The Bees Algorithm: modelling foraging behaviour to solve continuous optimization problems

D T Pham* and **M Castellani**
Manufacturing Engineering Centre, Cardiff University, Cardiff, UK

The manuscript was received on 19 December 2008 and was accepted after revision for publication on 13 May 2009.

DOI: 10.1243/09544062JMES1494

Abstract: The Bees Algorithm models the foraging behaviour of honeybees in order to solve optimization problems. The algorithm performs a kind of exploitative neighbourhood search combined with random explorative search. This article describes the Bees Algorithm in its basic formulation, and two recently introduced procedures that increase the speed and accuracy of the search. A critical review of the related swarm intelligence literature is presented. The effectiveness of the proposed method is compared to that of three state-of-the-art biologically inspired search methods. The four algorithms were tested on a range of well-known benchmark function optimization problems of different degrees of complexity. The experimental results proved the reliability of the bees foraging metaphor. The Bees Algorithm performed optimally, or near optimally, in almost all the tests. Compared to the three control algorithms, the Bees Algorithm was highly competitive in terms of learning accuracy and speed. The experimental tests helped also to shed further light on the search mechanisms of the Bees Algorithm and the three control methods, and to highlight their differences, strengths, and weaknesses.

Keywords: optimization, swarm intelligence, artificial intelligence, honeybees

1 INTRODUCTION

New developments in engineering and technology and the increasing globalization of social and economic networks have created large and highly complex systems that are difficult to model. Many real-world engineering problems require the manipulation of a number of system variables in order to optimize a given quality parameter such as the reliability or accuracy of a process, or the value or performance of a product. Optimization will become even more important as resources diminish and there is a greater need to achieve sustainability.

Unfortunately, without an explicit input–output relationship between the system variables and the desired quality parameter, system optimization is often difficult. Several state and input variables are needed to describe the input–output relationship, and this relationship is often highly non-linear and ill behaved. Implicit discontinuities and constraints on both state and input variables are also frequent. This article addresses the generic problem of optimizing a user-defined performance index that depends on a number of variables.

Progress in the natural sciences has shed light on the mechanisms underlying the ability of many biological systems to perform hard optimization tasks, such as the natural adaptation of species to the environment, animal foraging behaviours, and the response of the human immune system to different kinds of infections.

As several studies have revealed, nature's problem-solving strategies often rely on stochastic search methods based on the exploration capability of large and decentralized ensembles of individuals. Through cooperation between individuals and group self-organization, biological societies achieve near-optimal efficiency in fundamental survival tasks such as locomotion, predator avoidance, and foraging. In particular, information sharing among group members results in collective cognitive capabilities that greatly exceed the capabilities of individuals.

The collective problem-solving capabilities of social animals are often referred to as swarm intelligence (SI) [1], and are the object of several interdisciplinary studies. On the one hand, projects originating in the

*Corresponding author: Manufacturing Engineering Centre, Cardiff University, Queen's Building, Newport Road, Cardiff CF24 3AA, UK.
email: phamdt@Cardiff.ac.uk

fields of biology and zoology explain the mechanisms of animal cooperation through precise mathematical models [**2–7**]. On the other hand, engineers and mathematicians take inspiration from the problem-solving strategies of social animals to generate new optimization algorithms [**8–12**].

This article concerns a recently developed optimization method [**11**] that mimics the foraging behaviour of honeybees. In nature, scout bees search the environment for flower patches where food is abundant. Once back at the hive, they communicate information on the direction and quality of food sources to resting mates. Through this exchange of information, the bees colony is capable of locating the richest food sources, and direct the bulk of the foragers towards these sources using a totally decentralized decision system.

The proposed Bees Algorithm randomly samples the solution space looking for areas of high performance. These areas are selected for further local search, until either a satisfactory solution is found or a predefined number of iterations have elapsed. Throughout the search, the Bees Algorithm balances random explorative and local exploitative search using a mechanism that is inspired by the foraging strategy of bees. The amount of problem domain knowledge required for the implementation of the Bees Algorithm is small, and in some cases is limited to the definition of the target performance for the implementation of the fitness evaluation function. Moreover, unlike other optimization algorithms, the Bees Algorithm does not require making assumptions about the problem domain such as the differentiability of the search space. Various versions of the Bees Algorithm were applied to different engineering problems, such as manufacturing cell formation [**13**], mechanical design [**14**], printed-circuit board (PCB) assembly optimization [**15**], machine shop scheduling [**16**], control system tuning [**17**], and pattern classifier training [**18**].

This article compares the results for the Bees Algorithm to those obtained using three well-known population-based optimization methods. Twelve benchmark function optimization problems of different characteristics and degrees of complexity are employed to test the algorithms. The experimental results are used to highlight the search mechanisms, strengths, and weaknesses of the Bees Algorithm and the three control methods. Section 2 presents the Bees Algorithm. Section 3 reviews the literature on related SI algorithms. Section 4 describes the experiments conducted. The results obtained are presented in section 5 and discussed in section 6. Section 7 concludes the article and gives suggestions for further work.

2 THE BEES ALGORITHM

The Bees Algorithm is inspired by the foraging behaviour of honeybees in nature.

2.1 Bees foraging process in nature

During the harvesting season, a bee colony employs part of its population to scout [**6**, **19**] the fields surrounding the hive. Scout bees move randomly looking for food sources. In particular, scout bees look for flower patches where nectar is abundant, easy to extract, and rich in sugar content.

When they return to the hive, scout bees deposit the nectar (or pollen) that they have collected during the search process. Those bees that found a high-quality food source (i.e. giving high energetic yield) signal the position of their discovery to resting mates through a ritual known as the ‘waggle dance’ [**20**]. The waggle dance is performed in a particular area of the hive called the ‘dance floor’, and communicates three basic pieces of information regarding the flower patch: the direction where it is located, its distance from the hive, and its quality rating [**19**, **21**]. After the waggle dance, the dancer bee goes back to the flower patch, followed by other nestmates recruited from the hive. The number of recruited bees depends on the quality rating of the patch. Flower patches that contain rich and easily available nectar or pollen sources attract the largest number of foragers [**1**, **20**]. Once a recruited forager returns to the hive, it will in turn waggle dance to direct other idle bees towards the food source. Thanks to this mechanism, the most profitable food sources attract the largest number of foragers [**22**], and thus the bee colony optimizes the efficiency of the food collection process (i.e. the amount of food collected versus the cost of harvesting it).

2.2 The basic Bees Algorithm

Without loss of generality, it will be assumed in the rest of this article that the optimization problem requires the minimization of a given measure of cost. This measure is evaluated through an objective function (fitness function) that is applied to the candidate solutions.

As mentioned above, the Bees Algorithm takes inspiration from the food foraging strategy of honeybees to search for the best solution to a given optimization problem. Each point in the search space (i.e. each potential solution) is thought of as a food source. ‘Scout bees’ randomly sample the space (i.e. solutions are randomly generated) and, via the fitness function, report the quality of the visited locations (i.e. the solutions are evaluated). The sampled solutions are ranked, and other ‘bees’ are recruited to search the fitness landscape in the neighbourhood of the highest ranking locations (i.e. other potential solutions close to the best of the generated solutions are explored). The neighbourhood of a solution is called a ‘flower patch’. The Bees Algorithm locates the most promising solutions, and selectively explores their neighbourhoods looking for the global minimum of the objective function (i.e. the solution that minimizes the given cost measure). The rest of this section describes the proposed method in detail. The flowchart of the Bees Algorithm is shown in Fig. 1, and the main algorithm parameters are given in Table 1.

2.2.1 Representation scheme

Given the space of feasible problem solutions $U = \{x \in R^n; \max_i < x_i < \min_i i = 1, \ldots, n\}$, and a fitness function $f(x) : U \to R$, each candidate solution is expressed as an n-dimensional vector of decision variables $x = \{_1, \ldots, x_n\}$.

2.2.2 Initialization

The population is fixed to *ns* scout bees, and is randomly scattered with uniform probability across the solution space. Each scout bee evaluates the visited site (i.e. solution) via the fitness function. The algorithm then enters the main loop, which is composed of four phases. The sequence of evolution cycles is interrupted when the stopping criterion is met.

2.2.3 Waggle dance

The *ns* visited sites are ranked according to the fitness information collected by the scout bees, and the *nb* locations of highest fitness (i.e. minimum measure of cost) are selected for local exploration. The local exploration is performed by other bees (foragers) that are directed to the neighbourhood of the selected sites by the scouts. For each selected site, the number of foragers is allocated deterministically as follows.

Each scout that returned from one of the *nb* best sites performs the 'waggle dance', that is it recruits nestmates for local exploration. The scout bees that visited the first *ne* elite (top-rated) sites among the best *nb* sites recruit *nre* foragers for neighbourhood search. The scouts that visited the remaining (*nb*–*ne*) sites recruit $nrb \leqslant nre$ foragers for neighbourhood search.

According to the above procedure, more bees are assigned to search the solution space in the vicinity of the *ne* points of highest fitness. The local search is thus more thorough in the neighbourhood of the elite sites, which are considered the most promising locations of the solution space. This fitness-based differential recruitment is a key operation of the Bees Algorithm, since it defines the extent of the exploitative search.

Table 1 Bees Algorithm parameters

ns	number of scout bees
ne	number of elite sites
nb	number of best sites
nre	recruited bees for elite sites
nrb	recruited bees for remaining best sites
ngh	initial size of neighbourhood
stlim	limit of stagnation cycles for site abandonment

2.2.4 Local search

For each of the *nb* selected sites, the recruited bees are randomly placed with uniform probability in a neighbourhood of the high fitness location marked by the scout bee. This neighbourhood (flower patch) is defined as an n-dimensional hyper box of sides $a_1, \ldots, a_n$ that is centred on the scout bee. For each flower patch, the fitness of the locations visited by the recruited bees is evaluated. If one of the recruited bees lands in a position of higher fitness than the scout bee,

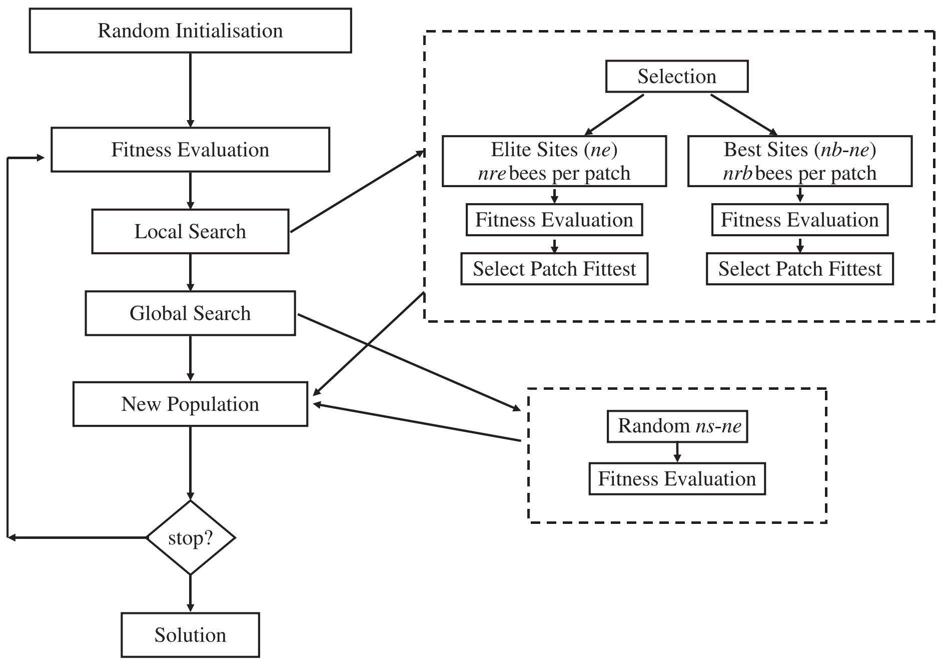

Fig. 1 Flowchart of the basic Bees Algorithm

that recruited bee is chosen as the new scout bee. At the end, only the fittest bee of each patch is retained. The fittest solution visited so-far is thus taken as a representative of the whole flower patch. This bee becomes the dancer once back at the hive. In nature, the feedback mechanism is different since all the bees involved in the foraging process perform the waggle dance.

2.2.5 Global search

In the global search phase, *ns–nb* bees are placed randomly across the fitness landscape to scout for new flower patches. Random scouting represents the exploration effort of the Bees Algorithm.

2.2.6 Population update

At the end of each iteration, the new population of the bee colony is formed out of two groups. The first group comprises the *nb* scout bees associated with the centre (the best solution) of each flower patch, and represents the results of the local exploitative search. The second group is composed of the *ns–nb* scout bees associated with a randomly generated solution, and represents the results of the global explorative search.

2.2.7 Stopping criterion

The stopping criterion depends on the problem domain, and can be either the location of a solution of fitness above a predefined threshold or the completion of a predefined number of evolution cycles.

2.3 The improved Bees Algorithm

The above procedure represents the Bees Algorithm in most basic formulation. Two new procedures were recently introduced [**15**, **23**] to increase the search accuracy of the algorithm and to avoid superfluous computations.

2.3.1 Neighbourhood shrinking

According to the first new procedure, the size $a = \{a_1, \ldots, a_n\}$ of the flower patches is initially set to a large value. For each variable a_i, it is set as follows

$$\begin{aligned} &a_i(t) = ngh(t)^*(\max_i - \min_i) \\ &ngh(0) = 1.0 \end{aligned} \tag{1}$$

where t denotes the tth iteration of the Bees Algorithm main loop. The size of a patch is kept unchanged as long as the local search procedure yields higher points of fitness. If the local search fails to bring any improvement in fitness, the size a is decreased. The updating of the neighbourhood size follows the following heuristic formula

$$ngh(t+1) = 0.8^* ngh(t) \tag{2}$$

Thus, following this strategy, the local search is initially defined over a large neighbourhood, and has a largely explorative character. As the algorithm progresses, a more detailed search is needed to refine the current local optimum. Hence, the search is made increasingly exploitative, and the area around the optimum is searched more thoroughly. The neighbourhood shrinking method bears some similarities with the simulated annealing [**24**] procedure.

2.3.2 Site abandonment

The second new procedure is applied when no fitness improvement is gained within a flower patch from applying the neighbourhood shrinking method. Each time the local search procedure fails to yield a solution of lower cost, the size of the local search is reduced. After a predefined number (*stlim*) of consecutive stagnation cycles, the local search procedure is assumed to have reached the top of the local fitness peak, and no further progress is possible. Consequently, the exploration of the patch is terminated and a new random solution is generated. If the site being abandoned corresponds to the best-so-far fitness value, the location of the peak is recorded. If no other flower patch will produce a better fitness measure during the remaining of the search, the recorded best fitness location is taken as the final solution.

3 RELATED WORK

The SI approach to optimization relies on the collective exploration capabilities of large groups of elementary agents. The agents are initially scattered across the search space, and through mutual information exchange they guide each other towards the most promising region(s) of the fitness landscape. This convergence process is realized through iterative steps of agent movements and information exchange. A random component is usually introduced into the agents' behaviour in order to foster the exploration of the fitness landscape. The search process is terminated when the optimal solution is found, or when a predefined number of iterations have elapsed.

3.1 Evolutionary algorithms

Although not regarded as true SI approaches by many practitioners, evolutionary algorithms (EAs) [**25–28**] share some features with SI algorithms.

EAs were the first search methods to employ a population of agents. In EAs, the population is driven towards the optimal point(s) of the search space by means of stochastic search operators inspired by the biological mechanisms of natural selection, mutation, and recombination.

The selection procedure picks the individuals that will generate the new population. It can be deterministic (only the best reproduce) [**25**, **27**] or stochastic (the best are more likely to reproduce) [**26–28**]. The crossover operator [**27**, **28**] creates new individuals by randomly mixing the features of couples of individuals. This recombination mechanism determines the exchange of information (social interaction) between candidate solutions. Random mutations (i.e. small perturbations) [**27**, **28**] of the parameters characterizing a solution are used to prevent premature convergence, and to encourage the exploration of the search space. The new individuals replace old individuals in the population at the end of each evolutionary cycle.

Traditional EAs differ from classical SI methods for the degree of decentralization of the search. In standard EAs, the parent selection and population replacement procedures are often centralized at the population level. For this reason, they are usually not regarded as proper SI algorithms. However, as the following of this section will show, several SI algorithms are characterized by similarly centralized population selection and update procedures. Also, many SI approaches borrowed their operators from EAs, or included phases of evolutionary search in their procedures. At the same time, some EAs use multi-population approaches [**28**] or population crowding methods [**29**] that make them fairly decentralized procedures.

Among the various EA paradigms, the Bees Algorithm bears the most similarities with $(\mu + \lambda)$-evolutionary strategies [**30**]. Like this class of EAs, the Bees Algorithm uses a deterministic selection procedure to generate the seeds of the next population, and creates a number of mutated solutions in the neighbourhood of each seed. Like some types of evolutionary strategies, the Bees Algorithm relies solely on the selection and mutation procedures to direct the search process. The lack of a crossover operator is characteristic also of evolutionary programming [**26**].

Differently from evolutionary strategies, the fitness-based recruitment mechanism allows the Bees Algorithm to sample more thoroughly the neighbourhood of the best solutions. Also, the population update procedure of the Bees Algorithm is defined locally, that is, for each patch. In $(\mu + \lambda)$-evolutionary strategies (and in most EAs), the whole parent population competes with the offspring for survival, or it is totally replaced by the offspring. The locality of the population update procedure entails a truly parallel distributed search, supports population diversity, and fosters the exploration of the solution space.

Finally, the exploration and exploitation capabilities of EAs are defined by the balance between the selection pressure, the rates of occurrence of the crossover and mutation procedures, and the disruptiveness of such procedures. Unfortunately, the interactions between the selection pressure and the search operators are not straightforward. This makes the choice and the tuning of the EA operators sometimes difficult. In the Bees Algorithm, exploration and exploitation are fully decoupled, and can be explicitly varied through the learning parameters. This feature makes it easy to tune the Bees Algorithm to particular classes of problems. For an example, parameters that encourage exploration (*ns* and *nb*) can be increased for strongly multi-modal problems, while parameters that encourage exploitation (*ne* and *nre*) can be increased for 'smooth' problems.

3.2 Particle swarm optimization and ant colony optimization

Particle swarm optimization (PSO) [**8**] and ant colony optimization (ACO) [**9**] are the first SI algorithms published in the literature. To date, they are also the most popular ones.

ACO was originally designed for combinatorial optimization problems. A number of authors have proposed algorithms that extended the ACO metaphor to continuous optimization domains. These algorithms include CACO [**31**], CIAC [**32**], and ACO_R [**33**].

Both PSO and ACO model social interactions through a medium that affects the behaviour of the individual agents. Although PSO uses attraction forces to direct the swarm towards the global fitness optimum, ACO uses stigmergy, that is the laying of a marker that is reinforced at a higher rate on the most profitable trails [**9**]. In both cases, the social mechanism is described through a set of mathematical equations that define respectively the individual particle dynamics (PSO) and the marker deposition/evaporation rate (ACO).

Even though the equations characterizing the interactions between the members of PSO and ACO populations are relatively simple, the overall swarm dynamics are difficult to understand fully. In particular, again there is no clear separation between random explorative search and opportunistic exploitative search. The explorative/exploitative nature of PSO and ACO can be adjusted, respectively, by changing the scope of the global attraction force and the evaporation rate of the marker. However, in both cases an increased exploration effort comes at the expense of a slower convergence towards the optimal solution, and vice versa.

In summary, there are three main differences between the Bees Algorithm and PSO and ACO. The first difference is that the Bees Algorithm does not rely on an interaction medium for the exchange of information. Bees swarm towards high-fitness regions based directly on the sampling results of the search space. This allows the user to set explicitly the sampling rate (that is the number of foragers) around interesting areas of the fitness landscape. The second difference is that in the Bees Algorithm there is a clear differentiation between explorative search (scout bees) and exploitative search (recruited foragers). As mentioned before, this feature facilitates the tuning of the Bees Algorithm. In the case of PSO and ACO, exploration and exploitation are competing needs. The third difference is that in PSO and ACO there is normally no in-built mechanism to overcome states where the population is stuck at a local fitness optimum. In the Bees Algorithm, random global search and local site abandonment allow the population to focus on new promising areas of the search space once a fitness peak has been fully climbed. This feature is present in more complex PSO implementations such as the multi-swarm PSO proposed by Blackwell and Branke [**34**].

3.3 Algorithms modelling the foraging behaviour of honeybees

A number of optimization algorithms inspired by the foraging behaviour of honeybees emerged recently [**35–37**]. These algorithms use standard evolutionary [**35**, **36**] or random [**37**] explorative search to locate promising areas of the fitness landscape. Exploitative search is implemented through parallel EAs [**38**] that are initialized at the centre of the most promising regions [**35**, **36**], or sub-populations of bees (samplers) that are guided according to an evolutionary, PSO, or ACO algorithm [**37**]. Phases of explorative and exploitative search are repeated until a satisfactory solution is found. In the first two cases [**35**, **36**], the exploration and exploitation phases take place sequentially. In the third case [**37**] they are run in parallel, and the results of exploration and exploitation are compared after a predefined number of cycles. In the Bees Algorithm, the results of exploitative and explorative search are compared at the end of each cycle, and the sampling rates are re-assigned accordingly. The two phases can be considered to happen simultaneously, and this is believed to allow faster adaptation of the sampling rates to new information gained on the fitness lanscape.

Apart from hybrid approaches complementing evolutionary optimization with search strategies borrowed from honeybees, other authors took inspiration directly from the foraging behaviour of bees to build new optimization paradigms. For applications in the area of continuous function optimization, Karaboga and Basturk [**39**] proposed the artificial bee colony (ABC) algorithm. Although the two algorithms were developed independently, there are strong analogies between ABC and the Bees Algorithm. The two optimization methods can be described using the same flowchart, and the site abandonment procedure is used also in the ABC algorithm. Differently from the Bees Algorithm, ABC uses the roulette wheel selection method [**28**] to simulate the recruiting of foragers through the waggle dance. The main difference between the two algorithms is in the implementation of the local search procedure. ABC generates foragers by a floating point crossover operation [**30**] between the dancer bee and a second bee randomly selected from the population. This operator calculates the components of the new forager as a weighted average of the parents' components. The weight of each parent's component is randomly determined. Since the second bee is randomly selected from the whole population, the crossover operation may generate a forager bee that is relatively far from the dancer bee. In particular, the forager bee may be placed outside the fitness peak that it is meant to exploit. For this reason, the effectiveness of the exploitative search may be reduced, and the extent of the neighbourhood search is more difficult to control.

3.4 Other SI approaches

Several other SI algorithms for continuous optimization problems were inspired by the collective behaviour of animal populations. Honey Bees Mating Optimization [**40**] simulates the mating behaviour of honeybees, the Termite Routing Algorithm [**41**] is modelled on the nest building behaviour of termites, the Artificial Fish Swarm Algorithm [**12**, **42**] mimics some basic fish behaviours such as moving, foraging, and schooling, the Bacterial Foraging Optimization Algorithm [**10**] is inspired by the chemotactic movement and exchange of information of *Escherichia coli* bacteria.

These methods share with the Bees Algorithm the population-based stochastic search approach, and mimic different kinds of interactions between individuals. However, their structure is less directly comparable to the Bees Algorithm, and their application seems to appear less frequently in the literature.

4 TESTS

The performance of the Bees Algorithm was evaluated on a set of 12 continuous function minimization benchmarks. For each problem, the results obtained using the proposed method were compared with the results given by three control algorithms, namely EA, PSO, and ABC.

It should be stressed that the purpose of the tests was neither to prove the superiority of one algorithm over the others, nor to provide an exhaustive comparison with the state-of-the-art in the literature. The aim of the study was to characterize the behaviour of the Bees Algorithm, highlight its strengths and weaknesses, and highlight the differences between the performance and reliability of the proposed method to those of the three competing algorithms.

The function minimization problems represent a varied set of learning scenarios that were chosen among widely used benchmarks in the literature [**43**]. Although the results are used to compare the performance of the four optimization procedures, the *no free lunch theorem* [**44**] precludes any claims regarding the superiority of one algorithm over another.

PSO and EAs are among the best understood and better characterized population-based metaheuristics in the literature. For this reason, they represent ideal terms of comparison to analyse the behaviour of the Bees Algorithm and the kind of challenges that the benchmarks pose. ABC was added to the study to show the differences between the Bees Algorithm and a similarly inspired method, and to further understand the strengths and the mechanisms of the bees foraging metaphor.

For each algorithm, different operators and settings of the learning parameters were tested, in order to study their effect on the search procedures.

4.1 Function minimization benchmarks

Table 2 shows the equations of the 12 continuous function minimization benchmarks. For each function, the equation is given together with the range of the variables and the global minimum.

The *hypersphere* and *Martin and Gaddy* benchmarks are fairly simple unimodal functions. The *Rosenbrock* benchmark is unimodal, the minimum lies at the bottom of a long, narrow, parabolic-shaped valley. To find the valley is trivial, however to locate the minimum is difficult. The *Easom* benchmark is characterized by a large flat surface with a narrow and steep 'hole'. The *Ackley, Griewank,* and *Rastrigin* functions have an overall unimodal behaviour, with a rough multi-modal surface created by a cosinusoidal 'noise' component. The remaining six benchmarks are true multi-modal functions. The *Goldstein and Price* benchmark represents a fairly easy optimization task, while the other multi-modal functions map complex search spaces.

4.2 General settings

For all the control algorithms, the solutions were encoded using the same representation scheme used for the Bees Algorithm, and described in section 2.2.1. All the control algorithms employed the same initialization procedure described in section 2.2.2. The fitness value $F(x)$ of a solution $x = \{x_1, \ldots, x_n\}$ is evaluated as follows

$$F(x) = f(x) - \bar{f} \tag{3}$$

where $f(x)$ is the value that is taken on x by the benchmark f, and $\bar{f}$ is the global minimum of f.

All the algorithms were run until either the minimum of the function was approximated to better than 0.001, or a maximum number of learning cycles (5000) had elapsed. The parameters of all the algorithms were set in order to have 100 function evaluations per learning cycle.

4.3 Control algorithm I: evolutionary algorithm

The EA encodings $x = \{x_1, \ldots, x_n, a\}$ have one extra component a that represents the interval width of the mutation events. The EA uses the *fitness ranking* selection procedure [**28**]. At the end of each evolution cycle, the whole population is renewed according to the *generational replacement* [**28**] scheme. *Elitism* [**28**] is applied, that is, a copy of the fittest individual of the parent population survives into the new population. Figure 2 shows the flowchart of the EA algorithm.

The EA manipulates the encodings of the solutions using the two standard genetic operators of mutation and crossover. The mutation operator modifies all the variables of a randomly picked solution. The modification is sampled with uniform probability within $[-a, a]$. An EA mutation event is equivalent to the generation of a recruited bee in a flower patch of size a and centred on the mutant solution. For each individual, parameter a is adaptively tuned via random mutation events.

Four different kinds of crossover strategies were tested. In the first case, no crossover was applied, and the EA relied only on genetic mutations to search for the optimal solution. The lack of a recombination operator, the online adaptation of the mutation range, and the stochastic selection procedure make this approach close to evolutionary programming [**26**]. When no crossover is used, the interactions among individuals are restricted. The only exchange of information comes in this case from the selection procedure, which focuses the search towards the most promising areas of the solution space. The solutions act thus more independently and population diversity is encouraged.

The second recombination strategy randomly picks corresponding subsets of variables of two candidate solutions (*parents*), and swaps them. Two new individuals (*children*) with mixed features are generated. This method will henceforth be called *binary* crossover, and corresponds to the standard two-point recombination operator [**28**]. The action of binary crossover in a two-dimensional (2D) search space is shown in Fig. 3. This method helps the population to move in 'good directions' along the coordinate axes.

The third and fourth recombination strategies generate a new candidate solution by making a weighted average of the parameters of two individuals. *Interpolation* crossover generates a child inside a hyperbox having as extremes the two solutions (see Fig. 3). This strategy is more suitable for the exploitation

Table 2 Benchmark functions

Function	Equation	Minimum
Ackley (10D)	$f(\vec{x}) = 20 - 20\,\mathrm{e}^{-0.2\sqrt{(1/10)\sum_{i=1}^{10} x_i^2}} - \mathrm{e}^{(1/10)\sum_{i=1}^{10}\cos(2\pi x_i)} + \mathrm{e}, \quad -32 < x_i < 32$	$\vec{x} = (\vec{0})$ $f(\vec{x}) = 0$
Easom (2D)	$f(x_1, x_2) = -\cos(x_1)\cos(x_2)\,\mathrm{e}^{[(x_1-\pi)^2-(x_2-\pi)^2]}, \quad -100 < x_i < 100$	$\vec{x} = (\pi, \pi)$ $f(\vec{x}) = -1$
Goldstein and Price (2D)	$A(x_1, x_2) = 1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)$ $B(x_1, x_2) = 30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)$ $f(x_1, x_2) = AB, \quad -2 < x_i < 2$	$\vec{x} = (0, -1)$ $f(\vec{x}) = 3$
Griewank (10D)	$f(\vec{x}) = \frac{1}{4000}\sum_{i=0}^{10}(x_i - 100)^2 \prod_{i=0}^{i=10}\cos\left(\frac{x_i - 100}{\sqrt{i+1}}\right) + 1, \quad -600 < x_i < 600$	$\vec{x} = (\overrightarrow{100})$ $f(\vec{x}) = 0$
Hypersphere (10D)	$f(\vec{x}) = \sum_{i=0}^{10} x_i^2, \quad -100 < x_i < 100$	$\vec{x} = (\vec{0})$ $f(\vec{x}) = 0$
Langermann[a] (10D)	$f(\vec{x}) = \sum_{i=0}^{i=5} c_i \left[\mathrm{e}^{-(1/\pi)\sum_{j=0}^{j=10}(x_j - a_{ij})^2}\right]\left[\cos\left(\pi\sum_{j=0}^{j=10}(x_j - a_{ij})^2\right)\right], \quad 0 < x_i < 10$	$f(\vec{x}) = -0.705\,52$
Martin and Gaddy (2D)	$f(x_1, x_2) = (x_1 - x_2)^2 + \left[\frac{(x_1 + x_2 - 10)}{3}\right]^2, \quad -20 < x_i < 20$	$\vec{x} = (5, 5)$ $f(\vec{x}) = 0$
Rastrigin (10D)	$f(\vec{x}) = \sum_{i=1}^{i=10}\left[(x_i)^2 - 10\cos(2\pi x_i) + 10\right], \quad -5.12 < x_i < 5.12$	$\vec{x} = (\vec{0})$ $f(\vec{x}) = 0$
Rosenbrock (10D)	$f(\vec{x}) = \sum_{i=1}^{10} 100\left(x_{i+1} - x_i^2\right)^2 + (1 - x_i)^2, \quad -50 < x_i < 50$	$\vec{x} = (\vec{1})$ $f(\vec{x}) = 0$
Schaffer (2D)	$f(x_1, x_2) = 0.5 + \frac{\left(\sin\sqrt{x_1^2 + x_2^2}\right)^2 - 0.5}{\left[1.0 + 0.001\left(x_1^2 + x_2^2\right)\right]^2}, \quad -100 < x_i < 100$	$\vec{x} = (0, 0)$ $f(\vec{x}) = 0$
Schwefel (2D)	$f(x_1, x_2) = -x_1\sin\left(\sqrt{\lvert x_1\rvert}\right) - x_2\sin\left(\sqrt{\lvert x_2\rvert}\right), \quad -500 < x_i < 500\mathrm{C}$	$\vec{x} = (420.97, 420.97)$ $f(\vec{x}) = -837.97$
Shekel* (10D)	$f(\vec{x}) = \sum_{i=0}^{i=30}\frac{1}{\sum_{j=0}^{j=10}\left[(x_j - a_{ij})^2 + c_j\right]}, \quad 0 < x_i < 10$	$f(\vec{x}) = -10.2021$

*Coefficients A_{ij} and c_j as defined in Adorio [**43**].

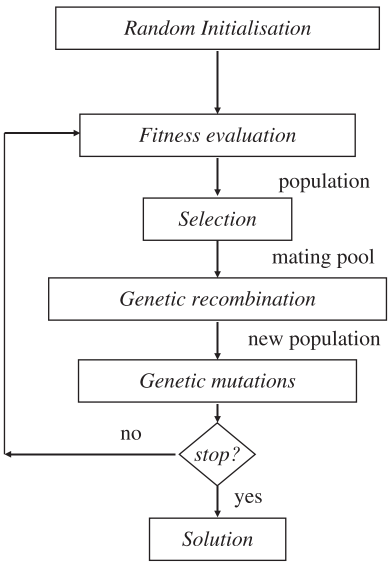

Fig. 2 Flowchart of the EA

of promising areas than the exploration of the fitness landscape. Interpolation crossover tends also to reduce population diversity quickly. *Extrapolation* crossover generates a new solution inside a rectangular box centred in the first parent and having as extreme the other parent (see Fig. 3). This is the most disruptive crossover strategy used in this study.

The learning parameters of the EA are given in Table 3. The equations of the genetic modification operators are given in the Appendix.

4.4 Control algorithm II: particle swarm optimization

The standard PSO procedure formulated by Kennedy and Eberhart [**8**] was used. The flowchart of the algorithm is given in Fig. 4.

The velocity vector is made of three components that model, respectively, the particle's inertia v, the particle's personal experience *pbest*, and the social

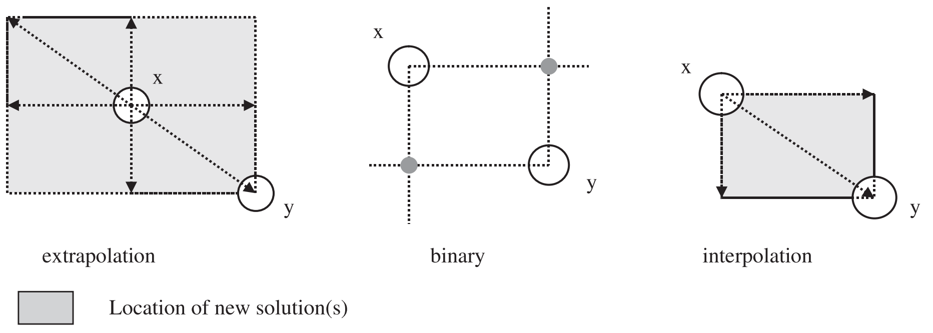

Fig. 3 Crossover operators

Table 3 EA learning parameters

Parameter	Crossover	No crossover
Population size	100	
Evolution cycles (max number)	5000	
Children per generation	99	
Crossover rate	1.0	0.0
Mutation rate (variables)	0.05	0.8
Mutation rate (mutation width)	0.05	0.8
Initial mutation interval width a (variables)	0.1	
Initial mutation interval width ρ (mutation width)	0.1	

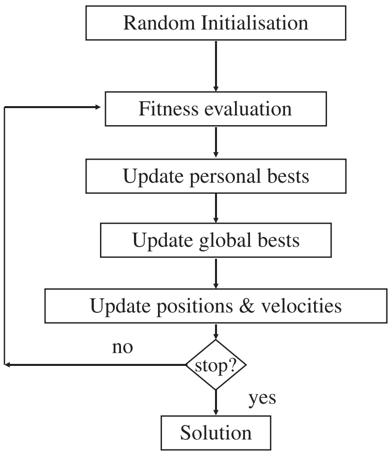

Fig. 4 Flowchart of the PSO algorithm

Table 4 PSO learning parameters: (a) fixed learning parameters and (b) matrix of velocity and connectivity combinations

Parameter	Value
(a)	
Population size	100
PSO cycles (max number) T	5000
Connectivity	See Table 4(b)
Maximum velocity	See Table 4(b)
C_1	2.0
C_2	2.0
$w_{\max}$	0.9
$w_{\min}$	0.4

		Max particle velocity u			
(b)		0.005	0.01	0.05	0.1
Connectivity (number of neighbours)	2				
	10				
	20				
	100				

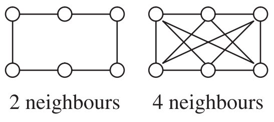

Fig. 5 PSO connectivity

interaction among particles *gbest*. The equations of the velocity update procedure are given in the Appendix. The learning parameters were set according to the standard values recommended in the literature [**45**]. They are given in Table 4(a).

Differently connected populations can be designed. Figure 5 shows the social networks resulting from connectivity models of respectively two and four neighbours. By varying the connectivity and $v^{\max}$ of the particles, different search strategies can be obtained. High connectivity and low $v^{\max}$ result in fast-converging population-driven exploitative search strategies. Low connectivity and high $v^{\max}$ generate more loosely connected particle networks that are more suitable for the exploration of the search space. In the proposed tests, 16 different combinations of $v^{\max}$ (obtained through u) and connectivity were tested. Table 4(b) gives the 4×4 matrix describing the 16 combinations of parameters. A connectivity of 100 neighbours means that each particle is connected to all the other particles.

4.5 Control algorithm III: ABC

The standard version of the ABC algorithm formulated by Karaboga and Basturk [**39**] was used. In ABC terminology, the 'employed' and 'onlooker' bees correspond, respectively, to the 'scouts' and 'foragers' of the Bees Algorithm.

The flowchart of the ABC algorithm is given in Fig. 6. For each flower patch, ABC uses proportional selection to recruit the pool of onlooker bees (waggle dance).

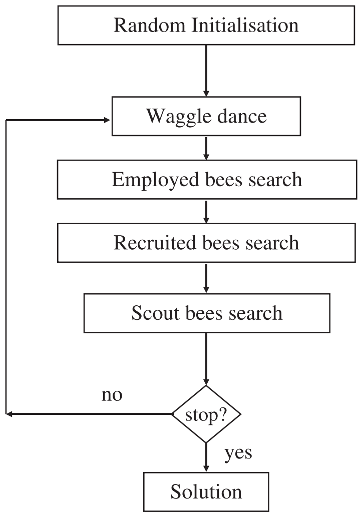

Fig. 6 Flowchart of the ABC algorithm

In the local search phase, a new solution is created for each employed bee through extrapolation crossover (employed bees search). That is, the employed bee is mated with a randomly picked partner, and a new solution is generated. If the new solution lies in a position of higher fitness than the employed bee, it replaces the employed bee as the representative of the flower patch. For each patch, the same procedure is repeated for all the onlooker bees (onlooker bees search).

Likewise the Bees Algorithm, ABC uses site abandonment (section 2.3.2), i.e. a flower patch is abandoned and a new solution is randomly created if local search fails to produce any fitness gain within a prefixed number of learning cycles. In the case of ABC, the stagnation limit *stlim* is calculated according to the following heuristic formula

$$stlim = n_e \cdot n \tag{4}$$

where n_e is the number of employed bees.

In ABC, the population is customarily split equally into 50 per cent explorers and 50 per cent onlookers. One random scout bee is generated every learning cycle for exploration of the search space (scout bees search). The learning parameters of the ABC algorithm are given in Table 5. It is worth noticing that the standard version of ABC does not need any optimization, since all learning parameters are heuristically pre-set.

Table 5 ABC learning parameters

Parameter	Crossover	No crossover
Population size		100
ABC cycles (max number)		5000
Employed bees, n_e		50
Onlooker bees, n_e		49
Random scouts		1
Stagnation limit for site abandonment *stlim*		$50 \times n$

In every learning cycle, the ABC algorithm tests 50 employed bees, 49 onlooker bees, and 1 random bee for a total of 100 function evaluations. According to equation (4), the stagnation limit for site abandonment corresponds to $50 \times n$ learning cycles. This figure corresponds to 100 cycles for the 2D benchmark functions, and 500 cycles for the 10-dimensional benchmarks. Hence, each flower patch is sampled hundreds of times before being abandoned. Considering that proportional selection tends to focus the search on the flower patches of highest fitness, it is clear that the ABC algorithm is characterized by a very exploitative search strategy. That is, the algorithm tends to focus on a limited number of selected areas, and samples them thoroughly. Such concentrated search comes at the expense of the exploration of the remaining of the fitness landscape.

4.6 The Bees Algorithm

The exploitative/explorative nature of the Bees Algorithm can be tuned by varying the learning parameters (Table 1). In the proposed tests, 12 different combinations of parameters were tested (Table 6).

In order to achieve precisely 100 function evaluations per learning cycle, the least fit of the selected *nb* bees recruits (*nrb* − 1) foragers.

5 EXPERIMENTAL RESULTS

All the algorithms were run 50 times for each parameter setting on each benchmark problem. For each of the 50 trials, the optimization procedure was run until either it located a solution ξ of fitness (3) $F(\xi) < 0.001$, or 5000 learning cycles had elapsed. The final accuracy result is the following

$$E = \begin{cases} 0 \Leftarrow F(x_f) \leqslant 0.001 \\ F(x_f) \Leftarrow \text{else} \end{cases} \tag{5}$$

Table 6 Combinations of Bees Algorithm parameters tested

No.	*stlim*	*ne*	*nre*	*nb*	*nrb*
1	10	2	30	4	10
2	5	2	30	4	10
3	5	1	40	3	20
4	5	2	20	6	10
5	5	1	40	6	10
6	10	1	30	7	10
7	5	1	30	7	10
8	10	1	30	14	5
9	5	1	30	14	5
10	5	1	20	8	10
11	5	1	20	16	5
12	5	1	10	18	5

where x_f is the final solution generated by the algorithm. For each trial, the learning speed S was fixed to the optimization cycle when ξ was located, or 5000.

For each benchmark, the average accuracy E (μ_E) and speed S (μ_S) over the 50 runs were computed for each parameter configuration of the four algorithms.

The learning results of the four optimization methods were analysed according to two criteria, the absolute performance on each benchmark, and the overall robustness. The performance of the four algorithms was evaluated taking for each function the parameter setting that produced the best results. The parameter setting that gave the most consistent performance across all the benchmarks was found for each algorithm. The robustness of the four optimization methods was then compared, considering for each algorithm the results given by the most consistent parameter configuration. The performance of an algorithm concerns its ability to produce a solution that satisfies the given quality criteria. The robustness of an algorithm is its reliability, in other words, its capability of producing consistently good solutions. In order to compare the performance of two algorithms (parameter settings), the following method was used.

5.1 Comparison criterion

Given a minimization task τ, it is assumed that an algorithm (or algorithm configuration) A_1 solves τ with average accuracy μ_{E1} and average speed μ_{S1}, and another algorithm (or algorithm configuration) A_2 solves τ with average accuracy μ_{E2} and average speed μ_{S2}. A_1 is said to perform better on τ than A_2 if $\mu_{E1} < \mu_{E2}$, and the difference between μ_{E1} and μ_{E2} is statistically significant.

If the difference between the average accuracy of A_1 and A_2 is not statistically significant, the average speed of A_1 and A_2 is compared. If there is no statistically significant difference between μ_{S1} and μ_{S2}, the performance of A_1 and A_2 is said to be the same. Otherwise, the faster algorithm is said to perform better on τ.

The comparison method follows therefore a hierarchical criterion, where accuracy has priority over speed.

The statistical significance of the difference between two average accuracies (speeds) was evaluated through student's t-tests. The t-tests were run for a 0.05 (5 per cent) level of significance.

5.2 Comparison of absolute performance

Tables 7 to 10 present the optimization results obtained by the four algorithms on the 12 test functions. The dotted lines divide the benchmarks into functions that are unimodal, functions that have an overall unimodal behaviour and local peaks created by a cosinusoidal component, and functions that are truly multi-modal. For each benchmark, the tables detail the results given by the best performing algorithm configuration.

Table 7 gives the results obtained by PSO. As expected, minimization tasks that required a fast convergence along a relatively uncomplicated slope (*Easom*, *Martin and Gaddy*, and *Goldstein and Price*) were usually solved more efficiently by fast-converging fully connected particle networks. The only exception was the *hypersphere* function, where the most sparsely connected population was the fastest to locate the minimum. In this case, the reduced cross talk between individuals might have helped the individual particles to stabilize the momentum along the radial gradient direction.

With the exception of the *Schaffer* benchmark, sparsely connected populations excelled in the most complex tasks. The ten neighbours configuration performed best on functions where an overall unimodal behaviour helped to locate the position of the minimum (*Ackley*, *Griewank*, and *Rastrigin*). The two neighbours configuration (minimal connectivity) performed best on three out of the four most complex multi-modal surfaces (*Langermann, Schwefel*, and *Shekel*), the *Ackley* benchmark, and the challenging *Rosenbrock* function.

The results presented in Table 7 confirm that 'looser' social models favour the exploration of the search space, and thus are more suitable for complex multi-modal tasks. This result was confirmed by the observation that the most successful problem solvers of the four most complex multi-modal surfaces used high maximum particle speeds.

Table 8 reports the results obtained by the EA. In three-quarter of the cases, the most successful configuration was the one not using genetic recombination. Crossover seemed to be beneficial mostly when searching multi-modal surfaces. In this case, the exchange of information might have helped to escape local minima, and enhanced the exploration capability of the EA. Unfortunately, genetic recombinations usually imply longer 'jumps' in the solution space than genetic mutations. For this reason, in many benchmarks the action of crossover might have proved too disruptive for the population convergence process.

Table 9 reports the results obtained by the standard configuration of the ABC algorithm.

Table 10 details the results achieved by the Bees Algorithm. In this case, the large number of variable parameters makes it more difficult to identify overall patterns. In general, it can be observed that the best configuration for all unimodal functions (and the easy *Goldstein and Price* benchmark) featured the largest stagnation limit. This result is due to the fact that, when the optimization surface is fairly regular, it is a good strategy to insist on searching in the proximity of the most promising solutions.

Table 7 PSO – best performances

Benchmark	Connectivity (no. of neighbours)	Max particle velocity u	μ_E	σ_E	μ_S	σ_S
Hypersphere (10D)	2	0.005	0.0000	0.0000	1717.54	77.32
Martin and Gaddy (2D)	100	0.05	0.0000	0.0000	17.78	6.12
Easom (2D)	100	0.005	0.0000	0.0000	161.24	159.42
Rosenbrock (10D)	2	0.1	0.5998	1.0436	4929.12	293.81
Ackley (10D)	2	0.01	0.0000	0.0000	2365.62	91.19
Griewank (10D)	10	0.005	0.0008	0.0026	2904.66	745.01
Rastrigin (10D)	10	0.1	0.1990	0.4924	4124.40	678.14
Goldstein and Price (2D)	100	0.01	0.0000	0.0000	32.62	8.22
Langermann (10D)	2	0.1	0.0000	0.0000	178.46	182.71
Schaffer (2D)	100	0.05	0.0000	0.0000	280.72	217.17
Schwefel (2D)	2	0.05	4.7376	23.4448	845.72	903.73
Shekel (10D)	2	0.1	7.1194	3.3744	4537.54	998.06

Table 8 EA – best performances

Benchmark	Crossover type	μ_E	σ_E	μ_S	σ_S
Hypersphere (10D)	None	0.0000	0.0000	363.76	27.36
Martin and Gaddy (2D)	Extrapolation	0.0000	0.0000	15.12	3.85
Easom (2D)	None	0.0000	0.0000	364.40	281.21
Rosenbrock (10D)	None	61.5213	132.6307	5000.00	0.00
Ackley (10D)	None	0.0000	0.0000	503.44	39.49
Griewank (10D)	None	0.0210	0.0130	4907.92	651.10
Rastrigin (10D)	Interpolation	2.9616	1.4881	5000.00	0.00
Goldstein and Price (2D)	Extrapolation	0.0000	0.0000	20.02	3.90
Langermann (10D)	None	0.3133	0.2228	4135.62	1866.41
Schaffer (2D)	None	0.0009	0.0025	2193.76	1833.73
Schwefel (2D)	Binary	4.7379	23.4448	2980.58	1496.38
Shekel (10D)	None	7.9152	2.3911	4624.80	1285.29

Table 9 ABC – best performances

Benchmark	μ_E	σ_E	μ_S	σ_S
Hypersphere (10D)	0.0000	0.0000	131.14	4.80
Martin and Gaddy (2D)	0.0000	0.0000	14.98	3.29
Easom (2D)	0.0000	2.0096	15.42	2.01
Rosenbrock (10D)	0.0965	0.0880	4977.28	160.65
Ackley (10D)	0.0000	0.0000	186.64	6.27
Griewank (10D)	0.0052	0.0078	3574.38	1491.29
Rastrigin (10D)	0.0000	0.0000	2074.86	575.68
Goldstein and Price (2D)	0.0000	0.0001	20.82	4.35
Langermann (10D)	0.0000	0.0000	373.42	364.78
Schaffer (2D)	0.0000	0.0000	211.56	137.14
Schwefel (2D)	0.0000	0.0000	47.50	11.97
Shekel (10D)	8.3544	1.7278	4916.96	414.31

Table 10 Bees Algorithm – best performances

Benchmark	*stlim*	*ne*	*nre*	*nb*	*nrb*	μ_E	σ_E	μ_S	σ_S
Hypersphere (10D)	10	2	30	4	10	0.0000	0.0000	82.88	4.03
Martin and Gaddy (2D)	10	1	30	7	10	0.0000	0.0000	21.44	4.65
Easom (2D)	10	1	30	14	5	0.0000	0.0000	38.28	4.94
Rosenbrock (10D)	10	2	30	4	10	0.0293	0.0068	5000.00	0.00
Ackley (10D)	5	1	20	8	10	0.0000	0.0000	128.82	29.77
Griewank (10D)	5	1	10	18	5	0.0022	0.0037	2659.06	1889.61
Rastrigin (10D)	5	1	40	3	20	7.4821	2.0325	5000.00	0.00
Goldstein and Price (2D)	10	2	30	4	10	0.0000	0.0000	27.14	4.54
Langermann (10D)	5	1	40	3	20	0.0000	0.0000	74.90	69.59
Schaffer (2D)	10	2	30	4	10	0.0000	0.0000	278.90	273.35
Schwefel (2D)	10	1	30	14	5	0.0000	0.0000	44.58	13.64
Shekel (10D)	5	2	20	6	10	0.0000	0.0000	442.72	438.16

Contrary to what had been expected, the most successful Bees Algorithm configurations corresponded by and large to exploitative search strategies. Only in three cases out of 12 did the number *nb* of flower patches exceed the ten units. In the case of the *Easom* benchmark (section 4.1), the large number of population clusters might have helped finding the narrow area where the minimum is located.

Two reasons might explain the success of predominantly exploitative search strategies. First, the annealing-like shrinking of the local search neighbourhoods (section 2.3.1) might have facilitated the location of the lowest regions of the minimization surface. Second, the site abandonment procedure (section 2.3.2) helped the algorithm to escape sub-optimal function minima.

A comparison of the learning results presented in Tables 7 to 10 is proposed in Table 11. For each benchmark, Table 11 gives the number of successful runs ($F(x_f) < 0.001$) of the four optimization procedures. The parameter settings of the four algorithms are the ones given in Tables 7 to 10. For each function, the average accuracy μ_E, and the overall performance (section 5.1) of the four optimization methods were also compared. For each row (i.e. benchmark), the box corresponding to the algorithm that obtained the best average accuracy was crossed. In the case two or more algorithms gave comparable top μ_E values, all the corresponding boxes were crossed. The average accuracies of two algorithms were considered comparable if their difference was not statistically significant. Likewise, the boxes of the algorithms giving the best overall performance were crossed.

In terms of the total number of successful minimization trials, PSO, ABC, and the Bees Algorithm performed similarly on the given benchmarks. The differences between the results of the three SI methods correspond approximately to 3 per cent of the absolute figures. The EA was the only optimization procedure that significantly underperformed.

PSO, ABC, and the Bees Algorithm also performed similarly in terms of average accuracy results. PSO and ABC obtained top results in nine cases, and the Bees Algorithm achieved top results in ten cases.

Overall, the three SI methods were able to solve reliably most of the given optimization methods. The *Rosenbrock* benchmark turned out to be the most difficult problem, with no algorithm capable of obtaining anything more than sporadic successes. The *Shekel* and *Griewank* benchmarks also proved to be difficult tasks, particularly for the EA and ABC. The *Rastrigin* benchmark was hard for the EA and the Bees Algorithm.

In addition to the average accuracy μ_E, the evaluation of the performance includes also the average optimization speed μ_S (section 5.1). Comparing the accuracy results with the full performance evaluation, it is apparent that, given similar accuracies, the two bees-inspired methods outperformed PSO in terms of learning speed.

In terms of overall performance, the best results were attained by the Bees Algorithm that excelled in 8 out the 12 benchmarks considered, followed by ABC (5 out of 12), and PSO (2 out of 12). The Bees Algorithm excelled in the four most complex multimodal tasks, in the difficult *Rosenbrock* function, and in the *Hypersphere*, *Easom*, and *Ackley* benchmarks. The Bees Algorithm performed worst on the *Griewank* and *Rastrigin* functions. Examination of the learning process revealed that, for the latter two benchmarks, the Bees Algorithm found it difficult to set a suitable step size for the local search process. When the size of the flower patches was still large, the local search was prone to fall off the local peak area, usually ending up in neighbouring areas of lower fitness. When the size of the patches was shrunk as a consequence of the lack of improvement, the solutions quickly became trapped by one of the local minima.

The superior performance of the two bees-inspired methods on the 12 benchmarks considered was

Table 11 Comparison of best performances

	PSO			EA			ABC			BayBA		
Benchmark	succ.	acc.	perf.	succ.	acc.	perf.	succ.	acc.	perf.	succ.	acc.	perf.
Hypersphere (10D)	50	X		50	X		50	X		50	X	X
Martin and Gaddy (2D)	50	X		50	X	X	50	X	X	50	X	
Easom (2D)	50	X		50	X		50	X		50	X	X
Rosenbrock (10D)	3			0			1			0	X	X
Ackley (10D)	50	X		50	X		50	X		50	X	X
Griewank (10D)	45	X	X	1			28			36		
Rastrigin (10D)	42			0			50	X	X	0		
Goldstein and Price (2D)	50	X		50	X	X	49	X	X	50	X	
Langermann (10D)	50	X		9			50	X		50	X	X
Schaffer (2D)	50	X	X	41			50	X	X	50	X	X
Schwefel (2D)	48	X		41	X		50	X	X	50	X	X
Shekel (10D)	9			4			2			50	X	X
Total	497	9	2	346	6	2	480	9	5	486	10	8

probably due to the efficiency of their exploitation strategies. Contrarily to PSO and the EA, the bees-inspired algorithms can quickly mobilize a sizeable number of foragers around the most promising areas of the solution space. This exploitation effort does not affect the exploration capabilities of the algorithms. On the contrary, the exploitation capabilities of PSO and EAs are determined by the learning parameters and operators chosen, and come at the expense of reduced exploration capabilities.

The lack of an effective local search strategy particularly affected the EA implementation tested here. Indeed, while the exploitative search of PSO is guided by particle's momentum and social interactions with neighbouring particles, exploitation in EAs can rely only on the guidance of crossover (social interaction) and occasional random mutations. For this reason, EAs are known to be quick to locate the region of the global peak of fitness, but slow to converge to the actual optimum point. EAs were reported to perform worse than SI methods on similar optimization benchmarks also by Clerc and Kennedy [**46**], and Karaboga and Basturk [**39**].

5.3 Comparison of robustness

The first task was to pick the most consistent parameter setting for each algorithm. The method described in section 5.1 was used for each algorithm to compare the results given by the different parameter settings.

Sixteen algorithm configurations were tested for the PSO algorithm (Table 4(b)). The following heuristic criterion was used to select the most reliable parameter setting. For each benchmark function, each PSO configuration was picked once, and was compared to all the other PSO configurations. Each time that the performance of the selected parameter setting was better than the performance of another configuration, the given parameter setting was awarded two points. If the performances of the two configurations were comparable, the given configuration was awarded one point. For each benchmark, the points obtained by each parameter setting were added up, and the overall total score over the 12 test functions was calculated for each setting. The same procedure was followed for the EA and the Bees Algorithm. ABC used the parameter configuration recommended by Karaboga and Basturk [**39**].

The proposed comparison method aims to reward those algorithm configurations that performed best or nearly so on the largest number of optimization tasks.

Table 12(a) shows the results of the comparison for the PSO algorithm. The highest performing solutions correspond to the most explorative search strategies, that is, those configurations characterized by the lowest connectivity and largest maximum particle inertia (top right part of the table). Some fairly exploitative strategies of medium connectivity and small maximum momentum scored also moderately well (centre left part of the table). The best score is highlighted in bold, and the optimization results of the highest-scoring solution are given for each benchmark in Table 12(b). Comparing the results given in Table 12(b) to the results presented in Table 7, it can be concluded that the fine tuning of the PSO parameters was particularly important for the solution of the most difficult benchmarks, namely, *Rosenbrock*, *Griewank*, and *Rastrigin*.

Table 13(a) presents the results of the comparison for the EA. In this case, the algorithm not using crossover proved to be the clear winner. Among the EA configurations using crossover, interpolation and extrapolation crossover performed similarly, while binary crossover performed worst. The optimization results of the EA configuration without genetic recombination are given for each benchmark in Table 13(b). Comparing the results given in Table 13(b) to the results presented in Table 8, it seems that the use of crossover was particularly beneficial for the solution of the *Rastrigin* and *Schwefel* problems.

Table 14(a) shows the results of the comparison for the Bees Algorithm. The optimization results of the highest-scoring solution are given for each benchmark in Table 14(b). Two configurations obtained scores far better than the others. These two settings (no. 1 and 6) correspond to exploitative search strategies, where at least one elite site is searched by a large (30) number of foragers. All the three best scoring combinations used the longest stagnation limit (10). Comparing the results given in Table 4(b) to those presented in Table 10, it seems that the use of one unique configuration for the Bees Algorithm parameters did not significantly degrade the optimization results.

Table 15 presents a comparison of the optimization results of the four algorithms. For each algorithm, the most consistent configurations (Tables 12(b) to 14(b)) were used. The standard configuration (Table 9) was used for the ABC algorithm. Table 15 is similar to Table 11; the only difference is that here only one configuration is used for all the benchmarks.

Comparing the results given in Table 15 to the results presented in Table 11, the most apparent difference is the dramatic loss of competitiveness of the PSO algorithm. The use of a unique PSO configuration reduced the number of successful minimization trials by 100. The performance of the EA and especially of the Bees Algorithm was far less dependent on the choice of learning parameters.

When a unique configuration was used, PSO and the EA did not excel in any of the benchmark tasks in terms of overall performance. ABC and the Bees Algorithm totalled, respectively, 480 and 460 successful minimization trials. The difference between these two values is very modest, since it only amounts to approximately 4 per cent of the individual figures. Also

Table 12 PSO – robust performance

		Max particle velocity u			
		0.005	0.01	0.05	0.1
(a) *Comparison of performances*					
Connectivity (no. of neighbours)	2	110.00	169.00	268.00	247.00
	10	213.00	208.00	211.00	207.00
	20	197.00	201.00	211.00	181.00
	100	148.00	166.00	185.00	150.00

Benchmark	Connectivity (no. of neighbours)	Max particle velocity u	μ_E	σ_E	μ_S	σ_S
(b) *Optimization results*						
Hypersphere (10D)	2	0.05	0.0000	0.0000	2230.82	108.72
Martin and Gaddy (2D)	2	0.05	0.0000	0.0000	25.12	7.81
Easom (2D)	2	0.05	0.0000	0.0000	971.36	456.42
Rosenbrock (10D)	2	0.05	1.7879	1.5473	5000.00	0.00
Ackley (10D)	2	0.05	0.0000	0.0000	2616.08	91.65
Griewank (10D)	2	0.05	0.0199	0.0097	4977.14	161.64
Rastrigin (10D)	2	0.05	4.8162	1.4686	5000.00	0.00
Goldstein and Price (2D)	2	0.05	0.0000	0.0000	48.36	23.61
Langermann (10D)	2	0.05	0.0294	0.0905	1345.88	1558.99
Schaffer (2D)	2	0.05	0.0000	0.0000	354.74	271.51
Schwefel (2D)	2	0.05	4.7376	23.4448	845.72	903.73
Shekel (10D)	2	0.05	7.9537	2.3888	4778.48	759.35

Table 13 EA – robust performance

Crossover type	Wins
(a) *Comparison of performances*	
None	68
Binary	37
Interpolation	43
Extrapolation	44

Benchmark	Crossover type	μ_E	σ_E	μ_S	σ_S
(b) *Optimization results*					
Hypersphere (10D)	None	0.0000	0.0000	363.76	27.36
Martin and Gaddy (2D)	None	0.0000	0.0000	32.48	16.02
Easom (2D)	None	0.0000	0.0000	364.40	281.21
Rosenbrock (10D)	None	61.5213	132.6307	5000.00	0.00
Ackley (10D)	None	0.0000	0.0000	503.44	39.49
Griewank (10D)	None	0.0210	0.0130	4907.92	651.10
Rastrigin (10D)	None	17.4913	7.3365	5000.00	0.00
Goldstein and Price (2D)	None	0.0000	0.0000	58.16	22.59
Langermann (10D)	None	0.3133	0.2228	4135.62	1866.41
Schaffer (2D)	None	0.0009	0.0025	2193.76	1833.73
Schwefel (2D)	None	9.4751	32.4579	514.68	1336.32
Shekel (10D)	None	7.9152	2.3911	4624.80	1285.29

in terms of the optimization accuracy ABC and the Bees Algorithm excelled in an equal number of tasks (ten).

In terms of the overall performance, the Bees Algorithm excelled in a higher number of tasks. ABC performed better than the Bees Algorithm in two out of the three multi-modal benchmarks that are characterized by an overall unimodal behaviour and local periodical peaks. The Bees Algorithm excelled in the fully multi-modal functions, and in three out of the four unimodal benchmarks.

6 DISCUSSION OF THE RESULTS

The results of Table 14(a) highlight the importance for the Bees Algorithm of a sustained and thorough exploitation of the most promising flower patches. This fact can be explained as follows.

On the one hand, it should be noted that the version of the Bees Algorithm that was used here employs random mutations as the only means of local search. Particularly in the case of high-dimensional and difficult fitness landscapes, the likelihood of a favourable

Table 14 Bees Algorithm – robust performance

No.	*stlim*	*ne*	*nre*	*nb*	*nrb*	*wins*
(a) *Comparison of performances*						
1	10	2	30	4	10	203
2	5	2	30	4	10	146
3	5	1	40	3	20	121
4	5	2	20	6	10	146
5	5	1	40	6	10	151
6	10	1	30	7	10	195
7	5	1	30	7	10	151
8	10	1	30	14	5	164
9	5	1	30	14	5	125
10	5	1	20	8	10	125
11	5	1	20	16	5	121
12	5	1	10	18	5	80

Benchmark	*stlim*	*ne*	*nre*	*nb*	*nrb*	μ_E	σ_E	μ_S	σ_S
(b) *Optimization results*									
Hypersphere (10D)	10	2	30	4	10	0.0000	0.0000	82.88	4.03
Martin and Gaddy (2D)	10	2	30	4	10	0.0000	0.0000	22.48	3.29
Easom (2D)	10	2	30	4	10	0.0000	0.0000	38.66	8.19
Rosenbrock (10D)	10	2	30	4	10	0.0293	0.0068	5000.00	0.00
Ackley (10D)	10	2	30	4	10	0.0000	0.0000	121.86	35.53
Griewank (10D)	10	2	30	4	10	0.0089	0.0059	4470.64	1265.12
Rastrigin (10D)	10	2	30	4	10	8.8201	2.2118	5000.00	0.00
Goldstein and Price (2D)	10	2	30	4	10	0.0000	0.0000	27.14	4.54
Langermann (10D)	10	2	30	4	10	0.0000	0.0000	115.68	102.39
Schaffer (2D)	10	2	30	4	10	0.0000	0.0000	278.90	273.35
Schwefel (2D)	10	2	30	4	10	0.0000	0.0000	50.06	21.10
Shekel (10D)	10	2	30	4	10	0.0000	0.0000	809.16	978.85

Table 15 Comparison of robustness

	PSO			EA			ABC			BayBA		
Benchmark	succ.	acc.	perf.	succ.	acc.	perf.	succ.	acc.	perf.	succ.	acc.	perf.
Hypersphere (10D)	50	X		50	X		50	X		50	X	X
Martin and Gaddy (2D)	50	X		50	X		50	X	X	50	X	
Easom (2D)	50	X		50	X		50	X		50	X	X
Rosenbrock (10D)	0			0			1			0	X	X
Ackley (10D)	50	X		50	X		50	X		50	X	X
Griewank (10D)	1			1			28	X	X	10		
Rastrigin (10D)	0			0			50	X	X	0		
Goldstein and Price (2D)	50	X		50	X		49	X	X	50	X	
Langermann (10D)	44			9			50	X		50	X	X
Schaffer (2D)	50	X		41			50	X	X	50	X	X
Schwefel (2D)	48	X		46			50	X	X	50	X	X
Shekel (10D)	4			4			2			50	X	X
Total	397	7	0	351	5	0	480	10	6	460	10	8

random mutation decreases with the fitness of the solution. For this reason, a longer stagnation period and a large recruitment of foragers improves the chance of reaching the local peak of performance. The chance of convergence is also helped by the neighbourhood shrinking procedure that narrows down the local search towards the local peak area. On the other hand, thanks to the site abandonment procedure, despite a fairly exploitative search strategy, the algorithm does not risk being stuck forever in local sub-optima of accuracy. Once a site is abandoned, the foragers are quickly re-directed to other flower patches.

The best search strategy for the Bees Algorithm is therefore to focus large resources on one (a few) selected sites, and move them quickly somewhere else once the flower patch becomes 'exhausted'. This approach is in very good agreement with the foraging behaviour of real honeybees [**22**].

A similarly exploitative strategy is recommended by Karaboga and Basturk for the ABC algorithm. In this case, proportional selection and a high stagnation limit guarantee a very large and long-sustained exploration of the patches of highest fitness. However, the very long stagnation limit may be the cause of the non-optimal results in some highly multi-modal benchmarks like the *Langermann* and *Shekel* functions. Unfortunately, the ABC algorithm uses the disruptive extrapolation crossover for neighbourhood search. This makes the local search procedure more likely to stray away from the fitness peak. The risk of falling off the fitness peak is increased by the fact that the employed bee can be mated with any other bee, including a recently generated distant bee of mediocre fitness. For this reason, a long stagnation limit is required to ensure that a flower patch is searched with sufficient accuracy.

The disruptiveness of the extrapolation crossover operator is probably also the cause of the non-optimal results of the ABC algorithm in benchmarks that required a highly accurate local search, like the *Easom* (a deep and narrow 'hole'), *Rosenbrock* (a long and narrow valley), and Ackley (a fairly narrow 'hole') functions.

ABC seems to perform best in those cases where the fitness landscape gives overall guidance on where to concentrate the sampling, but the presence of many local optima requires a fairly disruptive local search strategy. This is the case with the *Griewank* and *Rastrigin* benchmarks and the *Schaffer* function (sinusoidal wave of rapidly decaying amplitude). For all the above three functions, ABC achieved top performances.

PSO and EAs have a search mechanism that differs from those of the two bees-inspired algorithms. PSO and EAs initially spread the population across the search space, and via mutual attraction and/or natural selection make solutions converge towards the most promising area of the fitness landscape. Since no equivalent of the site abandonment procedure exists for the standard versions of PSO and EAs, a more cautious approach is required before committing the bulk of the resources to the exploitation of the most promising area(s). For this reason, the winning strategies for the PSO and EA versions tested here were the most explorative ones. However, in PSO and EAs exploration comes at the expense of exploitation, and for this reason the two algorithms did not excel in terms of learning speed. With no mechanism to escape sub-optimal convergence, the success of PSO also relies more heavily on the tuning of the learning parameters. EAs seemed to be more consistent in terms of performance, but this consistency came at the expense of sub-optimal accuracy results and a slower learning speed.

The contribution of crossover to the evolutionary search seemed to be of limited benefit for the solution of the benchmark tasks considered in this study. A likely explanation is in the disruptiveness of the operator, which involves larger jumps in the solution space than mutation.

7 CONCLUSIONS

The performance of the Bees Algorithm was evaluated on 12 benchmark minimization tasks. The results were compared with those produced by three control methods: PSO, EA, and ABC algorithm. For each algorithm, different parameter settings and operators were tested.

Despite the care taken in selecting a varied range of test functions, it should be stressed that the results presented in this article are strongly dependent on the benchmarks used. Given the large number of PSO and EA implementations, the validity of the results is also limited to the very standard versions examined in this study. Nonetheless, some interesting observations can be made.

The tests proved the strength of the Bees Algorithm in terms of accuracy, learning speed, and robustness. In 10 of the 12 benchmark cases considered, the Bees Algorithm ranked among the top performing optimization procedures. Conservatively, it can be claimed that the Bees Algorithm performed at least as well as PSO, EAs, and ABC.

The two algorithms modelled on the bees foraging metaphor achieved the best results, particularly in terms of learning speed and consistency. The ABC algorithm is the most easy to design, since all the learning parameters are heuristically pre-set. In the case of the Bees Algorithm, once a good parameter setting was found, this configuration was able to solve with top performance a larger number of benchmarks than any of its competitors. Using a custom-made parameter setting for each test function did not significantly enhance the performance of the Bees Algorithm. This result is encouraging, since it suggests that there exist settings of the learning parameters that perform consistently on large classes of problems.

The most effective search strategy for the Bees Algorithm consisted of focusing a large number of foragers on a few selected sites, and narrowing down the local search until a local minimum is found. This strategy is in agreement with the foraging behaviour of bees in nature, which quickly direct a large number of foragers to the best rated flower patches, and abandon them for other patches once their food content is exhausted. Keeping this approach in mind, selecting an effective set of parameters for the Bees Algorithm may prove easy.

The ABC method is characterized by a similarly exploitative strategy, but focuses for longer times on the most promising regions, and has a more disruptive local search strategy.

The Bees Algorithm and ABC seem to perform at their best in different classes of problems. The former seems more suitable for searching accurately narrow valleys and holes, and for highly multi-modal functions. The latter seems more suitable for functions where an overall unimodal or quasi-unimodal behaviour may be used to focus the emphasis of the search, but a rough, multi-pocketed fitness landscape can make progress difficult.

PSO and EAs use a more gradual approach, i.e. the population is initially spread evenly over the solution space, and is slowly made to converge towards the global peak of performance. The convergence process of these is not easily reversible, and this makes them more prone to becoming trapped by sub-optimal peaks of performance. For this reason, PSO and EAs seemed to benefit from the use of explorative search strategies. Unfortunately, exploitation and exploration in PSO and EAs are related, and their resources must be traded off. This made the performance of PSO more sensitive to the choice of an optimal set of learning parameters. Moreover, since exploration comes at the expense of exploration, PSO was slower to converge than its bees-inspired competitors. The EA version tested in this study fared better in terms of consistency of the results, but was often less accurate and slower than its competitors.

Future work should include further comparisons with other SI approaches, in particular ant-inspired continuous function optimization methods like ACO_R, CACO, and CIAC [**31–33**]. It should also be pointed out that all the optimization problems considered in this article were defined on a relatively small number of independent variables. A thorough study of the behaviour of the Bees Algorithm and other SI approaches should also include search spaces of very high dimensionality. The characterization of the behaviour of these algorithms in very large search spaces could be of great use in the design of effective search strategies for a wide number of tasks. A typical example of a high-dimensional search problem is the optimization of the usually large number of neural network weights.

ACKNOWLEDGEMENTS

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

REFERENCES

1 **Bonabeau, E., Dorigo, M.,** and **Theraulaz, G.** *Swarm intelligence: from natural to artificial systems*, 1999 (Oxford University Press, New York).
2 **Stephens, D.** and **Krebs, J.** *Foraging theory*, 1986 (Princeton University Press, Princeton, NJ).
3 **Deneubourg, J. L., Aron, S., Goss, S.,** and **Pasteels, J. M.** The self-organising exploratory pattern of the Argentine ant. *J. Insect Behav.*, 1990, **3**, 159–168.
4 **Bonabeau, E.** Social insect colonies as complex adaptive systems. *Ecosystems*, 1998, **1**, 437–443.
5 **Cox, M. D.** and **Myerscough, M. R.** A flexible model of foraging by a honey bee colony: the effects of individual behaviour on foraging success. *J. Theor. Biol.*, 2003, **223**(2), 179–197.
6 **Tereshko, V.** and **Loengarov, A.** Collective decision-making in honey bee foraging dynamics. *J. Comput. Inf. Syst.*, 2005, **9**(3), 1–7.
7 **Passino, K. M., Seeley, T. D.,** and **Visscher, P. K.** Swarm cognition in honey bees. *Behav. Ecol. Sociobiol.*, 2008, **62**(3), 401–414.
8 **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.
9 **Dorigo, M., Di Caro, G.,** and **Gambardella, L. M.** Ant algorithms for discrete optimization. *Artif. Life*, 1999, **5**(2), 137–172.
10 **Passino, K. M.** Biomimicry of bacterial foraging for distributed optimisation and control. *IEEE Control Syst. Mag.* (June), 2002, 52–67.
11 **Pham, D. T., Ghanbarzadeh, A., Koc, E., Otri, S., Rahim, S.,** and **Zaidi, M.** The Bees Algorithm, a novel tool for complex optimisation problems. In Proceedings of the Second International Virtual Conference on *Intelligent production machines and systems (IPROMS 2006)*, Elsevier, Oxford, 2006, pp. 454–459.
12 **Jiang, M., Mastorakis, N., Yuan, D.,** and **Lagunas, M. A.** Multi-threshold image segmentation with improved artificial fish swarm algorithm block-coding and antenna selection. In Proceedings of the ECC 2007 European Computing Conference, 2007.
13 **Pham, D. T., Afify, A. A.,** and **Koç, E.** Manufacturing cell formation using the Bees Algorithm. In Proceedings of the Third International Virtual Conference on *Intelligent production machines and systems (IPROMS 2007)*, Whittles, Dunbeath, Scotland, 2007.
14 **Pham, D. T., Castellani, M.,** and **Ghanbarzadeh, A.** Preliminary design using the Bees Algorithm. In Proceedings of the Eighth LAMDAMAP International Conference on *Laser metrology, CMM and machine tool performance*, Cardiff, UK, 2007, pp. 420–429.
15 **Pham, D. T., Otri, S.,** and **Haj Darwish, A.** Application of the Bees Algorithm to PCB assembly optimisation. In Proceedings of the Third International Virtual Conference on *Intelligent production machines and systems (IPROMS 2007)*, Whittles, Dunbeath, Scotland, 2007, pp. 511–516.
16 **Pham, D. T., Koc, E., Lee, J. Y.,** and **Phrueksanant, J.** Using the Bees Algorithm to schedule jobs for a machine. In Proceedings of the Eighth International Conference on *Laser metrology, CMM and machine tool performance (LAMDAMAP)*, Cardiff, Euspen, UK, 2007, pp. 430–439.
17 **Pham, D. T., Haj Darwish, A.,** and **Eldukhri, E. E.** Optimisation of a fuzzy logic controller using the Bees Algorithm. *Int. J. Comput. Aided Eng. Technol.*, 2009, **1**, 250–264.
18 **Pham, D. T., Soroka, A. J., Ghanbarzadeh, A., Koc, 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.
19 **Von Frisch, K.** *Bees: their vision, chemical senses and language*, revised edition, 1976 (Cornell University Press, Ithaca, NY).

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

21 **Camazine, S., Deneubourg, J.-L., Franks, N. R., Sneyd, J., Theraulaz, G.,** and **Bonabeau, E.** *Self-organization in biological systems*, 2003 (Princeton University Press, Princeton).

22 **Tereshko, V.** and **Lee, T.** How information-mapping patterns determine foraging behaviour of a honey bee colony. *Open Syst. Inf. Dyn.*, 2002, **9**(2), 181–193.

23 **Pham, D. T., Castellani, M.,** and **Fahmy, A. A.** Learning the inverse kinematics of a robot manipulator using the Bees Algorithm. In Proceedings of the INDIN 2008, 2008, pp. 493–498.

24 **Pham, D. T.** and **Karaboga, D.** *Intelligent optimisation techniques: genetic algorithms, tabu search, simulated annealing and neural networks*, 2000 (Springer-Verlag, London, UK).

25 **Rechenberg, I.** *Cybernetic solution path of an experimental problem*, library translation no. 1122, 1965 (Ministry of Aviation, Royal Aircraft Establishment, Farnborough, Hants, UK).

26 **Fogel, L. J., Owens, A. J.,** and **Walsh, M. J.** *Artificial intelligence through simulated evolution*, 1966 (John Wiley, New York).

27 **Holland, J. H.** *Adaptation in natural and artificial systems*, 1975 (University of Michigan Press, Ann Arbor).

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

29 **Mengshoel, O. J.** and **Goldberg, D. E.** The crowding approach to niching in genetic algorithms. *Evol. Comput.*, 2008, **16**(3), 315–354.

30 **Baeck, T., Hoffmeister, F.,** and **Schwefel, H. P.** A survey of evolution strategies. In Proceedings of the Fourth International Conference on *Genetic algorithms*, San Mateo, USA, 1991, pp. 2–9 (Morgan Kaufmann).

31 **Bilchev, G.** and **Parmee, I. C.** The ant colony metaphor for searching continuous design spaces. *Lect. Notes Comput. Sci.*, 1995, **993**, 25–39.

32 **Dréo, J.** and **Siarry, P.** Continuous interacting ant colony algorithm based on dense heterarchy. *Future Generation Computer Systems*, 2004, **20**, 841–856.

33 **Socha, K.** and **Dorigo, M.** Ant colony optimisation for continuous domains. *Eur. J. Oper. Res.*, 2008, **185**, 1155–1173.

34 **Blackwell, T.** and **Branke, J.** Multi-swarm optimization in dynamic environments, Applications of evolutionary computing. In *Lecture notes in computer science* (Ed. G. R. Raidl), vol. 3005, 2004, pp. 489–500 (Springer-Verlag, Berlin, Germany).

35 **Sato, T.** and **Hagiwara, M.** Bee system: finding solution by a concentrated search. In Proceedings of the IEEE International Conference on *Systems, man, and cybernetics*, 1997, pp. 3954–3959.

36 **Olague, G.** and **Puente, C.** Parisian evolution with honeybees for three-dimensional reconstruction. In Proceedings of the Eighth GECCO Annual Conference on *Genetic and evolutionary computation*, 2006, pp. 191–198.

37 **Baig, A. R.** and **Rashid, M.** Honey bee foraging algorithm for multimodal and dynamic optimization problems. In Proceedings of the Ninth GECCO Annual Conference on *Genetic and evolutionary computation*, 2007, p. 169.

38 **Baeck, T.** and **Schwefel, H. P.** An overview of evolutionary algorithms for parameter optimization. *Evol. Comput.*, 1993, **1**(1), 1–23.

39 **Karaboga, D.** and **Basturk, B.** On the performance of artificial bee colony (ABC) algorithm. *Appl. Soft Comput.*, 2008, **8**(1), 687–697.

40 **Haddad, O. B., Afshar, A.,** and **Marino, M. A.** Honey-bees mating optimization (HBMO) algorithm: a new heuristic approach for water resources optimization. *Water Resour. Manage.*, 2006, **20**, 661–680.

41 **Roth, M.** and **Wicker, S.** *Termite: emergent ad-hoc networking.* In Proceedings of the Second Mediterranean Workshop on *Ad-hoc networks*, 2003.

42 **Xiao, J. M., Zheng, X. M.,** and **Wang, X. H.** A modified artificial fish-swarm algorithm. In Proceedings of the Sixth IEEE World Congress on *Intelligent control and automation*, Dalian, China, 2006, pp. 3456–3460.

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

44 **Wolpert, D. H.** and **Macready, W. G.** No free lunch theorems for optimization. *IEEE Trans. Evol. Comput.*, 1997, **1**(1), 67–82.

45 **Eberhart, R. C., Simpson, P.,** and **Dobbins, R.** *Computational intelligence PC tools*, 1996, ch. 6, pp. 212–226 (Academic Press Professional, San Diego, CA).

46 **Clerc, M.** and **Kennedy, J.** The particle swarm – explosion, stability, and convergence in a multidimensional complex space. *IEEE Trans. Evol. Comput.*, 2002, **6**(1), 58–73.

APPENDIX – CONTROL ALGORITHMS

Evolutionary algorithm

Mutation

The genetic mutation operator modifies all the variables of a randomly picked solution according to the following equation

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

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 a Bees Algorithm's flower patch. For each individual, a is adaptively tuned via random mutation events as follows

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

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

Interpolation crossover

Given two parent solutions $x = \{x_1, \ldots, x_n, a_x\}$ and $y = \{y_1, \ldots, y_n, a_y\}$, interpolation crossover creates a new solution $z = \{z_1, \ldots, z_n, a_z\}$ as follows

$$z_i = x_i(1 - random) + y_i\ random \qquad i = 1, \ldots, n \tag{8}$$

where $random \in [0,1]$ is a randomly generated number. The mutation range a_z is also calculated according to equation (8).

Extrapolation crossover

Given two candidate solutions $x = \{x_1, \ldots, x_n, a_x\}$ and $y = \{y_1, \ldots, y_n, a_y\}$, extrapolation crossover creates a child $z = \{z_1, \ldots, z_n, a_z\}$ as follows

$$z_i = x_i + random(x_i - y_i) \qquad i = 1, \ldots, n \tag{9}$$

where $random \in [-1,1]$ is a randomly generated number.

Particle swarm optimization

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

$$x_i(t+1) = x_i(t) + v_i(t+1) \qquad i = 1, \ldots, n \tag{10}$$

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

$$\begin{aligned} p_i(t) &= random_1[p\text{best}(t)_i - x(t)_i] \\ s_i(t) &= random_2[g\text{best}(t)_i - x(t)_i] \\ v_i(t+1) &= w(t)v(t)_i + c_1 p_i(t) + c_2 s_i(t), \quad i = 1, \ldots, n \end{aligned} \tag{11}$$

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], $p$best(t) (personal best) is an n-dimensional vector that describes the best position (most fit) attained so far by the particle, and $g$best(t) (global best) describes the best position attained so far by the social neighbours of the particle.

The weight $w(t)$ is decayed as follows

$$w(t) = w_{\max} - \frac{w_{\max} - w_{\min}}{T} t \tag{12}$$

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 $[-v_i^{\max}, v_i^{\max}]$, where

$$v_i^{\max} = u \frac{\max_i - \min_i}{2} \tag{13}$$