Information Sciences 230 (2013) 183–196

Contents lists available at SciVerse ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

An artificial bee colony algorithm for the maximally diverse grouping problem

Francisco J. Rodriguez [a,*], M. Lozano [a], C. García-Martínez [b], Jonathan D. González-Barrera [c]

[a] Department of Computer Science and Artificial Intelligence, University of Granada, Granada, Spain
[b] Department of Computing and Numerical Analysis, University of Córdoba, Córdoba, Spain
[c] Department of Statistics, Operational Research, and Computer Science, University of La Laguna, San Cristobal de La Laguna, Spain

A R T I C L E I N F O

Article history:
Received 25 June 2012
Received in revised form 14 December 2012
Accepted 21 December 2012
Available online 8 January 2013

Keywords:
Maximally diverse grouping problem
Artificial bee colony algorithm
Iterated greedy

A B S T R A C T

In this paper, an artificial bee colony algorithm is proposed to solve the maximally diverse grouping problem. This complex optimisation problem consists of forming maximally diverse groups with restricted sizes from a given set of elements. The artificial bee colony algorithm is a new swarm intelligence technique based on the intelligent foraging behaviour of honeybees. The behaviour of this algorithm is determined by two search strategies: an initialisation scheme employed to construct initial solutions and a method for generating neighbouring solutions. More specifically, the proposed approach employs a greedy constructive method to accomplish the initialisation task and also employs different neighbourhood operators inspired by the iterated greedy algorithm. In addition, it incorporates an improvement procedure to enhance the intensification capability. Through an analysis of the experimental results, the highly effective performance of the proposed algorithm is shown in comparison to the current state-of-the-art algorithms which address the problem.

1. Introduction

The maximally diverse grouping problem (MDGP) consists of partitioning a set of n elements (E) into m disjoint groups so that the diversity among the elements in each group is maximised. The diversity among the elements in a group is calculated as the sum of the dissimilarity values between each pair of elements. Let d_{ij} be the dissimilarity value between the elements i and j and $x_{ig} = 1$ if the element i is in the group g and 0 if not. The problem can thus be stated as:

$$\textbf{Maximise} \quad \sum_{g=1}^{m}\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} d_{ij}\cdot x_{ig}\cdot x_{jg} \tag{1}$$

$$\textbf{Subject to} \quad \sum_{g=1}^{m} x_{ig} = 1 \quad i = 1,\ldots,n, \tag{2}$$

$$a_g \leqslant \sum_{i=1}^{n} x_{ig} \leqslant b_g \quad g = 1,\ldots,m, \tag{3}$$

$$x_{ig} \in \{0,1\} \; i = 1,\ldots,n; \quad g = 1,\ldots,m. \tag{4}$$

* Corresponding author.
E-mail address: fjrodriguez@decsai.ugr.es (F.J. Rodriguez).

0020-0255/$ - see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ins.2012.12.020

where constraint (2) ensures that each element is located in exactly one group and constraint (3) guarantees that the number of elements contained in group g is at least a_g and at most b_g. The MDGP also appears in the literature by other names, such as *the k-partition problem* in Feo et al. [12,13] and as *the equitable partition problem* in O'Brien and Mingers [36].

Different real-world applications of the MDGP can be found in the literature, covering a wide variety of fields. One of the most extended applications of the MDGP is to the assignment of students to groups [4,10,49–51,53]. The MDGP also arises in further fields such as the creation of diverse peer review groups [19], the design of VLSI circuits [13,49,50], the storage of large programs onto paged memory [32], and the creation of diverse groups in companies so that people from different backgrounds work together [6].

Several metaheuristic approaches were proposed to obtain high quality solutions to the MDGP. These include memetic algorithms [10,38], the tabu search [15], simulated annealing [38], and the variable neighbourhood search [38]. In this paper, we explore an alternative approach using the artificial bee colony (ABC) [5,24,26,27] method. The ABC algorithm is a new, population-based, metaheuristic approach inspired by the intelligent foraging behaviour of honeybee swarms. It consists of three essential components: food source positions, nectar-amount and three honeybee classes (employed bees, onlookers and scouts). Each food source position represents a feasible solution for the problem under consideration. The nectar-amount for a food source represents the quality of each solution (represented by an objective function value). Each bee-class symbolises one particular operation for generating new candidate food source positions. Specifically, employed bees search for food around the food source in their memory; meanwhile they pass their food information onto onlooker bees. Onlooker bees tend to select good food sources from those found by the employed bees, and then further search for food around the selected food source. Scout bees are a small group of employed bees that are transferred, abandoning their food sources and searching for new ones. Due to its simplicity and ease of implementation, the ABC algorithm has drawn much attention and has exhibited state-of-the-art performances for a considerable number of problems [28].

The rest of this paper is organised as follows. In Section 2 we provide an overview of particular metaheuristics that have previously been applied to the MDGP. In Section 3 we analyse the background to the ABC algorithm, which provides the basis of the specific algorithmic design we employ. In Section 4 we describe our ABC approach to the MDGP. In Section 5 we analyse the performance of the proposed ABC and draw comparisons with the existing literature. Finally, Section 6 contains a summary of results and conclusions.

2. Metaheuristics for the MDGP

In this section, we detail metaheuristics from the literature that are most relevant to the MDGP. The first approach to this problem was a *multistart algorithm* introduced by Arani and Lotfi [3]. This algorithm starts by generating a random solution that is then partially destroyed. Then, all the possible reconstructed solutions are explored by the algorithm in order to select the best one. This process is repeated until there is no improvement in the reconstructed solution.

Feo and Khellaf [13] presented several heuristics based on *graph theory*. Later, Weitz and Lakshminarayanan [50] carried out an experimental comparison of different heuristics for the MDGP and concluded that the best results were achieved by the *Lotfi–Cerveny–Weitz* [50] (LCW) method. This consists of an enhancement procedure that refines a random solution or a solution constructed by means of the *Weitz–Jelassi* [51] (WJ) algorithm. There are no significant differences in terms of the accuracy of the solutions between the two LCW variants; however, LCW with a random initial solution is considerably faster.

Fan et al. [10] proposed a *memetic algorithm* to deal with the MDGP (LSGA), combining a genetic algorithm with a local search procedure. The genetic algorithm uses a special encoding to group problems proposed by Falkenauer [8] and employs a procedure to initialise the population which ensures that the initial solutions conform to the constraints of the MDGP. LSGA employs a rank-based roulette-wheel strategy to select parents for the crossover operator. The latter is based on a special crossover operator proposed for grouping problems [8]. The local search procedure of LSGA implements a best improvement strategy based on exchanging elements between the groups [4]. Computational results show that LSGA is more effective than a pure genetic algorithm and LCW [50].

Later, Gallego et al. [15] developed a multistart metaheuristic for the MDGP that is based on the *tabu search methodology* (TS-SO). It consists of three main elements:

- A greedy constructive procedure to generate the initial solutions.
- A local search procedure based on the LCW method (T-LCW). This expands the LCW neighbourhoods to include insertions and adds a short-term tabu memory to prevent the recently moved elements from changing the group to which they belong for a number of iterations.
- A strategic oscillation method to explore solutions for which the group size limits may be amended. The strategic oscillation is coupled with the T-LCW improvement method, so that T-LCW is applied, relaxing lower and upper limits for all groups ($a_g = a_g - k$ and $b_g = b_g + k$). The value of k is reset to 1 after each successful application of T-LCW, otherwise k is increased by 1. In this way, the oscillation pattern is created. As this method works with unfeasible solutions, it is necessary to employ a repair mechanism. The repair mechanism consists of removing elements from groups that exceed their upper bounds and adding these elements to groups that fall below their lower bounds.

An extensive experimental study is performed to compare the performance of TS-SO with those of LCW, T-LCW, and LSGA. This study shows that TS-SO outperforms its competitors in two out of three sets of instances. For the other set of instances, the best results are achieved by T-LCW.

Finally, Palubeckis and Karciauskas [38] present three different metaheuristics to face the MDGP:

- A *multistart simulated annealing* (MSA) that at each iteration of the algorithm chooses probabilistically a neighbourhood function (*interchange or relocation neighbourhoods*) which is used to obtain a neighbouring solution from the current one. The neighbouring solution is accepted as the new current solution according to the simulated annealing acceptance criterion.
- A *hybrid steady-state genetic algorithm* (HGA) that uses a local search procedure to refine random initial solutions and solutions obtained by the crossover operator. The new offspring replaces the worst solution in the old population only if its fitness value is better.
- A *variable neighbourhood search* (VNS) that executes three phases iteratively: shaking, local search, and neighbourhood change. In the shaking phase, the algorithm selects $r/2$ pairs of elements in such a way that the elements of each pair belong to different groups and no element is selected twice. Then, the algorithm interchanges the groups of each pair of elements. The local search procedure is applied in the initialisation and after shaking step. The neighbourhood change principle of the VNS method is applied, changing the r value.

The performance of these algorithms is compared by means of an experimental study [38] on two sets of problem instances of a size of up to 2000 elements. The results show that there is no clear winner. For smaller instances, the best results were achieved by HGA. However, with large MDGP instances VNS showed the best performance.

3. The artificial bee colony algorithm

In a real bee colony, some tasks are carried out by specialised individuals. Bees try to maximise the nectar amount unloaded to the food stores in the hive by the division of labour and self-organisation, which are essential components of swarm intelligence. The minimal model of swarm intelligent forage selection in a honeybee swarm consists of three kinds of bees: *employed* bees, *onlooker* bees, and *scout* bees [47]. Employed bees are responsible for exploiting the previously explored nectar sources and they provide information to the onlooker bees in the hive about the quality of the food source that they are exploiting. The onlookers tend to select good food sources from those found by the employed bees, and then they perform a further search for food around the selected food source. Specifically, employed bees share information about food sources by dancing in a common area in the hive called the dance area (the duration of a dance is proportional to the nectar content of the food source currently being exploited by the dancing bee). In the natural behaviour of real bees, if a food source is not worth exploiting anymore, it is abandoned, and the employed bee of that source becomes a scout, searching the environment randomly or through an internal motivation. Scout bees perform the job of exploration, while employed and onlooker bees perform the job of exploitation.

Inspired by the foraging behaviour of honeybees, Karaboga [5,24,26,27] proposed the ABC algorithm. In this algorithm, the position of a food source represents a solution to the optimisation problem and the nectar amount of a food source represents the quality (fitness) of the solution represented by that food source. This algorithm assumes the existence of a set of computational agents called honeybees (employed, onlookers and scouts) and the process of bees seeking good food sources is that which is used to find the optimal solution. The algorithm begins with a population of randomly distributed positions of food sources. The number of employed bees and of onlookers is equal to the number of food sources around the hive.

ABC is an iterative algorithm that starts by generating random solutions as the food sources and then the following activities (performed by each category of bees) are repeated until a stopping condition is met:

1. *Employed bees phase*. The employed bees start the search process and employ cognitive skills to move towards the food source with the greatest nectar amount. Each employed bee finds a new food source near its currently assigned food source (using a *neighbourhood operator*) and checks the nectar amount of the newly discovered position. If the employed bee finds it to be greater than the nectar amount of the previous position, it refreshes its memory by replacing the old position with the new one; otherwise, it keeps the old position in its memory. When all the employed bees have finished this exploitation process, they share the nectar information of the food sources with the onlookers in the dance area.
2. *Onlooker bees phase*. Onlookers are the bees that wait in the dance area and decipher the information about the position and nectar amount of the food source conveyed in the form of dance by the employed bees. Each onlooker selects a food source according to the traditional roulette wheel selection method. After that, in a similar way to the employed bees, each onlooker tries to discover hidden potential food sources near its selected food source (using the neighbourhood operator) and calculates the nectar amount of the neighbouring food source. If it finds a more attractive food source, it retains the information regarding the new food source and discards the old one.
3. *Scout bees phase*. If the employed bee and the onlookers associated with a food source cannot find a better neighbouring food source within a certain number of iterations, the food source is abandoned and the employed bee associated with the food source becomes a scout. The scout will search for the location of a new food source in the vicinity of the hive. After the scout finds a new food source, it becomes an employed bee again.

Although the ABC algorithm was originally designed for continuous optimisation problems [26,27] and much work has been dedicated to the development of extended models for this kind of problem [1,9,16,23], other versions of the ABC algorithm have been successfully applied to a wide range of optimisation problems, such as the quadratic minimum spanning tree problem [44], the leaf-constrained minimum spanning tree problem [43], symbolic regression [30], designing and operating assembly lines [45], binary optimisation problems [31], image segmentation [34], constrained optimisation problems [25], the designing of recurrent neural networks [21], multi-objective optimisation problems [2,37], the development of routing protocols for wireless sensor networks [41,42], the planning of hardware solutions for communication systems [35], and clustering [29]. A complete review of ABC applications can be found in [28].

4. ABC algorithm for the MDGP

In this section, we explore how the ABC framework may be adapted to obtain high quality solutions for the MDGP. In Section 4.1, we provide a general overview of the overall algorithm. In Section 4.2, we describe the initialisation method employed to construct good starting solutions. In Section 4.3, we describe the neighbourhood procedures, which play a fundamental role in our ABC proposal. Finally, in Section 4.4, we provide details of the local improvement approach that is embedded in the proposed ABC algorithm to enhance its local search intensification capability.

```
    Input: $n$, $m$, $t_{max}$, $limit$, $NP$, $p_{ls}$
    Output: $S^*$
    //Initialization phase
 1  for $i = 1$ to $NP$ do
 2      $S_i \leftarrow$ Construct-Solution$(n, m)$;
 3      if $U(0,1) < p_{ls}$ then
 4          $S_i \leftarrow$ Local-Improvement $(S_i)$;
 5      end
 6  end
 7  while computation time $t_{max}$ not reached do
        //Employed bees phase
 8      for $i = 1$ to $NP$ do
 9          $E \leftarrow$ Generate-Neighbouring$(S_i)$;
10          if $U(0,1) < p_{ls}$ then
11              $E \leftarrow$ Local-Improvement $(E)$;
12          end
13          if $E$ is better than $S_i$ then
14              $S_i \leftarrow E$;
15          end
16      end
        //Onlooker bees
17      for $i$=1 to $NP$ do
18          $j \leftarrow$ Binary-Tournament $(1, ..., NP)$;
19          $O \leftarrow$ Generate-Neighbouring$(S_j)$;
20          if $U(0,1) < p_{ls}$ then
21              $O \leftarrow$ Local-Improvement $(O)$;
22          end
23          if $O$ is better than $S_j$ then
24              $S_j \leftarrow O$;
25          end
26      end
        //Scout bees phase
27      for $i$=1 to $NP$ do
28          if $S_i$ does not change for limit iterations then
29              $S_i \leftarrow$ Construct-Solution$(n, m)$;
30              if $U(0,1) < p_{ls}$ then
31                  $S_i \leftarrow$ Local-Improvement $(S_i)$;
32              end
33          end
34      end
        //Remember the best food source found so far
35      $S^* \leftarrow$ Best-Solution-Found ();
36  end
```

Fig. 1. Pseudocode for ABC-MDGP.

4.1. General scheme of the proposed ABC algorithm

An outline of our proposal, called ABC-MDGP, is depicted in Fig. 1. The approach begins with a population of solutions (food sources) generated by the function Construct-Solution (Line 2), which implements a greedy constructive method to create feasible solutions for the MDGP (Section 4.2). Then, the following steps are repeated until a termination criterion is met:

- *The employed bees phase* produces new solutions for the employed bees via the function Generate-Neighbouring (Line 9). For this function, we employ three different neighbourhood operators; two of them are destructive–constructive methods, which are formulated by borrowing the base idea from the iterated greedy algorithm [33] and the other is based on a standard swap move (Section 4.3).
- *The onlooker bees phase* produces new solutions for the onlookers from the solutions selected by the function Binary-Tournament (Line 18). In the basic ABC algorithm, an onlooker bee selects a food source depending on its winning probability value, in a similar way to the roulette wheel selection in genetic algorithms [18]. However, the *tournament selection* is widely used in ABC applications due to its simplicity and ability to escape from local optima [44,46]. For this reason, we employ a binary tournament selection in the ABC-MDGP algorithm. Specifically, an onlooker bee selects the better of two food sources that are selected from the population. Then, each onlooker determines a food source in its neighbourhood in the same way as the employed bees (Line 19).
- *The scout bees phase* determines the abandoned solutions for the scouts and replaces them with newly produced solutions via the function Construct-Solution (Line 29).

In order to enhance the exploitation capability of ABC-MDGP, a local improvement procedure (function Local-Improvement) is embedded in the proposed algorithm. Specifically, after generating new solutions (in the initialisation and scout phases, lines 4 and 31) and producing neighbouring food sources (in the employed and onlooker phases, lines 11 and 21), the local improvement procedure is applied (with a probability p_{ls}) to further refine the quality of the solutions. In addition, our ABC algorithm updates at each step the best food source found so far (function Best-Solution-Found, line 35).

The ABC-MDGP algorithm requires the input of values for four parameters: t_{max} denotes the computation time limit, NP determines the number of food sources, which is equal to the number of employed and onlooker bees, *limit* controls the generation of scout bees, and p_{ls} is the probability of applying the local optimiser. The algorithm returns the best food source found so far. A detailed description of all the above-mentioned functions is provided in the following sections.

4.2. Greedy constructive method

In this section, we describe the greedy constructive method employed by ABC-MDGP to generate initial solutions. It starts with a partial solution generated by selecting m random elements and adding each element to a different group. Then, this partial solution is completed by adding a new element to it iteratively. At each step, the greedy constructive method first chooses a group g at random from among those whose current size is below its minimum capacity (a_g). If the current size of all the groups is greater than or equal to their minimum capacity, a group with a current size less than its maximum capacity (b_g) is randomly selected. Secondly, the greedy constructive method chooses the element e from the set of unassigned elements (U) so that the diversity of group g is maximised.

$$e = \text{argmax}\{D_{ig} : i \in U\}, \tag{5}$$

where D_{ig} is the contribution value of the element i to the diversity of group g, and is obtained as $D_{ig} = \sum_{j \in E_g} d_{ij}$. E_g is the set composed of the elements assigned to g. This process is repeated until the set U is empty. The fitness value of the solution obtained at each step may be calculated as follows:

$$z = \sum_{g=1}^{m} \Delta(g), \tag{6}$$

where $\Delta(g)$ is the contribution of the group g to the fitness value of the solution and is determined as:

$$\Delta(g) = \frac{1}{2} \sum_{i \in E_g} D_{ig}. \tag{7}$$

4.3. Neighbourhood operators

The Generate-Neighbouring function creates a neighbouring solution by invoking one of the following operators:

- Destructive-constructive operators (NO_1 and NO_2). We propose two neighbourhood operators for the MDGP that are to some extent inspired by the iterated greedy algorithm [33]. Specifically, they remove a number of components from the solution and apply the greedy constructive heuristic in order to construct a new complete solution. NO_1 selects those components at random, whereas NO_2 employs a more sophisticated strategy. In particular, NO_2 deletes, at each step, the element that brings less diversity to its group. The number of elements deleted by NO_1 and NO_2 is determined randomly in the set $1, \ldots, n_d$, where n_d is a parameter dubbed *destruction size*. We should point out that different neighbourhood operators based on destructive-constructive procedures have appeared before in the ABC area [39,46].

- Swap operator (NO_3). This is based on the *swap move* [4,15,38] that swaps the group assignments of a pair of elements. Specifically, NO_3 performs q swap moves to obtain a neighbouring solution, where q is randomly chosen in the set $\{1, \ldots, p\}$ (p is a parameter named *perturbation strength*).

4.4. Local improvement procedure

Different authors have proposed enhancing the exploitation capability by embedding a local improvement procedure in the ABC algorithm. For example, in [44], after completion of the ABC algorithm, an attempt is made to further improve the quality of the best solution obtained through the ABC run by using a local search. An alternative approach is suggested in [46], where the local search technique is applied to the neighbouring food source found by an employed bee with a predefined probability. The purpose of these hybridisation schemes stems from the fact that ABC carries out the global search through the exploration of the search space, whereas the local improvement procedure intensifies the search in promising regions. Therefore, a balance of global and local searches may be effectively achieved.

In this section, we present a local improvement procedure for the MDGP, which has been fused into the ABC-MDGP algorithm. This local optimiser is invoked with a probability p_{ls} immediately after a new solution is built by the Construct-Solution function or generated by the Generate-Neighbouring procedure (see Fig. 1). The local improvement procedure consists of two independent local search methods, which are executed in a pipeline fashion. Both local search methods follow the so-called first-improvement strategy, but differ in the neighbourhood operator used. The first local search method employs *movement* operations that allocate an element to a different group, providing that the size limitations are not violated. The second one uses *swap* operations that interchange two elements from two different groups.

- *Local search method based on movements.* This implements the first choice strategy that scans movements in search of the first one yielding an improvement in the objective function. It is executed until no further improvement is achieved. After performing a movement, there is no need to completely evaluate the new solution. It is sufficient to update the contribution value of the groups involved in the movement operation to obtain the fitness value of the new solution. Let y be an element belonging to the group k. If we move the element y to a new group t, we update the contribution value of the two groups affected by the movement as follows:

$$\Delta'(k) = \Delta(k) - D_{yk}, \tag{8}$$
$$\Delta'(t) = \Delta(t) + D_{yt}. \tag{9}$$

Then, the new fitness function value z' will be calculated as follows:

$$z' = z - [\Delta(k) + \Delta(t)] + [\Delta'(k) + \Delta'(t)] \tag{10}$$
$$= z - D_{yk} + D_{yt}. \tag{11}$$

After each movement, the contribution value of the elements $i \in E/y$ for groups k and t (D_{ik} and $D_{it}, \forall i \in E/y$) is updated as shown below:

$$D_{ik} = D_{ik} - d_{iy}, \tag{12}$$
$$D_{it} = D_{it} + d_{iy}. \tag{13}$$

In this way, the cost of calculating the contribution of each group to the value of the objective function is reduced.

- *Local search method based on swaps.* The first swap operation that produces an improvement in the fitness value is applied. In the same way as in the local search method based on movements, we apply a procedure for the faster calculation of the objective function after each swap operation. Let x and y be two elements belonging to groups k and t, respectively. The contribution to the fitness value of the new solutions obtained after performing a swap operation is calculated as follows (Note that contribution values D_{ij} employed in formulas 14, 15, and 17 are those obtained before updating them according to 18 and 19):

$$\Delta'(k) = \Delta(k) - D_{xk} + D_{yk} - d_{xy}, \tag{14}$$
$$\Delta'(t) = \Delta(t) - D_{yt} + D_{xt} - d_{xy}. \tag{15}$$

Then, the fitness value of the new solution is obtained in this way:

$$z' = z - [\Delta(k) + \Delta(t)] + [\Delta'(k) + \Delta'(t)] \tag{16}$$
$$= z - [D_{xk} + D_{yt}] + [D_{yk} + D_{xt}] - 2d_{xy}. \tag{17}$$

After each swap operation, the contribution values of the elements $i \in E$ for groups k and t (D_{ik} and $D_{it}, \forall i \in E$) are updated:

$$D_{ik} = D_{ik} - d_{ix} + d_{iy}, \tag{18}$$

$$D_{it} = D_{it} - d_{iy} + d_{ix}. \tag{19}$$

Palubeckis and Karciauskas [38] propose a local improvement method for MDGP that, unlike the one presented in this work, integrates swap and movement operations within a single local search procedure.

5. Computational experiments

This section describes the computational experiments that we have performed in order to assess the performance of the ABC-MDGP algorithm. First, we detail the experimental setup (Section 5.1), then, we analyse the results obtained from different experimental studies carried out with this algorithm. Our aims are: (1) to analyse the influence of the parameters and settings associated with ABC-MDGP and show the benefits of the use of the proposed neighbourhood operators and the local improvement procedure (Section 5.2) and (2) to compare the results of ABC-MDGP with those of other metaheuristic approaches for the MDGP from the literature (Section 5.3).

5.1. Experimental setup

The code of ABC-MDGP has been implemented in C and the source code has been compiled with gcc 4.6. The experiments were conducted on a computer with a 2.8 GHz Intel® Core™ i7 processor with 12 GB of RAM running Fedora™ Linux V15. We considered four groups of benchmark instances for our experiments (in total constituting 360 instances), which are described below.

- RanInt, RanReal, and Geo. This set was compiled by Gallego et al. in [15] and is available at http://www.optsicom.es/mdgp/. Each of these three sets consists of 80 instances. RanInt, RanReal, and Geo instances differ in the way that the dissimilarity between the elements is generated. For RanInt instances, the dissimilarity $d_{ij} = d_{ji}, \forall i, j = 1, \dots, n$, is a uniform random integer between $[0, 100]$. In RanReal instances, the dissimilarity between each pair of elements is a real number chosen at random from within the interval $[0, 100]$. For Geo instances, the dissimilarity between two elements is calculated by means of the Euclidean distance, with coordinates randomly generated in $[0, 10]$. The number of coordinates for each element is generated randomly in the 2–21 range. For each combination of number of elements and number of groups, there are two sets of 10 instances. In the first set (*lower bound equal to upper bound*, EB), the group size is the same for each of the 10 instances and is calculated as $a_g = b_g = n/m, \forall g = 1, \dots, m$. In the second (*lower bound different to upper bound*, DB), upper and lower limits for each of the 10 instances are chosen at random in the interval $[l_{min}, l_{max}]$ for the lower and $[u_{min}, u_{max}]$ for the upper limit. In Table 1, we show for each combination of n and m, the group size for EB instances and the values of $l_{min}, l_{max}, u_{min}$, and u_{max} for DB instances. Moreover, in this table, the maximum CPU time allotted for each kind of instance is specified.
- type1_22. This set of instances was presented in [7]. In this set, the dissimilarity d_{ij} for each pair of elements i and j is chosen uniformly at random in the interval $[1, 10]$. We have considered two different sets with 10 instances for each combination of n and m. In the first set (*lower bound equal to upper bound*, EB), the group size is fixed to a predefined value so that $a_g = b_g = n/m, \forall g = 1, \dots, m$. In the second set (*lower bound different to upper bound*, DB), the group size is bound in a certain range $a_g \leqslant |E_g| \leqslant b_g, \forall g = 1, \dots, m$. Table 2 presents the details of Type1_22 instances for each combination of n and m, specifying a_g and b_g values and the time limit.

In order to analyse the statistical significance of the experimental results, we have considered non-parametric tests, given that they do not require explicit conditions in order to be carried out [17]. In particular, we have considered two alternative non-parametric tests to analyse the experimental results:

- The first method is the application of Iman and Davenport's test [22] and Holm's method [20] as a post hoc procedure. The first test is used to see whether there are significant statistical differences among a certain group of algorithms. If such differences are detected, then Holm's test is employed to compare the best algorithm with the remaining ones.

Table 1
Details of the *RanInt*, *RanReal*, and *Geo* sets of instances.

n	m	EB	DB				Time limit (s)
		$a_g = b_g = n/m$	l_{min}	l_{max}	u_{min}	u_{max}	
120	10	12	8	12	12	16	3
240	12	20	15	20	20	25	20
480	20	24	18	24	24	30	120
960	24	40	32	40	40	48	600

Table 2
Details of the Type1_22 set of instances.

n	m	EB	DB		Time limit (s)
		$a_g = b_g = n/m$	a_g	b_g	
2000	2	1000	866	1134	1200
2000	10	200	173	227	1200
2000	25	80	51	109	1200
2000	50	40	26	54	1200
2000	100	20	13	27	1200
2000	200	10	6	14	1200

- The second method is the utilisation of the Wilcoxon matched-pairs signed-ranks test [52] to perform pairwise comparisons. In order to compute the test, we measure the difference between the performance scores of the two compared algorithms in each instance and then these differences are ranked according to their absolute values. Let $R+$ be the sum of ranks for the functions in which the first algorithm outperforms the second, and $R-$ be the sum of ranks for the reverse case. Ranks corresponding to zero differences are split evenly among the sums. If $min\{R+, R-\}$ is less than or equal to the critical value, this test detects significant differences between the algorithms, which means that an algorithm outperforms its opponent.

5.2. Study of ABC-MDGP with different parameters

In this section, we investigate the effect of the different parameters and strategies used in ABC-MDGP. First, we present a series of preliminary experiments that were conducted to set the values of the key parameters of this algorithm: $limit, n_d$ (destruction size of NO_1 and NO_2), and p (perturbation strength of NO_3). In particular, we have built 15 ABC-MDGP configurations with $limit = \{0.5n, n, 2n\}$ and $n_d = \{0.1n, 0.2n, 0.3n, 0.4n, 0.5n\}$ (we have considered the same values for p as for n_d). Note that only a few possible combinations are explored, so performance may be further improved by examining more possible combinations/values. The population size is 20 and $p_{ls} = 1$. With the purpose of fine-tuning our proposal and avoiding overfitting, we have employed only half of the instances of the four sets detailed in Section 5.1 for the study of the parameters. Specifically, we have used the first five instances of each set of 10 for the fine-tuning of ABC-MDGP. The remaining five instances of each set will be employed for the experimental comparison of ABC-MDGP with the state-of-the-art metaheuristics for the MDGP (Section 5.3). All ABC-MDGP variants studied in this section were stopped using the time limit specified in Tables 1 and 2 and one run was performed for each problem instance.

In each experiment, we compute for each instance the overall best solution value, *BestValue*, obtained by the execution of all methods under consideration. Then, for each method, we compute the relative percentage deviation between the best solution value found by the method and the *BestValue*. In Table 3, we report the average of this relative deviation (*Dev*) across all the instances considered in each particular experiment and the number of instances (*#Best*) for which the value of the best solution obtained by a given method matches *BestValue* (note that the sum of all values in column *#Best* may exceed the total number of instances due to more than one algorithm achieving the *BestValue* in some instances). We also show the average *rankings* obtained by these ABC-MDGP variants. This measure is obtained by computing, for each problem instance, the ranking r_a of the observed results for algorithm a, assigning to the best of them the ranking 1, and to the worst

Table 3
Results of ABC-MDGP with different parameter values.

ABC	Geo–RanInt–RanReal					ABC	Type1_22				
Limit	$n_d = p$	Av. ran.	Dev (%)	#Best	Holm	Limit	$n_d = p$	Av. ran.	Dev (%)	#Best	Holm
$2n$	$0.4n$	9.33	0.17	1	No	$2n$	$0.4n$	10.44	0.14	0	Yes
$0.5n$	$0.4n$	9.12	0.16	1	No	n	$0.5n$	10.36	0.14	1	Yes
n	$0.4n$	8.93	0.16	2	No	$0.5n$	$0.5n$	10.31	0.13	1	Yes
n	$0.5n$	8.65	0.17	7	No	$2n$	$0.5n$	10.27	0.13	1	Yes
$2n$	$0.5n$	8.61	0.17	7	No	$0.5n$	$0.4n$	9.93	0.13	0	Yes
$0.5n$	$0.5n$	8.56	0.17	6	No	n	$0.4n$	9.93	0.13	0	Yes
n	$0.2n$	8.5	0.13	4	No	n	$0.3n$	9.12	0.12	4	Yes
$2n$	$0.2n$	8.5	0.13	4	No	$0.5n$	$0.3n$	9.08	0.12	4	Yes
$0.5n$	$0.2n$	8.38	0.13	5	No	$2n$	$0.3n$	8.94	0.12	4	Yes
$2n$	$0.3n$	7.45	0.09	8	No	$2n$	$0.2n$	6.34	0.08	10	No
n	$0.3n$	7.43	0.09	9	No	$0.5n$	$0.2n$	6.30	0.08	10	No
$0.5n$	$0.1n$	7.02	0.16	10	No	n	$0.2n$	6.30	0.08	10	No
$2n$	$0.1n$	6.83	0.13	9	No	$2n$	$0.1n$	4.40	0.02	20	No
$0.5n$	$0.3n$	6.36	0.09	10	No	$0.5n$	$0.1n$	4.19	0.02	20	No
n	$0.1n$	6.27	0.12	10	Winner	n	$0.1n$	4.02	0.01	21	Winner

the ranking $|A|$ (A is the set of algorithms). Then, an average measure is obtained from the rankings of this algorithm for all test problems. For example, if a certain algorithm achieves rankings 1, 3, 1, 4, and 2, in five problem instances, the average ranking is $\frac{1+3+1+4+2}{5} = 2.2$. Note that the lower an average ranking is, the better its associated algorithm is.

For the results in Table 3, Iman and Davenport's test finds significant performance differences between the considered algorithms for Geo–RanInt–RanReal and Type1_22 instances with a level of significance $\alpha = 0.05$. In addition, we report the results of Holm's method indicating whether or not there are significant differences between the best ranked algorithm (*winner*) and the compared method.

We can draw the following conclusions from Table 3:

- The n_d and p parameters have the largest impact on the ABC-MDGP performance; in general, the best ranked configurations employ low values for these parameters; $n_d = p = 0.1n$ is the best configuration for both sets of instances. This shows that ABC-MDGP behaviour is better when it emphasises the local character of the neighbourhood operator.
- Holm's test shows that there are no significant differences between the best configuration and the remaining ones for Geo–RanInt–RanReal instances. However, for Type1_22 instances, Holm's test indicates that there are significant differences between the control configuration and those with a value greater than or equal to 0.3 for n_d and p. Interestingly, according to these results, once we have fixed the value of n_d and p, the *limit* value does not seem to greatly affect the results of the considered configuration, which is in line with the conclusions outlined in the above paragraph.
- We choose $limit = n$ and $n_d = p = 0.1n$ for all the remaining experiments, because the best ranked configuration employs this setting.

Our next empirical study was performed to show the merit of the three neighbourhood operators invoked inside ABC-MDGP. Specifically, we study the quality of the results obtained by NO_1, NO_2, and NO_3 when they work in isolation (the same neighbourhood operator is used along the whole run) and in combination (the Generate-Neighbouring function randomly invokes a neighbourhood operator from the considered set). The remaining parameters are set as follows: $limit = n, n_d = p = 0.1n, NP = 20$, and $p_{ls} = 1$. Iman and Davenport's test result shows significant performance differences between the considered algorithms for both sets of instances (level of significance $\alpha = 0.05$). Table 4 reports the results for the different combinations.

We can make the following observations from Table 4:

- For the set with the largest instances, Type1_22, the best outcomes (rankings) are obtained when the three operators are jointly applied.
- For instances RanInt, RanReal, and Geo, the NO_2 operator obtains very good results when applied in isolation. We should also point out that this operator shows remarkable values for the measures *Dev* and *#Best* in Type1_22 instances. NO_2 is a destruction-construction operator that drops those elements which provide a lower contribution to the diversity of the group. These results show that the NO_2 strategy is highly effective in locating promising neighbouring solutions.
- Moreover, we should highlight the good results achieved by the combination of the NO_2 operator (high intensification level) with the operator that presents the highest diversification level, NO_3. This combination attains the second best ranking for both RanInt–RanReal–Geo instances and Type1_22 ones.
- The performance of NO_1 in isolation is quite poor. However, if we combine NO_1 with other operators, the combined neighbouring operator presents a positive synergy enabling a relatively robust performance in the two kinds of instances.
- With regards to Holm's test results, we may emphasise that the best configuration only shows statistically significant, better results than those with the isolated NO_1 operator for Type1_22 instances and those with NO_1 and NO_3 operators applied in isolation or combined for Geo–RanInt–RanReal instances. Thus, the Holm's test results confirm, as indicated above, that the NO_2 operator is a key element when designing an efficient neighbouring operator for both sets of instances.

Taking into account the above conclusions drawn from Table 4, we have chosen the combination of the three different neighbouring operators (NO_1, NO_2, and NO_3) in ABC-MDGP. We should point out that the simultaneous application of different neighbourhood operators has been considered by different authors in the ABC field [39,46].

Table 4
Results of ABC-MDGP with different NO combinations.

ABC	Geo–RanInt–RanReal				ABC	Type1_22			
Comb. NOs	Av. ran.	Dev (%)	#Best	Holm	Comb. NOs	Av. ran.	*Dev* (%)	#Best	Holm
NO_1	6.66	0.54	0	Yes	NO_1	5.11	0.11	5	Yes
NO_1 NO_3	4.27	0.14	3	Yes	NO_3	4.51	0.15	3	No
NO_3	3.87	0.11	6	Yes	NO_1 NO_2	4.25	0.10	1	No
NO_1 NO_2	3.73	0.18	9	No	NO_2	3.90	0.08	7	No
NO_1 NO_2 NO_3	3.33	0.11	3	No	NO_1 NO_3	3.63	0.11	7	No
NO_2 NO_3	3.12	0.10	7	No	NO_2 NO_3	3.34	0.11	7	No
NO_2	2.98	0.13	10	Winner	NO_1 NO_2 NO_3	3.23	0.09	6	Winner

Table 5
Results of ABC-MDGP with different p_{ls} values.

ABC	Geo–Ranint–Ranreal				ABC	Type1_22			
p_{ls}	Av. ran.	Dev (%)	#Best	Holm	p_{ls}	Av. ran.	Dev (%)	#Best	Holm
0	6	3.67	0	Yes	0	6	3.59	0	Yes
0.1	3.8056	0.17	2	No	0.1	3.3611	0.06	4	No
0.5	2.9028	0.10	12	No	0.5	3.1389	0.07	7	No
0.25	2.8889	0.08	6	No	0.75	3.0972	0.06	6	No
0.75	2.8611	0.11	6	No	0.25	3.0278	0.05	6	No
1	2.5417	0.08	10	Winner	1	2.375	0.03	14	Winner

Table 6
ABC-MDGP versus MS-LS (critical value = 208).

Inst. set	Algorithm	Dev (%)	#Best	$R+$	$R-$	Diff?
Geo–RanInt–RanReal	MS-LS	0.80	2	624	42	+
	ABC-MDGP	0.01	34			
Type1_22	MS-LS	0.88	0	666	0	+
	ABC-MDGP	0	36			

Now, we undertake a study to analyse the influence of the local improvement procedure on the performance of ABC-MDGP. Specifically, we investigate the effects of varying the p_{ls} parameter associated with the proposed local improvement procedure. In order to do this, we have investigated the performance of 6 ABC-MDGP configurations with $p_{ls} = \{0, 0.1, 0.25, 0.5, 0.75, 1\}$. The other parameters are fixed according to the above studies: $limit = n, n_d = p = 0.1n, NP = 20$, and a neighbouring operator combining NO_1, NO_2, and NO_3. Table 5 has the results of these algorithms.

The results in Table 5 reveal the clear superiority of the ABC-MDGP versions that apply a local improvement procedure ($p_{ls} > 0$) to both types of instances, which demonstrates the effectiveness of incorporating it into our ABC-MDGP algorithm. In addition, we observe that the best results are achieved with the highest values for p_{ls}, and, in particular, with $p_{ls} = 1$. Iman and Davenport's test reveals the existence of significant differences between the compared configurations for a level of significance $\alpha = 0.05$. Moreover, the Holm's test results outlined in this table show that there are only significant differences between the best configuration ($p_{ls} = 1$) and the configurations that do not employ a local search.

Although high p_{ls} values may seem to imbalance the relationship between exploration (global search) and exploitation in ABC-MDGP (because most computational time is spent on the local search), the previous study has shown that applying a local search to all generated solutions is the best performing option. This may suggest that the good results obtained by the proposal are mainly due to the local search procedure (the other ABC parts not being so important). An interesting idea could therefore be to compare the performance of ABC-MDGP to that of a multi-start local search algorithm (MS-LS) that only applies a local search to randomly generated solutions, returning the best solution found over all the starts.

In order to detect the differences between ABC-MDGP and MS-LS, we have applied Wilcoxon's test. Table 6 summarises the results of this procedure with a level of significance $\alpha = 0.05$. If $R-$ (associated with MS-LS) is smaller than $R+$ (associated with ABC-MDGP) and the critical value, ABC-MDGP is statistically better than MS-LS; if $R+$ is inferior to $R-$ and the critical value, ABC-MDGP is statistically worse than its competitor; if neither $R+$ nor $R-$ is smaller than the critical value, Wilcoxon's test does not find statistical differences. The last column indicates whether ABC-MDGP performs statistically better (+) than, worse ($-$) than, or without significant differences ($\sim$) to MS-LS.

Table 6 shows the clear advantage of ABC-MDGP over MS-LS. Specifically, its *#Best* and *Dev* results are much better and, according to Wilcoxon's test, it is the winner. This result justifies the design efforts dedicated to building all the constituent phases of our ABC algorithm.

5.3. ABC-MDGP vs. state-of-the-art Metaheuristic for the MDGP

In this section, we undertake a comparative analysis of our proposal with the current best algorithms for the MDGP, the tabu search with strategic oscillation (TS-SO) [15], simulated annealing (MSA) [38], the hybrid genetic algorithm (HGA) [38], the variable neighbourhood search (VNS) [38], and the memetic algorithm (LSGA) [10]. We have used the implementation of TS-SO and LSGA developed by Gallego et al. [15]. This implementation is available at http://www.optsicom.es/mdgp/. The code for MSA, HGA, and VNS was also provided by the authors. We should point out that ABC-MDGP and its competitors were run under the same computational conditions (in order to enable a fair comparison between them) in the second half of instances listed in Section 5.1 (180 different instances), which were not used for the fine-tuning of ABC-MDGP in Section 5.2, in order to avoid overfitting. These algorithms were run once on each problem instance and the cutoff time for each run was that specified in Tables 1 and 2.

Table 7
Comparison by problem size on the RanInt, Geo, RanReal sets (critical value = 137).

Inst. size	Algorithm	Dev	#Best	$R+$	$R-$	Diff?
$n = 120$	TS-SO	0.45	0	449.0	16.0	+
	MSA	0.37	3	426.0	39.0	+
	VNS	0.31	8	376.0	89.0	+
	HGA	0.23	3	384.0	81.0	+
	LSGA	1.22	0	458.0	7.0	+
	ABC-MDGP	0.08	16			
$n = 240$	TS-SO	0.23	3	331.0	134.0	+
	MSA	0.22	2	369.0	96.0	+
	VNS	0.29	8	327.0	138.0	∼
	HGA	0.24	6	361.0	104.0	+
	LSGA	1.08	1	458.0	7.0	+
	ABC-MDGP	0.12	10			
$n = 480$	TS-SO	0.27	8	420.0	45.0	+
	MSA	0.19	0	460.0	5.0	+
	VNS	0.37	1	449.0	16.0	+
	HGA	0.65	0	465.0	0.0	+
	LSGA	1.40	0	465.0	0.0	+
	ABC-MDGP	0.02	21			
$n = 960$	TS-SO	0.22	8	404.0	61.0	+
	MSA	0.20	4	420.0	45.0	+
	VNS	0.26	3	432.0	33.0	+
	HGA	0.67	0	465.0	0.0	+
	LSGA	1.27	0	465.0	0.0	+
	ABC-MDGP	0.03	15			

Table 8
Comparison on the RanInt, Geo, RanReal sets depending on the group size configuration (critical value = 52).

Inst. set	Algorithm	Dev	#Best	$R+$	$R-$	Diff?
RanInt EB	TS-SO	0.29	3	175.0	35.0	+
	MSA	0.37	0	199.0	11.0	+
	VNS	0.32	2	192.0	18.0	+
	HGA	0.53	5	175.0	35.0	+
	LSGA	1.05	1	201.5	8.5	+
	ABC-MDGP	0.11	9			
RanInt DB	TS-SO	0.35	2	180.0	30.0	+
	MSA	0.26	3	190.0	20.0	+
	VNS	0.42	1	195.0	15.0	+
	HGA	0.69	1	205.0	5.0	+
	LSGA	2.11	0	210.0	0.0	+
	ABC-MDGP	0.07	13			
Geo EB	TS-SO	0.01	8	83.0	127.0	∼
	MSA	0.03	0	195.0	15.0	+
	VNS	0.01	6	85.0	125.0	∼
	HGA	0.04	0	210.0	0.0	+
	LSGA	0.04	0	210.0	0.0	+
	ABC-MDGP	0.01	6			
Geo DB	TS-SO	0.36	1	207.0	3.0	+
	MSA	0.02	4	141.0	69.0	∼
	VNS	0.44	3	202.0	8.0	+
	HGA	0.09	0	210.0	0.0	+
	LSGA	0.85	0	210.0	0.0	+
	ABC-MDGP	0.02	12			
RanReal EB	TS-SO	0.29	4	141.0	69.0	∼
	MSA	0.34	1	203.0	7.0	+
	VNS	0.34	5	170.0	40.0	+
	HGA	0.59	2	195.0	15.0	+
	LSGA	1.16	0	210.0	0.0	+
	ABC-MDGP	0.11	8			
RanReal DB	TS-SO	0.46	1	203.0	7.0	+
	MSA	0.45	1	206.0	4.0	+
	VNS	0.32	3	183.0	27.0	+
	HGA	0.74	1	204.0	6.0	+
	LSGA	2.24	0	210.0	0.0	+
	ABC-MDGP	0.06	14			

Table 9
Comparison on 60 instances belonging to the Type1_22 set (critical value = 137).

Inst. set	Algorithm	Dev (%)	#Best	*R*+	*R*−	Diff?
Type1_22 EB	TS-SO	0.50	3	453	12	+
	VNS	0.18	4	449.5	15.5	+
	LSGA	1.47	0	465	0	+
	ABC-MDGP	0.01	23			
Type1_22 DB	TS-SO	0.39	0	460	5	+
	VNS	0.14	4	400	35	+
	LSGA	3.18	0	465	0	+
	ABC-MDGP	0.01	27			

In order to detect the differences between ABC-MDGP and the other algorithms, we have applied Wilcoxon's test. Tables 7 and 8 have the *#Best* and *Dev* measures and summarise the results of Wilcoxon's test with a level of significance $\alpha = 0.05$, where the values of $R+$ (associated with ABC-MDGP) and $R-$ of the test are specified. The last column indicates whether Wilcoxon's test found statistical differences between these algorithms.

Tables 7 and 8 show the results of the different algorithms on the RanInt, RanReal, and Geo instances from two different points of view. In Table 7, results are grouped by problem size, whereas in Table 8 results are displayed by instance type (RanInt, RanReal, and Geo) and by configuration of the bounds (EB and DB). From results in these tables, we notice that ABC-MDGP obtains improvements with respect to the other algorithms on the RanInt, RanReal, and Geo instances (see the metrics *Dev* and *#Best*). These improvements are statistically significant (because $R-$ values are lower than both $R+$ values and the critical value) in the majority of the comparisons performed: specifically in 45 out of 50 cases. In the remaining ones, Wilcoxon's test does not find significant differences among the algorithms compared. Thus, we can conclude that ABC-MDGP overcomes its competitors in the RanInt, RanReal, and Geo instances.

Moreover, in Table 9, we show the results of the experimental comparison on the set of largest instances. For this study, we have compared TS-SO, VNS (the best metaheuristic in [38] in large instances), LSGA, and ABC-MDGP on the Type1_22 set. According to the results in Table 9, ABC-MDGP again significantly outperforms its competitors on the set formed by the largest instances.

This experimental analysis shows that our algorithm is currently a state-of-the-art method for solving the MDGP, which evidences the great potential of the ABC metaheuristic as a search algorithm for this problem.

6. Conclusions

In this paper, the ABC-MDGP algorithm, based on mimicking the food foraging behaviour of honeybee swarms, is proposed as a method for solving the MDGP. Our algorithm employs an initialisation scheme based on a greedy constructive method proposed for this problem and two neighbourhood operators inspired by the iterated greedy metaheuristic, which were combined with another based on swap moves. In addition, the ABC approach has been enhanced by incorporating a local improvement routine that exploits two simple local modifications of the solutions (movement and swap operations). The designed algorithm has proven to be a very high performing algorithm for the MDGP, showing itself to be very competitive with respect to state-of-the-art algorithms. These results, along with the simplicity and flexibility of the ABC approach, allow us to conclude that this metaheuristic arises as a tool of choice to face this problem.

We believe that the ABC framework presented in this paper is a significant contribution, worthy of future study. We intend to explore the following two interesting avenues of research:

- *The employment of iterated-greedy based neighbourhood operators in ABC approaches dealing with other challenging optimisation problems*. Iterated greedy has exhibited state-of-the-art performances for a considerable number of problems such as scheduling in parallel machines [11] and flowshop scheduling [14]. Thus, the development of ABC methods with iterated-greedy based neighbourhood operators for these problems seems to be a promising research field.
- *The study of different schemes for parallel implementations of ABC-MDGP*. Parallel hardware (multicore processors, clusters, etc.) has become very affordable and widely available nowadays, favouring the parallel implementation of ABC methods [48]. We intend to implement our ABC-MDGP approach on parallel hardware, following different schemes proposed in the literature such as master–slave, multi-hive with migrations, and hybrid hierarchical [40], in order to improve results through the speed-up of the search process. These improvements are especially important when dealing with large-size instances.

Acknowledgments

We would like to thank Professors M. Gallego, M. Laguna, R. Martí and A. Duarte for providing TS-SO and LSGA implementation and Professors G. Palubeckis and A. Karciauskas for supplying the MSA, HGA, and VNS codes.

This work was supported by the Research Projects TIN2011–24124 and P08-TIC-4173.

References

[1] B. Akay, D. Karaboga, A modified artificial bee colony algorithm for real-parameter optimization, Information Sciences 192 (2012) 120–142.
[2] R. Akbari, R. Hedayatzadeh, K. Ziarati, B. Hassanizadeh, On the performance of artificial bee colony (ABC) algorithm, Swarm and Evolutionary Computation 2 (2012) 39–52.
[3] T. Arani, V. Lofti, A three phased approach to final exam scheduling, IIE Transactions 21 (1989) 86–96.
[4] K. Baker, S. Powell, Methods for assigning students to groups: a study of alternative objective functions, Journal of the Operational Research Society 53 (2002) 397–404.
[5] B. Basturk, D. Karaboga, An artificial bee colony (ABC) algorithm for numeric function optimization, in: Proceedings of the IEEE swarm intelligence, symposium, 2006, pp. 12–14.
[6] J. Bhadury, E. Mighty, H. Damar, Maximizing workforce diversity in project teams: a network flow approach, Omega 28 (2000) 143–153.
[7] A. Duarte, R. Martí, Tabu search and grasp for the maximum diversity problem, European Journal of Operational Research 178 (2007) 71–84.
[8] E.E. Falkenauer, Genetic Algorithms and Grouping Problems, John Wiley & Sons, Inc., New York, NY, USA, 1998.
[9] M. El-Abd, Performance assessment of foraging algorithms vs. evolutionary algorithms, Journal of the Operational Research Society 182 (2012) 243–263.
[10] Z. Fan, Y. Chen, J. Ma, S. Zeng, A hybrid genetic algorithmic approach to the maximally diverse grouping problem, Journal of the Operational Research Society 62 (2010) 92–99.
[11] L. Fanjul-Peyro, R. Ruiz, Iterated greedy local search methods for unrelated parallel machine scheduling, European Journal of Operational Research 207 (2010) 55–69.
[12] T. Feo, O. Goldschmidt, M. Khellaf, One-half approximation algorithms for the k-partition problem, Operations Research 40 (1992) 170–173.
[13] T. Feo, M. Khellaf, A class of bounded approximation algorithms for graph partitioning, Networks 20 (1990) 181–195.
[14] J. Framinan, R. Leisten, A multi-objective iterated greedy search for flowshop scheduling with makespan and flowtime criteria, OR Spectrum 30 (2008) 787–804.
[15] M. Gallego, M. Laguna, R. Martí, A. Duarte, Tabu search with strategic oscillation for the maximally diverse grouping problem, Journal of Operational Research Society (in press), <http://dx.doi.org/10.1057/jors.2012.66>.
[16] W.F. Gao, S. Liu, L. Huang, A global best artificial bee colony algorithm for global optimization, Journal of Computational and Applied Mathematics 236 (2012) 2741–2753.
[17] S. Garcia, D. Molina, M. Lozano, F. Herrera, A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 special session on real parameter optimization, Journal of Heuristics 15 (2008) 617–644.
[18] D. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley Longman Publishing Co., 1989.
[19] S. Hettich, M.J. Pazzani, Mining for proposal reviewers: lessons learned at the national science foundation, in: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, 2006, pp. 862–871.
[20] S. Holm, A simple sequentially rejective multiple test procedure, Scandinavian Journal of Statistics 6 (1979) 65–70.
[21] T. Hsieh, H. Hsiao, W. Yeh, Forecasting stock markets using wavelet transforms and recurrent neural networks: an integrated system based on artificial bee colony algorithm, Applied Soft Computing 11 (2011) 2510–2525.
[22] R. Iman, J. Davenport, Approximations of the critical region of the Friedman statistic, in: Communications in, Statistics, 1980, pp. 571–595.
[23] F. Kang, J. Li, Z. Ma, Rosenbrock artificial bee colony algorithm for accurate global optimization of numerical functions, Information Sciences 181 (2011) 3508–3531.
[24] D. Karaboga, An idea based on honey bee swarm for numerical optimization, Technical Report TR06, Computer Engineering Department, Erciyes University, Turkey, 2005.
[25] D. Karaboga, B. Akay, A modified artificial bee colony (ABC) algorithm for constrained optimization problems, Applied Soft Computing 11 (2011) 3021–3031.
[26] D. Karaboga, B. Basturk, A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm, Journal of Global Optimization 39 (2007) 459–471.
[27] D. Karaboga, B. Basturk, On the performance of artificial bee colony (ABC) algorithm, Applied Soft Computing 8 (2008) 687–697.
[28] D. Karaboga, B. Gorkemli, C. Ozturk, N. Karaboga, A comprehensive survey: artificial bee colony (ABC) algorithm and applications, Artificial Intelligence Review (in press) 1–37, <http://dx.doi.org/10.1007/s10462-012-9328-0>.
[29] D. Karaboga, C. Ozturk, A novel clustering approach: artificial bee colony (ABC) algorithm, Applied Soft Computing 11 (2011) 652–657.
[30] D. Karaboga, C. Ozturk, N. Karaboga, B. Gorkemli, Artificial bee colony programming for symbolic regression, Information Sciences 209 (2012) 1–5.
[31] M. Kashan, N. Nahavandi, A. Kashan, DisABC: a new artificial bee colony algorithm for binary optimization, Applied Soft Computing 12 (2012) 342–352.
[32] J. Kral, To the problem of segmentation of a program, Information Processing Machines 2 (1965) 116–127.
[33] M. Lozano, D. Molina, C. García-Martínez, Iterated greedy for the maximum diversity problem, European Journal of Operational Research 214 (2011) 31–38.
[34] M. Ma, J. Liang, M. Guo, Y. Fan, Sar image segmentation based on artificial bee colony algorithm, Applied Soft Computing 11 (2011) 5205–5214.
[35] V. Manoj, E. Elias, Artificial bee colony algorithm for the design of multiplier-less nonuniform filter bank transmultiplexer, Information Sciences 192 (2012) 193–203.
[36] F. O'Brien, J. Mingers, A heuristic algorithm for the equitable partitioning problem, Omega 45 (1997) 215–223.
[37] S. Omkar, J. Senthilnath, R. Khandelwal, G. Naik, S. Gopalakrishnan, Artificial bee colony (ABC) for multi-objective design optimization of composite structures, Applied Mathematics and Computation 11 (2011) 489–499.
[38] G. Palubeckis, A.R.E. Karciauskas, Comparative performance of three metaheuristic approaches for the maximally diverse grouping problem, Information Technology and Control 40 (2011) 277–285.
[39] Q.K. Pan, M. Tasgetiren, P. Suganthan, T. Chua, A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem, Information Sciences 181 (2011) 2455–2468.
[40] R.S. Parpinelli, C. Vargas Benitez, H. Lopes, Parallel approaches for the artificial bee colony algorithm, in: B. Panigrahi, Y. Shi, M.H. Lim (Eds.), Handbook of Swarm Intelligence, vol. 8 of Adaptation, Learning, and, Optimization, 2011, pp. 329–345.
[41] M. Saleem, G.A.D. Caro, M. Farooq, Swarm intelligence based routing protocol for wireless sensor networks: survey and future directions, Information Sciences 181 (2011) 4597–4624.
[42] M. Saleem, I. Ullah, M. Farooq, Beesensor: an energy-efficient and scalable routing protocol for wireless sensor networks, Information Sciences 200 (2012) 38–56.
[43] A. Singh, An artificial bee colony algorithm for the leaf-constrained minimum spanning tree problem, Applied Soft Computing 9 (2009) 625–631.
[44] S. Sundar, A. Singh, A swarm intelligence approach to the quadratic minimum spanning tree problem, Information Sciences 180 (2010) 3182–3191.
[45] P. Tapkan, L. Ozbakir, A. Baykasoglu, Modeling and solving constrained two-sided assembly line balancing problem via bee algorithms, Applied Soft Computing 12 (2012) 3343–3355.
[46] M.F. Tasgetiren, Q.K. Pan, P. Suganthan, A.L. Chen, A discrete artificial bee colony algorithm for the total flowtime minimization in permutation flow shops, Information Sciences 181 (2011) 3459–3475.
[47] V. Tereshko, Reaction-diffusion model of a honeybee colonys foraging behaviour, in: M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J.J. Merelo, H.P. Schwefel, (Eds.), Parallel Problem Solving from Nature PPSN VI, LNCS, vol. 1917, 2000, pp. 807–816.
[48] C. Vargas Benítez, H. Lopes, Parallel artificial bee colony algorithm approaches for protein structure prediction using the 3DHP-SC model, in: M. Essaaidi, M. Malgeri, C. Badica, (Eds.), Intelligent Distributed Computing IV, vol. 315 of Studies in, Computational Intelligence, 2010, pp. 255–264.

[49] R. Weitz, S. Lakshminarayanan, An empirical comparison of heuristic and graph theoretic methods for creating maximally diverse groups, VLSI design and exam scheduling, Omega 25 (1997) 473–482.
[50] R. Weitz, S. Lakshminarayanan, An empirical comparison of heuristic methods for creating maximally diverse groups, Journal of Operation Research Society 49 (1998) 635–646.
[51] R.R. Weitz, M.T. Jelassi, Assigning students to groups: a multi-criteria decision support system approach, Decision Sciences 23 (1992) 746–757.
[52] F. Wilcoxon, Individual comparisons by ranking methods, Biometrics 1 (1945) 80–83.
[53] H.K. Yeoh, M.I. Mohamad Nor, An algorithm to form balanced and diverse groups of students, Computer Applications in Engineering Education 19 (2011) 582–590.