to idle nest-mates the location of the newly discovered flower patch via a sophisticated ritual known as the 'waggle dance' [**22**]. As a result of this ritual, more foragers join the scout to exploit the food source. When the recruited foragers return to the hive to unload the food harvested, they may in turn waggle dance to recruit additional nest-mates. The likelihood that a bee performs the waggle dance, and the length of the dance, is proportional to the quality rating that the bee attaches to the food source. Consequently, the best flower patches are visited by more bees, and the colony maximizes the efficiency of the food harvesting process [**21**].

2.2 The bees algorithm

Without loss of generality, it will be assumed henceforth that the optimization problem entails the maximization of a given performance index. The solutions are encoded as n-dimensional vectors of decision variables, where each variable is bound within a pre-defined interval. Each candidate solution is regarded as a food source, and its quality (i.e. performance) is rated via an objective function (fitness function). An artificial bee colony of N individuals is considered.

At the beginning of the search the whole population is used to scout the search space, and evaluate the quality of the visited sites. That is, the solution space is randomly sampled and the fitness of the picked solutions is measured. The visited locations are ranked according to their quality rating, and 'forager bees' are recruited to harvest (i.e. search) the neighbourhood of the highest ranking food sources. That is, new solutions similar to the best-so-far solutions are generated and tested. At each iteration, the best m sites are selected for local neighbourhood search. Of these m sites, the top ranking e 'elite' sites are assigned the largest number of recruited foragers (ne), and the remaining $m - e$ sites are assigned $nb \leqslant ne$ foragers. For each assigned forager, a new location is sampled near the solution space site marked by the scout bee. In practice, the parameters of a high-fitness solution are randomly perturbed via a mutation or similar local search operation [**1**]. For each selected site, the search area is initialized to a large value and adjusted as the solution is progressively refined in subsequent iterations (*neighbourhood shrinking* procedure [**14**]).

At the end of the local search procedure, the best site (i.e. solution) of each of the m neighbourhoods is retained in the population. If the local search procedure fails to improve the fitness of a selected solution for a pre-fixed number of iterations, the solution is thought to be a local peak. In this case, the solution is re-initialized to a new randomly selected position in the search space (*site abandonment* procedure [**14**]).

At every learning cycle, $N - m$ 'scouts' keep on exploring the search space looking for new regions of high fitness. That is, $N - m$ solutions are randomly generated and evaluated.

The bees algorithm balances the exploration and exploitation efforts through the allocation of scout and forager bees. Scout bees perform a global explorative search of the solution space, and direct the local exploitative search towards regions of highest fitness. The pseudocode of the standard bees algorithm is given in Fig. 1.

For more information about the standard bees algorithm the reader is referred to Pham and Castellani [**14**].

3 THE MODIFIED BEES ALGORITHM

The distinguishing features of the bees algorithm are:

(a) continuous creation of new, randomized solutions;
(b) favouring of the best solutions by conducting more neighbourhood search steps around these solutions;
(c) use of random search within a shrinking neighbourhood window.

However, it is known that each class of problems needs a different set of search operators that best match its challenges. The proposed modified bees algorithm uses the following collection of operators which were designed for real-valued search spaces: mutation, creep, crossover, interpolation and extrapolation.

As in the original bees algorithm, a solution is encoded as an n-dimensional vector of real-valued

```
Initialise N random solutions.
Iterate until a stopping criterion is met
    Evaluate the fitness of the population.
    Sort the solutions according to their fitness.
    // waggle dance
    Select the highest-ranking m solutions for neighbourhood search.
    Assign x = ne foragers to each of the e≤m top-ranking elite solutions.
    Assign x = nb ≤ ne foragers to each of the remaining m-e selected solutions.
    // local search
    For each of the m selected solutions
        Create x new solutions by perturbing the selected solution randomly or otherwise.
        Evaluate the new solutions.
        Retain the best solution amongst the selected and new solutions.
        Adjust the neighbourhood of the retained solution.
    Re-initialise sites where search has stagnated for a given number of iterations.
    // global search
    Repeat N-m times
        Generate randomly a new solution.
        Evaluate the new solution.
    Form new population (m solutions from local search, N-m from global search)
```

Fig. 1 Pseudocode of standard bees algorithm

decision variables bounded to a continuous interval. Mutation randomly picks elements (i.e. variable values) of a solution's encoding and re-initializes them to random values drawn with uniform probability from the variables' range. Creep applies a small Gaussian perturbation (standard deviation equal to 0.001 of the variable's range) to all the elements of the solution vector. Crossover takes two 'parent' solutions and generates a 'child' solution composed of the first $k < n$ elements of the vector defining the first solution and the remaining $n - k$ elements of the vector defining the second. Interpolation takes two solutions and creates a child by adding element-by-element 0.7 times the components of the fittest parent and 0.3 times the components of the least fit parent, giving a solution which is biased to the first. Extrapolation takes two solutions and creates a child by adding element-by-element 1.3 times the components of the fittest parent and -0.3 times the components of the least fit parent. When a two-parent operator is applied to a solution, the second parent is chosen randomly from the rest of the population.

The probabilities of the reproduction operators are not tuned in this work but are based on previous results [**13**].

At each generation, the best of the population is retained (i.e. truncation selection is applied). Each surviving solution or 'bee' is given an opportunity to improve through a few evolution phases (i.e. steps) using the above operators. At each phase, one operator is applied once to the selected solution. If the change results in an improvement, the new solution replaces the parent. Otherwise, the original solution is retained. The number of evolution steps m_s is a decreasing function of the rank of the solution in the population (see Fig. 2)

$$m_s = \text{int}\left[(n_0 - 1)\left(\frac{\text{Rank}_{\text{last}} - \text{Rank}}{\text{Rank}_{\text{last}} - 1}\right)^f\right] + 1 \qquad (1)$$

where Rank is 1 for the fittest member, and n_0 is the number of evolution phases allocated to the fittest member of a generation. The least fit of the selected members (of $\text{Rank} = \text{Rank}_{\text{last}}$) is always given only one phase. The exponent f allows tuning of the distribution of evolution steps: when $f = 1$ the number of phases varies linearly with Rank, while when $f > 1$ the number of phases decreases more quickly.

As in the previous version of the bees algorithm, the least fit members of the population are deleted and new scout bees are randomly created to replace them. However, forcing new solutions to compete immediately with solutions already in the population, which may have evolved for a long time, is counterproductive, just as it would be in a society where newborn members were forced to compete with adults without going through a period of nurturing. This is particularly true for multimodal objective functions, where, for example, a new solution may land at the foot of a high peak and still be worse than old solutions that have stalled at a lower peak. Therefore, this paper introduces a new category of bees, the 'young members', which have evolved for fewer than a given number M of steps. The young members compete for survival only among themselves, until they reach 'adulthood' after evolving M steps. Each surviving young is given at least one evolution step to improve.

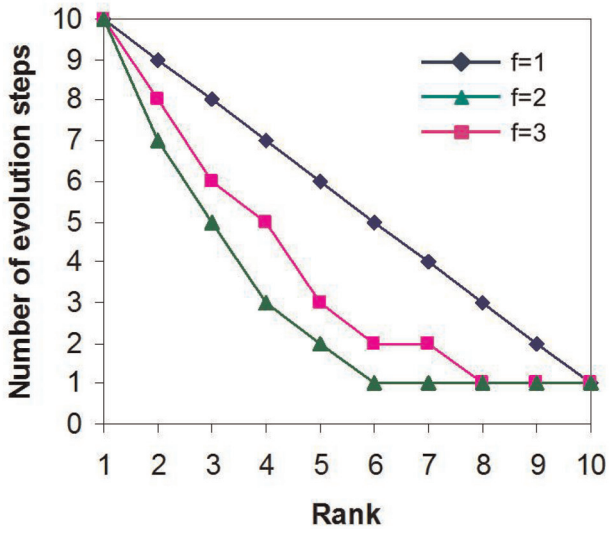

Fig. 2 Number of search steps as a function of rank for $n_0 = 10$, $\text{Rank}_{\text{last}} = 10$, and varying f

In summary, the modified bees algorithm strongly favours the best bees, but also continuously creates young bees and nurtures them, so that the pool of 'best bees' may be continuously rejuvenated if necessary. The pseudocode of the modified bees algorithm is given in Fig. 3.

The fraction e_r of selected bees as a percentage of the whole population is

$$e_r = \frac{e_s}{N} \qquad (2)$$

Initialise N random solutions.
Iterate until stopping criterion is met.
  Evaluate the fitness of the whole population.
  Sort the whole population according to fitness.
  Select the e_s fittest solutions for the new population.
  Allocate m_s evolution steps to every surviving solution (see eq.(1)).
  Sort the young bees (age $\leq M$) according to their fitness.
  Select the g fittest young bees for the new population.
  Create $N-e_s-g$ randomly initialised new bees (age=0) and add them to the new population.
  Allocate one evolution step to every young bee.
  Apply evolution steps.

Fig. 3 Pseudocode of modified bees algorithm

Table 1 Minimization benchmark problems

Problem	Solution space	Minimum	Objective function
1. Rosenbrock 2D	2D $-2.048 < x_i < +2.048$	$F_{opt} = 0$ $\mathbf{x}_{opt} = (1, 1)$	$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$
2. Griewangk	10D $-600 < x_i < +600$	$F_{opt} = 0$ $\mathbf{x}_{opt} = (0)$	$f(\vec{x}) = \frac{1}{4000} \cdot \sum_{i=0}^{10} x_i^2 \cdot \prod_{i=0}^{i=10} \cos\left(\frac{x_i}{\sqrt{i+1}}\right) + 1$
3. Shekel	2D $-65.536 < x_i < +65.536$	$F_{opt} = -1$ $\mathbf{x}_{opt} = (-32, -32)$	$f(x_1, x_2) = -\sum_{1}^{25} \frac{1}{j + (x_1 - a_{1j})^6 + (x_2 - a_{2j})^6}$
4. Schwefel	6D $-500 < x_i < +500$	$F_{opt} = -2513.9$ $\mathbf{x}_{opt} = (420.97)$	$f(x) = \sum_{i=1}^{10} x_i \sin\sqrt{\lvert x_i \rvert}$
5. Steps (de Jong)	5D $-5.12 < x_i < +5.12$	$F_{opt} = -25$ $\mathbf{x}_{opt} = (-5.12)$	$f(x) = -\sum_{i=1}^{5} \text{int}(x_i)$
6. Rosenbrock 5D	5D $-2.048 < x_i < +2.048$	$F_{opt} = 0$ $\mathbf{x}_{opt} = (1)$	$f(\vec{x}) = \sum_{i=1}^{5} 100 \cdot (x_{i+1} - x_i^2)^2 + (1 - x_i)^2$
7. Goldstein and Price	2D $-2 < x_i < +2$	$F_{opt} = 3$ $\mathbf{x}_{opt} = (0, -1)$	$f(x) = [1 + (x_1 + x_2 + 1)^2 (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)]$ $X[30 + (2x_1 - 3x_2)^2 (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$
8. Rastrigin	20D $-5.12 < x_i < +5.12$	$F_{opt} = 0$ $\mathbf{x}_{opt} = (0)$	$f(\vec{x}) = \sum_{i=1}^{i=20} \left[(x_i)^2 - 10 \cdot \cos(2 \cdot \pi \cdot x_i) + 10\right]$

Similarly, the fraction of selected young bees, g_r, as a percentage of the whole population of juveniles is

$$g_r = \frac{g}{(N - e_s)} \tag{3}$$

The pseudocode implies that a very good young bee may be copied in the new population twice, once as one of the fittest bees, and another time as one of the fittest young bees. This not only makes the algorithm simpler but also encourages exceptional newcomers. However, this case is not expected to happen frequently, except in the early stages of the algorithm.

4 EXPERIMENTAL EVALUATION

The proposed version of the bees algorithm was tested on eight well-known continuous function minimization benchmark problems [**23**]. The benchmarks are shown in Table 1.

The number of system parameters that define the behaviour of the standard bees algorithm is modest. Consequently, it is possible to tune the procedure effectively through trial and error [**14**]. However, due to the presence of additional operators, a much larger number of system parameters needs to be adjusted in order to optimize the performance of the modified bees algorithm. The tuning of the modified procedure is also complicated by the possible interactions amongst different operators. For this reason, trial and error tuning is likely to be very time consuming, and may yield sub-optimal results. Therefore, the statistical approach proposed by Pham [**13**] is used to optimize the modified bees algorithm procedure.

As in reference [**13**], a two-level full factorial experiment was performed to obtain a first estimate of the recommended parameter values for the proposed modified version of the bees algorithm. In this experiment, each parameter can take a low or high value.

The values used for the experiments were chosen after some preliminary tests and are shown in Table 2.

For each of the 2^6 combinations of parameters, the algorithm was run 10 times on every test problem. Each run of the algorithm corresponded to around 1000 function evaluations, except for *Rosenbrock*'s problem in 2D which was run for *ca.* 500 evaluations, the 5D *Steps* problems which was run for about 100 evaluations, and *Schwefel*'s problem, which was run for approximately 2000 evaluations. The exact numbers of evaluations for each benchmark are given in section 7, which details the test conditions. These numbers of evaluations were chosen to reflect the difficulty of each benchmark. If too few function evaluations are performed, the optimum will not be found. If too many function evaluations are carried out, it becomes more difficult to distinguish between bad and good parameter settings. For each run, the final value of the objective function is taken as the measure of performance.

As stated earlier, the operator probabilities were set based on previous work [**13**]. They are given in Table 3.

5 PRELIMINARY RESULTS

A brief introduction to the measures used is given, as statistical terms may be unfamiliar to some

Table 2 Parameter values in factorial experiment

Parameter	Symbol	Low value	High value
Population size	N	10	50
Selected bees as a fraction of the whole population	e_r	0.5	0.8
Selected young bees (with fewer than M evolution steps) as a fraction of all young bees	g_r	0	0.5
Number of evolution steps per generation for fittest bee	n_0	2	10
Linearity parameter	f	2	5
Number of evolution steps for 'adulthood'	M	5	10

Table 3 Operator probabilities

Parameter	Symbol	Value
Mutation rate	m_r	0.07
Creep rate	cp_r	0.43
Crossover rate	cx_r	0.07
Interpolation rate	ip_r	0.07
Extrapolation rate	xp_r	0.36

readers. The (main) *effect* of a parameter A is defined by

$$E(A) = \frac{\Sigma_A - \Sigma_a}{2r} \tag{4}$$

where Σ_A is the sum of the results (in this case the best fitness values) of all runs with parameter A set to its high value (see Table 2), Σ_a is the sum of the results of all runs with parameter A set to its low value (see Table 2), and r is the number of runs performed per parameter setting (10 in this study). If parameter A has no effect on the performance, the difference between the sums of the results obtained using the high and low setting of A ($\Sigma_A - \Sigma_a$) is very small, and $E(A)$ approaches zero.

The effect of the *interaction* between two parameters A and B is defined by:

$$E(A*B) = \frac{\Sigma_{AB} - \Sigma_{Ab} - \Sigma_{aB} + \Sigma_{ab}}{2r} = \frac{\Sigma_{AB} - \Sigma_{Ab}}{2r} - \frac{\Sigma_{aB} - \Sigma_{ab}}{2r} \tag{5}$$

where Σ_{Ab} is the sum of the results of all runs with A high and B low, and so forth. $E/2r$ is the difference between the effect of varying parameter B when A is high ($\Sigma_{AB} - \Sigma_{Ab}$) and that obtained when A is low ($\Sigma_{aB} - \Sigma_{ab}$). Equivalently it is the difference between the effect of varying parameter A when B is high ($\Sigma_{AB} - \Sigma_{aB}$) and that obtained when B is low ($\Sigma_{Ab} - \Sigma_{ab}$)

For each parameter or parameter combination (interaction), T is the effect, E, divided by its standard error, and P indicates the level of confidence in the effect, E. For instance, a P value of 0.01 corresponds to a confidence level of 99 per cent [**24**].

As an example, Table 4 gives the effects of the various factors on the best fitnesses found for Problem 1.

It shows with a high level of confidence ($P < 0.0005$, or better than 99.95 per cent confidence level) that the population size N has a definite effect on fitness. This is reflected also by the large ratio of the effect over its standard error, abs(T) = 5.68. The effect of N on the best fitness found is represented by E, which appears very small (−0.0104), but as most results are crowded closely near the best fitness value of 0, it is in fact considerable. The negative values of E and T indicate that a high level of the factor N results in a decrease (i.e. improvement) in the best fitness found. Other effects are listed in the table in order of decreasing confidence levels.

The effects tabulated in Table 4 indicate that the various factors interact very strongly, for example, the effect of $e_s{*}g_r$ is as great as that of g_r and much greater than that of e_s. Hence, it is not clear from the table what particular selection of parameter gives the best effect. For example, although g_r gives a positive effect (row 4), if a low population, N, is chosen, g_r may in fact have a negative effect due to the positive sign of the interaction term, $N{*}g_r$ in row 6. Other problems yield similar results.

Table 4 Summary of effects of the various parameters for Problem 1

Term	Effect	T	P
N	−0.0104	−5.68	<0.0005
$g_r = g/(N\text{-}e_s)$	0.006 98	3.81	<0.0005
$e_s{*}g_r$	−0.006 33	−3.46	0.001
$N{*}g_r$	0.005 78	3.16	0.002
n_0	0.005 24	2.86	0.004
$N{*}g_r{*}\ n_0$	−0.004 95	−2.71	0.007
$N{*}g_r{*}f{*}M$	−0.004 67	−2.55	0.011
$N{*}\ n_0$	0.003 86	2.11	0.036
$e_r = e_s/N$	0.003 51	1.92	0.056
$e_s{*}g_r{*}M$	−0.0034	−1.86	0.063
$N{*}f{*}M$	0.003 38	1.85	0.065
$e_s{*}M$	0.003 09	1.69	0.092
$e_s{*}g_r{*}M$	−0.002 98	−1.63	0.104
$g{*}\ n_0{*}f{*}M$	0.002 89	1.58	0.115

6 A NEW TUNING PROCEDURE FOR THE MODIFIED BEES ALGORITHM

In view of the significant interactions between the learning parameters, it is difficult to tune the bees algorithm according to the simple method of Pham [**13**]. Therefore, an alternative method was devised, based on the following principles:

(a) avoid any parameter setting that leads to poor performance;
(b) if two parameter settings are equally likely to cause poor performance, then choose the parameter setting that is more likely to give a good performance.

These principles are based on the observation that, while the best runs tend to give fitness results that are very close to each other, the results of the worst runs are distributed along a very long tail. In addition, the fitness results of the worst runs are much further below the median than the results of the best runs are above the median (see Fig. 4). Thus, the overriding concern in tuning an algorithm must be preventing the occurrence of bad runs.

The proposed tuning procedure calculates the best and worst level for each parameter. These best and worst settings are based on the weighted means of the top 10 per cent and bottom 10 per cent of the runs, respectively. Preliminary investigations indicate that if much fewer runs are used, statistical fluctuations may become significant, while if more runs are considered, the weighted average tends towards the overall mean, and thus the effect of the various parameters is more difficult to evaluate (see Fig. 5).

The detailed tuning procedure is composed of four steps as follows.

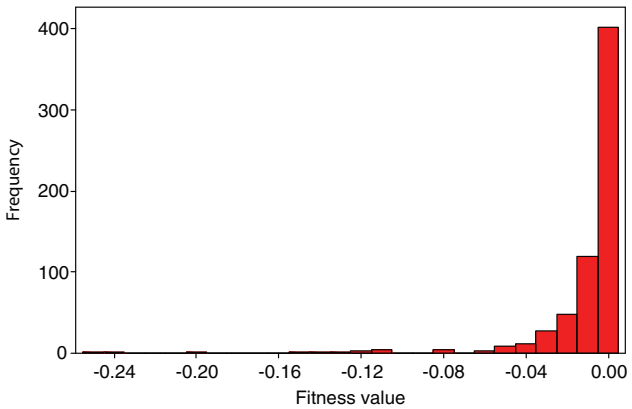

Fig. 4 Performance distribution for different parameter settings, Problem 1, showing long tail for poorly-performing settings

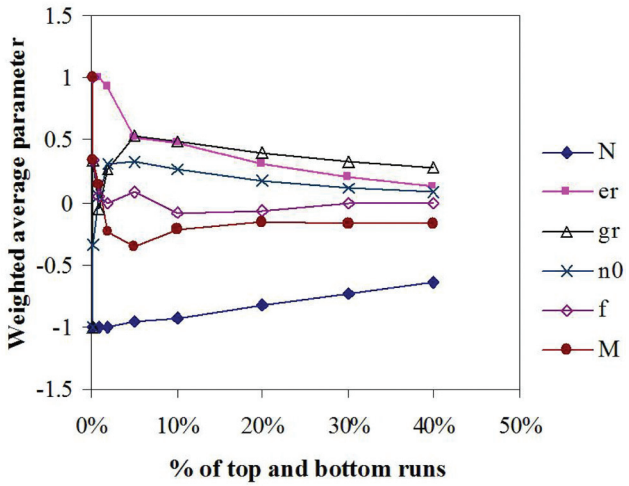

Fig. 5 Effect of per cent of runs included on weighted averages of the best parameter values for Problem 1

1. Calculate the weighted mean level of the parameters in the worst 10 per cent of all the runs for a given test problem. (–1 is the low level of the factor, +1 is the high level). Weights are adjusted to vary linearly with the rank, the worst run having a weight of 1, the 10th percentile a weight of 0. This gives the *mean worst settings* for the problem.
2. Calculate the weighted mean level of the parameters in the best 10 per cent of the runs in the same manner (the best run has the maximum weight). This gives the *mean best settings* for the given test problem.
3. Repeat steps 1 and 2 for all the other test problems and calculate the overall mean 'best' and 'worst' level of each parameter for each problem.
4.
   (a) If the mean worst level of a parameter is low (−1 to −0.33), set that parameter to the maximum level.
   (b) If the mean worst level of a parameter is high (+0.33 to +1), set that parameter to the minimum level.
   (c) In other cases (−0.33 to +0.33), set that parameter to its mean best level.

Tables 5 and 6 give the best and worst levels of the parameters for the test problems, and their average.

The clearest effect is that of the population size *N*. For all the problems, a small population gives better results. The effect is weakest for Problem 3 (*Shekel*), where the worst runs are just as likely to

Table 5 Best levels of parameters

Problem	N	$e_r = e_s/N$	$g_r = g/(N\text{-}e_s)$	n_0	f	M
1. Rosenbrock 2D	−0.93	0.48	0.49	0.26	−0.09	−0.22
2. Griewangk	−1.00	0.35	0.44	0.68	−0.20	0.19
3. Shekel	−0.55	0.41	0.53	0.23	−0.16	−0.03
4. Schwefel	−0.91	0.42	0.48	−0.03	−0.22	−0.06
5. Steps (de Jong)	−1.00	−0.70	−0.30	0.37	−0.18	−0.23
6. Rosenbrock 5D	−0.49	0.00	−0.05	0.15	−0.13	0.04
7. Goldstein- Price	−0.81	0.57	0.38	0.28	−0.24	−0.17
8. Rastrigin	−0.95	0.40	0.31	0.36	−0.14	0.16
Mean best level	−0.83	0.24	0.28	0.29	−0.17	−0.04

Table 6 Worst levels of parameters

Problem	N	$e_r = e_s/N$	$g_r = g/(N\text{-}e_s)$	n_0	f	M
1. Rosenbrock 2D	0.5	−0.3	−0.2	−0.4	0.0	0.1
2. Griewangk	1.0	−0.3	−0.9	−0.9	−0.1	0.0
3. Shekel	0.1	−0.2	−0.1	−0.2	0.1	−0.3
4. Schwefel	0.9	−0.3	−0.8	−0.5	0.3	−0.1
5. Steps (de Jong)	0.5	0.0	−0.1	−0.5	0.2	0.2
6. Rosenbrock	0.8	−0.4	−0.7	−0.7	−0.2	−0.2
7. Goldstein- Price	0.7	0.1	−0.8	−0.4	0.2	−0.1
8. Rastrigin	1.0	−0.5	−0.7	−0.8	0.0	0.0
Mean worst level	0.68	−0.23	−0.55	−0.56	0.06	−0.06

Table 7 Tuning of algorithm

Parameter	N	$e_r = e_s/N$	$g_r = g/(N\text{-}e_s)$	n_0	f	M
Worst level (−1 to +1)	0.68	−0.23	−0.55	−0.56	0.06	−0.06
Worst (High/Low/Indifferent)	H	I	L	L	I	I
Best level (−1 to +1)	−0.79	0.32	0.32	0.30	−0.16	−0.03
Recommended level (−1 to +1)	−1	0.32	1	1	−0.16	−0.03
Level −1 parameter value	10	0.5	0.0	2	2.0	5
Level +1 parameter value	50	0.8	0.5	10	5.0	10
Recommended value	10	0.7	0.5(*)	10	3	7

(*)For g_r the calculated recommended value is 0.5 which would give a number of surviving young bees g of 1.5. Hence g_r will have to be adjusted up to give $g = 2$.

involve a small population as a large one (Row 3, Column 2 of Table 6). This can be explained since, given a total number of function evaluations, the number of generated new random bees (and so the exploration capability) does not depend on the population size. However, population size does affect the number of bees that are favoured by neighbourhood search: a small population means that local exploitative search (i.e. extra evolution steps) is concentrated in fewer locations in the solution space. For highly multi-modal functions like the *Shekel* benchmark, this predominantly exploitative strategy is likely to be detrimental.

Table 7 details how the recommended tuning is calculated using the proposed tuning procedure. The first row (after the headings) gives the weighted mean worst levels (bottom 10 per cent of runs) as calculated in the first step. The second row converts the values to qualitative judgments. The third row gives the weighted mean best levels as calculated in the second step of the procedure. The fourth row calculates the recommended levels for the parameters according to step 4 of the procedure. The actual parameter settings are calculated from the recommended levels (row 4) and the definitions of high (row 5) and low (row 6) levels given in Table 2. The final recommended settings are given in the last row.

For example, the mean worst level for N is 0.68, which indicates that the algorithm performs worst when N is 'High', therefore N will be set to its lowest possible level, –1, or $N = 10$ (Rule 4a). The mean worst level for e_r is –0.23, which falls in the 'Indifferent' range, therefore e_r will be set to its best mean value, at a level of 0.32 on the –1 to + 1 scale, or $e_r = 0.7$ (Rule 4c).

7 PERFORMANCE OF THE TUNED ALGORITHM

With the parameters of the modified bees algorithm as shown in Table 7, a set of tests was run to assess the effectiveness of the proposed algorithm. In this case, the performance of the modified bees algorithm was compared to that of the standard bees algorithm, an EA, and particle swarm optimization (PSO) [**25**]. The tests were run on the eight continuous function minimization benchmarks given in Table 1.

7.1 Experimental Conditions

All the four optimization algorithms encode the candidate solutions as n-dimensional vectors of decision variables $\boldsymbol{x} = \{x_1, \ldots, x_n\}$, where $\boldsymbol{U} = \{x \in \mathrm{R}^n; \max_i < x_i < min_i, i = 1, \ldots, n\}$ is the solution space.

The EA (a brief description of which is given in Appendix 2) implements the selection procedure via *fitness ranking* [**26**], and uses *generational replacement* [**26**] to renew the population at the end of every evolution cycle. A copy of the fittest individual of the population survives into the next generation (*elitism* [**26**]). As Pham and Castellani [**14**] have found, the EA uses mutation as the only genetic modification operator. The values of the EA learning parameters are shown in Table 8.

The standard formulation of the PSO algorithm (see Appendix 2) set forth by Kennedy and Eberhart [**25**] is used in this study. The velocity vector of each particle is a function of the weighted sum of three elements, the particle's momentum (v), personal experience (*pbest*), and social interaction with its neighbours (*gbest*). Appendix 2, section 2 details the velocity and position update procedures. The PSO learning parameters were set to the standard values originally recommended by Shi and Eberhart [**27**], except for the connectivity (i.e. the number of neighbours per particle) and maximum speed which were set according to the results of Pham and Castellani [**14**]. As described in equation (13), the maximum speed was adjusted via the multiplication factor u. The learning parameters of the PSO algorithm are detailed in Table 9.

Table 8 EA learning parameters

Parameter	Value
Population size	20
Children per generation	19
Mutation rate (variables)	0.8
Mutation rate (mutation width)	0.8
Initial mutation interval width a (variables)	0.1
Initial mutation interval width ρ (mutation width)	0.1
Number of evaluations per iteration	20

Table 9 PSO learning parameters

Parameter	Value
Population size	20
Connectivity (no. of neighbours)	1
Maximum velocity (u)	0.05
c_1	2.0
c_2	2.0
w_{max}	0.9
w_{min}	0.4
Number of evaluations per iteration	20

Table 10 Standard bees algorithm learning parameters

Parameter	Name	Value
Number of scout bees	N	5
Number of elite sites	e	1
Number of best sites	m	3
Recruited bees for elite sites	ne	10
Recruited bees for remaining best sites	nb	5
Initial size of neighbourhood	ngh	0.5
Limit of stagnation cycles for site abandonment	$stlim$	10
Number of evaluations per iteration	ev	26

The learning parameters of the standard bees algorithm are set according to the results of Pham and Castellani [**14**]. They are given in Table 10.

For each benchmark, the number of learning cycles was adjusted in order to obtain a number of function evaluations as close as possible to that used by the modified bees algorithm (see section 5 above). Table 11 shows the number of learning cycles and the number of evaluations for the four algorithms. For each of the three control algorithms, the table also gives the differences between the total number of function evaluations performed by each method in comparison with the modified bees algorithm. The latter figure is expressed for each function as the variation in percentage from the total number of evaluations assigned to the modified bees algorithm. As shown in Table 11, the differences in duration are modest and in all cases less than five percentage points.

7.2 Results

For each benchmark, each optimization method was run 20 times with different random initializations. At the end of every run, the fittest individual (i.e. the one that obtained the maximum objective function evaluation) of the last generation was picked as the final solution.

Table 11 Learning cycles and function evaluations

Function		Rosenbrock 2D	Rosenbrock 5D	Griewangk	Goldstein–Price	Rastrigin	Steps	Schwefel	Shekel
Dimensions		2	5	10	2	20	5	6	2
Modified bees algorithm	evaluations	503	1026	1026	1026	1026	126	2011	1026
	iterations	50	100	100	100	100	10	200	100
EA	evaluations	500	1020	1020	1020	1020	120	2020	1020
	iterations	25	51	51	51	51	6	101	51
	difference (%)	−0.60	−0.58	−0.58	−0.58	−0.58	−4.76	0.45	−0.58
PSO	evaluations	500	1020	1020	1020	1020	120	2020	1020
	iterations	25	51	51	51	51	6	101	51
	difference (%)	−0.60	−0.58	−0.58	−0.58	−0.58	−4.76	0.45	−0.58
Standard bees algorithm	evaluations	494	1040	1040	1040	1040	130	2002	1040
Iterations	19	40	40	40	40	5	77	40	
Difference (%)	−1.79	1.36	1.36	1.36	1.36	3.17	−0.45	1.36	

Table 12 Comparison of fitness results

(a) mean μ and median m values of best fitnesses from 20 runs

Problem	EA		PSO		Standard bees algorithm		Modified bees algorithm	
	mean	median	mean	median	mean	median	mean	median
Rosenbrock (2D)	0.1258	0.0182	0.0540	0.0270	0.0224	0.0078	**0.0014**	**0.0003**
Griewangk	4.0311	3.6050	1.1522	1.1361	**1.0774**	**1.0887**	**1.0774**	**1.0718**
Shekel	−0.6773	−0.8535	−0.5468	−0.3333	**−0.9787**	**−0.9905**	−0.8475	−1.0000
Schwefel	−1660.6420	−1660.6000	−1633.6380	−1577.3250	−1893.5530	−1941.4250	**−2420.8317**	**−2440.4031**
Steps (de Jong)	−14.5000	−14.5000	−11.2000	−10.5000	**−20.8000**	**−20.5000**	**−21.1500**	**−21.0000**
Rosenbrock (5D)	2.7920	2.6838	2.30	1.21	**1.2090**	**1.3728**	**1.3038**	**0.8540**
Goldstein and Price	3.0192	3.0159	4.3504	3.0002	**3.0000**	**3.0000**	**3.0011**	**3.0000**
Rastrigin	194.5484	190.8840	97.4266	95.5743	116.2691	120.6735	**83.7651**	**77.8710**
Wins	0		0		5		7	

(b) 10th–90th percentiles of best fitnesses from 20 runs

Problem	EA		PSO		Standard bees algorithm		Modified bees algorithm	
	10th percentile	90th percentile	10th percentile	90th percentile	10th percentile	90th percentile	10th percentile	90th percentile
Rosenbrock (2D)	0.0009	0.2200	0.0012	0.1252	0.0005	0.0748	0.0000	0.0034
Griewangk	2.3444	6.1970	1.0736	1.2684	0.9531	1.1446	1.0074	1.1670
Shekel	−0.9970	−0.0145	−1.0000	−0.1543	−0.9985	−0.9490	−1.0000	−0.1667
Schwefel	−1946.4550	−1288.5500	−1841.8050	−1422.7800	−2069.3700	−1655.1400	−2512.5453	−2275.2209
Steps (de Jong)	−19.0000	−9.0000	−16.5000	−7.5000	−24.5000	−17.5000	−25.0000	−16.5000
Rosenbrock (5D)	1.0008	4.7889	0.7923	3.8608	0.0226	2.2720	0.2797	3.9566
Goldstein and Price	3.0011	3.0414	3.0000	3.0018	3.0000	3.0001	3.0000	3.0034
Rastrigin	167.6495	220.3270	81.9500	118.7630	89.4015	136.8870	64.4140	110.7925

The average and the median of the fitness of the final solutions are shown for each algorithm in Table 12(a). For each function, the best result is highlighted in bold. If two or more learning procedures excelled with comparable performances, their results are all likewise highlighted. The minimization results of the four algorithms were pairwise compared using a Mann–Whitney U-test. The U-tests are run for a 0.05 (5 per cent) level of significance. Table 12(b) reports for each benchmark the 10–90 per cent percentiles of the results obtained by the four algorithms. Figures 6–13 display for each test function the median and the 10–90 per cent percentile range of the minimization results obtained by the four algorithms.

The results of the tests prove the effectiveness of the proposed method. The modified bees algorithm achieved top optimization results on all the benchmarks with the only exception of the *Shekel* function. The standard version of the bees algorithm attained top results in five out of the eight comparisons. Overall, the modified versions of the bees algorithm seemed to perform slightly better than the standard bees algorithm on the eight benchmarks considered. Regardless of the implementation chosen, the bees algorithm outperformed the other optimization procedures. The EA gave the poorest results and was never amongst the top performing methods in any of the eight tests.

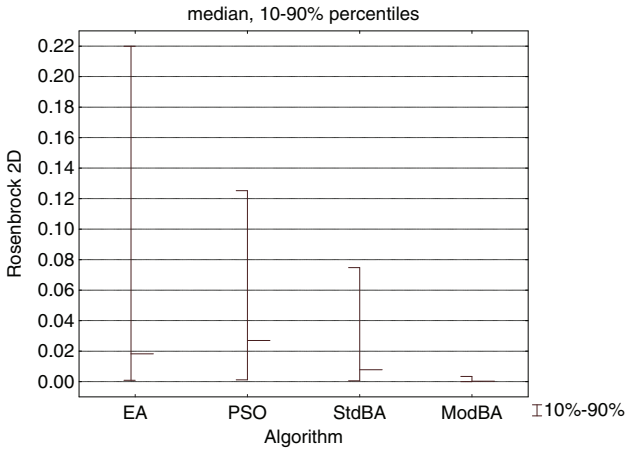

Fig. 6 Rosenbrock 2D – optimization results (median, 10 per cent and 90 per cent percentile) for algorithms tested

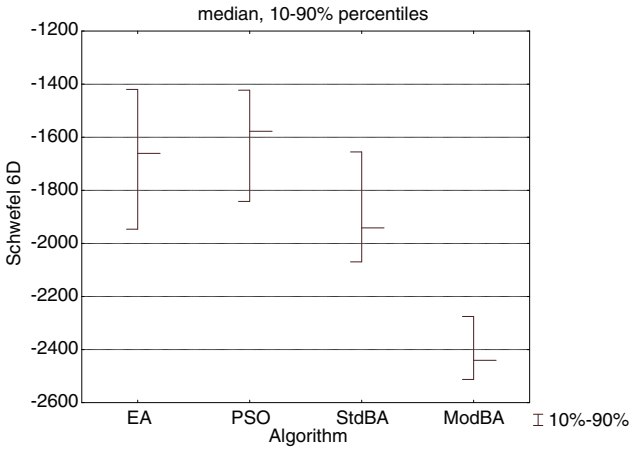

Fig. 9 Schwefel 6D – optimization results (median, 10 per cent and 90 per cent percentile) for algorithms tested

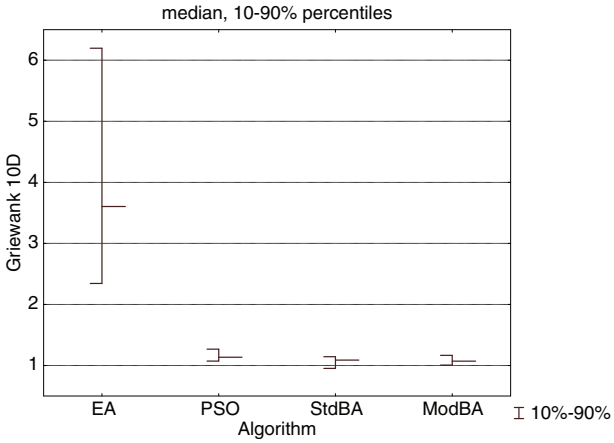

Fig. 7 Griewank 10D – optimization results (median, 10 per cent and 90 per cent percentile) for algorithms tested

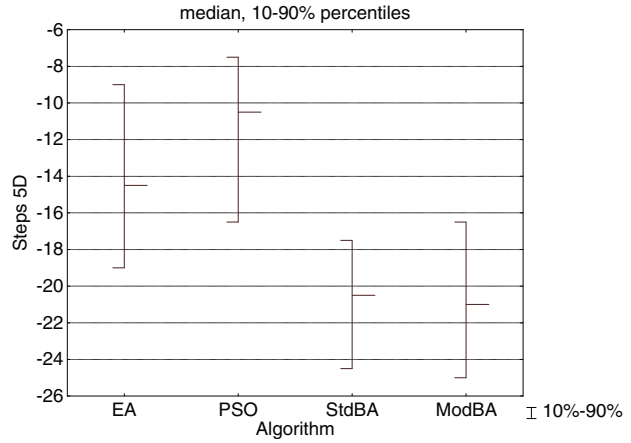

Fig. 10 Steps 5D – optimization results (median, 10 per cent and 90 per cent percentile) for algorithms tested

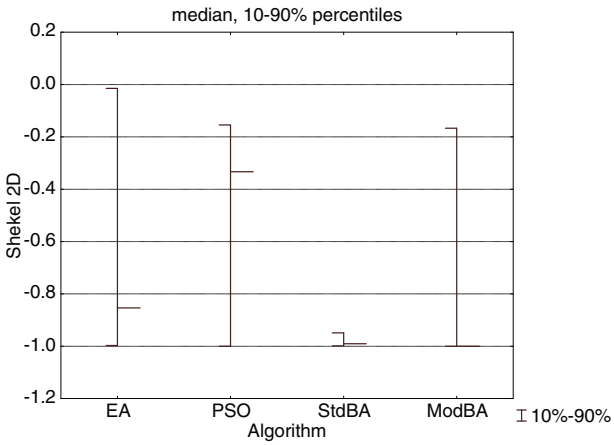

Fig. 8 Shekel 2D – optimization results (median, 10 per cent and 90 per cent percentile) for algorithms tested

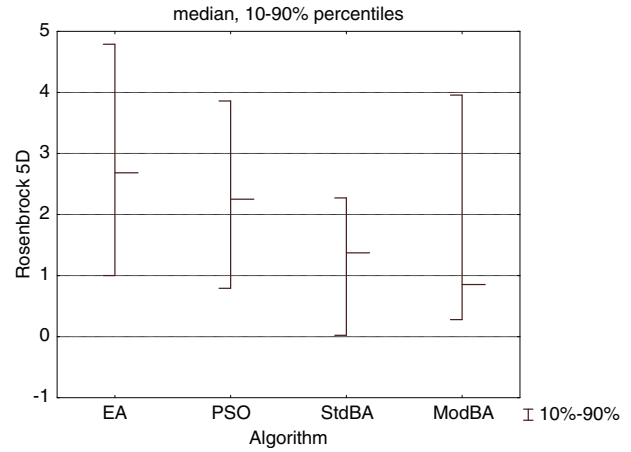

Fig. 11 Rosenbrock 5D – optimization results (median, 10 per cent and 90 per cent percentile) for algorithms tested

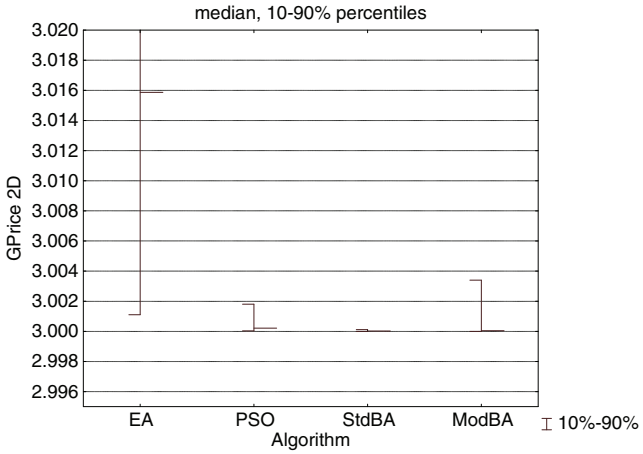

Fig. 12 Goldstein and Price 2D – optimization results (median, 10 per cent and 90 per cent percentile) for algorithms tested

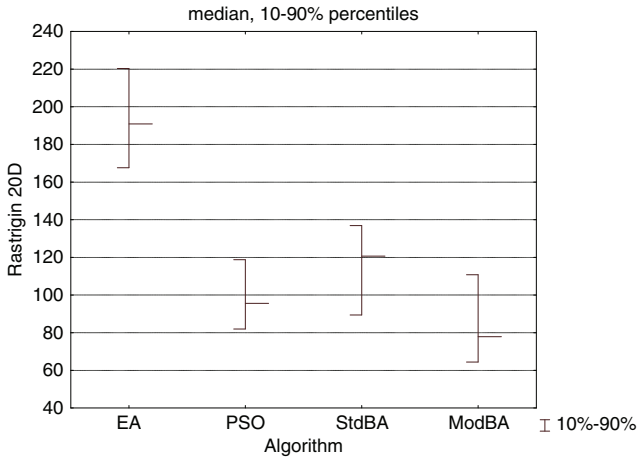

Fig. 13 Rastrigin 20D – optimization results (median, 10 per cent and 90 per cent percentile) for algorithms tested

Experimental evidence also confirmed the usefulness of the proposed parameter tuning procedure. Although designed only for a first-hand coarse setting of the learning parameters (it implicitly assumes linear relationships between parameter values and performance measures), the proposed method produced a highly effective and consistent parameter configuration for the modified bees algorithm. The advantages of this method are clear when it is compared against the uncertain trial-and-error procedure used for tuning the bees algorithm [**14**].

7.3 Test of robustness

The last test was designed to show the robustness of the proposed modified bees algorithm. In general, a tuning procedure can never cover all possible operational cases. It is, therefore, important to try the performance of the proposed algorithm against optimization problems defined outside the set of conditions used in the tuning set. In this case, the dimensionality of the Rastrigin function was increased to 30, corresponding to an increase of 50 per cent in the number of search parameters. The four algorithms were tested on the new problem keeping the same learning parameters employed for the Rastrigin 20D benchmark. Thus, the four optimization procedures were tested on a solution space much larger than the space where they were tuned.

Each algorithm was run 20 times, and the mean, median, and 10–90 per cent percentiles of the best solutions are reported in Table 13. The minimization results of the four algorithms were pairwise compared using a Mann–Whitney U-test, and the performance obtained by the modified bees algorithm and PSO was significantly better than the performance of the EA and standard bees algorithm. The U-test did not reveal any significant difference between the results obtained by PSO and the modified bees algorithm. In Table 13, the figures referring to the two best performing methods are highlighted in bold.

Table 13 also shows the growth of the mean and median minimization errors as the dimensionality of the Rastrigin function is increased from 20 to 30. In general, the performance of the EA and PSO degrades more gracefully than that of the two versions of the bees algorithm. The decrease in accuracy of the two bees algorithms is comparable, and this indicates that this feature is probably intrinsic to the bees optimization paradigm. Therefore, it can be concluded that, in terms of robustness to modification of the solution space, the modified bees algorithm is comparable to the standard version.

Table 13 Comparison of fitness results – Rastrigin 30D, 20 runs

Rastrigin 30D	mean	Increase mean (%)	median	Increase median (%)	10th percentile	90th percentile
EA	327.0062	68.08	328.9	72.30	284.1355	361.1120
PSO	**169.2338**	73.70	**172.602**	80.59	**139.1460**	**192.8390**
Standard bees algorithm	228.6955	96.69	226.8505	87.99	198.3980	262.2310
Modified bees algorithm	**159.5407**	90.46	**162.1418**	108.22	**112.7831**	**193.1309**

8 CONCLUSIONS

This paper has presented a new version of the bees algorithm. The proposed procedure is based on an improved set of local search operators, and the introduction of an effective method of handling young bees. These are the most recently generated solutions whose neighbourhood has not been yet adequately explored to give an informed evaluation on their goodness. In the modified version of the bees algorithm, young bees are protected from competition with more evolved individuals for a predefined number of optimization cycles.

Compared to the standard implementation of the bees algorithm, this new version requires the optimization of an extra set of learning parameters. Hence, a statistical tuning procedure has been formulated to determine a first coarse setting of the parameters.

Tuned using the proposed procedure, the modified bees algorithm was tested on a total of nine well-known function minimization benchmarks. The results obtained proved the effectiveness and robustness of the proposed optimization method. Compared to the standard version of the bees algorithm, PSO, and an EA, the modified bees algorithm achieved top results in nearly all the benchmarks.

Owing to the 'No Free Lunch' Theorem [**5**], the reader should not take the results of this paper as an overall evaluation of the effectiveness of the algorithms considered. Nevertheless, this study has demonstrated that the proposed modified version of the bees algorithm is capable of achieving highly competitive results on a representative set of problems. The new parameter tuning procedure has proved very effective and is much easier to implement than the complex and subjective trial-and-error methods that are commonly adopted. The only necessary manual intervention needed is some preliminary exploration to determine a reasonable range (low and high levels) for each parameter to be set.

The proposed tuning procedure was applied successfully to EAs [**13**], and in the present investigation to the modified bees algorithm. Further work is necessary to assess the effectiveness of the tuning procedure for other complex optimization algorithms.

Further work should extend the study to more benchmark functions. The effectiveness and computational effort of the proposed statistical parameter tuning procedure should be compared to that of other procedures such as the Taguchi method [**28**].

FUNDING

The research described in this article was partially supported by the EC FP6 Innovative Production Machines and Systems (I*PROMS) Network of Excellence.

REFERENCES

1 **De Jong, K. A.** *Evolutionary computation. A unified approach*, 2006 (MIT Press, Cambridge, Massachusetts).
2 **Bonabeau, E., Dorigo, M.,** and **Theraulaz, G.** *Swarm intelligence: from natural to artificial systems*, 1999 (Oxford University Press, New York).
3 **Dorigo, M., Di Caro, G.,** and **Gambardella, L. M.** Ant algorithms for discrete optimization. *Artificial Life*, 1999, **5**(2), 137–172.
4 **Pham, D. T., Ghanbarzadeh, A., Koç, E., Otri, S., Rahim, S.,** and **Zaidi, M.** The bees algorithm, a novel tool for complex optimisation problems. In Proceedings of the 2nd International Virtual Conference on Intelligent Production Machines and Systems (IPROMS 2006), (Eds D. T. Pham, E. E. Eldukhri, and A. J Soroka), 2006, pp. 454–459 (Elsevier, Oxford).
5 **Wolpert, D. H.** and **Macready, W. G.** No free lunch theorems for optimization. *IEEE Trans Evolutionary Computation*, 1997, **1**(1), 67–82.
6 **Michalewicz, M., Eben, A. E.,** and **Hinterding, R.** Parameter selection. In *Evolutionary optimisation* (Eds R. Sarker, M. Mohammadian, and X. Yao), 2002, pp. 279–306 (Kluwer, Dordrecht).
7 **De Jong, K. A.** *An analysis of the behavior of a class of genetic adaptive systems*. PhD Thesis, Computer and Communication Sciences Department, University of Michigan, 1975.
8 **Grefenstette, J.** Optimization of control parameters for genetic algorithms. *IEEE Trans Systems, Man and Cybernetics*, 1986, **16**, 122–128.
9 **Rechenberg, I.** *Evolutionsstrategie: Optimierung technischer Systeme und Prinzipien der biologischen Evolution*, 1973 (Frommann–Holzboog, Stuttgart).
10 **Davis, L.** Adapting operator probabilities in genetic algorithms. In Proceedings of the Third International Conference on *Genetic algorithms and their applications*, 1989 (Morgan Kaufman, San Mateo, California).
11 **Whitacre, J. M., Pham, Q. T.,** and **Sarker, R. A.** Use of statistical outlier detection method in adaptive evolutionary algorithms. In Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (GECCO'06), Seattle, 8–12 July 2006.
12 **Pham, Q. T.** Competitive evolution: a natural approach to operator selection. In *Progress in Evolutionary Computation, Lecture Notes in Artificial Intelligence*, 1995, **956**, 49–60 (Springer, Heidelberg).

13 **Pham, Q. T.** Using statistical analysis to tune an evolutionary algorithm for dynamic optimisation with progressive step reduction. *Computers Chemical Eng.*, 2007, **31**(11), 1475–1483.

14 **Pham, D. T.** and **Castellani, M.** The bees algorithm – modelling foraging behaviour to solve continuous optimization problems. *Proc. IMechE, Part C: J. Mechanical Engineering Science*, 2009, **223**(12), 2919–2938.

15 **Pham, D. T., Soroka, A. J., Ghanbarzadeh, A., Koç, E., Otri, S.,** and **Packianather, M.** Optimising neural networks for identification of wood defects using the bees algorithm. In Proceedings of the IEEE International Conference on *Industrial informatics*, Singapore, 2006, pp. 1346–1351 (IEEE Press, New York).

16 **Pham, D. T., Afify, A. A.,** and **Koç, E.** Manufacturing cell formation using the bees algorithm. In Proceedings of the Third Virtual International Conference on *Innovative production machines and systems*, 2–13 July 2007 (Eds D. T. Pham, E. E. Eldukhri, and A. J. Soroka), 2007, pp. 523–528 (Whittles, Dunbeath).

17 **Pham, D. T., Castellani, M.,** and **Ghanbarzadeh, A.** Preliminary design using the bees algorithm. In Proceedings of the 8th International Conference on *Laser metrology, machine tool, CMM and robotic performance (LAMDAMAP 2007)*, Cardiff, UK, June 2007, pp. 420–429.

18 **Pham, D. T., Koç, E., Lee, J. Y.,** and **Phrueksanant, J.** Using the bees algorithm to schedule jobs for a machine. In Proceedings of the 8th International Conference on *Laser metrology, machine tool, CMM and robotic performance (LAMDAMAP 2007)*, Cardiff, UK, June 2007, pp. 430–439.

19 **Pham, D. T., Darwish, A. H.,** and **Eldukhri, E. E.** Optimization of a fuzzy logic controller using the bees algorithm. *Int. J. Computer Aided Engng and Technol.*, 2009, **1**(2), 250–264.

20 **Pham, D. T.** and **Koç, E.** Design of a two-dimensional recursive filter using the bees algorithm, *Int. J. Automation and Computing*, 2010, **7**(3), 399–402.

21 **Tereshko, V.** and **Loengarov, A.** Collective decision-making in honey bee foraging dynamics. *J. Computing and Inf.* 2005, **9**(3), 1–7.

22 **Seeley, T. D.** *The Wisdom of the hive: the social physiology of honey bee colonies*, 1996 (Harvard University Press, Cambridge, Massachusetts).

23 **Adorio, E. P.** *MVF – Multivariate test functions library in C for unconstrained global optimization*, 2005, http://geocities.com/eadorio/mvf.pdf.

24 **Boslaugh, S.** and **Watters, P. A.** *Statistics in a nutshell*, 2008 (O'Reilly Media, Cambridge, Massachusetts).

25 **Kennedy, J.** and **Eberhart, R.** Particle swarm optimization. In Proceedings of the 1995 IEEE International Conference on *Neural networks*, Perth, Australia, 1995, vol. 4, pp. 1942–1948 (IEEE Press, New York).

26 **Fogel, D. B.** *Evolutionary computation: toward a new philosophy of machine intelligence*, 2nd edition, 2000 (IEEE Press, New York).

27 **Shi, Y.** and **Eberhart, R.** Parameter selection in particle swarm optimization. In Proceedings of the Seventh Annual Conference on *Evolutionary programming*, San Diego, California, 1998, pp. 591–600 (Springer-Verlag, Berlin).

28 **Roy, R. K.** *Design of experiments using the Taguchi approach: 16 steps to product and process improvement*, 2001 (John Wiley and Sons, New York).

APPENDIX 1

Notation

e	number of top-ranking sites ('elite' sites, standard bees algorithm)
e_r	e_s/N, surviving bees as a fraction of the whole population (modified bees algorithm)
e_s	number of solutions chosen to survive into the next generation based on fitness alone (modified bees algorithm)
f	linearity exponent
g	number of young bees (with fewer than M evolution steps) chosen to survive into the next generation (modified bees algorithm)
g_r	$g/(N\text{-}e_s)$, surviving young members as a fraction of all young members (modified bees algorithm)
m	number of sites selected for neighbourhood search (standard bees algorithm)
m_s	number of search steps per generation for a chosen member (modified bees algorithm)
M	number of search steps that define adulthood (modified bees algorithm)
nb	number of bees recruited to search a selected site that is not an elite site (standard bees algorithm)
ne	number of bees recruited to search a top-ranking (elite) site (standard bees algorithm)
ngh	initial size of the neighbourhood search area (standard bees algorithm)
n_0	number of search steps per generation applied to the fittest member (modified bees algorithm)
N	population size
Rank	rank of a selected solution (modified bees algorithm)
$\text{Rank}_{\text{last}}$	rank of the poorest selected solution (modified bees algorithm)

APPENDIX 2

Control algorithms

Evolutionary algorithm (EA)

The only genetic modification operator used in this EA is mutation. The genetic mutation operator modifies all the variables of a randomly picked solution according to the following equation

$$x_j^{\text{mutated}} = x_j^{\text{old}} + a \cdot random \cdot \frac{\max_i - \min_i}{2} \tag{6}$$

where *random* is a number drawn with uniform probability in the interval [–1,1], and a is the parameter encoding the maximum width of the mutation events. Parameter a is equivalent to the size of the standard bees algorithm's flower patch. For each individual, a is adaptively tuned via random mutation events as follows

$$a^{\text{mutated}} = a^{\text{old}} + \rho \cdot random \tag{7}$$

where ρ is a pre-defined learning parameter, and *random* is a number drawn with uniform probability in the interval [–1,1].

Particle swarm optimization (PSO)

In each cycle, the position of a particle $\boldsymbol{x} = \{x_1, \ldots, x_n\}$ is updated according to the following formula

$$\begin{aligned} &x_i(t+1) = x_i(t) + v_i(t+1) \\ &i = 1, \ldots, n \end{aligned} \tag{8}$$

where t is the tth PSO cycle and $\boldsymbol{v} = \{v_1, \ldots, v_n\}$ is the velocity vector of particle x. The velocity of a particle is updated as follows

$$\begin{aligned} &p_i(t) = random_1 \cdot \left[pbest(t)_i - x(t)_i\right] \\ &s_i(t) = random_2 \cdot \left[gbest(t)_i - x(t)_i\right] \\ &v_i(t+1) = w(t) \cdot v(t)_i + c_1 \cdot p_i(t) + c_2 \cdot s_i(t) \\ &i = 1, \ldots, n \end{aligned} \tag{9}$$

where c_1 and c_2 are system parameters, $random_1$ and $random_2$ are random numbers drawn with uniform probability in the interval [0,1], *pbest(t)* (personal best) is an n-dimensional vector that describes the best position (most fit) attained so far by the particle, and *gbest(t)* (global best) describes the best position attained so far by a particle in the swarm.

The weight $w(t)$ decays as follows

$$w(t) = w_{\max} - \frac{w_{\max} - w_{\max}}{T} \cdot t \tag{10}$$

where $w_{\max}$ and $w_{\min}$ are system parameters, and T is the maximum number of PSO cycles.

Every velocity vector component was clamped to the range $[-v^{\max}{}_i, v^{\max}{}_i]$, where

$$v_i^{\max} = u \cdot \frac{\max_i - \min_i}{2} \tag{11}$$