A hybrid metaheuristic for the cyclic antibandwidth problem

Manuel Lozano\textsuperscript{a,}\textsuperscript{*}, Abraham Duarte\textsuperscript{b}, Francisco Gortázar\textsuperscript{b}, Rafael Martí\textsuperscript{c}

\textsuperscript{a}Department of Computer Science and Artificial Intelligence, University of Granada, Granada 18071, Spain
\textsuperscript{b}Department of Computer Science, University Rey Juan Carlos, Madrid, Spain
\textsuperscript{c}Department of Statistics and Operations Research, University of Valencia, Valencia, Spain

\textbf{A R T I C L E  I N F O}

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.

\textbf{A B S T R A C T}

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.

© 2013 Elsevier B.V. All rights reserved.

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 \textit{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].
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 $\varphi$ 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 $\varphi$, let us define the clockwise distance $d^c(u,v) = |\varphi(u) - \varphi(v)|$ with $(u,v) \in E$ and, similarly the counterclockwise distance $d^d(u,v) = n - |\varphi(u) - \varphi(v)|$ with $(u,v) \in E$. Then, for a given circular labeling $\varphi$, the cyclic antibandwidth of $G$, referred to as $\text{CAB}(G,\varphi)$, is computed as follows:

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

The CAB problem consists of maximizing the value of $\text{CAB}(G,\varphi)$ over the set $\mathcal{P}$ of all possible labelings:

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

Note that in optimization terms, any labeling $\varphi$ of $G$ is a solution of the CAB problem stated in (2), with an objective function value, simply called value or $\text{CAB}(G,\varphi)$, 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 $\varphi$ of $G$, arranging the vertices in a cycle. Additionally, clockwise ($d^c$) and counterclockwise ($d^d$) 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 $d^c(A,B) = |\varphi(A) - \varphi(B)| = 7$. Similarly, the counterclockwise distance between these two vertices is $d^d(A,B) = 8 - |\varphi(A) - \varphi(B)| = 8 - 7 = 1$. In order to compute $\text{CAB}(G,\varphi)$, we evaluate $d^c$ and $d^d$ for the remaining 7 edges (shown in Table 1), reporting the minimum of all of them. Therefore, $\text{CAB}(G,\varphi) = 1$.

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 $P_{n_1,n_2}$ where each vertex $i$ in $P_{n_1,n_2}$ 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$^5$ 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$^5$ solutions is rather small ($r = 0.13$). In addition, the labeling with maximum AB value, out of the 10$^5$ generated, is $d^c(A,B) = 7 - 8 = 1$, $d^d(A,B) = 8 - 7 = 1$, $d^c(B,C) = 8 - 8 = 0$, $d^d(B,C) = 8 - 8 = 0$, $d^c(C,D) = 8 - 8 = 0$, $d^d(C,D) = 8 - 8 = 0$, $d^c(D,E) = 8 - 8 = 0$, $d^d(D,E) = 8 - 8 = 0$, $d^c(E,F) = 8 - 5 = 3$, $d^d(E,F) = 8 - 5 = 3$, $d^c(F,G) = 8 - 3 = 5$, $d^d(F,G) = 8 - 3 = 5$, $d^c(F,H) = 8 - 4 = 7$, $d^d(F,H) = 8 - 4 = 7$, $d^c(G,H) = 8 - 3 = 5$, $d^d(G,H) = 8 - 3 = 5$.

Table 1

<table>
<thead>
<tr>
<th>$d^c$</th>
<th>$d^d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d^c(A,B) = 7 - 8 = 1$</td>
<td>$d^d(A,B) = 8 - 7 = 1$</td>
</tr>
<tr>
<td>$d^c(B,C) = 8 - 8 = 0$</td>
<td>$d^d(B,C) = 8 - 8 = 0$</td>
</tr>
<tr>
<td>$d^c(C,D) = 8 - 8 = 0$</td>
<td>$d^d(C,D) = 8 - 8 = 0$</td>
</tr>
<tr>
<td>$d^c(D,E) = 8 - 5 = 3$</td>
<td>$d^d(D,E) = 8 - 5 = 3$</td>
</tr>
<tr>
<td>$d^c(F,G) = 8 - 3 = 5$</td>
<td>$d^d(F,G) = 8 - 3 = 5$</td>
</tr>
<tr>
<td>$d^c(F,H) = 8 - 4 = 7$</td>
<td>$d^d(F,H) = 8 - 4 = 7$</td>
</tr>
<tr>
<td>$d^c(G,H) = 8 - 3 = 5$</td>
<td>$d^d(G,H) = 8 - 3 = 5$</td>
</tr>
</tbody>
</table>

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

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

We can see that both labelings are very different.
The associated decision version of this problem consists of determining if $CAB(G, \phi)$ 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 departed 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:

- **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 algo-
rithm (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 $p_{\text{LS}}$) 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: $t_{\text{max}}$ denotes the computation time limit, $N_{P}$ determines the number of food sources which is equal to the number of employed and onlooker bees, $\text{limit}$ controls the generation of scout bees, and $p_{\text{LS}}$ 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, $L_{1}, \ldots, L_{n}$, 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 $\text{CAB}(\phi, u)$ of vertex $u$ for a labeling $\phi$. In mathematical terms it can be computed as follows:

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

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

$$\text{CAB}(G, \phi) = \min_{u \in V} \{\text{CAB}(\phi, u)\}. \quad (4)$$

From Eq. (4), we may intuit that a prominent strategy to discover enhanced solutions starting from a given solution involves the reassignment 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 $\text{CAB}(\phi, u)$. This is made by scanning a subset $L$ of possible labels for $u$, selected at random, and locating the label $l_{\text{best}} \in L$ that produces the fit

test CAB value for $u$. If $l_{\text{best}}$ 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, $v_{\text{free}}$ (the node having previously $l_{\text{best}}$ as label), and a free label, $l_{\text{free}}$ (the one initially assigned to $u$). We can consider labeling $v_{\text{free}}$ with $l_{\text{free}}$; 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 $L \cup \{l_{\text{free}}\}$ that induces the highest CAB value for $v_{\text{free}}$ is finally assigned to this vertex.

3. If the best label for $v_{\text{free}}$ is $l_{\text{free}}$, 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, $v_{\text{free}}$, and the procedure for labeling this vertex should be launched.

4. After performing $t_{\text{max}}$ trial interchanges, NO$_{1}$ finishes the construction of a neighborhood by finally assigning $l_{\text{free}}$ to $v_{\text{free}}$.

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).

#### Fig. 4. Pseudocode algorithm for NO$_{1}$.

**Input:** $G$, $\phi$, $t_{\text{max}}$, $n_{L}$

**Output:** $\phi$

1. $\phi \leftarrow \phi_{i}$
2. $I \leftarrow 1$
3. $v_{\text{free}} \leftarrow \text{Select-Node-Random}(V)$
4. $l_{\text{free}} \leftarrow \phi(v_{\text{free}})$
5. while $I \leq t_{\text{max}}$ do
6. $L \leftarrow \text{Select-Labels-Random}\{(1, \ldots, n_{L})\}$
7. $l_{\text{best}} \leftarrow \text{Best-Label}(v_{\text{free}}, L \cup \{l_{\text{free}}\})$
8. $\phi(v_{\text{free}}) \leftarrow l_{\text{best}}$
9. if $l_{\text{best}} \equiv l_{\text{free}}$ then
10. $v_{\text{free}} \leftarrow \text{Select-Node-Random}(V)$
11. $l_{\text{free}} \leftarrow \phi(v_{\text{free}})$
12. else
13. Find $u \in V$ such as $\phi(u) = l_{\text{best}}$
14. $v_{\text{free}} \leftarrow u$
15. $I \leftarrow I + 1$
16. end
17. end
18. $\phi(v_{\text{free}}) \leftarrow l_{\text{free}}$
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 pLs) 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 pLs 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. 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, φ). Then, changing the label of a particular vertex u to increase its cyclic antibandwidth CAB(φ, u), does not necessarily imply that CAB(φ, u) also increases.

Therefore we 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(φ’, u) and CAB(φ’, v) in constant time (Step 7), after exchanging the label of u with the label of v. Specifically, it records, for each vertex w ∈ V, lmin(w), lmax(w), lmin(w), and lmax(w), which are defined as follows:

\[
\begin{align*}
\ell_{\text{min}}(w) = & \begin{cases} 
\min S_u & \text{if } S_u \neq 0 \\
\phi(w) & \text{otherwise}
\end{cases} \\
\ell_{\text{max}}(w) = & \begin{cases} 
\max S_u & \text{if } S_u \neq 0 \\
\infty & \text{otherwise}
\end{cases}
\end{align*}
\]

and

\[
\begin{align*}
\ell_{\text{min}}(w) = & \begin{cases} 
\min S_v & \text{if } S_v \neq 0 \\
\phi(w) & \text{otherwise}
\end{cases} \\
\ell_{\text{max}}(w) = & \begin{cases} 
\max S_v & \text{if } S_v \neq 0 \\
\infty & \text{otherwise}
\end{cases}
\end{align*}
\]

where

S_u = \{\phi(t) : (w, t) ∈ E ∧ \phi(t) < \phi(w)\}

and

S_v = \{\phi(t) : (w, t) ∈ E ∧ \phi(t) > \phi(w)\}.

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

\[
\text{CAB}(\phi’, u) = \min\{|l + 1 - \ell_{\text{min}}(u)|, n - |l + 1 - \ell_{\text{max}}(u)|\},
\]

and

\[
\text{CAB}(\phi’, v) = \min\{|l - \ell_{\text{min}}(v)|, n - |l - \ell_{\text{max}}(v)|\},
\]

where \(l = \phi(u)\) and remind that \(\phi’(u) = l + 1\) and \(\phi’(v) = l\).

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.

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.optiscom.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

\[
\text{CAB}(P_n) = \left\lceil \frac{n}{2} \right\rceil - 1.
\]

(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 500 to 1000. For a cycle \( C_n \) with \( n \) vertices the optimal cyclic antibandwidth is

\[
\text{CAB}(C_n) = \left\lceil \frac{n}{2} \right\rceil - 1,
\]

(10)
as shown in Sýkora et al. [40].

- **Grids.** This data set consists of 23 graphs constructed as the Cartesian product of two paths \( P_{n1} \) and \( P_{n2} \) [32]. The size of these instances ranges from 81 to 1089. They are also called two dimensional meshes. For a grid \( P_{n1} \times P_{n2} \) with \( n = n_1 \cdot n_2 \) vertices the optimal cyclic antibandwidth fulfills that [32]:

\[
\left\lceil \frac{n_1(n_1 - 1)}{2} \right\rceil \leq \text{CAB}(P_{n1} \times P_{n2}) \leq \left\lceil \frac{n_2(n_2 - 1)}{2} \right\rceil.
\]

(11)

if \( n_1 \) is even and \( n_2 \) is odd \( (n_1 \geq n_2) \), and

\[
\text{CAB}(P_{n1} \times P_{n2}) = \frac{n_2(n_1 - 1)}{2}.
\]

(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

\[
\text{CAB}(C_n \times C_n) = \frac{n(n-2)}{2},
\]

(13)

if \( n \) is even, and

\[
\text{CAB}(C_n \times C_n) = \frac{(n-2)(n+1)}{2},
\]

(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 \( d \geq 2 \) and \( 2 \leq n_1 \leq n_2 \leq \cdots \leq n_d \), then the optimal cyclic antibandwidth for this type of instances is given by:

\[
\text{CAB}\left(\prod_{k=1}^{d} K_{n_k}\right) = \begin{cases} 
0 & \text{if } n_{d-1} \neq n_d, d \geq 2 \\
1 & \text{if } n_{d-1} = n_d \text{ and } n_{d-2} \neq n_{d-1}, d \geq 3
\end{cases}
\]

(15)

and

\[
n_1n_2\ldots n_{d-1} - \min\{n_1n_2\ldots n_{d-2}, n_{q+1}\ldots n_{d-1}\} \leq \text{CAB}\left(\prod_{k=1}^{d} K_{n_k}\right) \leq n_1n_2\ldots n_{d-1} - 1.
\]

(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 \( n_q = n_p \).

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 \( K_{n_1} \) and \( K_{n_2} \) 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 \( 2^{d-1} \) 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, \( P_{n_1} \times K_{n_2} \), is constructed using the path \( P_{n_1} \) and \( n_2 \) 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 node is 1,2...d 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 \( i \)-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 \( M_{ij} \neq 0 \).

5.2. Study of the neighborhood operator and the FLS method

In this section, we investigate the effects of the two search strategies, NO1 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 executed once for each problem instance. In our first preliminary experiment, we study the effect of the search parameters, \( t_{max} \) and \( n_v \), in the neighborhood operator, NO1. Specifically, we implemented 16 different variants of HABC-CAB with
I_{max} = [0.1n, 0.25n, 0.5n, n] and nL = [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 nL 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 nL = 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 nL 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 nL = 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, NO1. In order to do this, it was compared with a standard neighborhood operator (NO2) that has been widely used in the literature for a variety of graph arrangement problems. Specifically, NO2 is a swap operator that interchanges the labels associated with two vertices chosen at random. We have also examined a third operator, NO3, 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 NO1 clearly outperforms NO2 and NO3. Actually, there are no significant differences among all NO1 variants. However, any NO1 variant is better than any NO2 or NO3 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.).

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, nL = 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_{LS} (n \in \{250, 500, 750, 1000\} and p_{LS} \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://scis.es/ubic/Results-CAB.xlsx (the results of the algorithms on the Harwell-Boeing instances may be found in Appendix A).
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 −−).

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 stars, 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 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 stars, 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 component 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.
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

<table>
<thead>
<tr>
<th>Instance</th>
<th></th>
<th>MACAB</th>
<th>HABC-CAB</th>
</tr>
</thead>
<tbody>
<tr>
<td>pores.1 mtx.rnd</td>
<td></td>
<td>30</td>
<td>6</td>
</tr>
<tr>
<td>ibm32 mtx.rnd</td>
<td></td>
<td>32</td>
<td>6</td>
</tr>
<tr>
<td>bcspwrl01 mtx.rnd</td>
<td></td>
<td>39</td>
<td>9</td>
</tr>
<tr>
<td>bcsstk01 mtx.rnd</td>
<td></td>
<td>48</td>
<td>6</td>
</tr>
<tr>
<td>bcspwrl02 mtx.rnd</td>
<td></td>
<td>49</td>
<td>12</td>
</tr>
<tr>
<td>curtis54 mtx.rnd</td>
<td></td>
<td>54</td>
<td>7</td>
</tr>
<tr>
<td>will57 mtx.rnd</td>
<td></td>
<td>57</td>
<td>7</td>
</tr>
<tr>
<td>impcol_b mtx.rnd</td>
<td></td>
<td>59</td>
<td>3</td>
</tr>
<tr>
<td>ash85 mtx.rnd</td>
<td></td>
<td>85</td>
<td>13</td>
</tr>
<tr>
<td>nos4 mtx.rnd</td>
<td></td>
<td>100</td>
<td>22</td>
</tr>
<tr>
<td>dwt_234 mtx.rnd</td>
<td></td>
<td>117</td>
<td>22</td>
</tr>
<tr>
<td>bcspwrl03 mtx.rnd</td>
<td></td>
<td>118</td>
<td>18</td>
</tr>
<tr>
<td>bcsstk07 mtx.rnd</td>
<td></td>
<td>420</td>
<td>12</td>
</tr>
<tr>
<td>bcsstk06 mtx.rnd</td>
<td></td>
<td>420</td>
<td>12</td>
</tr>
<tr>
<td>impcol_d mtx.rnd</td>
<td></td>
<td>425</td>
<td>30</td>
</tr>
<tr>
<td>can_445 mtx.rnd</td>
<td></td>
<td>445</td>
<td>36</td>
</tr>
<tr>
<td>494_bus mtx.rnd</td>
<td></td>
<td>494</td>
<td>53</td>
</tr>
<tr>
<td>dwt_503 mtx.rnd</td>
<td></td>
<td>503</td>
<td>14</td>
</tr>
<tr>
<td>sherman4 mtx.rnd</td>
<td></td>
<td>546</td>
<td>258</td>
</tr>
<tr>
<td>dwt_592 mtx.rnd</td>
<td></td>
<td>592</td>
<td>33</td>
</tr>
<tr>
<td>662_bus mtx.rnd</td>
<td></td>
<td>662</td>
<td>58</td>
</tr>
<tr>
<td>nos6 mtx.rnd</td>
<td></td>
<td>675</td>
<td>328</td>
</tr>
<tr>
<td>685_bus mtx.rnd</td>
<td></td>
<td>685</td>
<td>16</td>
</tr>
<tr>
<td>can_715 mtx.rnd</td>
<td></td>
<td>715</td>
<td>21</td>
</tr>
</tbody>
</table>

References


