Knowledge-Based Systems 54 (2013) 103–113

Contents lists available at ScienceDirect

Knowledge-Based Systems

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

A hybrid metaheuristic for the cyclic antibandwidth problem

Manuel Lozano [a,*], Abraham Duarte [b], Francisco Gortázar [b], Rafael Martí [c]

[a] Department of Computer Science and Artificial Intelligence, University of Granada, Granada 18071, Spain
[b] Department of Computer Science, University Rey Juan Carlos, Madrid, Spain
[c] Department of Statistics and Operations Research, University of Valencia, Valencia, Spain

ARTICLE INFO

Article history:
Received 31 January 2013
Received in revised form 19 July 2013
Accepted 18 August 2013
Available online 12 September 2013

Keywords:
Artificial bee colony
Tabu search
Local search
Cyclic antibandwidth problem.

ABSTRACT

In this paper, we propose a hybrid metaheuristic algorithm to solve the cyclic antibandwidth problem. This hard optimization problem consists of embedding an n-vertex graph into the cycle C_n, such that the minimum distance (measured in the cycle) of adjacent vertices is maximized. It constitutes a natural extension of the well-known antibandwidth problem, and can be viewed as the dual problem of the cyclic bandwidth problem.

Our method hybridizes the artificial bee colony methodology with tabu search to obtain high-quality solutions in short computational times. Artificial bee colony is a recent swarm intelligence technique based on the intelligent foraging behavior of honeybees. The performance of this algorithm is basically determined by two search strategies, an initialization scheme that is employed to construct initial solutions and a method for generating neighboring solutions. On the other hand, tabu search is an adaptive memory programming methodology introduced in the eighties to solve hard combinatorial optimization problems. Our hybrid approach adapts some elements of both methodologies, artificial bee colony and tabu search, to the cyclic antibandwidth problem. In addition, it incorporates a fast local search procedure to enhance the local intensification capability. Through the analysis of experimental results, the highly effective performance of the proposed algorithm is shown with respect to the current state-of-the-art algorithm for this problem.

1. Introduction

The cyclic antibandwidth (CAB) problem consists of embedding an n-vertex graph into the cycle C_n, such that the minimum distance (measured in the cycle) of adjacent vertices is maximized [25] and is known as a NP-hard problem [32]. The CAB problem has been exactly solved for some specific classes of graphs like paths [40], cycles [40], two dimensional meshes (Cartesian product of two paths), tori (Cartesian product of two cycles), and asymptotic results are obtained for hypercube graphs [32]. Dobrev et al. [6] extended these results to the case of Hamming graphs (Cartesian product of d-complete graphs). However, because this problem is NP-hard, for most instances one must resort to metaheuristics to obtain near optimal solutions within reasonable time. To the best of our knowledge, only one of such optimization techniques has been presented for finding the cyclic antibandwidth of general graphs, the *memetic algorithm* by Bansal and Srivastava [4]. The great interest to develop more efficient optimization algorithms for solving the CAB problem led us to propose a hybrid metaheuristic that combines the effective artificial bee colony methodology (ABC) [17,18] with tabu search (TS) [13], and that also integrates a fast local search routine where the neighborhood is visited in an intelligent way.

The ABC algorithm is a new population-based metaheuristic approach inspired by the intelligent foraging behavior of honeybee swarm. In essence, it implements memory structures based on the analogy with a bee population. Inspired by the types of bees and their different behavior this methodology considers different elements in the algorithm. In particular, 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 such solution (represented by an objective function value). Each bee-class symbolizes a particular operation for generating new candidate food source positions. Specifically, employed bees search the food around the food source in their memory; meanwhile they pass their food information to onlooker bees. Onlooker bees tend to select good food sources from those found by the employed bees, and then they search for food around the selected source. Scout bees are translated from a few employed bees, which abandon their food sources and search new ones. We should remark that previous studies on bee algorithms showed that they provide efficiency in solving optimization problems [43].

* Corresponding author. Tel.: +34 958244258; fax: +34 958 243317.
E-mail addresses: lozano@decsai.ugr.es (M. Lozano), abraham.duarte@urjc.es (A. Duarte), francisco.gortazar@urjc.es (F. Gortázar), rmarti@uv.es (R. Martí).

0950-7051/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.knosys.2013.08.026

From an algorithmic point of view, we can say that ABC is a population-based method with memory structures. Although this methodology has been recently proposed [17], the use of memory structures in optimization algorithms can be traced back to 1986 when Glover [12] introduced the TS methodology. Genetic algorithms (GAs) probably constitute one of the most successful memory-less methods based on semi-random sampling. It is a population based methodology proposed in the seventies in which solutions are selected from a population according to their fitness, to form a new population by means of some operators inspired by the natural evolution. From this perspective, if we focus on the algorithm elements, we can view ABC as a hybrid metaheuristic combining the elements of population based methods, such as GAs, and memory-based methods, such as TS. In this paper, we explore this hybrid perspective taking some advanced TS elements (such as the ejection chains) and integrating them in an population based procedure.

The rest of this paper is organized as follows. Section 2 introduces the CAB problem in detail. Section 3 gives a brief overview of the ABC algorithm and TS. Section 4 describes our hybrid ABC approach for the CAB problem. Section 5 provides an analysis of the performance of the proposed ABC and a comparison with the existing literature. Finally, Section 6 contains a summary of results and conclusions.

2. The cyclic antibandwidth problem

Let $G(V,E)$ be an undirected and unweighted graph, where V represents the set of vertices (with $|V| = n$) and E represents the set of edges (with $|E| = m$). A labeling φ of the vertices of G is a bijective function from V to the set of integers $\{1, \ldots, n\}$ where each vertex $v \in V$ receives a unique label $\varphi(v) \in \{1, \ldots, n\}$. A circular arrangement of a labeling, simply called circular labeling, arranges the vertices of the graph in a cycle C_n where the last vertex (the one with label n) is next to the first vertex (the one with label 1). Given a circular labeling φ, let us define the clockwise distance $d^+(u,v) = |\varphi(u) - \varphi(v)|$ with $(u,v) \in E$ and, similarly the counterclockwise distance $d^-(u,v) = n - |\varphi(u) - \varphi(v)|$ with $(u,v) \in E$. Then, for a given circular labeling φ, the cyclic antibandwidth of G, referred to as $CAB(G,\varphi)$, is computed as follows:

$$CAB(G,\varphi) = \min_{(u,v)\in E}\{d^+(u,v), d^-(u,v)\}. \tag{1}$$

The CAB problem consists of maximizing the value of $CAB(G,\varphi)$ over the set Π of all possible labelings:

$$CAB(G) = \max_{\varphi\in\Pi} CAB(G,\varphi). \tag{2}$$

Note that in optimization terms, any labeling φ of G is a solution of the CAB problem stated in (2), with an objective function value, simply called value or $CAB(G,\varphi)$, defined in (1). The optimal solution(s) is therefore the labeling, or labelings, with maximum value.

Fig. 1-left shows an example of a graph G with 8 vertices and 8 edges. Fig. 1-right shows a circular labeling φ of G, arranging the vertices in a cycle. Additionally, clockwise (d^+) and counterclockwise (d^-) distances for each edge are shown in Table 1. In particular, each row of this table reports the distance between each pair of adjacent vertices (those joined with an edge). For example, the clockwise distance between vertex A and B is $d^+(A,B) = |\varphi(A) - \varphi(B)| = |7 - 8| = 1$. Similarly, the counterclockwise distance between these two vertices is $d^-(A,B) = 8 - |\varphi(A) - \varphi(B)| = 8 - |7 - 8| = 7$. In order to compute $CAB(G,\varphi)$, we evaluate d^+ and d^- for the remaining 7 edges (shown in Table 1), reporting the minimum of all of them. Therefore, $CAB(G,\varphi) = 1$.

CAB is a natural extension of the antibandwidth problem [3,7]. Although these two optimization problems are related, we should not expect a method developed for the former problem to perform well on the latter. We illustrate this fact by considering the example shown in Fig. 2, which corresponds to a caterpillar $P_{5,4}$ graph. A caterpillar P_{n_1,n_2} is constructed using the path P_{n_1} and n_1 copies of the path P_{n_2}, where each vertex i in P_{n_1} is connected to the first vertex of the i-th copy of the path P_{n_2}. For such instance we apply the RBFS constructive procedure by Bansal and Srivastava [4] to generate 10^6 labelings (solutions) and compute for each one the objective function value for the cyclic antibandwidth (CAB) according to (1), and the objective function value of the antibandwidth problem (AB) according to Duarte et al. [7]. The correlation between both values computed over the 10^6 solutions is rather small ($r = 0.13$). In addition, the labeling with maximum AB value, out of the 10^6 generated, is

$$(0, 12, 3, 16, 6, 10, 1, 11, 2, 13, 4, 14, 5, 15, 7, 19, 9, 17, 8, 18),$$

while the labeling with maximum CAB value is

$$(16, 5, 14, 2, 11, 7, 19, 9, 17, 8, 18, 4, 15, 6, 12, 3, 13, 1, 10, 0).$$

We can see that both labelings are very different.

The CAB problem was proved to be NP-hard in Raspaud et al. [32]. This problem was originally introduced in Leung et al. [25] in connection with multiprocessor scheduling problems. It has been found to be relevant in allocating time slots for different sensors in a network such that two sensors that interfere each other have a large time interval between their periods of operation

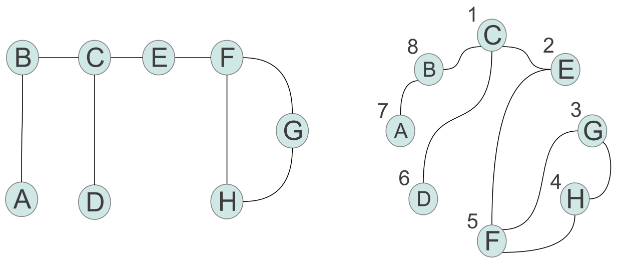

Fig. 1. A graph and a circular labeling layout.

Table 1
Clockwise and counterclockwise distances for the graph depicted in Fig. 1.

d^+	d^-
$d^+(A,B) = \|7 - 8\| = 1$	$d^-(A,B) = 8 - \|7 - 8\| = 7$
$d^+(B,C) = \|8 - 1\| = 7$	$d^-(B,C) = 8 - \|8 - 1\| = 1$
$d^+(C,D) = \|1 - 6\| = 5$	$d^-(C,D) = 8 - \|1 - 6\| = 3$
$d^+(C,E) = \|1 - 2\| = 1$	$d^-(C,E) = 8 - \|1 - 2\| = 7$
$d^+(E,F) = \|2 - 5\| = 3$	$d^-(E,F) = 8 - \|2 - 5\| = 5$
$d^+(F,G) = \|5 - 3\| = 2$	$d^-(F,G) = 8 - \|5 - 3\| = 6$
$d^+(F,H) = \|5 - 4\| = 1$	$d^-(F,H) = 8 - \|5 - 4\| = 7$
$d^+(G,H) = \|3 - 4\| = 1$	$d^-(G,H) = 8 - \|3 - 4\| = 7$

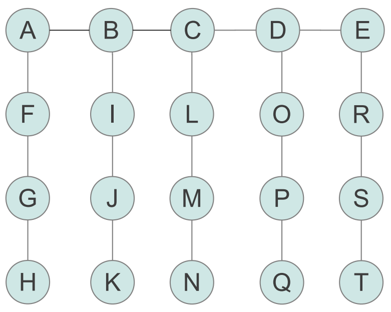

Fig. 2. A caterpillar $P_{5,4}$ graph.

[4]. The associated decision version of this problem consists of determining if $CAB(G,\varphi)$ is larger than a given value k. Specifically, when $k = 1$ the problem is equivalent to the existence of a Hamiltonian cycle in the complement of G. This belongs to standard textbooks on complexity and it is well known as a King Arthur's round table problem: "Is it possible to place knights around a table such that no two enemies are neighboring?" [40].

3. Methods: ABC and TS

In this section, we give a background of the ABC algorithm and TS, which set the stage for the later specific algorithmic design we employ to face the CAB problem.

3.1. The artificial bee colony algorithm

In a real bee colony, there are some tasks done by specialized individuals. Bees try to maximize the nectar amount unloaded to the food stores in the hive by this division of labor and self-organization. Division of labor and self-organization 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 [41]. Employed bees are responsible for exploiting the nectar sources explored before and they give information to the onlooker bees in the hive about the quality of the food source which they are exploiting. The onlookers tend to select good food sources from those found by the employed bees, and then further search food around the selected food source. Specifically, employed bees share information about food sources by dancing in a common area in the hive called 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 nature of real bees, if a food source is not worth exploiting anymore, it is abandoned by the bees, and the employed bee of that source becomes a scout searching the environment randomly or by an internal motivation. Scout bees perform the job of exploration, whereas employed and onlooker bees perform the job of exploitation.

Motivated by this foraging behavior of honeybees, Karaboga et al. [17] proposed the ABC algorithm. In this algorithm, the position of a food source represents a solution of the optimization 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 for good food sources is the process used to find the optimal solution. The algorithm begins with a population of randomly distributed positions of food sources. The number of employed bees is equal to the number of food sources existing around the hive, and the number of employed and onlookers bees is the same.

ABC is an iterative algorithm that starts by generating random solutions (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 skill to move towards the food source with greater nectar amount. Each employed bee finds a new food source near its currently assigned food source (using a *neighborhood operator*) and checks the nectar amount of the new discovered position. If the nectar amount of the new discovered position is greater than the previous one, the employed bee refreshes its associated memory by replacing the previous position by the new one; otherwise, the employed bee keeps the previous position in its memory. During exploration, they continuously investigate the probability of new food sources and update memory. After all employed bees finish this exploitation process, they share the nectar information of the food sources with the onlookers.
2. *Onlooker bees phase*. Communication among bees related to the quality of food sources occurs in the dancing area. Since information about all the current rich sources is available to an onlooker on the dance floor, they probably could watch numerous dances and choose to employ them at the most profitable source. Each onlooker selects a food source according to the traditional roulette wheel selection method [14]. After that, similar to employed bees, each onlooker tries to discover hidden potential food source near its selected food source (using the neighborhood operator) and calculates the nectar amount of the neighbor food source. If onlookers find more attractive food sources, they keep the information of new food sources and forget the previous ones.
3. *Scout bees phase*. If the employed bee and onlookers associated with a food source cannot find a better neighboring food source in *limit* number of iterations (a control parameter of the ABC algorithm), the food source is abandoned and the bee associated with it becomes a scout. The scout will search for the location of a new food source in the vicinity of the hive. When the scout finds a new food source, it becomes an employed bee again. Then, each employed bee is assigned to a food source and another iteration of the ABC algorithm begins.

The ABC algorithm was originally designed for continuous optimization problems [18] and much work has been devoted to the development of extended models for these problems [9–11,26,42]. However, other variants of the ABC algorithm have been successfully applied to a wide range of optimization problems, such as quadratic minimum spanning tree problem [39], leaf-constrained minimum spanning tree problem [37], binary optimization problems [22], image segmentation [28], constrained optimization problems [19], design of recurrent neural networks [16], multi-objective optimization problems [2,29], symbolic regression [21], estimation of electricity energy demand [23], and clustering [20].

3.2. Tabu search

TS is a meta-heuristic that guides a local heuristic search procedure to explore the solution space beyond local optimality. As a matter of fact, the term tabu search was coined in the same paper that the term metaheuristic was introduced [12]. One of the main components of TS is its use of adaptive memory, which creates a highly flexible search behavior. Memory-based strategies which are the hallmark of TS approaches, are founded on a quest for integrating principles, by which alternative forms of memory are appropriately combined with effective strategies for exploiting them. The adaptive memory feature of TS allows the implementation of procedures that are capable of searching the solution space economically and effectively. Since local choices are guided by information collected during the search, TS contrasts with memory-less designs that heavily rely on semi-random processes that implement a form of sampling.

A prospective neighborhood construction technique that is often used in connection with TS is ejection chain [13], which belongs to the class of local search algorithms with very large neighborhoods. Most of the local search algorithms in the literature use small neighborhoods since they explicitly enumerate and evaluate all neighbors. Generally, the larger the neighborhood, the better the quality of the best neighbor and the higher the likelihood of converging to the truly optimal solution. At the same time, the larger the neighborhood, the longer the time it takes to search the neighborhood at each iteration. Different authors were particularly interested in developing local search algorithms with very large neighborhoods and developing techniques that can identify an improved neighbor without explicitly enumerating and evaluating all neighbors in the neighborhood [1]. Ejection chain procedures attempt to accomplish this idea by generating a compound sequence of moves, leading from one solution to another by means of a linked sequence of steps. In each step, the changes in some elements cause other elements to be ejected from their current state.

4. Hybrid ABC for the CAB problem

In this section, we explore the adaptation of the ABC framework to obtain high quality solutions for the CAB problem. The original template of ABC is quite intuitive, focusing on the intelligent behavior of honeybee swarms. In our experience, however, this method requires of very large iterations for finding high-quality solutions to difficult problems. This was our motivation for exploring changes and extensions that resulted in a hybrid version of the original ABC methodology.

The proposed changes and extensions, which attempt to accelerate the ABC method, consist of using an initialization method to construct good starting solutions (food sources), a neighborhood technique based on the ejection chain methodology (which is frequently used in conjunction with TS), and an additional fast local search to improve upon the trial solutions that the method generates. We want to point out that it is not our intention to diminish the merit of the original ABC method, which was conceived as a generic procedure capable of providing solutions of reasonably good quality to a wide range of optimization problems. Our goal is to suggest a way of hybridizing the original proposal in order to make it more competitive when compared to specialized procedures. We view this process as being similar to the transformations experienced by the GA community [14], which over the years has incorporated elements such as local search, simulated annealing and elitism that are departured from the original methodology [24,34,27]. In this work, we use the CAB problem for illustration purposes.

The rest of the section is organized as follows. In Section 4.1, we provide a general overview of the overall algorithm. In Section 4.2, we describe the initialization method. In Section 4.3, we propose the neighborhood procedure, which plays a fundamental role in our ABC. Finally, in Section 4.4, we provide details for the local search approach that is embedded in the proposed hybrid ABC algorithm to enhance the local intensification capability.

4.1. General scheme of the proposed hybrid ABC algorithm

An outline of our proposal, called HABC-CAB, is depicted in Fig. 3. The approach begins with a population of solutions (food sources) generated by the function Construct-Solution-RBFS (Step 2), which implements the effective randomized breadth-first search (RBFS) method proposed by Bansal and Srivastava [4] to construct solutions for the CAB problem (Section 4.2). Then, the following steps are repeated until a termination criterion is met:

```
Input: $G$, $t_{max}$, $limit$, $NP$, $p_{LS}$
Output: $S_b$
// Initialization phase
1  for $i = 1$ to $NP$ do
2      $S \leftarrow$ Construct-Solution-RBFS($G$);
3      $r \leftarrow$ Generate Uniform Random Number in $(0, 1)$;
4      if $r < p_{LS}$ then
5          $S_i \leftarrow$ Fast-Local-Search ($S$);
6      end
7  end
8  while computation time $t_{max}$ not reached do
       // Employed bees phase
9      for $i = 1$ to $NP$ do
10         $E \leftarrow$ Generate-Neighboring($S_i$);
11         $r \leftarrow$ Generate Uniform Random Number in $(0, 1)$;
12         if $r < p_{LS}$ then
13             $E \leftarrow$ Fast-Local-Search ($E$);
14         end
15         if $E$ is better than $S_i$ then
16             $S_i \leftarrow E$;
17         end
18     end
       // Onlooker bees phase
19     for $i$=1 to $NP$ do
20         $S_j \leftarrow$ Binary-Tournament ($S_1, ..., S_{NP}$);
21         $O \leftarrow$ Generate-Neighboring($S_j$);
22         $r \leftarrow$ Generate Uniform Random Number in $(0, 1)$;
23         if $r < p_{LS}$ then
24             $O \leftarrow$ Fast-Local-Search ($O$);
25         end
26         if $O$ is better than $S_j$ then
27             $S_j \leftarrow O$;
28         end
29     end
       // Scout bees phase
30     for $i$=1 to $NP$ do
31         if $S_i$ does not change for limit iterations then
32             $S \leftarrow$ Construct-Solution-RBFS($G$);
33             $r \leftarrow$ Generate Uniform Random Number in $(0, 1)$;
34             if $r < p_{LS}$ then
35                 $S_i \leftarrow$ Fast-Local-Search ($S$);
36             end
37         end
38     end
       // Remember the best food source found so far
39     $S_b \leftarrow$ Best-Solution-Found ();
40 end
```

Fig. 3. Pseudocode algorithm for HABC-CAB.

- *Employed bees phase*. It produces new solutions for the employed bees using the neighborhood operator Generate-Neighboring (Step 10). We propose a neighborhood operator that implements ejection chains, which have been successfully applied in the context of TS (Section 4.3).
- *Onlooker bees phase*. It produces new solutions for the onlookers using the tournament operator Binary-Tournament (Step 20). In the basic ABC algorithm, an onlooker bee selects a food source depending on a probabilistic value, which is similar to the roulette wheel selection in GAs. However, the *tournament selection* is widely used in ABC applications due to its simplicity and ability to escape from local optima [38,39]. For this reason, we employ a binary tournament selection in the HABC-CAB algorithm. Specifically, an onlooker bee selects the best food source among two food sources that were randomly selected from the population. Then, each onlooker determines a food source in the neighborhood of its chosen solution similarly to the employed bees phase (Step 21).
- *Scout bees phase*. It determines when to abandon solutions and the corresponding employed bee becomes a scout bee. The scout bee starts to search a new food source by using the function Construct-Solution-RBFS (Step 32).

In order to enhance the exploitation capability of HABC-CAB, a fast local search method (FLS) is embedded in the proposed algorithm (Section 4.4). Specifically, after generating new solutions (in the initialization and scout phases) and producing neighboring food sources (in the employed and onlooker phases), it is applied (with a probability p_{LS}) to further improve the quality of the solutions. In addition, in each iteration, after all bees complete their searches, our ABC algorithm memorizes the best food source found so far.

The HABC-CAB algorithm requires four input values: t_{max} denotes the computation time limit, *NP* determines the number of food sources which is equal to the number of employed or onlooker bees, *limit* controls the generation of scout bees, and p_{LS} is the probability of applying FLS. 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. The RBFS constructive method

In this paper we use as initialization algorithm the randomized breadth-first search constructive procedure proposed by Bansal and Srivastava [4]. For the completeness of the paper we briefly describe this method. Level algorithms [5] are constructive procedures based on the partition of the vertices of a graph in different levels, $L_1, \ldots, L_s$, such that the endpoints of each edge in the graph are either in the same level L_i or in two consecutive levels, L_i and L_{i+1}. This level structure guarantees that vertices in alternative levels are not adjacent. Level structures are usually constructed using a breadth first search method, providing a root of the corresponding spanning tree and an order in which the vertices of the graph are visited.

Bansal and Srivastava [4] applied a level procedure to the CAB problem. Specifically, they proposed a randomized breadth first search, RBFS, wherein the spanning tree is constructed by selecting the root vertex (in level 1) as well as the neighbors of the visited vertices at random. Starting from an odd level (even level) the constructive procedure labels all the non-adjacent vertices of odd levels (even levels) sequentially. The remaining vertices are labeled using a greedy approach in which a vertex is labeled with the unused label that produces the minimum decreasing of the objective function. We refer the reader to Bansal and Srivastava [4] and Díaz et al. [5] for a more detailed description.

4.3. Neighborhood operator

Let us introduce the cyclic antibandwidth $CAB(\varphi,u)$ of vertex u for a labeling φ. In mathematical terms it can be computed as follows:

$$CAB(\varphi,u) = \min_{(u,v)\in E}\{d^+(u,v), d^-(u,v)\}. \tag{3}$$

Using this definition, the CAB problem can be alternatively enunciated as follows:

$$CAB(G,\varphi) = \min_{u\in V}\{CAB(\varphi,u)\}. \tag{4}$$

From Eq. (4), we may intuit that a prominent strategy to discover enhanced solutions starting from a given solution involves the reassignation of the labels of a set of nodes in such a way that all their CAB values are raised (increasing, in this way, the probability of improving the global CAB value of the graph). Our proposed neighborhood operator, NO_1 (see the detailed pseudocode in Fig. 4), implements this practice by following the steps explained below.

1. NO_1 begins by choosing, randomly, a vertex u and attempts to find a label that may result in an increment of $CAB(\varphi,u)$. This is made by scanning a subset L of possible labels for u, selected at random, and locating the label $l_{best} \in L$ that produces the fittest CAB value for u. If l_{best} allows the CAB value of u to be improved, it is assigned to this vertex. Otherwise, this entire process is repeated again.
2. In the case that NO_1 changed the label for u, the graph has currently an unlabeled vertex, v_{free} (the node having previously l_{best} as label), and a free label, l_{free} (the one initially assigned to u). We can consider labeling v_{free} with l_{free}; however, other labels may be better suited for this free vertex. To deal with this possibility, the subset L is created anew randomly (i.e., a different subset L for every interchange) and the label in $L \cup \{l_{free}\}$ that induces the highest CAB value for v_{free} is finally assigned to this vertex.
3. If the best label for v_{free} is l_{free}, then the graph will result completely labeled. In this case, NO_1 will attempt to improve the CAB value of another starting vertex, selected at random. In the other case, the best label is an element from L, which will lead to the appearance of a new unlabeled vertex, v_{free}, and the procedure for labeling this vertex should be launched.
4. After performing I_{max} trial interchanges, NO_1 finishes the construction of a neighborhood by finally assigning l_{free} to v_{free}.

In summary, NO_1 attempts to generate a series of interchanges for redistributing the labels among a set of nodes, looking for a relative improvement of their cyclic antibandwidth values. Clearly, this behavior bears a resemblance to the principles of the ejection chain methodology described in Section 3.2.

An important parameter of NO_1 is $n_L = |L|$, because it controls the size of the set of available labels. In other words, if $n_L = n$ this procedure is able to obtain the best label for the incumbent vertex (since all labels are available). On the other hand, if the size of n_L is close to 1, the procedure has fewer options of choosing a good label for the incumbent vertex. It is also important to remark that large values of n_L do not necessarily mean profitable benefits on the cyclic antibandwidth of the whole graph, since improving the cyclic antibandwidth of a given vertex does not imply an improvement of the cyclic antibandwidth of the graph. Thus, the impact of n_L on the performance of NO_1 should be carefully examined (see Section 5.2).

```
Input: $G$, $\varphi_i$, $I_{max}$, $n_L$
Output: $\varphi$
$\varphi \leftarrow \varphi_i$;
$I \leftarrow 1$;
$v_{free} \leftarrow$ Select-Node-Random ($V$);
$l_{free} \leftarrow \varphi(v_{free})$;
while $I \leq I_{max}$ do
    $L \leftarrow$ Select-Labels-Random ($\{1, ..., n\}$, $n_L$);
    $l_{best} \leftarrow$ Best-Label ($v_{free}$, $L \cup \{l_{free}\}$);
    $\varphi(v_{free}) \leftarrow l_{best}$;
    if $l_{best} == l_{free}$ then
        $v_{free} \leftarrow$ Select-Node-Random ($V$);
        $l_{free} \leftarrow \varphi(v_{free})$;
    else
        Find $u \in V$ such as $\varphi(u) = l_{best}$;
        $v_{free} \leftarrow u$;
    end
    $I \leftarrow I + 1$;
end
$\varphi(v_{free}) \leftarrow l_{free}$;
```

Fig. 4. Pseudocode algorithm for NO_1.

4.4. Fast local search method

Different authors proposed to enhance the exploitation capability by embedding a local search method in the ABC algorithm. For example, in Sundar and Singh [39], after the execution of the ABC algorithm, a local search method is applied to the best solution found. An alternative approach is proposed in Tasgetiren et al. [38] where the local search technique is applied (with a probability p_{LS}) to the neighboring food source found by an employed bee. The main purpose of hybridizing ABC with a local search method is to balance exploration vs. exploitation. Specifically, ABC performs the global search by exploring the search space, whereas the local search is responsible for exploiting a small region of the search space in order to find a local optimum.

We present a *fast* local search algorithm, FLS, for the CAB problem, which has been hybridized with the ABC algorithm. This local optimizer is invoked with a probability p_{LS} immediately after a new solution is built by Construct-Solution-RBFS or generated by Generate-Neighboring (see Fig. 3).

The proposed FLS procedure has the following main contributions (Fig. 5):

- *Consecutive swaps.* FLS is based on a systematic application of specific swap moves that exchange the label l of a vertex u with the label $l+1$ of a vertex v (Steps 5–6). This type of moves produce smooth changes in the objective function of the current labeling, because they will increment/decrement the objective function in one unit at most [35].
- *First-improvement local search.* Our method implements the so-called first choice strategy that scans moves in search for the first exchange yielding to an improvement in the objective function. However, in the CAB problem, there may be many different solutions with the same objective function value. We could say that the solution space presents a *flat landscape*. In this kind of problems, such as the min–max or max–min, local search procedures typically do not perform well because most of the moves have a null value [8,33]. In the case of the CAB problem, there may be multiple vertices with a cyclic antibandwidth value equal to $CAB(G,\varphi)$. Then, changing the label of a particular vertex u to increase its cyclic antibandwidth $CAB(\varphi,u)$, does not necessarily imply that $CAB(G,\varphi)$ also increases.
  We therefore extend the definition of "improving" to overcome the lack of information provided by the objective function. Specifically, we consider that the labeling φ' resulting from a swap move changing the labels of the vertices u and v improves the current labeling φ if the worst cyclic antibandwidth among the ones of vertices u and v becomes increased (Step 7). We have empirically found that this criterion allows the local search procedure to explore a larger number of solutions than a typical implementation that only performs moves when the objective function is improved. An advantage concerning this way of checking the suitability of a swap is that it surpasses the need for evaluating the whole solution, because decisions are made taking into account only the effects of the swap on the implied vertices.
- *Data structures.* FLS maintains data structures to be able to compute $CAB(\varphi',u)$ and $CAB(\varphi',v)$ in constant time (Step 7), after exchanging the label of u with the label of v. Specifically, it records, for each vertex $w\in V, l^-_{min}(w), l^-_{max}(w), l^+_{min}(w)$, and $l^+_{max}(w)$, which are defined as follows:

$$l^-_{min}(w)=\begin{cases}\min S^-_w & \text{if} |S^-_w|\neq 0\\ \varphi(w) & \text{otherwise}\end{cases},\quad l^-_{max}(w)=\begin{cases}\max S^-_w & \text{if} |S^-_w|\neq 0\\ -\infty & \text{otherwise}\end{cases} \tag{5}$$

$$l^+_{min}(w)=\begin{cases}\min S^w_+ & \text{if} |S^+_w|\neq 0\\ \infty & \text{otherwise}\end{cases},\quad l^+_{max}(w)=\begin{cases}\max S^+_w & \text{if} |S^+_w|\neq 0\\ \varphi(w) & \text{otherwise}\end{cases} \tag{6}$$

where

$$S^-_w=\{\varphi(t):(w,t)\in E\wedge\varphi(t)<\varphi(w)\}\text{and}$$

$$S^+_w=\{\varphi(t):(w,t)\in E\wedge\varphi(t)>\varphi(w)\}.$$

Then, using this information, the new cyclic antibandwidth values for u and v may be directly obtained by:

$$CAB(\varphi',u)=\min\{|l+1-l^+_{min}(u)|, n-|l+1-l^+_{max}(u)|, \\ |l+1-l^-_{max}(u)|, n-|l+1-l^-_{min}(u)|\}, \tag{7}$$

and

$$CAB(\varphi',v)=\min\{|l-l^+_{min}(v)|, n-|l-l^+_{max}(v)|, \\ |l-l^-_{max}(v)|, n-|l-l^-_{min}(v)|\}, \tag{8}$$

where $l=\varphi(u)$ and remind that $\varphi'(u)=l+1$ and $\varphi'(v)=l$.

In general, local search methods are a really computational expensive procedures, since they evaluate a huge number of solutions in a given neighborhood (exploitation). The FLS presented in this section significantly decrease the cost of computing the value of each visited solution, as we will show in Section 5.

```
Input: $G$, $\varphi_i$
Output: $\varphi$
$\varphi \leftarrow \varphi_i$;
$L \leftarrow \{1,...,n-1\}$;
while $|L|>0$ do
    Select $l \in L$ randomly;
    Find $u,v \in V$ such as $\varphi(u)=l$ and $\varphi(v)=l+1$;
    $\varphi' \leftarrow$ Swap($\varphi,u,v$);
    if $\min\{CAB(\varphi',u),CAB(\varphi',v)\} > \min\{CAB(\varphi,u),CAB(\varphi,v)\}$ then
        $\varphi \leftarrow \varphi'$;
        $L \leftarrow \{1,...,n-1\}$;
    else
        $L \leftarrow L/\{l\}$;
    end
end
```

Fig. 5. Pseudocode algorithm for FLS.

5. Computational experiments

This section describes the computational experiments that we conducted to assess the performance of the HABC-CAB algorithm presented in the previous section. Firstly, we detail the experimental setup (Section 5.1), then, we analyze the results obtained from different experimental studies carried out with this algorithm. Our aim is: (1) to analyze the influence of the parameters and settings associated with HABC-CAB and show the benefits of using the proposed neighborhood operator and local search method (Section 5.2); and (2) to compare the results of HABC-CAB with previous approaches for the CAB problem (Section 5.3).

5.1. Experimental setup

The code of HABC-CAB 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 3.2 GHz Intel ® Core® i7 processor with 12 GB of RAM running Fedora ® Linux V15. We have considered ten sets with a total of 295 instances classified in three groups: 132 instances with known optimum, 47 instances with conjectured optimum, and 116 instances with unknown optimum. All these instances are available at the following URL: http://www.optsicom.es/abp/. A detailed description of each set of instances follows.

Instances with known optimum

- *Paths*. This data set consists of 24 graphs constructed as a linear arrangement of vertices such that every vertex has a degree of two, except the first and the last vertex that have a degree of one. The size of these instances ranges from 50 to 1000. For a path P_n with n vertices, Sýkora et al. [40] proved that the optimal CAB value is

$$CAB(P_n) = \left\lceil \frac{n}{2} \right\rceil - 1. \tag{9}$$

- *Cycles*. This data set consists of 24 graphs constructed as a circular arrangement of vertices such that every vertex has a degree of two. The size of these instances ranges from 50 to 1000. For a cycle C_n with n vertices the optimal cyclic antibandwidth is

$$CAB(C_n) = \left\lceil \frac{n}{2} \right\rceil - 1, \tag{10}$$

as shown in Sýkora et al. [40].

- *Grids*. This data set consists of 23 graphs constructed as the Cartesian product of two paths P_{n_1} and P_{n_2} [32]. The size of these instances ranges from 81 to 1089. They are also called two dimensional meshes. For a grid $P_{n_1} \times P_{n_2}$ with $n = n_1 \cdot n_2$ vertices the optimal cyclic antibandwidth fulfills that [32]:

$$\left\lfloor \frac{n_2(n_1-1)}{2} \right\rfloor \leqslant CAB(P_{n_1} \times P_{n_2}) \leqslant \left\lceil \frac{n_2(n_1-1)}{2} \right\rceil, \tag{11}$$

if n_1 is even and n_2 is odd ($n_1 \geqslant n_2$), and

$$CAB(P_{n_1} \times P_{n_2}) = \frac{n_2(n_1-1)}{2}, \tag{12}$$

otherwise.

- *Toroidal* grids (Tori). This data set consists of 37 graphs constructed as the Cartesian product of two cycles (i.e., $C_n \times C_n$). The size of these instances ranges from 16 to 1600. The optimal cyclic antibandwidth is

$$CAB(C_n \times C_n) = \frac{n(n-2)}{2}, \tag{13}$$

if n is even, and

$$CAB(C_n \times C_n) = \frac{(n-2)(n+1)}{2}, \tag{14}$$

if n is odd (see [32]).

- *Hamming* graphs. This data set consists of 24 graphs constructed as the Cartesian product of d complete graphs K_{n_k}, for $k = 1,2,\ldots,d$ Dobrev et al. [6]. The size of these instances ranges from 80 to 1152. The vertices in these graphs are d-tuples $(i_1,i_2 \ldots,i_d)$, where $i_k \in \{0,1\ldots,n_{k-1}\}$. Two vertices $(i_1,i_2,\ldots,i_d)$ and $(j_1,j_2,\ldots,j_d)$ are adjacent if and only if the two tuples differ in exactly one coordinate. These graphs are referred to as Hamming graphs. Dobrev et al. [6] proved that if $d \geqslant 2$ and $2 \leqslant n_1 \leqslant n_2 \leqslant \ldots \leqslant n_d$, then the optimal cyclic antibandwidth for this type of instances is given by:

$$CAB\left(\prod_{k=1}^{d} K_{n_k}\right) = \begin{cases} n_1n_2\ldots n_{d-1} & \text{if} n_{d-1} \neq n_d, d \geqslant 2 \\ n_1n_2\ldots n_{d-1}-1 & \text{if} n_{d-1} = n_d \text{ and } n_{d-2} \neq n_{d-1}, d \geqslant 3 \end{cases} \tag{15}$$

and

$$n_1n_2\ldots n_{d-1} - \min\{n_1n_2\ldots n_{d-2}, n_{q+1}\ldots n_{d-1}\} \leqslant CAB\left(\prod_{k=1}^{d} K_{n_k}\right) \leqslant n_1n_2\ldots n_{d-1} - 1, \tag{16}$$

where $n_{d-2} = n_{d-1} = n$, $d \geqslant 3$ and q is the minimal index such that $q \leqslant d-2$ and $n_q = n_d$.

Instances with conjectured optimum

Our algorithm was checked as well using instances of three-dimensional meshes, hypercubes, and double stars. Based on the experiments carried out with the memetic algorithm by Bansal and Srivastava [4], these authors conjectured the optimal cyclic antibandwidth for these three types of graphs. We have compared the results of the executed algorithms with the conjectured optimal values found by these authors.

- *Three-dimensional meshes*. They are 20 graphs defined by the Cartesian product of three paths $P_{n_1} \times P_{n_2} \times P_{n_3}$ with $n = n_1 \times n_2 \times n_3$. The size of these instances ranges from 12 to 3600.
- *Double stars*. A double star $s(n_1,n_2)$ consists of two stars K_{1,n_1} and K_{1,n_2} with one edge in common. We generated 20 instances whose size ranges from 20 to 180.
- *Hypercubes*. The hypercube Q_d of dimension d is a d-regular graph with 2^d vertices and $d \cdot 2^{d-1}$ edges. Each vertex is labeled by a distinct d-bit binary string, and two vertices are adjacent if they differ in exactly one bit. 7 instances where obtained, ranging from 16 to 1024 vertices.

Instances with unknown optimum

- *Random connected graphs*. These graphs are generated by including each of $|V|2$ possible edges independently with probability p_d. Through the experiments, we fixed the parameter p_d such that different edge densities are taken into account. The procedure for building these graphs ensures that they are connected.
- *Caterpillars*. This data set consists of 40 graphs. Each caterpillar, P_{n_1,n_2} is constructed using the path P_{n_1} and n_1 copies of the path P_{n_2} (usually referred to as "hairs"), where each vertex i in P_{n_1} is connected to the first vertex of the i-th copy of the path P_{n_2}. The size of these instances ranges from 20 to 1000.
- *Complete binary trees* (CBT). This data set consists of 24 trees, where every tree-level is completely filled except possibly the last level and all nodes are as far left as possible. The size of these instances ranges from 30 to 950.
- *Harwell-Boeing*. We derived 24 matrices from the Harwell-Boeing sparse matrix collection [15]. The size of these instances ranges from 30 to 715. This collection consists of a set of standard test matrices arising from problems in linear systems, least squares, and eigenvalue calculations from a wide variety of science and engineering problems. The problems range from small matrices, used as counter-examples to hypotheses in sparse matrix research, to large matrices arising in practical applications. Graphs are derived from these matrices as follows. Let M_{ij} denote the element of the ith row and jth column of the $n \times n$ sparse matrix M. The corresponding graph has n vertices. Edge (i,j) exists in the graph if and only if $M_{ij} \neq 0$.

5.2. Study of the neighborhood operator and the FLS method

In this section, we investigate the effects of the two search strategies, NO$_1$ and FLS, proposed for HABC-CAB. For these experiments, we consider a set of 70 instances of random connected graphs that range from 100 to 1000 vertices with different densities ($p_d \in [0.1, 0.7]$). For each combination of graph size and edge density, two random instances are generated for the experiments. In order to have a fair comparison, all the HABC-CAB variants were stopped using a time limit of 150 s. Additionally, each algorithm was executed once for each problem instance. In our first preliminary experiment, we study the effect of the search parameters, I_{max} and n_L, in the neighborhood operator, NO$_1$. Specifically, we implemented 16 different variants of HABC-CAB with

$I_{max} = \{0.1n, 0.25n, 0.5n, n\}$ and $n_L = \{0.01n, 0.1n, 0.25n, 0.5n\}$. The population size is set to 20, $limit = 0.5n$, and $p_{LS} = 1$.

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 deviation between the best solution value found by the method and the *BestValue*. In Table 2, we report the average of this relative deviation in percentage (*%Dev*) across all the instances considered in each particular experiment and the percentage of instances (*%Best*) for which the value of the best solution obtained by a given method matches *BestValue*. We also show the average *rankings*, computed by the Friedman test, obtained by these algorithms (*Av. Ran.*). This measure is obtained by computing, for each 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 the ranking $|A|$ (where 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, on five instances, the average ranking is $\frac{1+3+1+4+2}{5} = 2.20$. Note that the lower the ranking, the better the algorithm.

Results in Table 2 show that the n_L parameter has an important effect on the quality of the solution obtained by the method. According to the rankings, (*Av. Ran.*), the HABC-CAB variants with $n_L = 0.1n$ and $0.25n$ obtain, in general, the best results. Actually, these variants also outperform the others in terms of *%Dev* and *%Best*. These results support our conjecture that large values of n_L do not necessarily improve the cyclic antibandwidth of the whole graph, since improving the cyclic antibandwidth of a given vertex does not imply an improvement of the cyclic antibandwidth of the graph. Therefore, attending to the results shown in Table 2, we set $n_L = 0.1n$ and $I_{max} = n$ for the rest of our experimentation.

Our next empirical study was performed to show the merit of the proposed neighborhood operator, NO_1. In order to do this, it was compared with a standard neighborhood operator (NO_2) that has been widely used in the literature for a variety of graph arrangement problems. Specifically, NO_2 is a swap operator that interchanges the labels associated with two vertices chosen at random. We have also examined a third operator, NO_3, performing two consecutive swap moves. We have also studied the effect of *limit* search parameter within each neighborhood operator. Specifically, we consider $limit = \{0.25n, 0.5n, n, 2n\}$, resulting in 12 different HABC-CAB variants. Table 3 shows the associated results, reporting again the same statistics.

Results in Table 3 show that NO_1 clearly outperforms NO_2 and NO_3. Actually, there are no significant differences among all NO_1 variants. However, any NO_1 variant is better than any NO_2 or NO_3 variant. Considering the results presented in this table, we set $limit = 0.5n$ for the rest of our experimentation, since exhibits the best performance (*Av. Ran.*).

Table 2
Results of HABC-CAB with different parameter values for NO_1.

I_{max}	n_L	*Av. Ran.*	*%Dev*	*%Best*
n	$0.5n$	8.97	10.93	57.14
$0.5n$	$0.5n$	9.33	12.86	47.14
$0.25n$	$0.5n$	8.64	9.71	54.29
$0.1n$	$0.5n$	9.26	11.81	47.14
n	$0.25n$	6.67	3.89	77.14
$0.5n$	$0.25n$	7.35	7.07	64.29
$0.25n$	$0.25n$	7.88	7.86	61.43
$0.1n$	$0.25n$	8.92	9.41	54.29
n	$0.1n$	6.47	3.98	81.43
$0.5n$	$0.1n$	6.95	5.75	64.29
$0.25n$	$0.1n$	7.507	7.11	60.00
$0.1n$	$0.1n$	8.05	8.17	58.57
n	$0.01n$	10.75	14.85	40.00
$0.5n$	$0.01n$	9.73	12.59	44.29
$0.25n$	$0.01n$	9.27	10.32	50.00
$0.1n$	$0.01n$	10.19	12.69	44.29

Finally, in Table 4, we analyze the influence of FLS on the performance of HABC-CAB. Specifically, we investigate the effects of varying the p_{LS} parameter associated with this local search procedure. In order to do this, we have tested the performance of four HABC-CAB configurations with $p_{LS} = \{0, 0.25, 0.5, 1\}$. Notice that $p_{LS} = 0$ means that the HABC-CAB method does not use the local search. On the other hand, $p_{LS} = 1$ means that FLS is invoked after a new solution is built by Construct-Solution-RBFS or generated by Generate-Neighboring.

Results in Table 4 clearly show the merit of the local search method. Specifically, the ranking of the variant without local search is the worst of the four analyzed HABC-CAB variants. Notice that all methods were executed for the same CPU time (150 s), therefore the local search procedure is worth using within the ABC template. In fact, the more often we apply the local search procedure, the better the HABC-CAB variant performs. In particular, when $p_{LS} = 1$, we obtain the best results in terms of ranking, *%Dev*, and *%Best*. Consequently, we set $p_{LS} = 1$ for the rest of the experiments.

5.3. HABC-CAB vs. state-of-the-art metaheuristic for the CAB problem

In this section, we undertake a comparative analysis among HABC-CAB ($I_{max} = n$, $n_L = 0.1n$, $limit = 0.5n$, and $p_{LS} = 1$) and the current best algorithm for the CAB problem, the memetic algorithm (named MACAB) proposed by Bansal and Srivastava [4]. This algorithm starts by creating an initial population of solutions using the RBFS method (Section 4.2). As it is customary in evolutionary methods, the initial population evolves by applying three steps: selection, recombination and mutation. The selection strategy is implemented by means of a classical tournament operator. The recombination operator is implemented using a modified version of the RBFS procedure, in which a solution is obtained by copying part of its "father" (up to a random point) and then completing it with the RBFS constructive procedure. The mutation strategy is implemented by swapping the labels of two vertices selected at random in a solution. These three main steps are repeated until a maximum number of iterations (generations) is reached.

We should point out that HABC-CAB and MACAB were run under the same computational conditions (in order to enable a fair comparison between them) on all instance sets listed in Section 5.1. For this experimental study, we have generated 28 new instances of random connected graphs by combining different values for n and p_d ($n \in \{250, 500, 750, 1000\}$ and $p_d \in [0.1, 0.7]$). The parameter values used for MACAB are the ones recommended in the original work. These algorithms were run once on each instance and the cutoff time for each execution was set to 150 s. Detailed results for all test instances may be found at http://sci2s.ugr.es/gabic/Results-CAB.xlsx (the results of the algorithms on the Harwell-Boeing instances may be found in Appendix A).

Table 3
Results of the HABC-CAB instances with different neighborhood operators.

NO	*Limit*	*Av. Ran.*	*%Dev*	*%Best*
NO_1	$2n$	4.33	1.50	91.43
NO_1	n	4.39	1.59	90.00
NO_1	$0.5n$	4.21	0.71	94.29
NO_1	$0.25n$	4.27	1.27	94.29
NO_2	$2n$	7.44	16.09	41.43
NO_2	n	7.62	16.88	38.57
NO_2	$0.5n$	7.42	16.01	41.43
NO_2	$0.25n$	7.61	16.56	41.43
NO_3	$2n$	7.88	17.01	38.57
NO_3	n	7.76	16.83	38.57
NO_3	$0.5n$	7.53	15.73	41.43
NO_3	$0.25n$	7.55	15.36	45.71

Table 4
Results of HABC-CAB with different p_{LS} values.

p_{LS}	*Av. Ran.*	*%Dev*	*%Best*
0	3.47	21.71	32.86
0.25	2.21	2.72	81.43
0.5	2.16	2.26	82.86
1	2.15	2.02	84.29

Firstly, we compare HABC-CAB with MACAB using the Wilcoxon matched-pairs signed ranks test. With this test, the results of two algorithms may be directly compared. In statistical terms, this test answers the question: Do the two samples represent two different populations? When comparing two algorithms with the Wilcoxon test, it determines if the results of two methods are significantly different. Table 5 summarizes the results of this procedure for a level of significance $\alpha = 0.05$, where the values of R^+ (associated to HABC-CAB) and R^- (associated to MACAB) of the test are specified, together with the critical values. If R^- is smaller than both, R^+ and the critical value, HABC-CAB is statistically better than MACAB (represented with the sign +); if R^+ is smaller than both, R^- and the critical value, our algorithm is statistically worse than its competitor (represented with the sign −); if neither R^+ nor R^- is smaller than the critical value, the test does not find statistical differences among them (represented with the sign $\sim$).

The Wilcoxon test reveals that: (1) HABC-CAB has the upper hand in the statistical comparison over its competitor on the instance sets *CBT*, *Hamming*, *Harwell-Boeing*, *Caterpillars*, *Tori*, and *Random connected*, (2) MACAB statistically outperforms our proposal in the case of *Grids*, *Double stars*, *3-d meshes*, and *Hypercubes*, and (3) there are not significant differences between these algorithms for the *Cycles* and *Paths* sets.

To complement this information, we analyze, in detail, the results returned by these algorithms, which are outlined in Table 6. For those instance sets with known (or conjectured) optimum, the last two columns (*%Dev-Opt* and *%Best-Opt*) display the *%Dev* and *%Best* measures computed with respect to the corresponding optimum values (or conjectured ones). For those instance sets with unknown optimum, these two columns display an hyphen. Attending to the values of *%Dev-Opt* and *%Best-Opt*, the instances of the sets *Hamming*, *Tori*, and *Cycles* seem to be the hardest ones. Therefore, we believe that there is still room for improvement. Similarly, the sets of instances *Paths*, *Grids*, *Double starts*, *3-d meshes*, and *Hypercubes* could be considered "easy to solve" since, in general, both methods are able to obtain solutions with a relative low percentage deviation with respect to the optimum.

Table 6 shows that our HABC-CAB method obtains better results in terms of *%Dev* and *%Best* (and, when available, in terms of *%Dev-Opt* and *%Best-Opt*) than MACAB in 6 sets of instances out of 12. Specifically, our method clearly outperforms MACAB in *CBT*, *Harwell-Boeing*, *Caterpillars*, in the complex *Hamming* and *Tori* instances, and in the *Random connected* set (in fact, the last

Table 5
HABC-CAB vs. MACAB (Wilcoxon's test).

Inst. set	R^+	R^-	Critical val.	Sig. differences?
CBT	300.0	0.0	81	+
Hamming	290.5	9.5	81	+
Harwell-Boeing	269.0	7.0	81	+
Caterpillars	525.0	255.0	264	+
Tori	508.0	158.0	221	+
Random connected	367.5	10.5	116	+
Cycles	161.0	115.0	81	$\sim$
Paths	126.5	149.5	81	$\sim$
Grids	19.0	234.0	81	−
Double stars	39	171	52	−
3-d Meshes	5	185	52	−
Hypercubes	1.5	19.5	7	−

Table 6
HABC-CAB vs. MACAB.

Inst.	*Alg.*	*%Dev*	*%Best*	*%Dev-Opt*	*%Best-Opt*
CBT	HABC-CAB	0.00	100.00	–	–
	MACAB	18.05	0.00	–	–
Hamming	HABC-CAB	0.76	95.83	23.76	0
	MACAB	45.10	4.17	55.09	0
Harwell-Boeing	HABC-CAB	0.07	91.67	–	–
	MACAB	42.89	12.50	–	–
Caterpillars	HABC-CAB	0.09	80	–	–
	MACAB	1.85	47.50	–	–
Tori	HABC-CAB	0.15	70.27	16.70	18.92
	MACAB	32.99	48.65	37.52	13.51
Random connected	HABC-CAB	0	100	–	–
	MACAB	31.11	25	–	–
Cycles	HABC-CAB	0.42	54.17	4.94	8.33
	MACAB	14.44	66.67	17.69	54.17
Paths	HABC-CAB	0.01	95.83	0.01	95.83
	MACAB	0	100	0	100
Grids	HABC-CAB	0.64	34.78	0.80	30.43
	MACAB	0.10	95.65	0.26	56.52
Double stars	HABC-CAB	5.22	60	5.22	60
	MACAB	0	100	0	100
3-d meshes	HABC-CAB	0.3	25	0.39	25
	MACAB	0	100	0.09	35
Hypercubes	HABC-CAB	3.22	42.86	3.33	42.85
	MACAB	0	100	0.11	71.42

instances have posed a real challenge for MACAB, probably due to their lack of fixed structure and property). On *Cycles*, our algorithm reached better *%Dev* performance and worse *%Best* one. On the other hand, MACAB performs better than our HABC-CAB procedure in *Grids*, *Paths*, and in all the instance sets with conjectured optima (*Double starts*, *3-d meshes*, and *Hypercubes*). Although Wilcoxon's test shows that this advantage is statistically significant, we should remark that, in general, there are not great differences in the *%Dev* performance between the two algorithms in these cases. We should highlight that HABC-CAB employs the RBFS method proposed by Bansal and Srivastava [4], which is a decisive components of MACAB that ensures appropriate performance on this type of graphs [4]. Thus, the invocation of RBFS as constructive mechanism has allowed our algorithm to achieve an acceptable level of robustness when facing these problems.

In summary, this experimental analysis confirms that HABC-CAB arises as a very attractive alternative to the existing memetic approach for the CAB problem.

6. Conclusions

In this paper, an ABC algorithm, based on mimicking the food foraging behavior of honeybee swarms, is proposed as a method for solving the CAB problem. We propose changes and extensions to the original template of ABC, resulting in a hybrid ABC approach. In particular, our algorithm employs the initialization scheme presented in Bansal and Srivastava [4] (instead of random initialization), and a tournament operator (instead of a roulette wheel-based operator). Additionally, we propose to use a local search procedure after a new solution is built by the initialization operator or generated by the neighboring operator. Specifically, we present a new effective and efficient local search for the CAB problem, where the neighborhood is visited in an intelligent way. We have also introduced a new neighborhood operator specifically designed for the CAB problem, which follows principles of the ejection chain methodology.

Based on a series of preliminary experiments to identify effective ways to coordinate the underlying strategies, we are able to produce a method that reaches high quality solutions on previously reported instances. In fact, the designed algorithm has experimentally proved to be competitive with the state-of-the-art (the memetic algorithm by Bansal and Srivastava [4]). Specifically, the empirical study reveals a clear superiority when tackling the hardest instances.

We believe that the hybrid ABC framework presented in this paper is a significant contribution, worthy of future study. We will intend to explore other interesting avenues of research, such as the adaptation of the proposed approach for its application to other challenging graph layout problems, including linear arrangement [36], bandwidth [31], and cutwidth [30] problems.

Acknowledgment

This work was supported by the Research Projects TIN2012-37930-C02-01, TIN2012-35632, TIN2009-07516, and P08-TIC-4173.

Appendix A. Results of the algorithms on Harwell-Boeing

Instance	$\|V\|$	MACAB	HABC-CAB
pores_1.mtx.rnd	30	6	6
ibm32.mtx.rnd	32	6	8
bcspwr01.mtx.rnd	39	9	13
bcsstk01.mtx.rnd	48	6	7
bcspwr02.mtx.rnd	49	12	16
curtis54.mtx.rnd	54	7	10
will57.mtx.rnd	57	7	11
impcol_b.mtx.rnd	59	3	7
ash85.mtx.rnd	85	13	19
nos4.mtx.rnd	100	22	30
dwt_234.mtx.rnd	117	22	43
bcspwr03.mtx.rnd	118	18	29
bcsstk07.mtx.rnd	420	12	29
bcsstk06.mtx.rnd	420	12	29
impcol_d.mtx.rnd	425	30	84
can_445.mtx.rnd	445	36	73
494_bus.mtx.rnd	494	53	163
dwt_503.mtx.rnd	503	14	48
sherman4.mtx.rnd	546	258	256
dwt_592.mtx.rnd	592	33	97
662_bus.mtx.rnd	662	58	165
nos6.mtx.rnd	675	328	325
685_bus.mtx.rnd	685	16	114
can_715.mtx.rnd	715	21	98

References

[1] R.K. Ahuja, O. Ergun, J. Orlin, A. Punen, A survey of very large-scale neighborhood search techniques, Discrete Applied Mathematics 123 (13) (2002) 75–102.

[2] R. Akbari, R. Hedayatzadeh, K. Ziarati, B. Hassanizadeh, A multi-objective artificial bee colony algorithm, Swarm and Evolutionary Computation 2 (2012) 39–52.

[3] R. Bansal, K. Srivastava, Memetic algorithm for the antibandwidth maximization problem, Journal of Heuristics 17 (2009) 39–60.

[4] R. Bansal, K. Srivastava, A memetic algorithm for the cyclic antibandwidth maximization problem, Soft Computing 15 (2) (2011) 397–412.

[5] J. Díaz, J. Petit, M. Serna, A survey of graph layout problems, Journal ACM Computing Surveys 34 (3) (2002) 313–356.

[6] S. Dobrev, R. Královic, D. Pardubská, L. Trörök, I. Vrt'o, Antibandwidth and cyclic antibandwidth of Hamming graphs, Electronic Notes in Discrete Mathematics 34 (2009) 295–300.

[7] A. Duarte, R. Martí, M.G.C. Resende, R.M.A. Silva, GRASP with path relinking heuristics for the antibandwidth problem, Networks 58 (3) (2011) 171–189.

[8] A. Duarte, L.F. Escudero, R. Martí, N. Mladenovic, J.J. Pantrigo, J. Sánchez-Oro, Variable neighborhood search for the vertex separation problem, Computers and Operations Research 39 (12) (2012) 3247–3255.

[9] M. El-Abd, Performance assessment of foraging algorithms vs. evolutionary algorithms, Information Sciences 182 (1) (2012) 243–263.

[10] W.-F. Gao, S.-Y. Liu, A modified artificial bee colony algorithm, Computers and Operations Research 39 (2012) 687–697.

[11] 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.

[12] F. Glover, Future paths for integer programming and links to artificial intelligence, Computers and Operations Research 13 (1986) 533–549.

[13] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, 1997.

[14] D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, AddisonWesley, New York, 1989.

[15] Harwell-Boeing, 2011. <http://math.nist.gov/MatrixMarket/data/Harwell-Boeing/>.

[16] T.J. Hsieh, H.F. Hsiao, W.C. Yeh, Forecasting stock markets using wavelet transforms and recurrent neural networks: an integrated system based on artificial bee colony algorithm, Applied Soft Computing 11 (2) (2011) 2510–2525.

[17] Karaboga, D., 2005. An idea based on honey bee swarm for numerical optimization. Technical Report TR06, Computer Engineering Department, Erciyes University, Turkey.

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

[19] D. Karaboga, B. Akay, A modified artificial bee colony (ABC) algorithm for constrained optimization problems, Applied Soft Computing 11 (3) (2011) 3021–3031.

[20] D. Karaboga, C. Ozturk, A novel clustering approach: artificial bee colony (ABC) algorithm, Applied Soft Computing 11 (1) (2011) 652–657.

[21] D. Karaboga, C. Ozturk, N. Karaboga, B. Gorkemli, Artificial bee colony programming for symbolic regression, Information Sciences 209 (2012) 1–15.

[22] M.H. Kashan, N. Nahavandi, A.H. Kashan, DisABC: a new artificial bee colony algorithm for binary optimization, Applied Soft Computing 12 (1) (2012) 342–352.

[23] M.S. Kiran, E. Ozceylan, M. Gunduz, T. Paksoy, Swarm intelligence approaches to estimate electricity energy demand in Turkey, Knowledge-Based Systems 36 (2012) 103–931.

[24] N. Krasnogor, J.E. Smith, A tutorial for competent memetic algorithms: model, taxonomy, and design issue, IEEE Transactions on Evolutionary Computation 9 (5) (2005) 474–488.

[25] J.Y.-T. Leung, O. Vornberger, J.D. Witthoff, On some variants of the bandwidth minimization problem, SIAM Journal of Computing 13 (1984) 650–667.

[26] G.Q. Li, P.F. Niu, X.J. Xiao, Development and investigation of efficient artificial bee colony algorithm for numerical function optimization, Applied Soft Computing 12 (1) (2012) 320–332.

[27] M. Lozano, C. García-Martínez, Hybrid metaheuristics with evolutionary algorithms specializing in intensification and diversification: overview and progress report, Computers and Operations Research 37 (3) (2010) 481–497.

[28] M. Ma, J.H. Liang, M. Guo, Y. Fan, Y.L. Yin, SAR image segmentation based on artificial bee colony algorithm, Applied Soft Computing 11 (8) (2011) 5205–5214.

[29] S.N. Omkar, J. Senthilnath, R. Khandelwal, G.N. Naik, S. Gopalakrishnan, Artificial bee colony (ABC) for multi-objective design optimization of composite structures, Applied Soft Computing 11 (1) (2011) 489–499.

[30] J.J. Pantrigo, R. Martí, A. Duarte, E.G. Pardo, Scatter search for the cutwidth minimization problem, Annals of Operations Research 199 (1) (2012) 285–304.

[31] E. Piñana, I. Plana, V. Campos, R. Martí, GRASP and path relinking for the matrix bandwidth minimization, European Journal of Operational Research 153 (2004) 200–210.

[32] A. Raspaud, H. Schröder, O. Sýkora, L. Torok, I. Vrt'o, Antibandwidth and cyclic antibandwidth of meshes and hypercubes, Discrete Mathematics 309 (2009) 3541–3552.

[33] M.G.C. Resende, R. Martí, M. Gallego, A. Duarte, GRASP and path relinking for the max-min diversity problem, Computers and Operations Research 37 (2010) 498–508.

[34] F.J. Rodríguez, C. García-Martínez, M. Lozano, Hybrid metaheuristics based on evolutionary algorithms and simulated annealing: taxonomy, comparison, and synergy test, IEEE Transactions on Evolutionary Computation 16 (6) (2012) 787–800.

[35] E. Rodriguez-Tello, J.-K. Hao, J. Torres-Jimenez, An improved simulated annealing algorithm for bandwidth minimization, European Journal of Operational Research 185 (2008) 1319–1335.

[36] E. Rodriguez-Tello, H. Jin-Kao, J. Torres-Jimenez, An effective two-stage simulated annealing algorithm for the minimum linear arrangement problem, Computer and Operations Research 35 (10) (2008) 3331–3346.

[37] A. Singh, An artificial bee colony algorithm for the leaf-constrained minimum spanning tree problem, Applied Soft Computing 9 (2) (2009) 625–631.
[38] M.F. Tasgetiren, Q.-K. Pan, P.N. Suganthan, A.H.-L. Chen, A discrete artificial bee colony algorithm for the total flowtime minimization in permutation flow shops, Information Sciences 181 (2011) 3459–3475.
[39] S. Sundar, A. Singh, A swarm intelligence approach to the quadratic minimum spanning tree problem, Information Sciences 180 (17) (2010) 3182–3191.
[40] O. Sýkora, L. Torok, I. Vrt'o, The cyclic antibandwidth problem, Electronic Notes in Discrete Mathematics 22 (2005) 223–227.
[41] V. Tereshko, Reaction-diffusion model of a honeybee colony's foraging behavior, in: Parallel Problem Solving from Nature VI, Lecture Notes in Computer Science, vol. 1917, Springer-Verlag, Berlin, 2000, pp. 807–816.
[42] G. Zhu, S. Kwong, Gbest-guided artificial bee colony algorithm for numerical function optimization, Applied Mathematics and Computation 217 (2010) 3166–3173.
[43] K. Ziarati, R. Akbari, V. Zeighami, On the performance of bee algorithms for resource-constrained project scheduling problem, Applied Soft Computing 11 (2011) 3720–3733.