A Hybrid Metaheuristic for the Cyclic Antibandwidth Problem

Manuel Lozano
Department of Computer Science and Artificial Intelligence,
University of Granada, Granada 18071, Spain, lozano@decsai.ugr.es

Abraham Duarte
Department of Computer Science,
University Rey Juan Carlos, Madrid, Spain, abraham.duarte@urjc.es

Francisco Gortázar
Department of Computer Science,
University Rey Juan Carlos, Madrid, Spain, francisco.gortazar@urjc.es

Rafael Martí
Department of Statistics and Operations Research
University of Valencia, Valencia, Spain, rmarti@uv.es

Abstract

In this paper, we propose a hybrid metaheuristic algorithm to solve the cyclic antibandwidth problem. This hard optimization problem consists in 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 (ABC) with tabu search (TS) to obtain high-quality solutions in short computational times. ABC 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, ABC and TS, 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.

Keywords. Artificial bee colony, local search, cyclic antibandwidth problem.

1 Introduction

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:

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

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

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

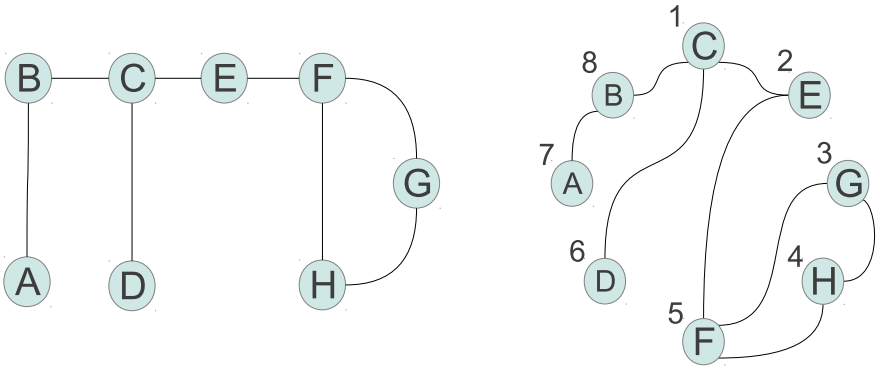

Figure 1: A graph and a circular labeling layout.

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

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

Figure 1-left shows an example of a graph G with 8 vertices and 8 edges. Figure 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 $d^+(\mathrm{A},\mathrm{B}) = |\varphi(\mathrm{A})-\varphi(\mathrm{B})| = |7-8| = 1$. Similarly, the counterclockwise distance between these two vertices is $d^-(\mathrm{A},\mathrm{B}) = 8-|\varphi(\mathrm{A})-\varphi(\mathrm{B})| = 8-|7-8| = 7$. 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 [2, 37]. 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 Figure 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 generate 10^6 solutions and compute for each one the value of the objective function for the antibandwidth problem (AB) and the cyclic antibandwidth (CAB). The correlation between both objective functions is rather small ($r = 0.13$). In addition, the best solution found for AB (among the generated solutions) is $\varphi_{AB_{best}} = (0, 12, 3, 16, 6, 10, 1, 11, 2, 13, 4, 14, 5, 15, 7, 19, 9, 17, 8, 18)$ whose objective function value is $AB(G, \varphi_{AB_{best}}) = 9$. This solution has a CAB-value of 7, which is relatively far from $\varphi(AB_{best})$ (with a percentage difference of 28.5%). Symmetrically, the best solution found for CAB problem $\varphi_{CAB_{best}} = (16, 5, 14, 2, 11, 7, 19, 9, 17, 8, 18, 4, 15, 6, 12, 3, 13, 1, 10, 0)$ has a value of $CAB(G, \varphi_{CAB_{best}}) = 8$ while the associated AB-value is 9 (with a percentage difference of

12.5%). Finally, remark that $\varphi_{AB_{best}}$ and $\varphi_{CAB_{best}}$ are completely different labelings.

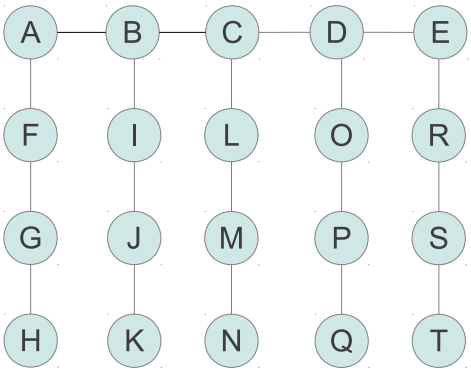

Figure 2: A caterpillar $P_{5,4}$ graph.

The CAB problem was proved to be NP-hard in Raspaud et al. [27]. This problem was originally introduced in Leung et al. [20] 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 [2]. The associated decision version of this problem consist in 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?" [34].

Exact results for the CAB problem are proved for some specific classes of graphs like paths [34], cycles [34], two dimensional meshes (Cartesian product of two paths), tori (Cartesian product of two cycles), and asymptotic results are obtained for hypercube graphs [27]. Dobrev et al. [4] extended these results to the case of Hamming graphs (Cartesian product of d-complete graphs). However, to the best of our knowledge, only one metaheuristic approach has been presented for finding the cyclic antibandwidth of general graphs, the *memetic algorithm* by Bansal and Srivastava [2]. (In Section 4.3, we compare our proposal with this algorithm).

In this paper, we propose a new approach based on the artificial bee colony methodology (ABC) [13, 14] for solving this problem. The ABC algorithm is a new population-based metaheuristic approach that is inspired by the intelligent foraging behavior of honeybee swarm. In essence, it implements memory structures based on the analogy with a bee population. Inspired in the types of bees and their different behaviour 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. Due to its simplicity and ease of implementation, the ABC algorithm has captured much attention and has exhibited state-of-the-art performances for a considerable number of problems, like forecasting stock markets [12], clustering [16], numerical function optimization [21], among others [1, 15, 18, 23, 24, 31, 33].

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 [13], the use of memory structures in optimization algorithms can be traced back to 1986 when Fred

Glover introduced the tabu search methodology. As a matter of fact, the term tabu search (TS) was coined in the same paper that introduced the term meta-heuristic [8]. TS is based on the premise that problem solving, in order to qualify as intelligent, must incorporate adaptive memory and responsive exploration. These elements allow 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 memoryless designs that heavily rely on semi-random processes that implement a form of sampling. 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 in 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 tabu search. In this paper, we explore this hybrid perspective taking some advanced tabu search 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 gives a brief overview of the ABC algorithm. Section 3 describes our hybrid ABC approach for the CAB problem. Section 4 provides an analysis of the performance of the proposed ABC and a comparison with the existing literature. Finally, Section 5 contains a summary of results and conclusions.

2 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 [35]. Employed bees are responsible from 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 [13, 14] proposed the artificial bee colony (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. 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 [14] and much work has been devoted to the development of extended models for these problems [5, 6, 7, 21, 36]. 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 [33], leaf-constrained minimum spanning tree problem [31], binary optimization problems [18], image segmentation [23], constrained optimization problems [15], design of recurrent neural networks [12], multi-objective optimization problems [1, 24], symbolic regression [17], and clustering [16].

3 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 behaviour 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 tabu search), 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 [10], which over the years

has incorporated elements such as local search, simulated annealing and elitism that are departures from the original methodology [19, 28, 22]. In this work, we use the cyclic antibandwidth problem for illustration purposes.

The rest of the section is organized as follows. In Section 3.1, we provide a general overview of the overall algorithm. In Section 3.2, we describe the initialization method. In Section 3.3, we propose the neighborhood procedure, which plays a fundamental role in our ABC. Finally, in Section 3.4, we provide details for the local search approach that is embedded in the proposed hybrid ABC algorithm to enhance the local intensification capability.

3.1 General Scheme of the Proposed Hybrid ABC Algorithm

An outline of our proposal, called HABC-CAB, is depicted in Figure 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 [2] to construct solutions for the CAB problem (Section 3.2). Then, the following steps are repeated until a termination criterion is met:

- *Empoyed bees phase.* It produces new solutions for the employed bees using the neighborhood operator Generate-Neighboring (Step 9). We propose a neighborhood operator that implements ejection chains [9], which have been successfully applied in the context of tabu search (Section 3.3).

- *Onlooker bees phase.* It produces new solutions for the onlookers using the tournament operator Binary-Tournament (Step 18). 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 [10]. However, the *tournament selection* is widely used in ABC applications due to its simplicity and ability to escape from local optima [32, 33]. 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 bee phase (Step 19).

- *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 29).

In order to enhance the exploitation capability of HABC-CAB, a fast local search method (FLS) is embedded in the proposed algorithm (Section 3.4). 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.

3.2 The RBFS Constructive Method

In this paper we use as initialization algorithm the Randomized Breadth-First Search constructive procedure proposed by Bansal and Srivastava [2]. For the completeness of the paper we briefly describe this method. Level algorithms [3] are constructive procedures based on the partition of the vertices of a graph in different levels, $L_1, ..., L_s$, such that the endpoints

```
   Input: G, t_max, limit, NP, p_LS
   Output: S_b
   // Initialization phase
 1 for i = 1 to NP do
 2     S ← Construct-Solution-RBFS(G);
 3     if U(0,1) < p_LS then
 4         S_i ← Fast-Local-Search (S);
 5     end
 6 end
 7 while computation time t_max not reached do
       // Empoyed bees phase
 8     for i = 1 to NP do
 9         E ← Generate-Neighboring(S_i);
10         if U(0,1) < p_LS then
11             E ← Fast-Local-Search (E);
12         end
13         if E is better than S_i then
14             S_i ← E;
15         end
16     end
       // Onlooker bees
17     for i=1 to NP do
18         S_j ← Binary-Tournament (S_1, ..., S_NP);
19         O ← Generate-Neighboring(S_j);
20         if U(0,1) < p_LS then
21             O ← Fast-Local-Search (O);
22         end
23         if O is better than S_j then
24             S_j ← 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 ← Construct-Solution-RBFS(G);
30             if U(0,1) < p_LS then
31                 S_i ← Fast-Local-Search (S);
32             end
33         end
34     end
       // Remember the best food source found so far
35     S_b ← Best-Solution-Found ();
36 end
```

Figure 3: Pseudocode algorithm for HABC-CAB.

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 breath 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 [2] 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 [2] and [3] for a more

detailed description.

3.3 Neighborhood Operator

The proposed neighborhood operator (NO_1) is inspired by the *ejection chain* methodology (Figure 4). This strategy is often used in connection with tabu search [9] and consists of 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.

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:

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

Given a solution φ, let us consider now two vertices, u and v, labeled with $\varphi(u)$ and $\varphi(v)$, respectively. In the context of the CAB problem, suppose that we assign the label $\varphi(v)$ to the vertex u, since this operation results in an increment of $CAB(\varphi, u)$. After this operation, we will have an unlabeled vertex, $v_{free} = v$, and a free label, $l_{free} = \varphi(u)$ (the one initially assigned to u). We can therefore consider labeling v_{free} with l_{free}. However, other labels may be better suited for this vertex. To check this possibility, we scan a subset L of possible labels for v_{free}, selected at random (Step 6), and find the label $l_{best} \in L \cup \{l_{free}\}$ that produces the best $CAB(\varphi, v_{free})$ (Step 7). Then, we assign l_{best} to v_{free} (Step 8). If the best label is l_{free}, then the graph will result completely labeled (Step 9). Otherwise, the best label belongs to L, and the new free vertex is the one which had this label (Steps 13-14). Then, the process is repeated for labeling this vertex. This process ends after performing I_{max} trial interchanges.

```
   Input: G, φ_i, I_max, n_L
   Output: φ
1  φ ← φ_i;
2  I ← 1;
3  v_free ← Select-Node-Random (V);
4  l_free ← φ(v_free);
5  while I ≤ I_max do
6      L ← Select-Labels-Random ({1, ..., n}, n_L);
7      l_best ← Best-Label (v_free, L ∪ {l_free});
8      φ(v_free) ← l_best;
9      if l_best = l_free then
10         v_free ← Select-Node-Random (V);
11         l_free ← φ(v_free);
12     else
13         Find u ∈ V such as φ(u) = l_best;
14         v_free ← u;
15     end
16     I ← I + 1;
17 end
18 φ(v_free) ← l_free;
```

Figure 4: Pseudocode algorithm for NO_1.

The main objective of NO_1 is 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. This operator starts with an initial vertex, u, chosen at random (Step 3) and performs the aforementioned process throughout I_{max} iterations. If during an iteration the graph becomes completely labeled, another initial vertex is selected at random (Steps 10-11). NO_1 finishes the construction of a neighborhood by assigning l_{free} to v_{free} (Step 18).

An important parameter of this operator 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 4.2).

3.4 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 [33], after the execution of the ABC algorithm, a local search method is applied to the best solution found. An alternative approach is proposed in [32] where the local search technique is applied (with a probability p_L) 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 versus 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 Figure 3).

Input: G, φ_i
Output: φ
1 $\varphi \leftarrow \varphi_i$;
2 $L \leftarrow \{1, ..., n-1\}$;
3 **while** $|L| > 0$ **do**
4 Select $l \in L$ randomly;
5 Find $u, v \in V$ such as $\varphi(u) = l$ and $\varphi(v) = l + 1$;
6 $\varphi' \leftarrow \text{Swap}(\varphi, u, v)$;
7 **if** $\min\{CAB(\varphi', u), CAB(\varphi', v)\} > \min\{CAB(\varphi, u), CAB(\varphi, v)\}$ **then**
8 $\varphi \leftarrow \varphi'$;
9 $L \leftarrow \{1, ..., n-1\}$;
10 **else**
11 $L \leftarrow L/\{l\}$;
12 **end**
13 **end**

Figure 5: Pseudocode algorithm for FLS.

The proposed FLS procedure has the following main contributions (Figure 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 [29].

- *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. 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:

$$l^-_{min}(w) = \min_{(w,t)\in E}\{\varphi(t) : \varphi(t) < \varphi(w), \}, l^-_{max}(w) = \max_{(w,t)\in E}\{\varphi(t) : \varphi(t) < \varphi(w)\}, \quad (5)$$

  and

$$l^+_{min}(w) = \min_{(w,t)\in E}\{\varphi(t) : \varphi(t) > \varphi(w)\}, l^+_{max}(w) = \max_{(w,t)\in E}\{\varphi(t) : \varphi(t) > \varphi(w)\}. \quad (6)$$

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

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

  and

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

  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 neighbourhood (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 4.

4 Computational Experiments

This section describes the computational experiments that we performed to assess the performance of the HABC-CAB algorithm presented in the previous section. Firstly, we detail the experimental setup (Section 4.1), 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 4.2); and (2) to compare the results of HABC-CAB with previous approaches for the CAB problem (Section 4.3).

4.1 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 236 instances classified in two groups: 164 instances with known optimum, and 72 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. [34] 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 [34].

- *Grids.* This data set consists of 24 graphs constructed as the Cartesian product of two paths P_{n_1} and P_{n_2} [27]. The size of these instances ranges from 81 to 1170. 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 [27]:

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

  if n_1 is even and n_2 is odd ($n_1 \geq n_2$), 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 [27]).

- *Hamming* graphs. This data set consists of 24 graphs constructed as the Cartesian product of d complete graphs K_{n_k}, for $k = 1, 2, ..., d$ [4]. The size of these instances ranges from 80 to 1152. The vertices in these graphs are d-tuples $(i_1, i_2, ..., i_d)$, where $i_k \in \{0, 1, ..., n_{k-1}\}$. Two vertices $(i_1, i_2, ..., i_d)$ and $(j_1, j_2, ..., j_d)$ 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. [4] proved that if $d \geq 2$ and $2 \leq n_1 \leq n_2 \leq ... \leq n_d$, then the optimal cyclic antibandwidth for this type of instances is given by:

$$CAB(\prod_{k=1}^{d} K_{n_k}) = \begin{cases} n_1 n_2 ... n_{d-1} & \text{if } n_{d-1} \neq n_d,\ d \geq 2 \\ n_1 n_2 ... n_{d-1} - 1 & \text{if } n_{d-1} = n_d \text{ and } n_{d-2} \neq n_{d-1},\ d \geq 3 \end{cases} \tag{15}$$

and

$$n_1 n_2 ... n_{d-1} - \min\{n_1 n_2 ... n_{d-2}, n_{q+1} ... n_{d-1}\} \leq CAB(\prod_{k=1}^{d} K_{n_k}) \leq n_1 n_2 ... n_{d-1} - 1, \tag{16}$$

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

Instances with unknown optimum

- *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 [11]. The size of these instances ranges from 30 to 900. 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 i-th row and j-th 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 $M_{ij} \neq 0$.

4.2 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 representative subset of instances (10 Harwell-Boeing, 10 complete binary trees and 10 Hamming instances with different densities and sizes). In order to have a fair comparison, all the ABC variants were stopped using a time limit of 150 seconds. 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 implement 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 ABC variants (*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.

I_{max}	n_L	*Av. Ran.*	*%Dev*	*%Best*
n	$0.5n$	11.01	5.05	6.67
$0.5n$	$0.5n$	10.15	4.87	10.00
$0.25n$	$0.5n$	10.48	4.84	6.67
$0.1n$	$0.5n$	9.83	5.05	6.67
n	$0.25n$	8.21	3.86	10.00
$0.5n$	$0.25n$	7.78	3.65	26.67
$0.25n$	$0.25n$	7.81	3.90	10.00
$0.1n$	$0.25n$	8.88	4.33	13.33
n	$0.1n$	4.51	1.95	56.67
$0.5n$	$0.1n$	4.56	2.30	43.33
$0.25n$	$0.1n$	5.86	3.22	26.67
$0.1n$	$0.1n$	7.33	3.76	6.67
n	$0.01n$	10.11	8.60	26.67
$0.5n$	$0.01n$	9.90	6.83	16.67
$0.25n$	$0.01n$	9.98	6.38	16.67
$0.1n$	$0.01n$	9.53	5.90	10.00

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

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 $n_L = 0.1n$ obtain the best results. Actually, these variants also outperform the others in terms of *%Dev.* and *%Best* (with the only exception of the HABC-CAB variant with $I_{max} = 0.5n$ and $n_L = 0.25n$). 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.

NO	*limit*	*Av. Ran.*	*%Dev*	*%Best*
NO_1	$2n$	4.96	0.51	66.67
NO_1	n	4.98	0.48	66.67
NO_1	$0.25n$	4.91	0.63	60.00
NO_1	$0.5n$	4.76	0.32	70.00
NO_2	$2n$	7.13	5.91	16.67
NO_2	n	6.80	5.90	13.33
NO_2	$0.5n$	7.31	6.35	20.00
NO_2	$0.25n$	6.61	6.27	16.67
NO_3	$2n$	7.70	6.31	23.33
NO_3	n	7.45	6.23	23.33
NO_3	$0.5n$	7.83	6.39	16.67
NO_3	$0.25n$	7.51	6.42	16.67

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

Results in Table 3 show that NO_1 clearly outperforms NO_2 and NO_3. Actually, there are no significant differencies 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.

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.

p_{LS}	*Av. Ran.*	*%Dev*	*%Best*
0	4.00	32.05	0.00
0.25	2.22	1.15	46.67
0.5	1.92	0.75	56.67
1	1.87	0.45	60.00

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

Results in Table 4 clearly show the merit of the local search method. Specifically, the ranking of the variant without local search is almost two times the ranking of the worst of the three remaining variants. Notice that all methods were executed for the same CPU time (150 seconds), 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.

4.3 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 [2]. This algorithm starts

by creating an initial population of solutions using the RBFS method (Section 3.2). As it is customary in evolutionary methods, the initial population evolves by applying three steps: selection, combination and mutation. The selection strategy is implemented by means of a classical tournament operator. The combination 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 two positions of 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 4.1. 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 seconds. Results are outlined in Table 5. For those instance sets with known optimum, the last two columns display the *%Dev* and *%Best* measures computed with respect to the corresponding optimum values. For those instance sets with unknown optimum, these two columns display an hyphen.

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	0.00
	MACAB	45.10	4.17	55.09	0.00
Harwell-Boeing	HABC-CAB	0.07	91.67	-	-
	MACAB	42.89	12.50	-	-
Caterpillars	HABC-CAB	0.09	80.00	-	-
	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
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	0.00	100.00	0.00	100.00
Grids	HABC-CAB	0.64	34.78	0.80	30.43
	MACAB	0.10	95.65	0.26	56.52

Table 5: HABC-CAB vs. MACAB.

Table 5 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 5 set of instances (out of 8). Specifically, our method outperforms the MACAB in *CBT*, *Hamming*, *Harwell-Boeing*, *Caterpillars*, and *Tori*. On the other hand, MACAB performs better than our HABC-CAB procedure in one of the instance sets (*Grids*). Regarding the set *Cycles*, our method obtains better results in terms of *%Dev*, but the MACAB method gets some more best solutions (*%Best*). Finally, both methods obtain similar results in the set of instances *Paths*.

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* and *Grids* 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.

To complement this information, 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 6 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 crit-

ical 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 ∼).

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	+
Cycles	161.0	115.0	81	∼
Paths	126.5	149.5	81	∼
Grids	19.0	234.0	81	−

Table 6: HABC-CAB vs. MACAB (Wilcoxon's test)

The Wilcoxon test reveals that: (1) HABC-CAB has the upper hand in the statistical comparison over its competitor on *CBT*, *Hamming*, *Harwell-Boeing*, *Caterpillars*, and *Tori*, which confirms the previous results, (2) MACAB statistically outperforms our proposal only in the case of the *Grids* set, and (3) there are not significant differences between these algorithms for the *Cycles* and *Paths* sets. In summary, this statistical analysis evidences that, in a global view, the comparison favors the proposed HABC-CAB algorithm.

5 Conclusions

In this paper, an artificial bee colony (ABC) algorithm, based on mimicking the food foraging behavior of honeybee swarms, is proposed as a method of solving the cyclic antibandwidth (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 [2] (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 [2]). 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 [30], bandwidth [26], and cutwidth [25] problems.

Acknowledgments

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

References

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

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

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

[4] 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.

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

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

[7] 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.

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

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

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

[11] Harwell-Boeing 2011. http://math.nist.gov/MatrixMarket/data/Harwell-Boeing/

[12] 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.

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

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

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

[16] D. Karaboga, C. Ozturk. A novel clustering approach: Artificial Bee Colony (ABC) algorithm. *Applied Soft Computing* 11:1 (2011) 652–657.

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

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] F.J. Rodriguez, 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.

[29] 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.

[30] 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.

[31] A. Singh. An artificial bee colony algorithm for the leaf-constrained minimum spanning tree problem. *Applied Soft Computing* 9:2 (2009) 625-631.

[32] 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.

[33] S. Sundar, A. Singh. A swarm intelligence approach to the quadratic minimum spanning tree problem. *Information Sciences* 180:17 (2010) 3182–3191.

[34] O. Sýkora, L. Torok, I. Vrt'o. The cyclic antibandwidth problem. *Electronic Notes in Discrete Mathematics* 22 (2005) 223–227.

[35] V. Tereshko. Reaction-diffusion model of a honeybee colony's foraging behavior. In: Parallel Problem Solving from Nature VI, vol. 1917 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 2000, 807-816.

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

[37] 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.