Tabu Search for the Cyclic Bandwidth Problem☆

Eduardo Rodriguez-Tello*,a, Hillel Romero-Monsivais[a], Gabriel Ramirez-Torres[a], Frédéric Lardeux[b]

[a] *CINVESTAV-Tamaulipas, Information Technology Laboratory*
Km. 5.5 Carretera Victoria-Soto La Marina, 87130 Victoria Tamps., Mexico
[b] *LERIA, Université d'Angers.*
2 Boulevard Lavoisier, 49045 Angers, France

Abstract

The *Cyclic Bandwidth* problem (CB) for graphs consists in labeling the vertices of a guest graph G by distinct vertices of a host cycle C_n (both of order n) in such a way that the maximum distance in the cycle between adjacent vertices in G is minimized. To the best of our knowledge, this is the first research work investigating the use of metaheuristic algorithms for solving this challenging combinatorial optimization problem in the case of general graphs.

In this paper a new carefully devised *Tabu Search* algorithm, called TScb, for finding near-optimal solutions for the CB problem is proposed. Different possibilities for its key components and input parameter values were carefully analyzed and tuned, in order to find the combination of them offering the best quality solutions to the problem at a reasonable computational effort.

Extensive experimentation was carried out, using 113 standard benchmark instances, for assessing its performance with respect to a Simulated Annealing (SAcb) implementation. The experimental results show that there exists a statistically significant performance amelioration achieved by TScb with respect to SAcb in 90 out of 113 graphs (79.646%). It was also found that our TScb algorithm attains 56 optimal solutions and establishes new better upper bounds for the other 57 instances. Furthermore, these competitive results were obtained expending reasonable computational times.

Key words:
cyclic bandwidth problem, tabu search, best-known bounds

1. Introduction

The *Cyclic Bandwidth* problem (CB) is a graph embedding problem. It was first stated by Leung *et al.* in 1984 in relation with the design of a ring interconnection network [1]. Their aim was to find an arrangement on a cycle for a set V of computers with a known communication pattern, given by the graph $G(V, E)$, in such a way that every message sent can arrive at its destination in at most k steps. The decision problem corresponding to the CB is known to be NP-complete [2], and arises also in other important application areas like VLSI designs [3], data structure representations [4], code design [5] and interconnection networks for parallel computer systems [6].

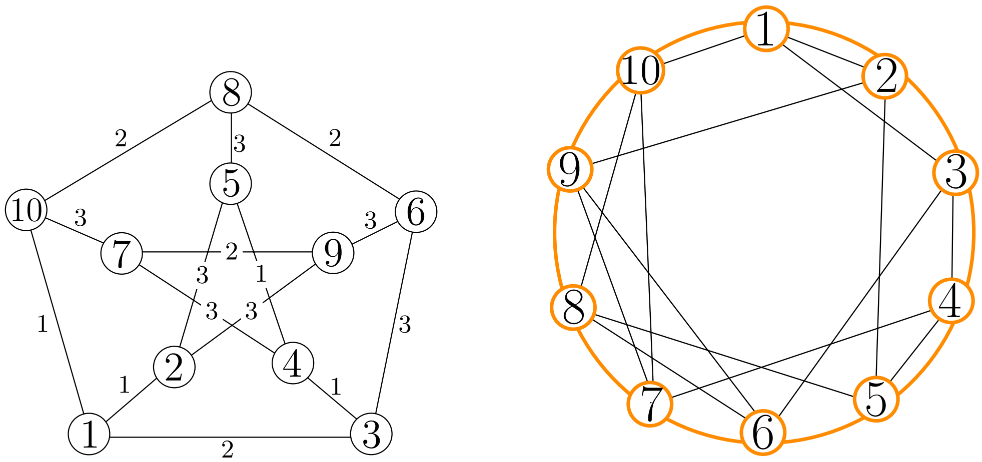
The CB problem can be formally defined as follows. Let $G(V, E)$ be a finite undirected graph (guest) of order n and C_n a cycle graph (host) with vertex set $|V'| = n$ and edge set E'. Given an injection $\varphi : V \to V'$, which represents an embedding of G in C_n, the cyclic bandwidth (the cost) for G with respect to φ is defined as:

$$B_C(G, \varphi) = \max_{uv \in E}\{|\varphi(u) - \varphi(v)|_n\}, \tag{1}$$

where $|x|_n = \min\{|x|, n - |x|\}$ $(1 < |x| < n - 1)$ is called the *cyclic distance*, and $\varphi(u)$ denotes the label associated to vertex u.

☆This research work was partially funded by the following projects: 51623 Fondo Mixto CONACyT y Gobierno del Estado de Tamaulipas; CONACyT 99276, Algoritmos para la Canonización de Covering Arrays.

*Corresponding author.
Email addresses: ertello@tamps.cinvestav.mx (Eduardo Rodriguez-Tello), hromero@tamps.cinvestav.mx (Hillel Romero-Monsivais), grtorres@tamps.cinvestav.mx (Gabriel Ramirez-Torres), lardeux@info.univ-angers.fr (Frédéric Lardeux)



(a) A labeling φ for graph G. **(b)** G embedded in the cycle C_n

Figure 1: Example of a cyclic bandwidth problem instance.

Then the CB problem consists in finding an embedding φ^*, such $B_C(G, \varphi^*)$ is minimum, i.e.,

$$B_C(G, \varphi^*) = \min_{\varphi \in \mathscr{E}} \{B_C(G, \varphi)\}, \tag{2}$$

where $\mathscr{E}$ is the set of all possible embeddings. The embedding φ^* satisfying this condition is called an optimal embedding.

Note that an embedding can also be seen as a labeling of the guest graph G using distinct vertices of the host graph C_n[1]. The cost of such an embedding is the maximum distance in C_n between two adjacent vertices in the guest graph G.

For instance, consider the graph $G(V, E)$ of order $n = 10$ depicted in Figure 1(a) with the labeling φ given by the numbers shown inside each vertex. The cyclic distance of each edge $uv \in E$ is calculated using the expression $\min\{|\varphi(u) - \varphi(v)|, n - |\varphi(u) - \varphi(v)|\}$ and represented by the number associated to that edge. For this particular labeling φ, the cyclic bandwidth of G is $B_C(G, \varphi) = 3$. The resulting embedding of the graph G in a cycle graph C_n is presented in Figure 1(b) for illustrative purposes.

The CB problem is a natural extension of the well-known *bandwidth minimization* (BM) problem for graphs [7], which consists in embedding the vertices of a guest graph G in a host path P (both of order n) in such a way that the maximum distance in the path between adjacent vertices in G is minimized. Formally, the bandwidth (the cost) for G with respect to φ is defined as:

$$B_P(G, \varphi) = \max_{uv \in E} \{|\varphi(u) - \varphi(v)|\}. \tag{3}$$

There exist some special graphs (2- and 3-dimensional meshes, hypercubes, complete trees) for which it has been demonstrated that their optimal cyclic bandwidth and bandwidth are equal [6, 8]. It has permitted to establish the following bound relation between bandwidth $B_P(G)$ and cyclic bandwidth $B_C(G)$ of a general graph [6]:

$$\frac{1}{2} B_P(G) \leq B_C(G) \leq B_P(G). \tag{4}$$

Even if both combinatorial optimization problems are related, an algorithm specially devised for one of them is not expected to perform well on the other. This is illustrated by considering example the graph *ibm32* from the Harwell-Boeing Sparse Matrix Collection[2] (other tested graphs give similar results), which is composed of $n = 32$

[1] Hereafter the terms *embedding* and *labeling* are used indistinctly.

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

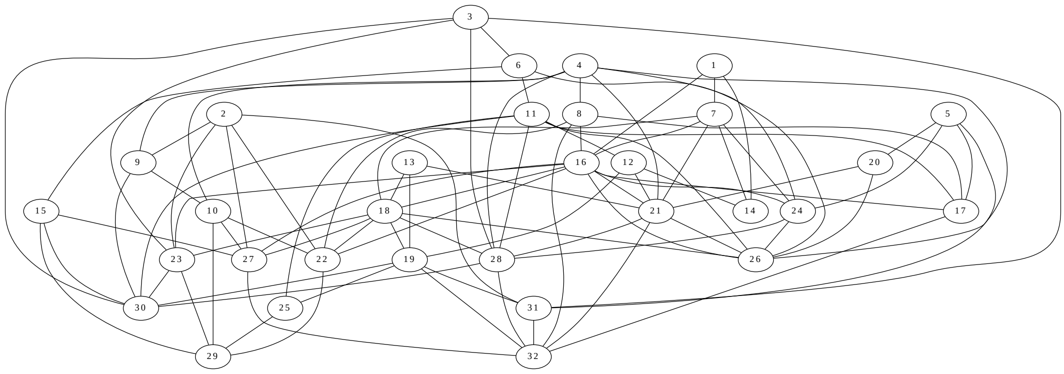


Figure 2: Graph *ibm32* from the Harwell-Boeing Sparse Matrix Collection.

vertices (see Figure 2). For this particular graph 1000 local optimal embeddings were generated by executing a first improvement descent algorithm for the CB problem, starting from different randomly generated initial solutions (embeddings). Then, for each of the resulting embeddings the cyclic bandwidth and the bandwidth objective functions were computed according to (1) and (3), respectively. The Pearson's correlation coefficient between these two sets of values is small ($r = -0.18$), showing the weak correlation (if any) among the solutions of both problems. Furthermore, the embedding with the optimal cyclic bandwidth ($B_C(G) = 9$) for this particular graph results in bandwidth value of 31, which is very far from the optimal value ($B_P(G) = 11$) [9]. In this context there exist a real need for devising new algorithms specially devoted for solving the CB problem.

This paper aims at developing, to the extent of our knowledge, the first tabu search (TS) algorithm implementation (hereafter called TSCB) for finding near-optimal solutions for the CB problem. To achieve this, different possibilities for its key components were carefully designed after an in-depth analysis of the given problem. The TSCB input parameter values yielding the best quality solutions to the problem at a reasonable computational effort were determined by employing a tuning methodology based on combinatorial interaction testing [10, 11]. The performance of the new solution approach is investigated and compared with respect to a Simulated Annealing (SACB) algorithm through extensive experimentation over a full test-suite composed of 113 benchmark graphs with a number of vertices ranging from 9 to 8192. These graphs are divided into two different sets. The first one is composed of 85 standard graphs with known optimal solutions from seven different families including paths, cycles, meshes, trees, hypercubes and caterpillars. Certain of them were previously used to evaluate a branch & bound (B&B) algorithm for the CB problem [12]. The second set contains 28 graphs produced from real-world scientific and engineering applications, some of which were recently used as benchmark instances for other graph labeling problem [13, 14]. The computational results for the first set show that TSCB attained the optimal solution for 49 graphs (57.647%) and found good quality near-optimal solutions for the rest of the instances. For the second set, TSCB was able to reach the optimal solution for 7 instances and to ameliorate the existing theoretical upper bounds [15] for the rest of the instances. The statistical analyses performed on both experiments confirm that there exists a statistically significant increase in performance achieved by TSCB with respect to SACB in 90 out of 113 graphs (79.646%), highlighting the suitability of the proposed TSCB algorithm. Furthermore, these competitive results were obtained expending reasonable computational times.

The remainder of this manuscript is organized as follows. In Section 2 a brief review of some representative related work is given. The main components of the proposed TSCB algorithm are detailed in Section 3. Then, computational experiments are presented in Section 4 which are devoted to determine the best parameter settings for TSCB and to compare its performance with respect to a Simulated Annealing (SACB) algorithm in terms of the best-known lower bounds of the state-of-the-art. Section 5 experimentally investigates the influence of some key features in the global performance of the proposed TSCB algorithm. Finally, the main conclusions of this work and some possible directions for future research are provided in Section 6.

2. Relevant Related Work

The CB problem has a strong connexion with other graph embedding problems like the bandwidth minimization [7, 16], antibandwidth [1, 17], and cyclic antibandwidth [1].

The bandwidth minimization problem has already been described in Section 1. On the other hand, the antibandwidth and the cyclic antibandwidth problems consist in labeling the vertices of a guest graph G of order n by distinct vertices of either a host path P or a host cycle C, respectively, in such a way that the minimum distance between adjacent vertices in G, measured in the host, is maximized.

The bandwidth problem has been the object of extensive research in the past. Different exact and heuristic algorithms for solving it have been reported in the literature. Some relevant heuristic algorithms for solving the bandwidth problem are: Tabu Search [18], GRASP with Path Relinking [19], Particle Swarm Optimization [20], Simulated Annealing [21] and Variable Neighborhood Search [22].

The work published about the antibandwidth problem was mainly devoted to find polynomial time exact algorithms for solving some special instances of the antibandwidth problem on specific classes of graphs: paths, cycles, special trees, complete and complete bipartite graphs, meshes, and tori [17, 23–27]. Some exceptions are metaheuristic algorithms including: Memetic Algorithms [28], GRASP with Path Relink [13] and Variable Neighborhood Search [14].

The cyclic antibandwidth problem has been exactly solved for some specific families of graphs like paths, cycles, two dimensional meshes and tori [29]. Asymptotic results were also obtained for hypercube and Hamming graphs [27, 30]. However, only two metaheuristic algorithms have been reported for this important problem, a Memetic Algorithm [31] and a hybrid algorithm combining the Artificial Bee Colony methodology with Tabu Search [32].

In spite of its practical and theoretical importance, less attention has been paid to the CB problem with regard to other graph embedding problems. Up to now, most of the research on this important problem has been concentrated upon the theoretical study of its properties, with the aim of finding exact solutions for certain specific families of graphs. Next, a brief review of these studies is presented.

In 2002, Zhou proposed a systematic method for obtaining lower bounds for the bandwidth and cyclic bandwidth problems in terms of some distance- and degree-related parameters of the graph [33]. The main idea of this method is to relax the condition of embedding the graph G on the host graph with the aid of a graphical parameter possessing some kind of monotonic property. This method has been demonstrated to be efficient when the parameters are chosen appropriately. The author concludes that this method yields to a number of lower bounds for the ordinary and cyclic bandwidths. In both cases, it gives rise to new estimations, as well as improvements of some known results.

Later, de Klerk *et al.* [34] proposed two new semidefinite programming (SDP) relaxations of the bandwidth and cyclic bandwidth based on the quadratic assignment problem (QAP) reformulation. The bounds produced by this method were tested for some special graphs showing that they are tight for paths, cliques and complete bipartite graphs. However, these bounds are not tight for hypercubes, rectangular grids and complete k-level t-ary trees.

In 1995 Yuan and Zhou [35] demonstrated that for unit interval graphs, there exists a labeling which is simultaneously optimal for the following seven labeling problems: bandwidth, cyclic bandwidth, profile, cutwidth, modified cutwidth and bandwidth sum. Following this idea, in [8] Lam *et al.* made a characterization of graphs with equal bandwidth and cyclic bandwidth which includes planar graphs, triangulation meshes and grids with some specific characteristics.

The CB problem has been exactly solved for twenty small instances ($n < 40$) using a branch & bound algorithm (B&B) recently proposed by Romero-Monsivais *et al.* [12]. However, this algorithm becomes impractical when the number of vertices n in the studied graph increases, since the size of the search space suffers a combinatorial explosion. Therefore, there is a need for heuristic methods to address the CB problem in reasonable time.

3. A new tabu search algorithm

The Tabu Search (TS) algorithm was first proposed by Glover [36] and it has been widely used for solving a large number of combinatorial optimization problems [37–43]. A particularity of TS is that it explicitly employs the history of the search, both to escape from local minima and to implement an explorative strategy.

The pseudo-code of our TS implementation, called TScb, is presented in Algorithm 1. It starts with a randomly generated solution φ (see Subsection 3.2), then it proceeds iteratively to visit a series of locally best configurations

Algorithm 1: Tabu Search algorithm

input: A finite undirected graph $G(V, E)$, neighborhood function $\mathcal{N}$, evaluation function B_C, maximum non-improving neighboring solutions *maxNI*
output: The best solution found φ^*

$\varphi \leftarrow$ `GenerateInitialSolution()`
$\varphi^* \leftarrow \varphi$
`InitializeTabuList()`
$NI \leftarrow 0$
while *stop condition not met* **do**
    $\varphi' \leftarrow$ `ChooseBestAdmissible(`φ`)` `/*` $\{\varphi' \in \mathcal{N}(\varphi) \mid \varphi'$ `non-tabu or aspiration condition holds`$\}$ `*/`
    `UpdateTabuListAndAspirationCondition()`
    $\varphi \leftarrow \varphi'$
    if $B_C(G, \varphi) < B_C(G, \varphi^*)$ **then**
        $\varphi^* \leftarrow \varphi$
        $NI \leftarrow 0$
    else
        $NI \leftarrow NI + 1$
    end
    if $NI > maxNI$ **then** `Diversification(`φ`)`
end
return φ^*

following a neighborhood function $\mathcal{N}(\varphi)$. At each iteration, a best neighbor φ' is chosen to replace the current configuration φ, even if the former does not improve the current one (refer to Subsection 3.3). This operation is called a *move*. In order to explore consecutive local optimal solutions and to avoid the occurrence of cycles, during the search, it is necessary to prohibit visiting twice the same configuration. Storing all the already visited configurations is very expensive and generally impossible in practice. An alternative approach is to only store the last moves in a *recency-based* memory structure, called *tabu list* $\mathcal{L}$ (see Subsection 3.4). The basic idea behind this memory is to record the attributes of each visited solution and to forbid the algorithm to visit again this configuration during the next $\mathcal{T}$ iterations ($\mathcal{T}$ is called the *tabu tenure*). In the case that more than one move have the same best cost value, one among them is randomly selected. In some cases, the tabu list may be too restrictive since certain forbidden moves could produce a solution better than the best solution found so far. To cope with this issue, an aspiration criterion is applied to accept those exceptional quality solutions (refer to Subsection 3.5).

Next, the main components of our TSCB implementation are discussed in detail. For some of these components different possibilities were analyzed (see Subsection 4.4) in order to find the combination of them which offers the best quality solutions at a reasonable computational effort.

3.1. Search space, representation and evaluation function

Given a guest graph $G = (V, E)$ of order $|V| = n$ and C_n a cycle graph (host) with vertex set $|V'| = n$ and edge set E', the search space $\mathscr{E}$ for the CB problem is composed of all possible embeddings (solutions) of G in C_n, $\varphi : V \to V'$. Therefore, there exist $(n-1)!/2$ possible embeddings for a graph with n vertices[3].

In our TSCB algorithm an embedding (labeling) φ is represented as an array l of integers with length n, which is indexed by the vertices and whose i-th value $l[i]$ denotes the label assigned to the vertex i. The quality $B_C(G, \varphi)$ of the embedding φ is evaluated by using (1).

3.2. Initial solution

The initial solution is the starting embedding used for the algorithm to begin the search of better configurations in the search space $\mathscr{E}$. In this implementation the starting solution is generated randomly.

[3] Because each one of the $(n-1)!$ embeddings can be reversed to obtain the same cyclic bandwidth.

3.3. Neighborhood functions

Given that TScb is a Local Search (LS) algorithm, then a neighborhood function must be defined. The main objective of the neighborhood function is to identify the set of potential solutions which can be reached from the current solution in an LS algorithm. Formally, a neighborhood relation is a function $\mathcal{N} : \mathscr{E} \rightarrow 2^{\mathscr{E}}$ that assigns to every potential solution (an embedding) $\varphi \in \mathscr{E}$ a set of neighboring solutions $\mathcal{N}(\varphi) \subseteq \mathscr{E}$, which is called the neighborhood of φ.

The results of our preliminary experimentations lead us to identify three suitable neighborhood structures for the CB problem. The logic behind these neighborhood relations, used in our TScb algorithm, is to identify those critical vertices in the graph which determine its cyclic bandwidth in order to "repair" them. These neighborhood functions are partially inspired by the work of Martí *et al.* on the bandwidth problem [18], which also employed a Tabu Search algorithm with a neighborhood function based on the reparation of critical vertices of the graph.

Before introducing these neighborhood structures, some preliminary concepts used in their definition are presented. Let us define the cyclic bandwidth $B_C(u, \varphi)$ for a vertex u with respect to the embedding φ as follows:

$$B_C(u, \varphi) = \max_{v \in \mathcal{A}(u)} \{|\varphi(u) - \varphi(v)|_n\}, \tag{5}$$

where $\mathcal{A}(u)$ denotes the set of adjacent vertices of u, with cardinality $deg(u)$. We define a vertex u as critical if its cyclic bandwidth $B_C(u, \varphi)$ is close to $B_C(G, \varphi)$. Thus, the set $C(\varphi) \subseteq V$ of critical non-tabu vertices can be defined with the following expression for $0 < \alpha < 1$:

$$C(\varphi) = \{u \in V \ : \ B_C(u, \varphi) \geq \alpha B_C(G, \varphi), \ u \notin \mathcal{L}\}. \tag{6}$$

Let $\mathcal{S}(u) \subseteq \mathcal{A}(u)$ be a set of suitable swapping vertices for u. $\mathcal{S}(u)$ contains those non-tabu vertices adjacent to u whose label values are closer to $mid(u)$ than $\varphi(u)$:

$$\mathcal{S}(u) = \{v \in \mathcal{A}(u) \ : \ |mid(u) - \varphi(v)|_n < |mid(u) - \varphi(u)|_n\}, \tag{7}$$

where $mid(u)$ corresponds to the middle point of the shortest path in the host graph C_n containing all the vertices adjacent to u and is delimited by the rightmost $r(u)$ and the leftmost $l(u)$ vertices:

$$mid(u) = \left\lfloor \frac{l(u) + r(u) + a}{2} \right\rfloor \bmod n, \tag{8}$$

with $a = n$, if $l(u) > r(u)$ and $a = 0$ otherwise. In order to identify the values $r(u)$ and $l(u)$, the ordered sequence $\mathcal{B} = \left\{b_1, b_2, \ldots, b_{deg(u)}\right\} \cup \{b_1 + n\}$ which contains the labels currently assigned to the vertices in $\mathcal{A}(u)$ is constructed. Then, the expression:

$$i^* = \arg\max_{i \leq deg(u)} \left((b_{i+1} - b_i)\right), \tag{9}$$

is evaluated over the elements of $\mathcal{B}$ to obtain the values $l(u) = b_{i^*+1}$ and $r(u) = b_{i^*}$.

Let $swap(\varphi, u, v)$ be a function allowing to exchange the labels of a pair of vertices u and v to produce a new embedding φ' where the new label for vertex u is $\varphi'(u)$, i.e., $\varphi'(u) = \varphi(v)$ and $\varphi'(v) = \varphi(u)$.

After introducing these preliminary concepts our first neighborhood function $\mathcal{N}_1(\varphi)$ can now be formally defined as follows:

$$\mathcal{N}_1(\varphi) = \{\varphi' = swap(\varphi, u, v) \ : \ u \in C(\varphi), \ v \in \mathcal{S}(u)\}. \tag{10}$$

It generates for every potential embedding $\varphi \in \mathscr{E}$ a set of neighboring solutions produced by exchanging the labels of a critical vertex $u \in C(\varphi)$ and the most suitable swapping vertex $v \in \mathcal{S}(u)$ for it. Every vertex $v \in \mathcal{S}$ is individually evaluated to identify the best one, *i.e.*, the one that not only produces the smaller cyclic bandwidth $B_C(u, \varphi')$ after the application of the function $swap(\varphi, u, v)$, but also the one that reduces the number of vertices adjacent to u or v whose cyclic bandwidth increases due to this label exchange operation. We consider that the cyclic bandwidth of a vertex $w \in \mathcal{A}(u)$ or $w \in \mathcal{A}(v)$ increased when:

$$\begin{array}{ll} |\varphi'(u) - \varphi(w)|_n > B_C(w, \varphi) & \text{and} \\ |\varphi'(u) - \varphi(w)|_n > \beta B_C(G, \varphi) & , \end{array} \tag{11}$$

for $0 \leq \beta \leq 1$ (acceptable cyclic bandwidth increase). If multiple vertices result into the same cyclic bandwidth $B_C(u, \varphi')$ and the same number of adjacent vertices whose cyclic bandwidth increases, then tie is broken at random.

The second neighborhood function $\mathcal{N}_2(\varphi, \gamma)$ is formally defined as follows:

$$\mathcal{N}_2(\varphi, \gamma) = \{\varphi' = swap(\varphi, u, v) \; : \; u \in C(\varphi), \; v \in \mathcal{R}_\gamma(u)\}\,, \tag{12}$$

where the set $\mathcal{R}_\gamma(u) \subseteq V$ contains γ non-tabu vertices randomly selected. It assigns to every potential embedding $\varphi \in \mathcal{E}$ a set of neighboring solutions which can be produced by exchanging the labels of a critical vertex $u \in C(\varphi)$ and a suitable swapping vertex $v \in \mathcal{R}_\gamma(u)$ for it. All the elements belonging to $\mathcal{R}_\gamma(u)$ are individually evaluated in order to identify the best of them. That is, the vertex v that produces the smaller cyclic bandwidth $B_C(u, \varphi)$ when its label is exchanged with that currently assigned to vertex u (ties are randomly broken).

The third neighborhood relation $\mathcal{N}_3(\varphi, \gamma, p)$ is a compound function partially inspired by the ideas reported in [21, 44]. It is a combination of both $\mathcal{N}_1(\varphi)$ and $\mathcal{N}_2(\varphi, \gamma)$ neighborhood functions. The former is applied with probability $(1 - p)$, while the latter is employed at a p rate. This combined neighborhood function $\mathcal{N}_3(\varphi, \gamma, p)$ is defined in (13), where *rnd* is a random number in the interval [0, 1].

$$\mathcal{N}_3(\varphi, \gamma, p) = \begin{cases} \mathcal{N}_1(\varphi) & \text{if } rnd \geq p \\ \mathcal{N}_2(\varphi, \gamma) & \text{if } rnd < p \end{cases} \tag{13}$$

3.4. Tabu list management

In our TScb algorithm the neighbor of a given solution φ is obtained by exchanging the label of a critical vertex u and a suitable swapping vertex v. When such a move is performed the vertex u is classified tabu for the next $\mathcal{T}$ iterations (tabu tenure). Therefore, the label of vertex u cannot be exchanged during this period.

The tabu tenure $\mathcal{T}$ for a move, in our TScb algorithm, can be either a constant prefixed value, or it can be dynamically calculated during the search using the approach introduced by Galinier *et al.* [39] and used later in [45]. It is based on the use of a periodic step function $\mathcal{PS}$ which takes as argument the number of iterations *iter*. Each period of this function is composed of 1500 iterations divided into 15 intervals. The value returned by $\mathcal{PS}$ for a particular iteration *iter* is given by $(a_j)_{j=1,2,\ldots,15} = (1, 2, 1, 4, 1, 2, 1, 8, 1, 2, 1, 4, 1, 2, 1)\tau$, where τ is a parameter that fixes the minimum tenure value and the index j is computed with (14).

$$j = \left\lfloor \frac{iter \mod 1500}{100} \right\rfloor + 1\,. \tag{14}$$

Therefore, the tabu tenure equals τ between iterations 1 and 99, 2τ between iterations 100 and 199, followed by τ again for iterations [200,299] and 4τ for iterations [300,399], etc. This variation scheme is periodically repeated by this function after every 1500 iterations.

3.5. Aspiration criteria

Since the attributes of a solution are recorded in the tabu list instead of the solutions themselves, it is possible that a candidate solution in the tabu list could lead to a better embedding than the best solution found so far. Therefore, in our TS algorithm, a simple aspiration criterion is applied: a tabu move leading to a configuration better than the best embedding found so far φ^* is always accepted.

3.6. Diversification strategy

A diversification mechanism was also implemented since our TScb algorithm could be trapped in deep local optima. In our case, the search is judged as stagnated each time the best solution found so far φ^* is not further improved after *maxNI* consecutive iterations. To help the search to escape from such deep local optima, a perturbation mechanism is applied to the current solution to bring diversification into the search.

The perturbation is composed by two complementary functions. The first one, called $\mathcal{D}_1(\varphi, \gamma)$, is defined in (15) and is applied ρ consecutive times, where ρ is a parameter that fixes the strength of this perturbation.

$$\mathcal{D}_1(\varphi, \gamma) = \{\varphi' = swap(\varphi, u, v) \; : \; u \in \mathcal{R}_\gamma(\varphi), \; v \in \mathcal{A}(u)\}\,. \tag{15}$$

It maps every potential embedding $\varphi \in \mathscr{E}$ to a set of solutions produced by exchanging the labels of a non-tabu randomly selected vertex $u \in \mathcal{R}_\gamma(\varphi)$ and the most appropriate adjacent vertex to it, $v \in \mathcal{A}(u)$. The most suitable vertex $v \in \mathcal{A}(u)$ for the $swap(\varphi, u, v)$ is the one that results into the smaller cyclic bandwidth $B_C(u, \varphi')$ for u, and at the same time minimizes the number of vertices $w \in \mathcal{A}(u)$ or $w \in \mathcal{A}(v)$ whose cyclic bandwidth increases due to this label exchange operation, see (11). If there exist ties they are randomly broken.

The perturbation function $\mathcal{D}_1(\varphi, \gamma)$ is applied a predefined maximum number of non-consecutive times $MaxS$ when the search stagnates. After that if the best solution found so far φ^* is not further improved the second perturbation function $\mathcal{D}_2(\varphi, \gamma)$ is applied at most $MaxH$ non-consecutive times. This perturbation constructs a partially ordered set $\mathcal{F}(V, \leq)$ whose elements are the vertices of the graph in ascending order with respect to their cyclic bandwidth. More formally $\mathcal{F}(V, \leq)$ can be defined with the expression:

$$\mathcal{F}(V, \leq) = \{u \ : \ \forall u, v \in V, \ B_C(u, \varphi) \leq B_C(v, \varphi)\}, \tag{16}$$

then the subset $\mathcal{H}_\gamma \subseteq \mathcal{F}(V, \leq)$, defined in (17), can be obtained.

$$\mathcal{H}_\gamma = \{u_i \ : \ 1 \leq i \leq \gamma, \ u_i \in \mathcal{F}(V, \leq)\} \tag{17}$$

The perturbation function $\mathcal{D}_2(\varphi, \gamma)$, which employs $(\gamma - 1)$ consecutive times the function $swap(\varphi, u, v)$ to construct a new embedding φ', can be formally expressed as follows:

$$\mathcal{D}_2(\varphi, \gamma) = \{\varphi' = swap(\varphi, u, v) \ : \ u, v \in \mathcal{H}_\gamma\}. \tag{18}$$

3.7. Stop condition

The TScb algorithm stops either if a predefined maximum number of iterations (*maxIter*) is reached, or when the algorithm ceases to make progress. In our implementation a lack of progress exists if the perturbation function $\mathcal{D}_2(\varphi, \gamma)$ has been applied $MaxH$ non-consecutive times.

3.8. Implementation considerations

In the following section we will see that the proposed TScb algorithm provides near-optimal solutions for the CB problem expending for that very reasonable computational times. This is possible thanks to the use of thoroughly designed data structures when implementing the algorithm.

For instance, in order to compute the cyclic bandwidth of an embedding φ, using the evaluation function $B_C(G, \varphi)$, every edge in the graph $G = (V, E)$ must be analyzed (see (1)). As a result $O(|E|)$ instructions must be executed by this *complete evaluation scheme*. Nevertheless, the proposed TScb algorithm employs an *incremental evaluation* of neighboring solutions. To this end, the cyclic distance of each edge in the graph is stored using an appropriate data structure. Indeed, suppose that the labels of two different vertices (u, v) are exchanged in an embedding φ to produce a neighboring solution φ', then we should only recompute the $|\mathcal{A}(u)| + |\mathcal{A}(v)|$ cyclic distances that change[4] in order to obtain the cyclic bandwidth of φ'. As it can be verified this is faster than the $O(|E|)$ operations originally required. As a consequence the TScb algorithm is able to analyze thousands of neighboring solution employing only a very small fraction of the time that would be required by the complete evaluation scheme.

Another example of a clever data structure implemented by the TScb algorithm is the one employed to manage the tabu list $\mathcal{L}$ (recency-based memory), which enables it to verify the tabu status of a move in constant time.

Important reductions in the total computational time expended by the proposed TScb algorithm are also possible because of the use of sorted sets in the implementation of neighborhood and diversification functions.

4. Computational experiments

In this section the three main experiments accomplished to evaluate the performance of the proposed TScb algorithm, as well as that of some of its components are presented. The objective of the first experiment is to determine

[4]$|\mathcal{A}(u)|$ and $|\mathcal{A}(v)|$ represent the number of adjacent vertices to u and v, respectively.

both a component combination, and a set of parameter values which permit TScb to attain the best trade-off between solution quality and computational effort.

The purpose of the second and third experiments is twofold: a) to carry out a performance evaluation of TScb over both a set of standard graphs with known optimal solutions and a set of graphs from real-world scientific and engineering applications; and b) to compare TScb with respect to a Simulated Annealing (SAcb) metaheuristic specially developed for the CB problem.

For all these experiments TScb and SAcb were coded in C and compiled with *gcc* using the optimization flag *-O*3. They were run sequentially into a CPU Xeon X5650 at 2.66 GHz, 2 GB of RAM with Linux operating system. Due to the non-deterministic nature of the studied algorithms, 31 independent runs were executed for each of the selected benchmark instances in each experiment presented in this section.

4.1. Simulated annealing algorithm

As it was mentioned in Section 2, to the best of our knowledge, there are not reported metaheuristic algorithms for the CB problem in the literature. Thus, for sake of comparison we have adapted the code of the Simulated Annealing algorithm for the bandwidth minimization problem reported in [21] to meet the special requirements of the CB problem. In particular, the new implementation, called SAcb, presents the following characteristics: a) it evaluates the fitness $B_C(G, \varphi)$ of the embedding φ by employing (1), b) the neighborhood $\mathcal{N}_4(\varphi)$ of a labeling φ is such that for each $\varphi \in \mathscr{E}$, $\varphi' \in \mathcal{N}_4(\varphi)$ if and only if φ' can be obtained by rotating the labels of any group of five consecutive vertices in the host graph C_n, c) a parameter configuration that (according to our preliminary experiments) gives a good trade-off between solution quality and computational effort: initial temperature $T_i = 5.0$, final temperature $T_f = 1.0E\text{-}07$, cooling rate $c_r = 0.97$, maximum number of visited neighboring labelings at each temperature $NV_{max} = \lceil maxIter/decTemp \rceil$, where $maxIter = 2.0E{+}07$ is a predefined global maximum number of visited neighboring labelings (iterations) and $decTemp = 435$ is the total number of temperature decrements from T_i to T_f using a geometrical cooling scheme $T_k = c_r T_{k-1}$.

4.2. Performance assessment and statistical significance analysis

To evaluate the efficiency of the new proposed TScb implementation two criteria were selected: the best cyclic bandwidth found for each instance (smaller values are better) and the expended CPU time in seconds.

A statistical significance analysis was performed for the second and third experiments presented below. Each analysis was conducted using the following methodology. First, *D'Agostino-Pearson's omnibus* K^2 test was used to evaluate the normality of data distributions. For normally distributed data, either *ANOVA* or the *Welch's t* parametric tests were used depending on whether the variances across the samples were homogeneous (*homoskedasticity*) or not. This was investigated using the *Bartlett's* test. For non-normal data, the nonparametric *Kruskal-Wallis* test was adopted. A significance level of 0.05 has been considered.

4.3. Benchmark instances

The performance evaluation of the new proposed TScb implementation was carried out with extensive experiments over 113 benchmark instances[5] grouped into two different sets, whose detailed descriptions are presented in the following subsections.

4.3.1. Standard graphs

For the first set we have generated a total of 85 standard graphs with known optimal solutions that belong to seven different families (3 *r-dimensional hypercubes*, 10 *three dimensional meshes*, 12 *complete r level k-ary trees*, and 15 graphs from each of the following classes: *paths, cycles, two dimensional meshes and caterpillars*). This set of benchmark instances is composed of 23 small graphs ($n < 100$), 24 medium graphs ($100 < n \leq 200$) and 38 large graphs ($200 < n \leq 8192$). A brief description of each selected class of graph and its corresponding optimal cyclic bandwidth value are summarized below.

[5] Available at `http://www..cinvestav.mx/~ertello/cbmp.php`

1. *Paths.* A path graph P_n is constructed as a linear sequence of n vertices. Two of them are terminal vertices of degree one, while the others (if any) have degree two. For a path P_n of order n the optimal cyclic bandwidth value is:

$$B_C(P_n) = 1 \ ,$$

as it was shown in [2].

2. *Cycles.* A cycle graph C_n is build as a circular arrangement of n vertices such that all of them have degree two. Yixun Lin [2] demonstrated that the CB problem has an optimal solution value for a cycle C_n given by:

$$B_C(C_n) = 1 \ .$$

3. *Two dimensional meshes.* These graphs are constructed as the Cartesian product of two paths P_{n_1} and P_{n_2}. Hromkovič *et al.* [6], based on the work of Chvátalová [46], demonstrated that the optimal cyclic bandwidth for a two dimensional mesh $P_{n_1} \times P_{n_2}$ of order $n = n_1 \cdot n_2$ (for $\max\{n_1, n_2\} > 3$) is:

$$B_C(P_{n_1} \times P_{n_2}) = \min\{n_1, n_2\} \ .$$

4. *Three dimensional meshes.* A three dimensional mesh is defined as the Cartesian product of three paths. Hromkovič *et al.* [6] proved that the optimal cyclic bandwidth for a three dimensional mesh $P_n \times P_n \times P_n$ with n^3 vertices (for $n > 3$) can be calculated with the following expression:

$$B_C(P_n \times P_n \times P_n) = \left\lfloor \frac{3n^2}{4} + \frac{n}{2} \right\rfloor .$$

5. *Complete r level k-ary trees.* These graphs are rooted complete trees in which the i-th level consists of k^{i-1} vertices and each vertex that belongs to level i has exactly k descendants at level $i+1$ (for $1 \leq i < r$). Such a tree, denoted $T_{k,r}$, has $n = (k^r - 1)/(k-1)$ vertices and its optimal cyclic bandwidth value is:

$$B_C(T_{k,r}) = \left\lceil \frac{k(k^{r-1} - 1)}{2(r-1)(k-1)} \right\rceil ,$$

see [5, 6, 47].

6. *Caterpillars.* A caterpillar is a special tree in which every vertex is on a central stalk, called spine, or within distance one of the stalk, *i.e.*, removal of its endpoints leaves a path graph. For a caterpillar T with spine $P(u_1, u_2, \ldots, u_m)$, Lin [2] proved that the optimal cyclic bandwidth is:

$$B_C(T) = \max_{1 \leq i \leq j \leq m} \left\lceil \frac{n_{ij} - 1}{j - i + 2} \right\rceil .$$

where n_{ij} denotes the order of the subtree T_{ij} induced by $u_i, u_{i+1}, \ldots, u_j$ and all the vertices adjacent to them $(i \leq j)$.

7. *r-dimensional hypercubes.* An r-dimensional hypercube Q_r is a graph usually defined as the Cartesian product of r path graphs with two vertices (P_2). It can be constructed using $n = 2^r$ vertices labeled with r-bit binary numbers and connecting two vertices by an edge whenever the Hamming distance of their labels is one. Hromkovič *et al.* [6] proved that the optimal cyclic bandwidth for Q_r ($r \geq 11$) is:

$$B_C(Q_r) = \sum_{k=0}^{r-1} \binom{k}{\lfloor \frac{k}{2} \rfloor} .$$

Although there exist other standard classes of graphs whose optimal solutions for the CB problem are reported in the literature (*stars*, *complete* and *complete bipartite* graphs), we have decided to not include them in this set since according to our preliminary experiments they can be solved optimally using any random labeling. In this sense these kind of instances can not be considered as good candidates for evaluating the performance of the proposed algorithm.

Table 1: Input parameters of the TScb algorithm and their selected values.

	p	$\mathcal{T}$	*maxS*	*maxH*	ρ	*maxNI*	β	γ
0	0.0	$\tau = 1$	90	80	1	10	0.4	0.2
1	0.2	$\tau = 2$	100	90	3	25	0.5	0.3
2	0.5	3	110	100	5	40	—	—
3	0.8	5	—	—	—	—	—	—

4.3.2. Harwell-Boeing graphs

The second set contains 28 problem instances with a number of vertices between 9 and 715, coming directly from the Harwell-Boeing Sparse Matrix Collection. This collection gathers standard test matrices arising from a wide variety of scientific and engineering practical problems which can be considered as adjacency matrices in order to construct graphs. Most of the graphs in our second set (24 of them) were previously used by Duarte *et al.* [13] and Lozano *et al.* [14] as benchmarks instances for the antibandwidth problem [1]. The rest of the instances (4 graphs) were collected by us to complement this set with small graphs that can be solved exactly using the B&B algorithm proposed in [12].

4.4. Components and parameters tuning

Optimizing parameter settings is an important task in the context of algorithm design. Different procedures have been proposed in the literature to find the most suitable combination of parameter values [48, 49]. In this paper we employed a tuning methodology based on combinatorial interaction testing (CIT) [10, 11], which was successfully used in [50, 51]. We have decided to use CIT, because it allows to significantly reduce the number of tests (experiments) needed to determine the best parameter settings of an algorithm. Instead of exhaustive testing all the parameter value combinations of the algorithm, it only analyzes the interactions of t (or fewer) input parameters by creating interaction test-suites that include, at least once, all the t-way combinations between these parameters and their values.

Covering arrays (CAs) are combinatorial objects which have been extensively used to represent those interaction test-suites. A covering array, $CA(N; t, k, v)$, of size N, strength t, degree k, and order v is an $N \times k$ array on v symbols such that every $N \times t$ sub-array includes, at least once, all the ordered subsets from v symbols of size t (t-tuples) [52]. The minimum N for which a $CA(N; t, k, v)$ exists is the *covering array number* and it is defined according to the following expression: $CAN(t, k, v) = min\{N : \exists\, CA(N; t, k, v)\}$.

CAs are used to represent an interaction test-suite as follows. In an algorithm we have k input parameters. Each of these has v values or levels. An interaction test-suite is an $N \times k$ array where each row is a test case. Each column represents an input parameter and a value in the column is the particular configuration. This test-suite allows to cover all the t-way combinations of input parameter values at least once. Thus, the costs of tuning the algorithm can be substantially reduced by minimizing the number of test cases N in the covering array.

In practice, algorithm's input parameters do not have exactly the same number of values (levels). To overcome this limitation of CAs, mixed level covering arrays (MCAs) are used. An $MCA(N; t, k, (v_1, v_2, \cdots, v_k))$ is an $N \times k$ array on v symbols ($v = \sum_{i=1}^{k} v_i$), where each column i ($1 \le i \le k$) of this array contains only elements from a set S_i, with $|S_i| = v_i$. This array has the property that the rows of each $N \times t$ sub-array cover all t-tuples of values from the t columns at least once. Next, we present the details of the tuning process, based on CIT, for the particular case of our TScb algorithm.

First, we have identified $k = 8$ input parameters used for TScb: application probability for the neighborhood function $\mathcal{N}_3$ (p), tabu tenure ($\mathcal{T}$), maximum number of non-consecutive calls to diversification functions $\mathcal{D}_1$ ($MaxS$) and $\mathcal{D}_2$ ($MaxH$), strength of perturbation $\mathcal{D}_1$ (ρ), maximum number of iterations without improvement (*maxNI*), percentage of acceptable cyclic bandwidth increase used in function $\mathcal{N}_1$ (β) and percentage γ of vertices employed in functions $\mathcal{N}_3, \mathcal{D}_1$ and $\mathcal{D}_2$. Based on some preliminary experiments, certain reasonable values were selected for each one of those input parameters (shown in Table 1). No important differences in performance were observed when varying the parameter α, which determines the number of critical vertices considered in functions $\mathcal{N}_1$ and $\mathcal{N}_2$, thus its value was fixed to 0.9 for all the experiments presented in this section.

The mixed level covering array $MCA(168; 4, 8, (4, 4, 3, 3, 3, 3, 2, 2))$, shown (transposed) in Table 2, was obtained by using the Memetic Algorithm reported in [53]. This covering array can be easily mapped into an interaction test-suite by replacing each symbol from a column to its corresponding parameter value. For instance, we can map 0 in

Table 2: Mixed level covering array MCA(168; 4, 8, (4, 4, 3, 3, 3, 3, 2, 2)) representing an interaction test-suite for tuning TSCB (transposed).

(a) Test cases 1 to 84

0 3 3 3 2 3 3 1 1 0 0 1 2 2 1 3 0 3 2 0 0 3 0 1 2 2 3 1 1 0 2 3 1 1 3 3 3 2 3 3 1 1 0 0 0 2 0 0 3 2 2 0 1 1 2 3 0 1 3 0 3 0 0 0 2 0 2 2 0 3 3 1 0 2 2 1 2 0 2 2 1 2 0 2
3 3 3 3 3 3 0 1 1 2 1 0 1 0 3 1 1 2 1 0 0 3 1 2 0 2 0 0 0 3 3 1 0 3 1 1 2 1 1 2 2 2 2 1 2 3 1 2 3 0 1 2 0 0 3 1 0 0 1 1 2 0 3 0 2 3 3 1 1 0 3 0 1 0 3 0 3 2 2 2 1 2 2 2
1 1 0 1 1 1 1 0 2 0 2 0 1 2 0 2 1 2 1 2 1 2 2 1 1 0 0 0 0 0 1 1 1 1 0 2 0 2 0 0 1 0 1 1 1 2 2 0 0 2 0 0 0 1 2 1 2 2 0 0 1 2 1 2 1 2 0 2 0 1 2 1 1 2 1 0 0 1 2 1 0 0 2 2
0 1 0 0 2 1 0 1 0 0 1 2 1 0 0 2 2 1 0 0 1 2 2 2 0 1 0 0 2 0 2 2 2 2 0 0 0 2 1 1 0 0 0 0 1 2 1 2 1 2 2 1 0 1 1 0 1 2 2 1 1 0 1 2 1 0 0 1 2 2 1 0 1 0 0 1 1 1 1 0 1 0 2 2
0 0 0 1 1 2 0 2 1 1 0 0 2 1 0 1 0 0 1 1 1 1 1 2 1 0 2 1 1 2 0 0 1 0 1 0 0 1 1 1 0 2 0 1 0 2 1 0 1 1 2 2 2 0 1 2 2 2 0 1 2 0 1 0 1 1 1 0 2 2 0 2 2 2 2 0 2 1 2 0 0 2 0 2
2 1 0 1 1 0 2 1 0 2 2 2 0 1 2 2 0 2 1 2 0 0 1 2 0 0 1 1 0 1 0 1 2 1 2 1 1 0 1 0 1 0 0 2 1 2 0 0 2 2 1 1 2 0 0 0 1 1 2 0 1 2 1 0 1 0 0 1 2 0 1 1 1 1 0 1 2 2 2 2 0 1 2 0
1 0 0 1 0 1 1 0 0 0 0 1 0 1 0 0 1 0 0 0 1 0 0 1 0 1 0 0 0 0 1 1 1 0 1 0 1 1 0 0 1 1 1 0 0 1 1 0 1 0 1 1 0 0 0 1 0 0 0 0 0 1 1 0 1 0 1 0 1 0 1 1 1 0 1 1 0 1 0 1 1 0 0 1
0 1 0 0 0 0 0 0 0 1 0 0 1 0 1 1 1 0 1 0 1 1 1 0 1 1 1 1 0 1 0 0 1 1 0 1 0 1 0 0 1 0 0 1 1 1 0 0 1 0 1 1 1 0 0 1 1 0 1 1 0 1 0 1 0 1 1 0 0 1 0 0 0 1 0 0 0 0 1 0 1 0 1 0

(b) Test cases 85 to 168

1 3 3 3 1 2 2 1 0 2 1 1 2 2 1 3 2 1 0 3 3 1 0 0 0 1 0 3 0 3 1 0 1 1 1 0 2 2 3 3 2 0 1 1 1 3 1 3 2 1 2 0 0 3 2 2 3 0 3 2 2 2 1 3 3 0 2 2 3 0 1 1 1 1 2 3 1 0 3 2 1 0 3 0
3 1 0 2 1 3 1 1 3 3 3 2 1 0 2 0 0 2 2 0 0 3 2 0 2 3 0 1 0 0 3 0 2 1 2 2 3 2 2 1 2 3 1 1 3 1 1 3 0 2 0 3 3 2 3 2 0 3 3 3 0 1 2 2 0 0 1 0 2 1 1 0 3 3 2 1 0 1 3 1 0 2 2 3
0 2 2 2 1 2 0 1 2 1 2 0 2 2 2 0 1 2 1 0 0 2 1 1 2 2 0 1 1 2 1 0 2 2 1 1 2 1 2 2 2 2 1 0 0 1 0 2 1 2 0 1 0 1 1 2 1 0 1 0 0 1 0 0 2 0 0 0 1 0 2 2 1 2 0 0 2 2 0 1 2 2 2 2
2 1 0 0 2 0 0 0 1 1 0 1 0 1 0 2 2 1 2 1 1 1 0 2 0 1 1 1 0 2 1 2 0 1 1 2 1 2 1 2 0 2 1 0 1 2 2 0 1 2 2 2 2 0 0 2 1 1 2 2 1 1 2 2 1 0 1 0 2 0 2 1 0 2 2 2 1 0 2 2 0 1 2 2
2 0 1 2 2 0 0 0 2 2 2 0 2 0 1 1 0 2 2 0 2 0 2 2 1 1 0 2 0 0 2 1 0 2 1 1 0 2 1 2 1 2 1 2 1 1 1 2 2 0 2 2 1 1 1 0 1 0 0 0 1 0 1 2 2 2 1 0 0 0 0 1 1 1 1 2 2 2 2 0 0 2 1 0
0 0 0 0 0 1 0 2 2 1 1 2 2 0 2 2 1 1 1 0 0 0 2 2 1 1 2 2 1 1 2 1 2 2 0 0 2 0 1 1 0 0 1 1 1 0 2 2 2 0 0 0 2 2 2 1 1 0 2 1 1 2 1 2 2 0 2 2 0 1 1 2 0 2 2 0 0 0 1 2 0 0 1 1
0 1 0 1 0 1 0 0 0 1 1 1 1 1 1 0 0 0 1 0 1 1 0 0 1 0 0 0 0 1 0 1 0 1 0 1 0 0 1 0 0 1 1 1 1 1 0 0 1 1 1 0 0 0 0 0 1 1 0 0 0 1 0 1 1 1 0 1 1 1 1 0 1 1 1 0 0 0 1 0 1 1 0 0
1 1 1 0 1 1 0 0 1 1 0 1 0 0 1 0 1 1 0 1 0 0 1 0 0 1 0 0 0 0 0 1 0 1 0 1 0 1 1 0 1 0 1 1 0 0 0 1 0 1 0 1 0 1 1 1 0 1 0 1 1 1 0 1 1 0 0 1 1 0 0 1 1 0 0 0 0 1 1 0 1 0 0 0

Table 3: Results from the 10 best test cases in the tuning experiments.

Num.	*Test case*	*Avg. CB*	*Avg. T*	*R-RMSE*
74	**20202101**	**45.008**	**361.923**	**0.1646**
79	22212201	44.944	557.292	0.1648
123	32211111	44.626	594.958	0.1657
122	22122001	45.237	284.369	0.1682
102	12212101	45.097	336.658	0.1740
124	31222100	45.059	585.365	0.1743
38	21221011	45.051	258.920	0.1763
115	13112200	45.274	368.028	0.1767
89	11122001	45.384	203.701	0.1788
112	31112200	45.293	722.984	0.1792

the first column (the first line in Table 2, corresponding to parameter p) to 0.0, 1 to 0.2, 2 to 0.5 and 3 to 0.8. The resulting interaction test-suite contains, thus, 168 test cases (parameter settings) which include at least once all the 4-way combinations between TSCB's input parameters and their values[6].

Each one of those 168 test cases was used to run 31 times the TSCB algorithm over the 12 larger Harwell-Boeing graphs described in Subsection 4.3, resulting in a total of 62,496 executions. For each test case the relative Root Mean Square Error (R-RMSE) and the average CPU time where computed.

The R-RMSE is a quadratic scoring rule which is frequently used to measure the differences between values predicted by a model or an estimator and the values actually observed [54]. It can be formally defined using (19):

$$\text{R-RMSE} = \sqrt{\frac{\sum_{i=1}^{R} \left(\frac{\hat{Y}_i - Y_i}{Y_i} \right)^2}{R}}\,, \tag{19}$$

where $\hat{Y}_i$ refers to the estimator of the parameter Y_i and R is the number of simulations. In optimization the R-RMSE can be used to evaluate the performance of an algorithm by measuring the differences between the solution values produced by it ($\hat{Y}_i$) with respect to the best-known solutions (Y_i) [55–57]. For a perfect performance, $\hat{Y}_i = Y_i$ and R-RMSE = 0. So, the R-RMSE ranges values from 0 to infinity, with 0 corresponding to the ideal.

Table 3 summarizes the 10 test cases which yield the best results. For each test cases it lists the average cyclic bandwidth (*Avg. CB*), the average CPU time (*Avg. T*) in seconds and the R-RMSE value. This table allowed us to observe that the parameter setting giving the best trade-off between solution quality and computational effort corresponds to the test case number 74 (shown in bold). The best R-RMSE value with an acceptable speed is thus reached

[6]In contrast, with an exhaustive testing which contains $(4^2)(3^4)(2^2) = 5184$ test cases.

Table 4: Overall performance comparison of the SAcb and TScb algorithms over 85 standard graphs from 7 different types all of them with known optimal solutions B_C^*.

		SAcb				TScb				
Graphs	*Num.*	*Avg. Best*	*Avg. T*	*R-RMSE*	*%Best*	*Avg. Best*	*Avg. T*	*R-RMSE*	*%Best*	*SS+*
path	15	67.533	1.139	111.575	20.000	2.933	7.943	3.235	46.667	14
cycle	15	67.600	1.086	112.084	26.667	2.667	7.219	3.066	66.667	13
mesh2D	15	91.533	1.618	6.590	20.000	27.667	6.423	1.977	60.000	12
mesh3D	10	351.600	3.644	3.836	0.000	199.900	17.711	1.853	10.000	10
tree	12	94.917	0.657	1.416	16.667	55.250	5.026	0.034	83.333	5
caterpillar	15	72.467	0.609	3.215	33.333	15.400	9.663	0.024	80.000	10
hypercube	3	2387.333	600.000	1.054	0.000	1375.000	600.000	0.185	0.000	3

with the following input parameter values: application probability for the neighborhood function $\mathcal{N}_3$, $p = 0.5$; dynamic tabu tenure with $\tau = 1$; maximum number of non-consecutive calls to diversification functions $MaxS = 110$ and $MaxH = 80$, for $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively; strength of perturbation $\mathcal{D}_1$, $\rho = 5$; maximum number of iterations without improvement, *maxNI* = 25; percentage of acceptable cyclic bandwidth increase $\beta = 0.4$ used in function $\mathcal{N}_1$ and percentage $\gamma = 0.3$ of vertices employed in functions $\mathcal{N}_3, \mathcal{D}_1$ and $\mathcal{D}_2$. These values are therefore used in the experimentation reported next.

4.5. Assessing the performance of TScb *over standard graphs*

For the first assessing performance experiment the set of 85 standard graphs with known optimal solutions, described in Subsection 4.3.1, was selected. Then, both TScb and SAcb were compiled and executed over these instances setting a cutoff time of 600 seconds for each execution.

The computational results produced in this experiment were first analyzed and classified by family of the tested graphs in order to separately observe the performance of the compared algorithms when solving each type of graph. Table 4 summarizes this classification. For the two compared algorithms the average (*Avg. Best*) of the best cyclic bandwidth cost attained by them over 31 independent executions and the average CPU time (*Avg. T*) in seconds expended are depicted. For each class of graph tested, the relative Root Mean Square Error (*R-RMSE*) with respect to the known optimal solution B_C^* was computed using (19), as well as the percentage of instances (*%Best*) for which the value of the best solution obtained by a given metaheuristic reaches B_C^*. Additionally, a statistical significance analysis was performed for this experiment by using the methodology described in Subsection 4.2. Last column (*SS+*) sums up how many times a statistically significant performance amelioration was achieved by TScb with respect to SAcb.

Table 4 shows that, for each of the seven types of standard graphs analyzed, our TScb algorithm clearly outperforms SAcb in terms of the average of the best solution cost reached (*Avg. Best*). Indeed, the statistical analysis carried out for this experiment established that there exists a statistically significant performance amelioration achieved by TScb with respect to SAcb on 67 benchmark instances (78.824% of the graphs). For the other 18 tested graphs there was not a significant difference between the performance of both compared algorithms.

In general, a high *R-RMSE* value, indicates that the solutions provided by the analyzed algorithm present a high deviation relative to the known optimal values B_C^*. This measure has the important property of considering a difference of only one unit with respect to an optimal value $B_C^* = 1$ worse than the same one unit difference with respect to a higher B_C^* value. In other words, it penalizes more the deviations with respect to small B_C^* values. In this experiment, the two higher *R-RMSE* values scored by TScb are for the *paths*, and *cycles*, since for these types of graphs $B_C^* = 1$.

Regarding the percentage of instances (*%Best*) for which the best solution found by TScb equals B_C^*, we observe that for five types of the studied graphs (*paths, cycles, two dimensional meshes, complete r level k-ary trees* and *caterpillars*) it is higher than 46.667% and goes up to 83.333%. In the case of the *three dimensional meshes* and the *r-dimensional hypercubes* the *%Best* value for TScb is 10.000 and 0.000, respectively, which reveals that these instances are challenging for TScb, probably due to their large size and specific topology.

The detailed results from this experiment can be found in Table A.1, from which it is possible to observe that TScb reached the optimal solution (B_C^*) for 57.647% (49 out of 85) of the selected graphs consuming in average 29.484 seconds, which is slightly higher than the average CPU time employed by the SAcb algorithm (22.484 seconds).

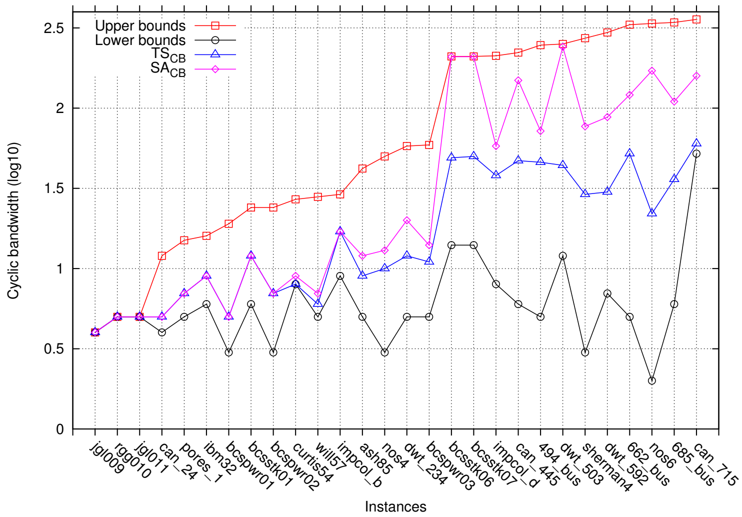


Figure 3: Performance evaluation of the best solutions achieved by SACB and TSCB, over 28 Harwell-Boeing graphs, with respect to the theoretical lower and upper bounds proposed by Lin [15].

For the other 36 benchmark instances TSCB found solutions which are close to the optimal cyclic bandwidth (*R-RMSE* = 2.145). Another important outcome is that the solution cost found by TSCB presents a relatively small standard deviation (in average 4.738). It is an indicator of the algorithm's precision and robustness since it shows that in average the performance of TSCB does not present important fluctuations.

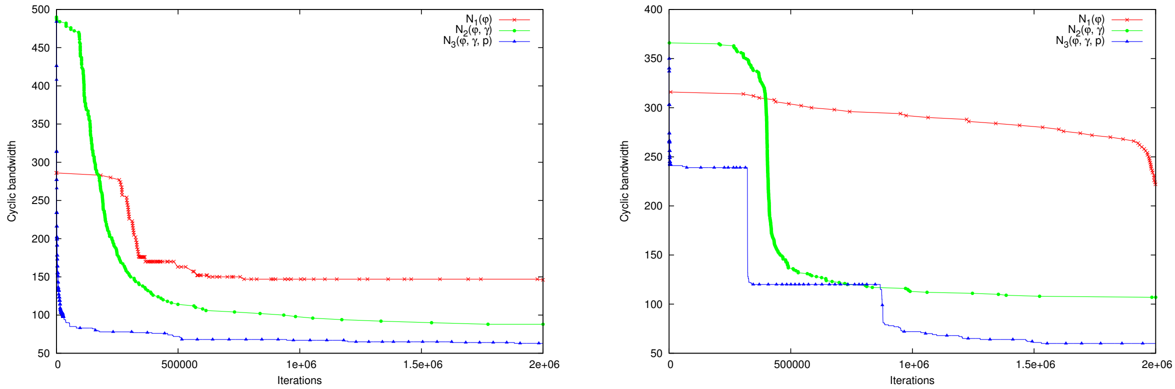
From this experiment we can conclude that TSCB is certainly an effective approach for finding good quality solutions for the CB problem in the case of standard graphs with known tight lower bounds. Below, we will present more computational results obtained from a performance evaluation carried out with TSCB employing graphs produced from real-world scientific and engineering applications.

4.6. *Assessing the performance of* TSCB *over Harwell-Boeing graphs*

In this experiment a performance evaluation of the best solutions achieved by SACB and TSCB with respect to the theoretical lower and upper bounds proposed by Lin [15] was carried out over the test-suite described in Subsection 4.3.2, using a cutoff time of 600 seconds for each execution. The results from this experiment are depicted in Table A.2 using the same column headings defined at the beginning of Appendix A.

Analyzing the data presented in Table A.2 lead us to the following main observations. First, our TSCB algorithm is able to outperform the solutions provided by SACB in 18 out of 28 graphs (64.286%) and to equal its results for the other 10 benchmark instances. The statistical analysis presented in the last two columns of Table A.2 confirms that there exists a statistically significant increase in performance achieved by TSCB with respect to SACB on 23 graphs (82.143% of the instances). This highlights the suitability of the studied TSCB approach. One can also remark that both SACB and TSCB are able to attain the optimal solutions, found by the B&B algorithm reported in [12], for the 6 smallest ($n \leq 32$) instances in this test-suite and to supply a solution cost separated only by one unit from the optimal cyclic bandwidth (5 vs. 4) for the instance *bcspwr01*. These results evidence that optimal solution costs could be higher than the theoretical lower bounds (L_B) proposed by Lin [15], i.e., these lower bounds are not tight.

Second, the solution quality provided by TSCB for the other 21 graphs (with $n \geq 39$) is very competitive, since it consistently improve the upper bounds (U_B) calculated according to [15] (see Figure 3). Indeed, the *R-RMSE* for TSCB, computed with respect to the corresponding best-known bound, is nearly one order of magnitude smaller than that of SACB (4.055 vs. 19.174), indicating that the cost of the labelings provided by TSCB are closer to the best-known solutions. Our TSCB algorithm is even able to equal the lower bound value $L_B = 8$ for the instance *curtis54* of size $n = 54$, which means that the optimal solution for this graph was found by it. Moreover, the cyclic bandwidth of the solutions reached by TSCB present in average a very reduced standard deviation (see Column *Dev.*).



(a) (b)

Figure 4: Average performance comparison of three neighborhood functions using TScb over the instances: (a) *tree2x9* and (b) *can_715*.

Third, one can notice that TScb is the most time-consuming algorithm, since it uses an average of 138.369 seconds for solving these 28 instances. On the contrary, SAcb employs only 9.515 seconds in average. However, we believe that the CPU time consumed by TScb is acceptable (at most 600 seconds) and is fully justified by considering that it is able to outperform the SAcb method in terms of cost. In addition, TScb's consumed computing time is reasonable compared with that expended by the B&B algorithm used in this experiment [12]. For instance, TScb employed only 600.000 seconds for finding a near-optimal solution for the larger instance in this test-suite (*can_715*), while the execution time for the B&B over the instance *bcspwr01* (a smaller instance) is 4.73E05 seconds.

5. Discussion and analysis

The main objective of this section is to experimentally analyze the extent to which certain key components of the proposed TScb implementation can influence its global performance. For all the experiments presented in this section the algorithms were compiled and executed independently 31 times over the two sets of benchmark instances introduced in Subsection 4.3. The parameter values employed for TScb were those previously identified in Subsection 4.4.

Given the space requirements for reporting the results of this experiment, our findings are presented using plots for only two representative instances. One from the set of standard graphs with known optimal solutions (*tree2x9*) and one from the set of Harwell-Boeing graphs (*can_715*). However, comparable results were obtained with all the other tested instances.

5.1. Influence of the neighborhood functions

The neighborhood function is a critical element for the performance of any local search algorithm. In order to further examine the influence of this element on the overall performance of our TScb implementation we have performed some experimental comparisons using the following three neighborhood functions (see Subsection 3.3): $\mathcal{N}_1(\varphi)$, $\mathcal{N}_2(\varphi,\gamma)$ and $\mathcal{N}_3(\varphi,\gamma,p)$.

For this experiment each one of the studied neighborhood functions was implemented within TScb. Figure 4 summarizes the results from this experiment. It shows three average execution (convergence) profiles which represent the evolution, during the search process (abscissa)[7], of the best solution quality attained by TScb (ordinate), when each one of the studied neighborhood relations is used to solve the instances *tree2x9* and *can_715*. As it can be seen from the plots, the worst performance is attained by TScb when the neighborhood function $\mathcal{N}_1(\varphi)$ is used. The function $\mathcal{N}_2(\varphi,\gamma)$ produces better results compared with $\mathcal{N}_1(\varphi)$ since it improves the solution cost faster. Nevertheless, it also causes that our TScb algorithm gets stuck on some local minima. Finally, the best performance is attained by TScb when the function $\mathcal{N}_3(\varphi,\gamma,p)$ is employed, which is a compound neighborhood combining the complementary characteristics of both $\mathcal{N}_1(\varphi)$ and $\mathcal{N}_2(\varphi,\gamma)$.

[7]Each iteration in the plots represents a call to the evaluation function (1).

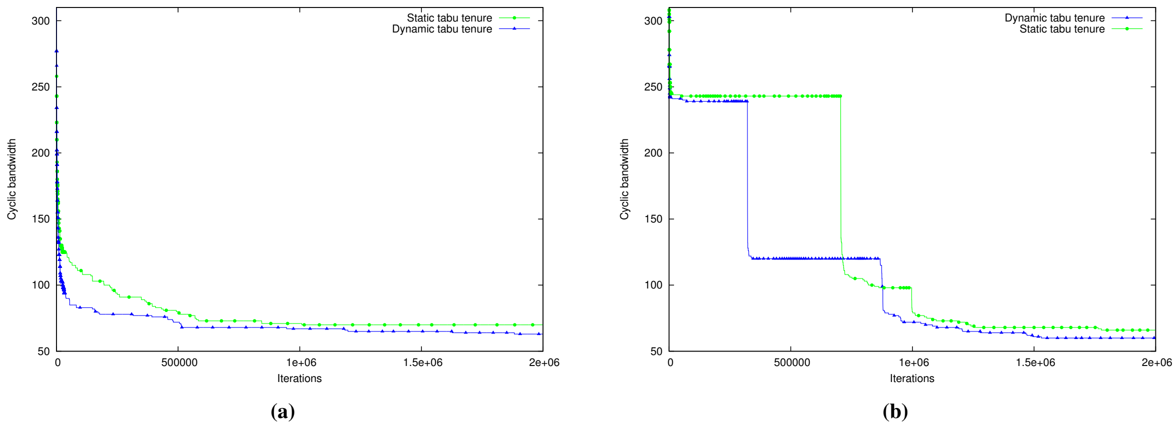


Figure 5: Average performance comparison of two different tabu tenure management strategies using TSCB over the instances: (a) *tree2x9* and (b) *can_715*.

5.2. Influence of the tabu tenure management

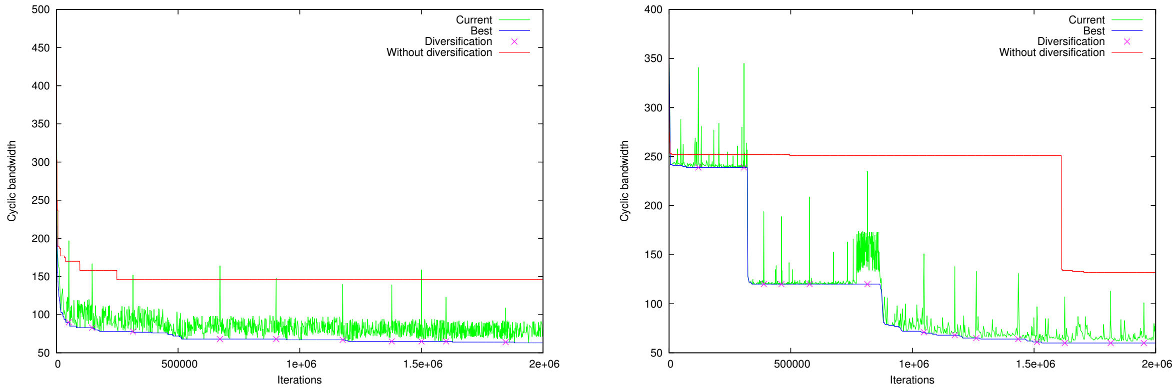
In this experiment we compare the performance of two different tabu tenure management strategies. The first one is commonly used in the literature [36, 42], and consists in prefixing a tabu tenure value that is maintained until the end of the algorithm's execution. The second one is the approach introduced in [39] (see Subsection 3.4), where the tabu tenure is dynamically calculated during the search using a periodic step function.

The two tabu tenure management approaches (called here *Static* and *Dynamic*, respectively) were integrated into the TSCB source code and executed for solving the selected benchmark instances. According to the results of our preliminary experiments, the tabu tenure value was fixed to $\mathcal{T} = 5$ for the *Static* approach, while the minimum tenure value is equal to $\tau = 1$ for the dynamic strategy. The results reached by TSCB over the instances *tree2x9* and *can_715* for this comparative experiment are illustrated in Figure 5. Each plot represents the iterations of TSCB (abscissa) against the average best solution quality attained with the used of the compared tabu tenure management mechanism (ordinate). This figure discloses that TSCB using a tabu tenure value which is dynamically calculated throughout the search, with a periodic step function, performs slightly better than the TSCB implementation that prefixes a tabu tenure value. Very similar results were obtained with the rest of the analyzed benchmark instances, thus this figure correctly summarizes the behavior of the compared tabu tenure management schemes.

5.3. Influence of the diversification strategy

The diversification strategy is an important component that must be carefully designed when implementing a tabu search algorithm, since it is related with the process of discovering new unexplored regions of the search space containing potentially good solutions [58, 59]. This experiment is thus devoted to investigate the extent to which this key component of TSCB can influence its global performance.

Two different versions of the TSCB algorithm were specially prepared to this end. The first one equipped with the diversification strategy described in Subsection 3.6. The second one without any diversification strategy implemented. Both versions were executed independently over the selected benchmark instances. Figure 6 displays, for the instances *tree2x9* and *can_715*, two convergence profiles representing the evolution through the search (abscissa) of the current and best solution quality provided by the TSCB algorithm employing the proposed diversification strategy (ordinate). The iteration when the perturbation function $\mathcal{D}_2(\varphi, \gamma)$ is applied to diversify the search is also marked in these plots with a symbol "x". Recall, that $\mathcal{D}_2(\varphi, \gamma)$ is only applied after a predefined maximum number ($MaxS$) of non-consecutive calls to the perturbation function $\mathcal{D}_1(\varphi, \gamma)$. For comparative purposes, the best solution quality attained by the TSCB version without any diversification strategy implemented is additionally presented in a third plot. From this figure it can be easily observed that the TSCB algorithm who does not implement a diversification strategy has the worst overall performance in this experiment. This point can be better explained by observing that the lack of a pertinent diversification strategy causes that the TSCB algorithm easily stagnates in deep local optima. On the contrary, the TSCB algorithm employing the proposed diversification strategy is able to detect when the search process



(a) **(b)**

Figure 6: Performance comparison of TSCB with and without a diversification strategy over the instances: (a) *tree2x9* and (b) *can_715*.

is trapped into some local minima and to guide it to more promising potential solutions. This is confirmed by the fact that TSCB with a diversification mechanism consistently attains better global performance in terms of cyclic bandwidth cost.

6. Conclusions

In this paper the first implementation of a Tabu Search algorithm (TSCB) for solving the *Cyclic Bandwidth* problem (CB) was presented. This algorithm integrates two key features that importantly determine its performance. First, a carefully designed composed neighborhood function which allows the search to quickly reduce the total cost of candidate solutions, while avoiding to get stuck on some local minima. Second, an effective tabu list management method where the tabu tenure for a move is dynamically calculated during the search using a specially designed periodic step function.

TSCB's components and parameter values were carefully determined, through the use of a tuning methodology based on combinatorial interaction testing [10, 11], to yield the best solution quality in a reasonable computational time. Then, the practical effectiveness of TSCB was assessed through extensive experimentation over a set of 113 standard benchmark instances [12–14], and compared against a Simulated Annealing implementation (SACB).

The first of these experiments employed 85 standard graphs with known optimal solutions to carefully compare the performance of both SACB and TSCB with respect to the known optimal solutions (B_C^*). This experiment demonstrated that there exists a statistically significant performance amelioration achieved by TSCB with respect to SACB in 67 out of 85 tested graphs (78.824%). Our proposed method attained the optimal solutions value B_C^* for 49 out of 85 (57.647%) selected graphs expending in average 29.484 seconds, which is slightly higher than the average CPU time employed by the SACB algorithm (22.484 seconds). For the rest of the graphs TSCB found good quality solutions, which are close to the optimal cyclic bandwidth (*R-RMSE* = 2.145 in average).

The second experiment aimed at performing a comparative evaluation of the best solutions achieved by SACB and TSCB with respect to the theoretical bounds proposed by Lin [15] for the CB problem. For this experiment 28 test instances produced from real-world scientific and engineering applications (Harwell-Boeing graphs) were used. The obtained results confirm that TSCB's performance was statistically superior than that reached by SACB on 82.143% of the tested instances (23 out of 28). In fact, TSCB was able to find the optimal solution B_C^* for 7 graphs in this test-suite and to establish new better upper bounds for the other 21 instances.

The third experiment was devoted to experimentally analyze the extent to which some key components of the proposed TSCB implementation (neighborhood function, tabu tenure management and diversification strategy) can influence its convergence process. Thanks to this analysis it was possible to identify that the use of the proposed compound neighborhood function coupled with a diversification strategy are the two key components which determine the global performance of our TSCB algorithm. The compound neighborhood function permits a fast improving of the solution cost while avoiding to get stuck on some local minima. The diversification strategy, on the other hand, allows the TScb algorithm to detect when the search process is trapped into some local minima and to guide it to more promising potential solutions, which results in a better global performance in terms of cyclic bandwidth cost.

All the experimental results presented confirm the practical advantages of using our TScb algorithm for solving the CB problem. It is a robust algorithm yielding good quality solutions for the CB problem in the case of general graphs at competitive computational time. In this sense this work represents an original contribution in this field.

Although promising results were obtained by the TScb algorithm, we believe that they could be still ameliorated. Our future work will concentrate on designing and evaluating alternative neighborhood and evaluation functions in order to boost the performance of the proposed TScb algorithm, since it is well-known that they are two important components which define the so-called landscape of the search problem [60, 61] and impact thus greatly the efficiency of the search algorithms [62].

Acknowledgements

The authors thankfully acknowledge the high performance computing resources (Neptuno cluster) and the technical assistance provided by the Information Technology Laboratory at CINVESTAV-Tamaulipas. The four reviewers of the paper are greatly acknowledged for their constructive comments which have aided to improve the presentation of this paper.

A. Detailed comparison of the SAcb and TScb algorithms

Tables A.1 and A.2 present detailed results from the experimental comparison carried out between the TScb and SAcb algorithms over both a set of standard graphs with known optimal solutions and a set of graphs from real-world scientific and engineering applications. The first three columns in these tables represent the graph name, its number of vertices ($|V|$) and edges ($|E|$). In Table A.1 the known optimal solution (B^*_C) was calculated using equations presented in Subsection 4.3.1. The theoretical lower (L_B) and upper (U_B) bounds for the graphs listed in Table A.2 (see Columns 4 and 5) were computed, as indicated by Lin [15], with the expressions $L_B = \lceil \Delta(G)/2 \rceil$ and $U_B = \lfloor |V|/2 \rfloor$, where $\Delta(G)$ denotes the maximum degree of the graph G. For the first seven instances (with $n \leq 40$) in Table A.2, the exact optimal solutions B^*_C were obtained using the B&B algorithm reported in [12]. In both tables, for each compared algorithm five columns are used to depict the best (*Best*), average (*Avg.*) and standard deviation (*Dev.*) of the cyclic bandwidth cost reached by that method over 31 independent executions, its average CPU time (T) in seconds, and the difference (D) between its best result (*Best*) and the corresponding best-known bound (either B^*_C or L_B). A statistical significance analysis was performed for these experiments by using the methodology described in Subsection 4.2 and the resulting *p-value* is presented. If a statistically significant difference exists between the performance of TScb and SAcb, the corresponding cells in the last column (SS) are marked either "+" or "−" depending on whether such a difference favors TScb or not. Unmarked cells indicate that there was not a significant difference between the compared algorithms.

Table A.1: Detailed performance assessment of the SAcb and TScb algorithms over 85 standard graphs from 7 different types all of them with known optimal solutions B^*_C.

				SAcb					TScb						
Graph	$\|V\|$	$\|E\|$	B^*_C	*Best*	*Avg.*	*Dev.*	*T*	*D*	*Best*	*Avg.*	*Dev.*	*T*	*D*	*p-value*	*SS*
path20	20	19	1	1	1.000	0.000	1.665	0	1	1.000	0.000	0.133	0	1.0E+00	
path25	25	24	1	1	1.500	0.527	1.876	0	1	1.000	0.000	0.603	0	1.2E-02	+
path30	30	29	1	1	1.600	0.843	3.107	0	1	1.000	0.000	1.422	0	3.0E-02	+
path35	35	34	1	2	2.600	0.516	3.292	1	1	1.000	0.000	3.420	0	3.8E-05	+
path40	40	39	1	2	2.800	0.422	3.711	1	1	1.000	0.000	3.330	0	2.7E-05	+
path100	100	99	1	15	18.500	2.173	0.030	14	1	1.500	0.527	1.920	0	1.4E-16	+
path125	125	124	1	21	25.100	3.348	0.037	20	1	1.700	0.483	5.865	0	4.6E-18	+
path150	150	149	1	28	31.000	2.108	0.055	27	2	2.000	0.000	3.295	1	6.6E-13	+
path175	175	174	1	31	35.300	2.111	0.083	30	2	2.000	0.000	6.836	1	5.0E-05	+
path200	200	199	1	40	42.300	2.791	0.103	39	2	2.000	0.000	9.612	1	4.1E-05	+
path300	300	299	1	63	67.400	3.134	0.191	62	3	3.200	0.422	10.470	2	6.8E-05	+
path475	475	474	1	115	120.600	4.300	0.375	114	5	5.600	0.516	14.566	4	1.2E-04	+

Continued on next page ...

Table A.1 – Continued from previous page

Graph	*|V|*	*|E|*	B_C^*	SACB					TSCB						
				Best	*Avg.*	*Dev.*	*T*	*D*	*Best*	*Avg.*	*Dev.*	*T*	*D*	*p-value*	*SS*
path650	650	649	1	171	176.700	4.473	0.592	170	7	7.000	0.000	17.428	6	7.0E-05	+
path825	825	824	1	229	237.600	3.373	0.870	228	8	8.000	0.000	19.831	7	3.9E-14	+
path1000	1000	999	1	293	300.600	5.082	1.104	292	8	8.900	0.316	20.409	7	5.0E-15	+
cycle20	20	20	1	1	1.300	0.483	1.746	0	1	1.000	0.000	0.356	0	6.7E-02	
cycle25	25	25	1	1	1.100	0.316	2.005	0	1	1.000	0.000	0.285	0	3.2E-01	
cycle30	30	30	1	1	2.400	0.966	2.210	0	1	1.000	0.000	0.573	0	1.4E-03	+
cycle35	35	35	1	1	2.400	0.843	3.315	0	1	1.000	0.000	0.571	0	5.1E-04	+
cycle40	40	40	1	2	3.000	0.816	3.535	1	1	1.000	0.000	0.548	0	4.7E-05	+
cycle100	100	100	1	16	19.200	3.393	0.031	15	1	1.000	0.000	1.898	0	4.0E-05	+
cycle125	125	125	1	22	24.500	1.841	0.047	21	1	1.000	0.000	1.920	0	3.8E-05	+
cycle150	150	150	1	25	29.000	3.127	0.067	24	1	1.000	0.000	3.300	0	3.8E-05	+
cycle175	175	175	1	32	35.600	2.797	0.085	31	1	1.000	0.000	8.192	0	4.8E-05	+
cycle200	200	200	1	38	40.200	1.317	0.114	37	1	1.000	0.000	9.254	0	5.0E-05	+
cycle300	300	300	1	64	67.300	2.263	0.186	63	3	3.100	0.316	11.528	2	7.1E-05	+
cycle475	475	475	1	113	119.800	4.780	0.388	112	5	5.800	0.422	14.703	4	3.8E-15	+
cycle650	650	650	1	170	178.100	4.067	0.605	169	7	7.600	0.516	16.348	6	3.4E-16	+
cycle825	825	825	1	237	241.500	3.749	0.885	236	7	7.900	0.316	18.262	6	7.3E-05	+
cycle1000	1000	1000	1	291	298.700	5.438	1.071	290	8	8.500	0.850	20.542	7	1.3E-20	+
mesh2D5x4	20	31	4	4	4.800	0.422	1.829	0	4	4.000	0.000	2.291	0	3.7E-04	+
mesh2D5x5	25	40	5	5	5.000	0.000	3.352	0	5	5.000	0.000	2.932	0	1.0E+00	
mesh2D5x6	30	49	5	5	6.200	1.317	4.702	0	5	5.000	0.000	1.633	0	5.0E-03	+
mesh2D5x7	35	58	5	6	7.100	1.449	2.790	1	5	5.000	0.000	1.393	0	4.0E-05	+
mesh2D5x8	40	67	5	6	6.500	0.972	4.401	1	5	5.000	0.000	1.770	0	3.5E-05	+
mesh2D10x10	100	180	10	26	28.000	2.309	0.047	16	10	10.500	0.527	3.832	0	5.6E-05	+
mesh2D5x25	125	220	5	28	33.500	2.415	0.090	23	5	5.000	0.000	2.790	0	2.0E-05	+
mesh2D10x15	150	275	10	39	42.400	3.596	0.121	29	11	11.000	0.000	3.031	1	2.0E-05	+
mesh2D7x25	175	318	7	45	47.200	1.751	0.179	38	7	8.700	3.335	6.183	0	6.3E-05	+
mesh2D8x25	200	367	8	52	53.700	1.059	0.231	44	8	8.200	0.422	3.588	0	7.2E-05	+
mesh2D15x20	300	565	15	81	88.800	3.553	0.434	66	16	33.900	28.368	6.330	1	4.1E-05	+
mesh2D19x25	475	906	19	154	160.200	4.614	0.824	135	119	119.900	0.316	9.267	100	1.8E-01	
mesh2D25x26	650	1249	25	233	236.600	4.061	1.246	208	164	164.000	0.000	14.154	139	1.9E-04	+
mesh2D28x30	840	1622	28	313	317.300	3.860	1.824	285	30	174.000	75.895	15.557	2	1.2E-01	
mesh2D20x50	1000	1930	20	376	386.600	6.484	2.206	356	21	113.100	117.825	21.600	1	1.0E-03	+
mesh3D4	64	300	14	19	20.700	0.949	0.032	5	16	16.000	0.000	0.060	2	1.3E-05	+
mesh3D5	125	540	21	34	38.500	2.718	0.168	13	21	21.700	0.483	0.260	0	4.7E-04	+
mesh3D6	216	882	30	62	76.700	6.897	0.370	32	31	31.000	0.000	6.578	1	7.7E-05	+
mesh3D7	343	1344	40	118	127.700	5.143	0.785	78	42	42.000	0.000	5.510	2	8.5E-05	+
mesh3D8	512	1344	52	200	207.400	3.239	1.341	148	129	129.600	0.516	17.460	77	2.1E-02	+
mesh3D9	729	1944	65	306	308.800	1.619	2.065	241	184	184.700	0.483	19.195	119	7.8E-14	+
mesh3D10	1000	2700	80	427	432.300	4.523	3.399	347	252	253.100	0.568	23.299	172	8.2E-09	+
mesh3D11	1331	3630	96	582	588.500	2.799	5.908	486	336	336.800	0.422	28.427	240	4.2E-11	+
mesh3D12	1728	4752	114	772	778.900	5.626	8.994	658	435	435.800	0.422	35.164	321	1.7E-16	+
mesh3D13	2197	6084	133	996	1002.600	3.658	13.382	863	553	553.600	0.699	41.161	420	6.3E-23	+
tree2x4	31	30	4	4	4.000	0.000	2.617	0	4	4.000	0.000	0.502	0	1.0E+00	
tree3x3	40	39	7	7	7.000	0.000	1.927	0	7	7.000	0.000	0.314	0	1.0E+00	
tree10x2	111	110	28	29	30.300	1.160	0.029	1	28	28.000	0.000	0.003	0	1.0E+00	
tree3x4	121	120	15	23	24.200	0.919	0.039	8	15	15.700	0.483	0.538	0	7.3E-07	+
tree5x3	156	155	26	33	36.900	1.912	0.049	7	26	26.000	0.000	5.869	0	3.7E-04	+
tree13x2	183	182	46	47	48.400	1.955	0.071	1	46	46.000	0.000	0.067	0	1.0E+00	
tree2x7	255	254	19	52	60.900	3.985	0.148	33	20	20.100	0.316	7.306	1	3.6E-09	+
tree17x2	307	306	77	83	88.900	3.213	0.196	6	77	77.000	0.000	0.525	0	1.0E+00	
tree21x2	463	462	116	133	139.700	3.529	0.376	17	116	116.000	0.000	0.811	0	1.0E+00	
tree25x2	651	650	163	203	207.400	3.658	0.620	40	163	163.000	0.000	1.098	0	3.2E-01	
tree5x4	781	780	98	219	229.800	5.884	0.716	121	98	98.200	0.422	19.670	0	6.0E-12	+
tree2x9	1023	1022	57	306	316.000	6.254	1.099	249	63	64.200	0.919	23.608	6	2.4E-15	+
caterpillar3	9	8	3	3	3.000	0.000	0.000	0	3	3.000	0.000	20.046	0	1.0E+00	
caterpillar4	14	13	3	3	3.000	0.000	0.368	0	3	3.000	0.000	0.329	0	1.0E+00	
caterpillar5	20	19	4	4	4.000	0.000	1.480	0	4	4.000	0.000	0.719	0	1.0E+00	
caterpillar6	27	26	5	5	5.000	0.000	1.644	0	5	5.000	0.000	0.540	0	1.0E+00	
caterpillar7	35	34	6	6	6.000	0.000	1.809	0	6	6.000	0.000	0.765	0	1.0E+00	
caterpillar13	104	103	10	16	20.400	3.307	0.032	6	10	10.000	0.000	4.058	0	2.0E-05	+
caterpillar14	119	118	11	20	25.000	4.190	0.036	9	11	11.000	0.000	5.391	0	3.4E-05	+
caterpillar16	152	151	13	30	33.800	4.467	0.065	17	13	13.000	0.000	6.835	0	3.4E-05	+
caterpillar17	170	169	14	32	36.000	2.749	0.085	18	14	14.000	0.000	4.902	0	3.8E-05	+
caterpillar19	209	208	15	41	45.200	1.751	0.121	26	15	15.900	0.316	9.500	0	1.1E-04	+

Continued on next page ...

Table A.1 – Continued from previous page

Graph	|V|	|E|	B_C^*	SAcb Best	SAcb Avg.	SAcb Dev.	SAcb T	SAcb D	TScb Best	TScb Avg.	TScb Dev.	TScb T	TScb D	p-value	SS
caterpillar23	299	298	19	65	72.500	4.478	0.181	46	19	19.300	0.483	13.259	0	2.0E-09	+
caterpillar29	464	463	24	118	124.100	3.604	0.358	94	24	25.800	0.919	15.077	0	3.9E-14	+
caterpillar35	665	664	29	182	196.000	8.524	0.657	153	31	32.300	1.252	18.250	2	6.8E-23	+
caterpillar39	819	818	33	241	251.000	8.367	0.941	208	34	38.500	4.275	21.364	1	1.8E-20	+
caterpillar44	1034	1033	37	321	334.700	6.977	1.358	284	39	54.000	7.732	23.906	2	4.4E-16	+
hypercube11	2048	11264	526	1021	1022.500	0.707	600.000	495	570	582.200	9.461	600.000	44	1.7E-21	+
hypercube12	4096	24576	988	2046	2046.900	0.316	600.000	1058	1175	1203.300	24.208	600.000	187	2.4E-18	+
hypercube13	8192	53248	1912	4095	4095.000	0.000	600.000	2183	2380	2462.400	116.956	600.000	468	5.3E-05	+
Average				191.812	195.909	2.686	22.484	131.176	88.435	93.345	4.738	29.484	27.800		

Table A.2: Detailed performance comparison of the SAcb and TScb algorithms over 28 Harwell-Boeing graphs.

Graph	|V|	|E|	Bounds L_B	Bounds U_B	B_C^*	SAcb Best	SAcb Avg.	SAcb Dev.	SAcb T	SAcb D	TScb Best	TScb Avg.	TScb Dev.	TScb T	TScb D	p-value	SS
jgl009	9	50	4	4	4	4	4.000	0.000	0.001	0	4	4.000	0.000	0.001	0	1.0E+00	
rgg010	10	76	5	5	5	5	5.000	0.000	0.001	0	5	5.000	0.000	0.001	0	1.0E+00	
jgl011	11	76	5	5	5	5	5.000	0.000	0.001	0	5	5.000	0.000	0.001	0	1.0E+00	
can_24	24	92	4	12	5	5	5.400	0.516	4.857	0	5	5.000	0.000	0.033	0	2.9E-02	+
pores_1	30	103	5	15	7	7	8.300	0.823	3.837	0	7	7.000	0.000	2.642	0	5.5E-04	+
ibm32	32	90	6	16	9	9	9.100	0.316	7.368	0	9	9.000	0.000	0.778	0	3.2E-01	
bcspwr01	39	46	3	19	4	5	5.800	0.789	2.603	1	5	5.000	0.000	1.044	1	4.9E-03	+
bcsstk01	48	176	6	24		12	12.400	0.843	10.073	6	12	12.000	0.000	3.699	6	1.5E-01	
bcspwr02	49	59	3	24		7	7.800	0.919	2.565	4	7	7.000	0.000	1.385	4	1.3E-02	+
curtis54	54	124	8	27		9	10.100	1.595	5.413	1	8	8.000	0.000	10.936	0	4.6E-05	+
will57	57	127	5	28		7	7.300	0.483	5.941	2	6	6.900	0.316	2.954	1	4.5E-02	+
impcol_b	59	281	9	29		17	17.800	0.422	12.136	8	17	17.000	0.000	6.387	8	3.7E-04	+
ash85	85	219	5	42		12	12.400	0.516	7.926	7	9	9.000	0.000	45.716	4	3.8E-05	+
nos4	100	247	3	50		13	14.700	4.001	8.172	10	10	10.000	0.000	5.801	7	4.3E-05	+
dwt_234	117	162	5	58		20	20.700	0.823	7.248	15	12	16.600	2.221	62.508	7	1.7E-04	+
bcspwr03	118	179	5	59		14	18.000	3.621	6.546	9	11	11.400	0.516	41.347	6	2.5E-04	+
bcsstk06	420	3720	14	210		209	209.000	0.000	0.434	195	49	51.800	2.098	178.202	35	5.1E-05	+
bcsstk07	420	3720	14	210		209	209.000	0.000	0.458	195	50	51.600	1.578	244.357	36	5.0E-05	+
impcol_d	425	1267	8	212		58	71.400	15.700	14.450	50	38	43.100	5.877	505.329	30	2.0E-04	+
can_445	445	1682	6	222		149	149.300	0.483	12.836	143	47	61.600	10.255	327.149	41	5.9E-10	+
494_bus	494	586	5	247		72	87.700	10.541	13.279	67	46	56.100	5.990	358.692	41	1.6E-07	+
dwt_503	503	2762	12	251		241	241.800	0.919	35.526	229	44	45.100	0.568	439.687	32	7.6E-40	+
sherman4	546	1341	3	273		77	126.600	26.154	14.124	74	29	29.800	0.632	103.231	26	9.4E-07	+
dwt_592	592	2256	7	296		88	166.200	51.923	17.978	81	30	31.100	0.876	338.423	23	1.8E-05	+
662_bus	662	906	5	331		121	137.000	11.489	14.672	116	52	56.100	4.433	284.321	47	1.5E-04	+
nos6	675	1290	2	337		171	172.700	1.337	12.893	169	22	23.500	0.972	93.658	20	1.2E-04	+
685_bus	685	1282	6	342		110	138.600	20.343	15.039	104	36	40.400	3.026	216.045	30	6.6E-08	+
can_715	715	2975	52	357		159	229.100	47.708	30.031	107	60	65.800	7.361	600.000	8	1.4E-06	+
Average						64.821	75.079	7.224	9.515	56.893	22.679	24.782	1.669	138.369	14.750		

References

[1] J. Leung, O. Vornberger, J. Witthoff, On some variants of the bandwidth minimization problem, SIAM Journal on Computing 13 (3) (1984) 650–667.

[2] Y. Lin, The cyclic bandwith problem, Journal of Systems Science and Complexity 7 (3) (1994) 282.

[3] S. N. Bhatt, F. Thomson Leighton, A framework for solving VLSI graph layout problems, Journal of Computer and System Sciences 28 (2) (1984) 300–343.

[4] A. L. Rosenberg, L. Snyder, Bounds on the costs of data encodings, Theory of Computing Systems 12 (1) (1978) 9–39.

[5] F. R. K. Chung, Labelings of graphs, in: L. W. Beineke, R. J. Wilson (Eds.), Selected topics in graph theory volume 3, Academic Press, 1988, Ch. 7, pp. 151–168.

[6] J. Hromkovič, V. Müller, O. Sýkora, I. Vrťo, On embedding interconnection networks into rings of processors, Lecture Notes in Computer Science 605 (1992) 51–62.

[7] L. H. Harper, Optimal assignment of numbers to vertices, Journal of SIAM 12 (1) (1964) 131–135.
[8] P. C. B. Lam, W. C. Shiu, W. H. Chan, Characterization of graphs with equal bandwidth and cyclic bandwidth, Discrete Mathematics 242 (3) (2002) 283–289.
[9] R. Martí, V. Campos, E. Piñana, A branch and bound algorithm for the matrix bandwidth minimization, European Journal of Operational Research 186 (2) (2008) 513–528.
[10] D. M. Cohen, S. R. Dalal, J. Parelius, G. C. Patton, The combinatorial design approach to automatic test generation, IEEE Software 13 (5) (1996) 83–88.
[11] D. M. Cohen, S. R. Dalal, M. L. Fredman, G. C. Patton, The AETG system: An approach to testing based on combinatorial design, IEEE Transactions on Software Engineering 23 (1997) 437–444.
[12] H. Romero-Monsivais, E. Rodriguez-Tello, G. Ramírez, A new branch and bound algorithm for the cyclic bandwidth problem, Lecture Notes in Artificial Intelligence 7630 (2012) 139–150.
[13] 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.
[14] M. Lozano, A. Duarte, F. Gortázar, R. Martí, Variable neighborhood search with ejection chains for the antibandwidth problem, Journal of Heuristics 18 (6) (2012) 919—938.
[15] Y. Lin, Minimum bandwidth problem for embedding graphs in cycles, Networks 29 (3) (1997) 135–140.
[16] R. Livesley, The analysis of large structural systems, Computer Journal 3 (1) (1960) 34–39.
[17] L. Yixun, Y. Jinjiang, The dual bandwidth problem for graphs, Journal of Zhengzhou University 35 (1) (2003) 1–5.
[18] R. Martí, M. Laguna, F. Glover, V. Campos, Reducing the bandwidth of a sparse matrix with tabu search, European Journal of Operational Research 135 (2) (2001) 211–220.
[19] 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.
[20] A. Lim, J. Lin, F. Xiao, Particle swarm optimization and hill climbing for the bandwidth minimization problem, Applied Intelligence 26 (3) (2007) 175–182.
[21] E. Rodriguez-Tello, J. K. Hao, J. Torres-Jimenez, An improved simulated annealing algorithm for bandwidth minimization, European Journal of Operational Research 185 (3) (2008) 1319–1335.
[22] N. Mladenovic, D. Urosevic, D. Pérez-Brito, C. G. García-González, Variable neighbourhood search for bandwidth reduction, European Journal of Operational Research 200 (1) (2010) 14–27.
[23] Z. Miller, D. Pritikin, On the separation number of a graph, Networks 19 (6) (1989) 651–666.
[24] W. Yao, Z. Ju, L. Xiaoxu, Dual bandwidth of some special trees, Journal of Zhengzhou University Natural Science Edition 35 (2003) 16–19.
[25] T. Calamoneri, A. Missini, L. Török, I. Vrt'o, Antibandwidth of complete k-ary trees, Electronic Notes in Discrete Mathematics 24 (2006) 259–266.
[26] L. Török, Antibandwidth of three-dimensional meshes, Electronic Notes in Discrete Mathematics 28 (2007) 161–167. doi:10.1016/j.endm.2007.01.023.
[27] A. Raspaud, H. Schröder, O. Sýkora, L. Török, I. Vrt'o, Antibandwidth and cyclic antibandwidth of meshes and hypercubes, Discrete Mathematics 309 (11) (2009) 3541–3552.
[28] R. Bansal, K. Srivastava, Memetic algorithm for the antibandwidth maximization problem, Journal of Heuristics 17 (1) (2011) 39–60.
[29] O. Sýkora, L. Török, I. Vrt'o, The cyclic antibandwidth problem, Electronic Notes in Discrete Mathematics 22 (2005) 233–227.
[30] S. Dobrev, R. Královic, D. Pardubská, L. Török, I. Vrt'o, Antibandwidth and cyclic antibandwidth of hamming graphs, Electronic Notes in Discrete Mathematics 34 (2009) 295–300.
[31] R. Bansal, K. Srivastava, A memetic algorithm for the cyclic antibandwidth maximization problem, Soft Computing 15 (2) (2011) 397–412.
[32] M. Lozano, A. Duarte, F. Gortázar, R. Martí, A hybrid metaheuristic for the cyclic antibandwidth problem, Knowledge-Based Systems 54 (2013) 103–113.
[33] S. Zhou, Bounding the bandwidths for graphs, Theoretical computer science 249 (2) (2002) 357–368.
[34] E. de Klerk, M. Eisenberg-Nagy, R. Sotirov, On semidefinite programming bounds for graph bandwidth, Technical report, Centrum Wiskunde & Informatica, (2011).
[35] J. Yuan, S. Zhou, Optimal labelling of unit interval graphs, Applied Mathematics, A Journal of Chinese Universities 10B (3) (1995) 337–344.
[36] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, 1997.
[37] Z. P. Lü, J. K. Hao, A memetic algorithm for graph coloring, European Journal of Operational Research 203 (1) (2010) 241–250.
[38] Z. P. Lü, J. K. Hao, Adaptive tabu search for course timetabling, European Journal of Operational Research 200 (1) (2010) 235–244.
[39] P. Galinier, Z. Boujbel, M. Coutinho Fernandes, An efficient memetic algorithm for the graph partitioning problem, Annals of Operations Research 191 (1) (2011) 1–22.
[40] J. Jin, T. G. Crainic, A. Løkketangen, A parallel multi-neighborhood cooperative tabu search for capacitated vehicle routing problems, European Journal of Operational Research 222 (3) (2012) 441–451.
[41] B. Cesaret, C. Oğuz, F. S. Salman, A tabu search algorithm for order acceptance and scheduling, Computers & Operations Research 39 (6) (2012) 1197–1205.
[42] F. Glover, M. Laguna, Tabu search, in: P. M. Pardalos, D. Z. Du, R. L. Graham (Eds.), Handbook of Combinatorial Optimization, 2nd Edition, Springer, 2013.
[43] R. Kothari, D. Ghosh, Tabu search for the single row facility layout problem using exhaustive 2-opt and insertion neighborhoods, European Journal of Operational Research 224 (1) (2013) 93–100.
[44] Z. P. Lü, J. K. Hao, F. Glover, Neighborhood analysis: a case study on curriculum-based course, Journal of Heuristics 17 (2) (2011) 97–118.
[45] Q. Wu, J. K. Hao, Memetic search for the max-bisection problem, Computers & Operations Research 40 (1) (2013) 166–179.
[46] J. Chvátalová, Optimal labelling of a product of two paths, Discrete Mathematics 11 (3) (1975) 249–253.
[47] L. Smithline, Bandwidth of the complete k-ary tree, Discrete Mathematics 142 (1–3) (1995) 203–212.
[48] W. A. de Landgraaf, A. E. Eiben, V. Nannen, Parameter calibration using meta-algorithms, in: In proceedings of the IEEE Congress on

Evolutionary Computation, IEEE Press, 2007, pp. 71–78.
[49] A. Gunawan, H. C. Lau, Lindawati, Fine-tuning algorithm parameters using the design of experiments, Lecture Notes in Computer Science 6683 (2011) 131–145.
[50] L. Gonzalez-Hernandez, J. Torres-Jimenez, MiTS: A new approach of tabu search for constructing mixed covering arrays., Lecture Notes in Artificial Intelligence 6438 (2010) 382–392.
[51] J. Richer, E. Rodriguez-Tello, K. E. Vazquez-Ortiz, Maximum parsimony phylogenetic inference using simulated annealing, in: O. Schütze, C. Coello Coello, A. Tantar, E. Tantar, P. Bouvry, P. Del Moral, P. Legrand (Eds.), EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation II, Vol. 175 of Advances in Intelligent Systems and Computing, Springer, 2013, pp. 189–203.
[52] C. J. Colbourn, Combinatorial aspects of covering arrays, Le Matematiche 58 (2004) 121–167.
[53] E. Rodriguez-Tello, J. Torres-Jimenez, Memetic algorithms for constructing binary covering arrays of strength three, Lecture Notes in Computer Science 5975 (2010) 86–97.
[54] H. Toutenburg, Shalabh, Statistical Analysis of Designed Experiments, 3rd Edition, Sp, 2009.
[55] T. Tsuchiya, Y. Takenaka, H. Taguchi, Multidisciplinary design optimization for hypersonic experimental vehicle, AIAA Journal 45 (7) (2007) 1655–1662.
[56] A. Azadeh, Z. S. Faiz, A meta-heuristic framework for forecasting household electricity consumption, Applied Soft Computing 11 (1) (2011) 614–620.
[57] P. Samui, Slope stability analysis using multivariate adaptive regression spline, in: X. S. Yang, A. H. Gandomi, S. Talatahari, A. H. Alavi (Eds.), Metaheuristics in Water, Geotechnical and Transport Engineering, Elsevier, 2013, Ch. 14, pp. 327–342.
[58] H. G. Santos, L. S. Ochi, M. J. F. Souza, A tabu search heuristic with efficient diversification strategies for the class/teacher timetabling problem, ACM Journal of Experimental Algorithmics 10 (Article No. 2.9) (2005) 1–16.
[59] T. James, C. Rego, F. Glover, Multistart tabu search and diversification strategies for the quadratic assignment problem, IEEE Transactions on Systems, Man, and Cybernetics, Part A 39 (3) (2009) 579–596.
[60] P. F. Stadler, Correlation in landscapes of combinatorial optimization problems, Europhysics Letters 20 (1992) 479–482.
[61] E. Pitzer, M. Affenzeller, A comprehensive survey on fitness landscape analysis, in: J. Fodor, R. Klempous, C. P. Suárez-Araujo (Eds.), Recent Advances in Intelligent Engineering Systems, Vol. 378 of Studies in Computational Intelligence, Springer, 2012, Ch. 8, pp. 161–191.
[62] E. Talbi, Metaheuristics: From design to implementation, John Wiley & Sons, 2009.