IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 7, 2019, accepted July 5, 2019, date of publication July 17, 2019, date of current version August 7, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2929410

An Iterated Three-Phase Search Approach for Solving the Cyclic Bandwidth Problem

JINTONG REN[1], JIN-KAO HAO[1,2], AND EDUARDO RODRIGUEZ-TELLO[3], (Member, IEEE)
[1]LERIA, Université d'Angers, 49045 Angers, France
[2]Institut Universitaire de France, 75231 Paris, France
[3]CINVESTAV—Tamaulipas, Ciudad Victoria 87130, Mexico

Corresponding authors: Jin-Kao Hao (jin-kao.hao@univ-angers.fr) and Eduardo Rodriguez-Tello (ertello@tamps.cinvestav.mx)

The work of J. Ren was supported by the China Scholarship Council (2016–2020) under Grant 201608070103. The work of E. Rodriguez-Tello was supported by the Mexican Secretariat of Public Education through SEP-CINVESTAV (2019–2020) under Grant 00114.

ABSTRACT The cyclic bandwidth problem (CBP) was initially introduced in the context of designing ring interconnection networks and has a number of other relevant applications, such as the design of computer networks and minimization of wire lengths in VLSI layout. However, the problem is computationally challenging since it belongs to the class of NP-hard problems. Existing studies on the CBP mainly focus on theoretical issues, and there are still very few practical methods devoted to this important problem. This paper fills the gap by introducing an iterated three-phase search approach for solving the CBP effectively. The proposed algorithm relies on three complementary search components to ensure a suitable balance of search intensification and diversification, guided by an enriched evaluation function. Computational assessments on a test-suite of 113 popular benchmark instances in the literature demonstrate the effectiveness of the proposed algorithm. In particular, it improves on 19 best-known computational results of the current best-performing algorithm for the problem and discovers 12 new record results (updated upper bounds). The key components of the proposed algorithm are investigated to shed light on their influences over the performance of the algorithm.

INDEX TERMS Combinatorial optimization, cyclic bandwidth minimization, multiple neighborhood search, threshold-based search, extended evaluation function.

I. INTRODUCTION

The Cyclic Bandwidth Problem (CBP) is a general and useful model able to formulate a number of practical applications. Initially introduced in the context of designing ring interconnection networks [1], the CBP involves finding an arrangement on a cycle for a set V of computers with a known communication pattern, given by the graph $G(V, E)$ to ensure that every message could be sent to its destination in at most k steps. The decision problem of the CBP is known to be $\mathcal{NP}$-complete [2]. Also, it has some other important applications in VLSI design [3], data structure representations [4], code designs [5] and parallel computer systems [6].

Let $G(V, E)$ be a finite undirected graph of order n and C_n a cycle graph. Given a bijection $\varphi : V \rightarrow V$ which represents an embedding (also called a labeling) of G in C_n, the cyclic bandwidth (the cost) for G with respect to φ is defined as:

$$\mathrm{Cb}(G, \varphi) = \max_{(u,v)\in E} \{|\varphi(u) - \varphi(v)|_n\}, \quad (1)$$

where $|x|_n = \min\{|x|, n - |x|\}$, with $|x| \in (0, n)$, is called the *cyclic distance*, and $\varphi(u)$ denotes the label associated to vertex u. The cyclic bandwidth $\mathrm{Cb}(u, \varphi)$ for a particular vertex u, with set of adjacent vertices $|A(u)| = deg(u)$, under the embedding φ can be computed as follows:

$$\mathrm{Cb}(u, \varphi) = \max_{v\in A(u)} \{|\varphi(u) - \varphi(v)|_n\}. \quad (2)$$

The main objective of the CBP is thus to find an embedding φ^* such that $\mathrm{Cb}(G, \varphi^*)$ is minimized:

$$\varphi^* = \underset{\varphi\in\Omega}{arg\,min} \{\mathrm{Cb}(G, \varphi)\}, \quad (3)$$

where Ω represents the set of all possible embeddings. The embedding φ^* satisfying this condition is called an optimal or exact embedding of the given graph.

The associate editor coordinating the review of this manuscript and approving it for publication was Huaqing Li.

TABLE 1. Table of cyclic distances for all the edges of graph G depicted in Fig. 1.

$x = (u, v) \in E$	(a, b)	(a, e)	(a, f)	(a, h)	(b, d)	(b, g)	(c, e)	(c, j)	(d, i)	(e, g)	(f, g)	(f, j)	(h, i)
$\|x\| = \|\varphi(u) - \varphi(v)\|$	4	3	2	1	8	1	4	1	7	8	7	4	4
$\|x\|_n = \min(\|x\|, n - \|x\|)$	4	3	2	1	2	1	4	1	3	2	3	4	4

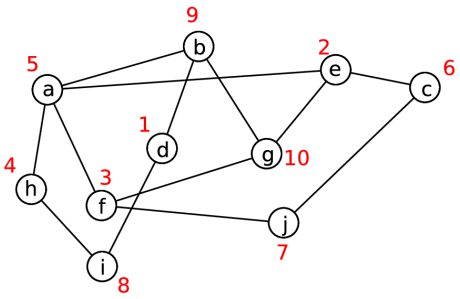

FIGURE 1. An illustrative example of graph G of order $n = 10$ where the vertices are named from a to j and a labeling is represented by the red numbers from 1 to 10.

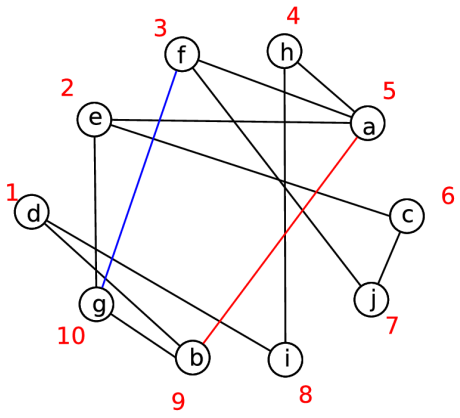

FIGURE 2. The graph G of Fig. 1 with its vertices a to j reordered clockwise on a cycle according to the label numbers 1 to 10 (in red).

Fig. 1 shows a graph with ten vertices ($n = 10$) named from a to j with an embedding or vertex labeling indicated in red from 1 to 10. In Fig. 2, the vertices of G are reordered clockwise on a cycle according to the label numbers (in red). So for each edge $(u, v) \in E$, it is easy to calculate the labels $\varphi(u)$ and $\varphi(v)$ to get the absolute distance $|x|$ and the cyclic distance $|x|_n$ (see Table 1). For instance, the edge $x = (f, g)$ (in blue) has an absolute distance $|x| = 7$ (i.e., the number of vertices from f to reach g in a clockwise direction in the cycle) while its cyclic distance $|x|_n$ equals 3 ($\min\{7, 10 - 7\}$, which is also the number of vertices from f to reach g in a counterclockwise direction). According to (1), the cyclic bandwidth $\mathrm{Cb}(G, \varphi)$ of this graph is the maximum value among all the $|x|_n$, i.e., $\mathrm{Cb}(G, \varphi) = 4$ which concerns the edge $x = (a, b)$ of Fig. 2 indicated in red.

Until now, the majority of the existing studies concern either special graphs whose exact cyclic bandwidths can be determined theoretically or propositions to define lower and upper bounds of a general graph. For instance, in [7], it was shown that for every unit interval graph, there exists a simultaneously optimal labeling for several labeling problems including the CBP. The study of [6] established the relationships between the bandwidth $B_P(G)$ and the cyclic bandwidth $\mathrm{Cb}(G)$: $B_P(G) \geq \mathrm{Cb}(G) \geq \frac{1}{2}B_P(G)$.

Following this result, studies of [8]–[10] identified the criterion conditions for two extreme cases $B_P(G) = \mathrm{Cb}(G)$ and $\frac{1}{2}B_P(G) = \mathrm{Cb}(G)$, and further obtained some exact values for special graphs including trees, planar graphs, triangulation meshes, grids with specific characteristics and some other graphs with particular conditions.

In [11], a systematic method was proposed to achieve a number of lower bounds for the bandwidth of a graph, which is then used to obtain lower bounds for the CBP in terms of some distance- and degree-related parameters.

The work of [12] was devoted to the upper bound of the cyclic bandwidth of a general graph with an edge added. By exploring the property that the cyclic distance between any pair of adjacent vertices will not be affected by shifting all vertices in the cyclic order the same distance, a sharp upper bound was obtained.

The study of [13] used the semi-definite programming (SDP) relaxations of the quadratic assignment problem to propose two new lower bounds on the bandwidth and cyclic bandwidth, which are shown to be better than two other previous SDP bounds.

In addition to these theoretical results, little effort has been made to develop practical solution methods for the CBP. To our knowledge, there are only three published algorithms on solving the CBP. In [14], a branch and bound algorithm was proposed that can solve some standard instances (like path, mesh and cycle) of small sizes limited to 40 vertices. To handle larger instances, a heuristic algorithm based on the tabu search metaheuristic (named TScb) was presented in [15]. The authors also adapted a highly effective simulated annealing designed for the related Bandwidth Minimization Problem (BMP) [16] to the cyclic bandwidth problem. Their experimental assessment on a set of benchmark instances demonstrated the superiority of TScb over the simulated annealing algorithm. As a result, TScb can be considered as the state-of-the-art algorithm for the CBP and will serve as the main reference for our computational study.

Our literature review indicates that contrary to the BMP, for which various solution methods have been proposed (e.g., [17]–[20]), effective algorithms dedicated to the CBP remains scarce. To enrich the practical solution arsenal

for this important optimization problem, we introduce in this work an iterated three-phase search algorithm (ITPS) for solving the CBP. The algorithm is characterized by the following features. First, the algorithm is composed of three key search components: a double neighbor decent phase to find a local optimal solution, a responsive threshold-based search phase to explore the nearby regions for the purpose of discovering better solutions and a special perturbation phase to displace the search to a new and distant region. The algorithm also integrates an extended evaluation function which enriches the optimization objective by additional information. This function is used to discriminate many solutions with the same cyclic bandwidth and provides a relevant means for guiding the search process.

We assess the proposed algorithm on a set of 113 well-known benchmark instances taken from the literature. This set of instances includes 85 standard graphs (e.g., paths, cycles, caterpillars, etc) and 28 Harwell-Boeing graphs which arise from diverse engineering and scientific real-world problems. The comparisons with the results produced by the state-of-the-art reference method show the competitiveness of our algorithm. For the set of 85 standard graphs, our algorithm improves on 19 best computational (upper) bounds and matches 60 best-known computational results from the literature. For the set of 28 Harwell-Boeing graphs, our algorithm discovers new record results (updated upper bounds) for 12 graphs and matches the best-known results for 15 other graphs.

The remainder of this manuscript is organized as follows: Section II first introduces the main scheme of the proposed algorithm. Then, the implementation details of the neighbor-based decent procedure as well as the responsive threshold-based search method are presented. In Section III a set of computational experiments is presented. They are devoted to determine the best input parameter values for the ITPS algorithm and to compare its performance with respect to the reference algorithm in the literature, TScb [15]. Section IV experimentally investigates the extent to which key components of the ITPS algorithm can influence its global performance. Finally, the main conclusions drawn from this work, and some future work ideas are discussed in Section V.

II. ITERATED THREE-PHASE SEARCH FOR THE CBP

A. MAIN SCHEME

The proposed ITPS algorithm was inspired by the three-phase approach presented in [21]. Even if the work of [21] concerns a particular optimization problem (i.e., the quadratic minimum spanning tree problem), the approach is of general interest and has been applied to other problems such as clique partitioning [22]. In this work, we adapted this three-phase approach to the CBP by reusing its general framework and making dedicated adaptations to deal with the particular features of our considered problem.

Let $G = (V, E)$ be a graph of order $|V| = n$ and a cycle graph $C_n = (V', E')$, the search space Ω considered by our ITPS algorithm is composed of all candidate embeddings (labellings or solutions) of G in C_n, $\varphi : V \rightarrow V'$. In our implementation, an embedding φ is represented by a permutation of $\{1, 2, \ldots, n\}$ such that the i-th element denotes the label assigned to vertex $i \in V$. To effectively explore the space Ω, ITPS combines a double neighborhood descent search, a responsive threshold-based search as well as a specific perturbation. To cope with the difficulty of discriminating many equal-cost candidate solutions, ITPS integrates an extended evaluation function using graph structure information.

Algorithm 1 ITPS Algorithm for the CBP

1: **Input**: Finite undirected graph $G(V, E)$, neighborhoods N_1 and N_2, extended evaluation function f_e, search depth δ and cutoff time limit T_{max}
2: **Output**: The best solution found φ^*
3: $\varphi \leftarrow InitialSolution()$
4: $\varphi^* \leftarrow \varphi$
5: **while** the cutoff time limit T_{max} is not reached **do**
6: $\quad NonImp \leftarrow 0$
7: $\quad$ **while** $NonImp < \delta$ **do**
8: $\quad\quad (\varphi, \varphi^*) \leftarrow DNDS(\varphi, \varphi^*, N_1, N_2)$ // Section II-C
9: $\quad\quad (\varphi, \varphi^*) \leftarrow RTBS(\varphi, \varphi^*, N_1, N_2)$ // Section II-D
10: $\quad\quad NonImp \leftarrow NonImp + 1$
11: $\quad$ **end while**
12: $\quad \varphi \leftarrow Purturbation(\varphi)$ // Section II-E
13: **end while**
14: **return** φ^*

The pseudo-code of the ITPS algorithm is presented in Algorithm 1. It starts with a randomly generated solution φ. Then the algorithm enters the main 'while' loop (lines 5-13), Alg. 1) to explore solutions of increasing quality in terms of the extended evaluation function f_e. At each iteration, the descent search (first phase, Section II-C) is first run to find a local optimal solution using two neighborhoods N_1 and N_2 (line 8, Alg. 1). This phase is followed by the responsive threshold-based search (second phase, Section II-D) to discover additional local optima of better quality from the incumbent solution (line 9, Alg. 1). These two phases are repeated δ times. At this point, the search is judged to be trapped in a deep local optimum. To overcome the trap, the perturbation procedure (third phase, Section II-E) is triggered to strongly transform the incumbent solution (line 12, Alg. 1). The search then goes back to the first phase with the perturbed solution as its new starting solution. During the search, each time a solution better than the previous best recorded solution is found, φ^* is updated. The whole search process stops when a given cutoff time limit (T_{max}) is reached. As the output of the algorithm, the best recorded solution φ^* is returned.

B. EXTENDED EVALUATION FUNCTION

A notable feature of the CBP is that many solutions may have the same objective value. This is because there are $(n-1)!/2$ possible solutions while there are only $\lfloor n/2 \rfloor$ different possible objective values, see equation (1). From the local optimization perspective, it is critical to discriminate the solutions with the same objective value. For this purpose, we devise an extended evaluation function f_e as follows.

Let $\varphi \in \Omega$ be a candidate solution with cyclic bandwidth cost $\mathrm{Cb}(G, \varphi)$. Let $NumE(\mathrm{Cb}(G, \varphi))$ represent the number of edges whose cyclic bandwidth equals $\mathrm{Cb}(G, \varphi)$:

$$NumE(\mathrm{Cb}(G, \varphi)) = \sum_{(u,v)\in E} X_{uv}, \tag{4}$$

where $X_{uv} = 1$ if $|\varphi(u) - \varphi(v)|_n = \mathrm{Cb}(G, \varphi)$; otherwise $X_{uv} = 0$. Then, the extended evaluation function f_e is given by:

$$f_e(\varphi) = \mathrm{Cb}(G, \varphi) + \frac{NumE(\mathrm{Cb}(G, \varphi))}{|E|}. \tag{5}$$

As we show below, this evaluation function is able to distinguish the solutions that under the conventional evaluation function presented in (1) have the same objective value. An analysis of the influence of the new evaluation function f_e is provided in Section IV-A.

Figure 3 shows an example of the extend evaluation function f_e applied to two solutions with the same objective value $\mathrm{Cb}(G, \varphi) = 4$. According to the extended evaluation function, $f_e(\varphi_1) = 4 + 1/13 = 4.0769$, while $f_e(\varphi_2) = 4 + 4/13 = 4.3076$. The embedding φ_1 is considered to be "better" than the embedding φ_2. This is reasonable, because one notices, from Fig. 3(a), that for reducing the cost value $\mathrm{Cb}(G, \varphi)$ of the embedding φ_1 it is necessary to deal with only one edge (marked in red), while for embedding φ_2, depicted in Fig. 3(b), there are four edges (marked in red) that should be considered. Thus, it is easier to operate with φ_1 than with φ_2 to reduce the cyclic bandwidth of G.

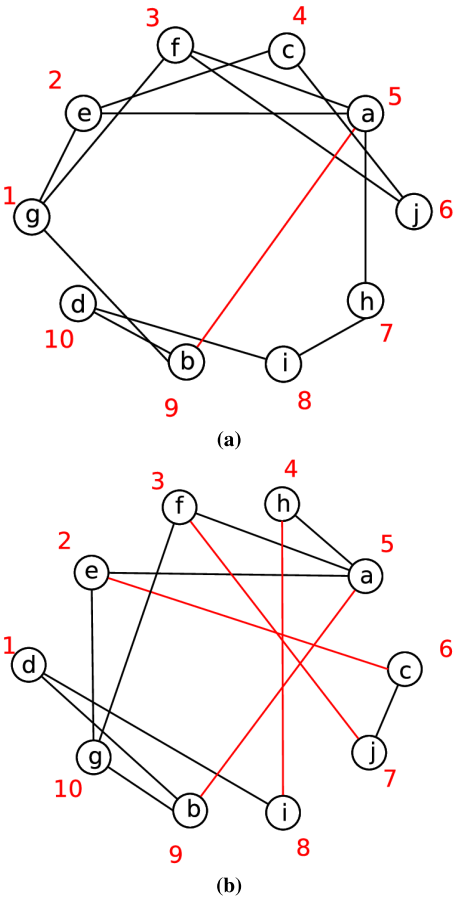

FIGURE 3. An illustration of the extended evaluation function f_e applied to two different embeddings. (a) φ_1. (b) φ_2. Both embeddings have the same cost (cyclic bandwidth) under the conventional evaluation function (1). However, the new function f_e discriminates these embeddings by assigning to them two different values $f_e(\varphi_1) = 4 + 1/13 = 4.0769$ and $f_e(\varphi_2) = 4 + 3/13 = 4.2307$.

C. FIRST PHASE - DOUBLE NEIGHBORHOOD DESCENT SEARCH

To explore the given search space, we first apply the double neighborhood decent search procedure (DNDS) whose general scheme is shown in Algorithm 2. Basically, DNDS explores the two neighborhoods N_1 and N_2 defined below and iteratively replaces the incumbent solution by a neighbor solution selected from a set of candidate neighbors. At each iteration, DNDS uses either N_1 or N_2 to create the candidate list (*CLst*) by identifying the solutions no worse than the incumbent solution in terms of the evaluation function f_e (lines 6-16, Alg. 2). A priority is always given to N_1 and N_2 is examined only if the neighbor solutions in N_1 are all worse than the incumbent solution. If the candidate list is not empty (i.e., it contains at least one improving or non-worsening neighbor solution), either one best neighbor solution, or a random neighbor solution is chosen from *CLst* to become the new current solution according to probability ρ_{best} (lines 18-22, Alg. 2). Notice that given the criterion used to build *CLst*, the selected neighbor solution is always at least as good as the replaced solution. In case *CLst* contains no candidate solution, DNDS moves to the next iteration without performing a solution transition (the number of consecutive non-improving iterations is indicated by *NonImpCounter*, line 31, Alg. 2). During the search, the best-found solution φ^* is updated each time a better solution is attained. The DNDS process terminates if the best-found solution φ^* cannot be updated during L_d consecutive iterations. In this case, DNDS has attained a local optimum and the ITPS algorithm switches to the responsive threshold-based search method for escaping this local optimum trap and to continue looking for new better quality solutions.

1) NEIGHBORHOODS

The two neighborhoods N_1 and N_2 explored by DNDS are defined by the general *swap* operator. Let φ be the incumbent solution, then a neighbor solution φ' can be generated by exchanging the labels of vertices u and v with the

Algorithm 2 Double Neighborhood Descent Search

Input: input solution φ, best optimum found φ^*, neighborhoods N_1 and N_2, evaluation function f_e, maximum non-improving limit L_d, and best neighbor move strategy probability ρ_{best}
Output:last local optimum φ, best optimum found φ^*
$NonImpCounter \leftarrow 0$
$Improving \leftarrow True$
while $NonImpCounter < L_d$ **do**
  if $Improving$ **then**
    $N \leftarrow N_1$
  else
    $N \leftarrow N_2$
  end if
  $CLst \leftarrow \emptyset$
  for each $\varphi' \in N(\varphi)$ **do**
    if $f_e(\varphi') \leq f_e(\varphi)$ **then**
      $CLst \leftarrow CLst \cup \{\varphi'\}$
    end if
  end for
  if $CLst \neq \emptyset$ **then**
    if $rand(0, 1) < \rho_{best}$ **then**
      $\varphi \leftarrow BestSol(CLst)$ // With probability ρ_{best}
    else
      $\varphi \leftarrow RandomSol(CLst)$
    end if
    $Improving \leftarrow True$
  else
    $Improving \leftarrow False$
  end if
  if $f_e(\varphi) < f_e(\varphi^*)$ **then**
    $NonImpCounter \leftarrow 0$
    $\varphi^* \leftarrow \varphi$
  else
    $NonImpCounter \leftarrow NonImpCounter + 1$
  end if
end while
return φ, φ^*

operation $swap(u, v)$. Without any restriction, the *swap* operator leads to a neighborhood of size of order $O(n^2)$, which is too large to be explored efficiently. Following the idea of [15], we use two constrained neighborhoods by imposing specific conditions on the vertices that take part in a *swap* operation.

The first neighborhood $N_1(\varphi)$ is given by the set of neighbor solutions obtained by swapping a critical vertex $u \in C(\varphi)$ and a specific vertex $v \in S(u)$ adjacent to u:

$$N_1(\varphi) = \{\varphi' = \varphi \oplus swap(u, v) : u \in C(\varphi), v \in S(u), swap(u, v) \notin TL\}, \quad (6)$$

where $\varphi' = \varphi \oplus swap(u, v)$ denotes the neighbor solution obtained by applying $swap(u, v)$ to transform φ, TL is the so-called tabu list that records the swaps that were recently performed (see Section II-C.2). The set $C(\varphi)$ contains a group of critical vertices $w \in V$ having a cyclic bandwidth $\mathrm{Cb}(w, \varphi) = \mathrm{Cb}(G, \varphi)$, while $S(u) \subseteq V$ is the set containing those vertices z currently labeled with values closer to $mid(u)$ than to $\varphi(u)$ (i.e., $|mid(u) - \varphi(z)|_n < |mid(u) - \varphi(u)|_n$). The value $mid(u)$ stands for the middle point of the shortest path in the cycle C_n containing all the vertices adjacent to u [15].

The descent procedure uses this strongly constrained neighborhood $N_1(\varphi)$ to make an intensified exploration of candidate solutions.

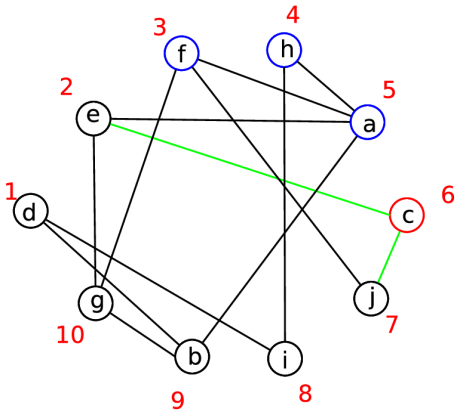

FIGURE 4. A simple illustration of the neighborhood $N_1(\varphi)$. The embedding φ containing a critical vertex $c \in C(\varphi)$ (marked in red), as well as the set $S(c) = \{a, h, f\}$ of suitable vertices eligible to be swapped with vertex c (highlighted in blue) are depicted.

Figure 4 depicts an illustrative example of the neighborhood $N_1(\varphi)$. It presents an embedding φ containing a critical vertex $c \in C(\varphi)$ (marked in red), which has the label 6 assigned to it. Using its adjacent vertices $A(c) = \{e, j\}$ (edge (c, e) and edge (c, j) marked in green), we identify the vertex h (having label 4) as the middle point $mid(c)$ of the shortest path in the cycle C_n containing all the vertices in $A(c)$. Thus, all the vertices highlighted in blue (i.e., a, h and f) are in the suitable set $S(c)$ and are eligible to be swapped with vertex c.

For the purpose of search diversification, the descent procedure employs also a larger neighborhood $N_2(\varphi)$ which is specified by the following expression:

$$N_2(\varphi) = \{\varphi' = \varphi \oplus swap(u, v) : u \in C(\varphi), v \in R_\gamma(u), swap(u, v) \notin TL\}, \quad (7)$$

where the set $R_\gamma(u) \subseteq V$ contains $\gamma * n$ randomly selected vertices ($\gamma \in (0, 1]$). Compared to $N_1(\varphi)$, the *swap* operator can exchange a critical vertex u with any other vertex in the graph, leading to a much higher freedom for a swap operation. Since the neighbor solutions of $N_2(\varphi)$ are more varied, this neighborhood promotes search diversification.

Compared to swapping all pairs of labels to generate neighbor solutions, the neighborhoods $N_1(\varphi)$ and $N_2(\varphi)$ are much smaller in size. Indeed, $N_1(\varphi)$ contains $|C(\varphi)|*|\overline{S(.)}|$ neighbor solutions, where $|\overline{S(.)}|$ is the average number of suitable vertices for a critical vertex with respect to the current solution φ, while $N_2(\varphi)$ has $|C(\varphi)| * \gamma * n$ neighbor solutions.

Our preliminary experiments indicated that for the tested instances $|C(\varphi)| \leq 0.1 * n$ and $|\overline{S(.)}| \leq 0.1 * n$ hold. For this reason the value of γ was set to 0.05 or 0.1 in our experiments. As a result, each iteration of the descent procedure only considers $0.01n^2$ candidate solutions, which significantly accelerates the search process.

Finally, we adopted a fast incremental technique to evaluate a neighbor solution φ' according to the evaluation function f_e. Let φ' be an embedding obtained by swapping u and v in φ. Then, to obtain $f_e(\varphi')$ from $f_e(\varphi)$, we need only to recalculate the changing part $|A(u)| + |A(v)|$ ($|A(u)|$ and $|A(v)|$ represent the number of adjacent vertices to u and v, respectively). This ensures that each iteration of the algorithm requires a time complexity bounded by $\mathcal{O}((|A(u)| + |A(v)|) * n^2)$.

2) TABU LIST MANAGEMENT

Since the double neighborhood descent search only accepts non-deteriorating (i.e., improving or equal cost) neighbor solutions, it is possible that a previously visited solution is reconsidered at a later iteration, leading to search cycling. To avoid this problem, the DNDS procedure integrates a tabu list that is a key concept of the tabu search method [23]. The idea is to keep track of the performed swaps and forbid the reverse swap operations during the next τ iterations (τ is an input parameter called the tabu tenure). So when $swap(u, v)$ is performed to transform the current solution, $swap(u, v)$ is added in the tabu list and it is forbidden to swap vertices v and u during the period fixed by the tabu tenure. In principle, the tabu tenure can take a fixed value or can be dynamically calculated during the search. We adopt a dynamic tabu tenure technique introduced in [24]. As shown in other studies [25], [26], this technique proves to be robust and effective in different settings and was also used in [15] for the CBP. This technique applies a periodic step function that takes as argument the number of iterations *iter* for computing the tabu tenure value. The value returned by this function for a particular iteration *iter* is given by $(a_j)_{j=1,2,...,15} = (1, 2, 1, 4, 1, 2, 1, 8, 1, 2, 1, 4, 1, 2, 1) \times d$, where d is a parameter fixing the minimum tabu tenure (set to 100 in this work) and index j is computed by $j = \lfloor \frac{iter \mod 1500}{100} \rfloor + 1$. Therefore, each period of this function is composed of 1500 iterations divided into 15 intervals.

3) DISCUSSIONS

Like [15], the first phase of our ITPS algorithm is based on two neighborhoods. However, there are some notable differences. First, our neighborhood N_1 uses a set $C(\varphi)$ of critical vertices defined by the condition $\text{Cb}(w, \varphi) = \text{Cb}(G, \varphi)$, which is more restrictive than the condition $\text{Cb}(w, \varphi) \geq \alpha * \text{Cb}(G, \varphi)$ (α is a prefixed parameter between 0 and 1) used in [15]. In this way, the set of critical vertices is reduced and each iteration needs to examine fewer candidate solutions. Second, we make a swap move after visiting all candidate solutions induced by all critical vertices in $C(\varphi)$ while in [15] a swap move is performed after visiting the candidate solutions of only one critical vertex. The advantage of our strategy is that we could encounter a better solution at each iteration, and have less chance to miss an elite solution. Third, in [15], the two neighborhoods are used according to a probability. In our work, N_1 is always applied with priority and N_2 is used only when N_1 is exhausted (i.e., when a local optimum is attained with N_1). Finally, our first phase uses the descent procedure to ensure an efficient search intensification (i.e., no worsening neighbor solution is allowed), while the algorithm of [15] uses tabu search which may accept worsening solution transitions.

D. SECOND PHASE - RESPONSIVE THRESHOLD-BASED SEARCH

As explained in Section II-C, the double neighborhood based descent search only accepts non-deteriorating neighbor solutions. As such, it can be trapped in local optima. When this happens, we trigger the second search phase and apply the responsive threshold-based search (RTBS) to escape such traps. During the second phase, both improving and deteriorating neighbor solutions can be accepted in order to favor a large exploration of the search space.

Like the double neighborhood based descent search, the responsive threshold-based search also relies on the neighborhoods N_1 and N_2. However, RTBS adopts the threshold accepting heuristic [27], [28] as the criterion for solution transitions. As such, a solution whose quality does not drop below a given threshold can be accepted to replace the incumbent solution. To further enforce search exploration, the two neighborhoods are considered alternatively according to a probability ρ_{N_1}. The general responsive threshold-based search procedure is described in Algorithm 3.

RTBS starts each iteration by calculating the responsive threshold, denoted by T (line 5, Alg. 3). Then it iteratively makes transitions from the current solution to a neighbor solution which is selected by examining the neighborhoods N_1 and N_2. The former is applied with probability ρ_{N_1}, while the latter is employed at a $(1 - \rho_{N_1})$ rate (lines 6-10, Alg. 3). This is simulated with a random number generated in the interval $(0, 1)$. Then all neighbor solutions whose quality is no worse than the threshold T are identified to form the *CLst* (lines 12-16, Alg. 3). Finally, according to the probability ρ_{best}, either a best solution or a random solution is selected from *CLst* (like DNDS does) and used to replace the current solution (lines 18-22, Alg. 3). best solution found φ^* during the search is updated each time a better solution is discovered (lines 24-29, Alg. 3). If φ^* is not updated, we increase the counter of non-improving iterations *NonImpCounterT* and move to the next iteration. This process stops if the best local optimum found during this run cannot be updated during L_t consecutive iterations. In this case, the search is supposed to be trapped in a deep local optimum.

One key issue concerns the threshold T. Indeed, if T takes a value that is far from the current objective value ($T - \text{Cb}(G, \varphi) \gg 0$), even very bad neighbor solutions can be accepted, leading to a random-like search. On the other hand, if T takes a value that is too close to the current

Algorithm 3 Responsive Threshold-Based Decent Procedures

Input: input solution φ, best found solution φ^*, neighborhoods N_1 and N_2, evaluation function f_e, maximum non-improving limit L_t, neighborhood N_1 application probability ρ_{N_1}, and best neighbor move strategy probability ρ_{best}
Output: best found solution φ^*, last solution φ
$NonImpCounterT \leftarrow 0$
while $NonImpCounterT < L_t$ **do**
  $T \leftarrow Threshold(\varphi)$
  if $rand(0, 1) < \rho_{N_1}$ **then**
    $N \leftarrow N_1$ // With probability ρ_{N_1}
  else
    $N \leftarrow N_2$
  end if
  $CLst \leftarrow \emptyset$
  for each $\varphi' \in N(\varphi)$ **do**
    if $f_e(\varphi') \leq T$ **then**
      $CLst \leftarrow CLst \cup \{\varphi'\}$
    end if
  end for
  if $CLst$ is not empty **then**
    if $rand(0, 1) < \rho_{best}$ **then**
      $\varphi \leftarrow BestSol(CLst)$ // With probability ρ_{best}
    else
      $\varphi \leftarrow RandomSol(CLst)$
    end if
  end if
  if $f_e(\varphi) < f_e(\varphi^*)$ **then**
    $NonImpCounterT \leftarrow 0$
    $\varphi^* \leftarrow \varphi$
  else
    $NonImpCounterT \leftarrow NonImpCounterT + 1$
  end if
end while
return φ, φ^*

objective value ($T - \mathrm{Cb}(G, \varphi) \approx 0$), the search will behave like the descent search and can hardly escape local optimum traps. To identify a suitable threshold T, we follow the work of [29] and use a responsive mechanism to dynamically tune T according to the current objective value $\mathrm{Cb}(G, \varphi)$ and a threshold ratio r. Specifically, we set T as follows $T = (1 + r) * \mathrm{Cb}(G, \varphi)$, where $r = 1/(a * \mathrm{Cb}(G, \varphi) + b) + c$. The coefficients a, b, and c were empirically fixed at 0.00891104, 0.52663736 and 0.16331589, respectively. It was carried out by solving simultaneously three equations produced with the following pairs of $(\mathrm{Cb}(G, \varphi), r)$ values obtained from preliminary experiments: $\{(2, 2), (150, 0.7), (3000, 0.2)\}$. As a result, the threshold T evolves according to $\mathrm{Cb}(G, \varphi)$ and the threshold ratio r. T tends to become small when the current solution is of high quality so that only improving or limited worsening neighbor solutions are accepted. Inversely, T tends to become large when the current solution is not so good in order to encourage more exploration.

E. THIRD PHASE - SHIFT-INSERT-BASED PERTURBATION

With its threshold accepting strategy, the responsive threshold-based search ensures a large exploration of solutions of various quality. When this second phase is exhausted, we trigger a strong perturbation to displace the search to a new and distant region of the search space. Specifically, this is achieved by applying the *ShiftInsert* operator to transform the current solution as follows.

Let φ be the current solution with cyclic bandwidth $\mathrm{Cb}(G, \varphi)$. Let $W = \{(u, v) \in E : |\varphi(u) - \varphi(v)|_n = \mathrm{Cb}(G, \varphi)\}$ be the set of edges whose cyclic distance equals $\mathrm{Cb}(G, \varphi)$. Let $e = (u, v)$ be an edge randomly taken from W such that $\varphi(u) > \varphi(v)$. The *ShiftInsert*(u, v) operator first removes u, then shifts all vertices between u and v clockwise or anti-clockwise at random, and finally inserts u at the position of v. In practice, *ShiftInsert*(u, v) is realized by performing $\mathrm{Cb}(G, \varphi) - 1$ successive *swap*(u, x) operations where x denotes the inverse clockwise nearby vertex of u in the solution undergoing transformation until x reaches vertex v.

An illustrative example is shown in Fig. 5(a) (solution before the *ShiftInsert* operation) and Fig. 5(b) (solution after the *ShiftInsert* operation). In this example, $\mathrm{Cb}(G, \varphi) = 4$ and edge (c, e) is chosen for *ShiftInsert* among $W = \{(c, e), (f, j), (h, i), (a, b)\}$, which is the set of edges with a cyclic distance of 4. *ShiftInsert*(c, e) is performed by three successive *swap* operations: *swap*(c, a), *swap*(c, h), and *swap*(c, f). Table 2 indicates the changes of the cyclic distances of the edges impacted by the *ShiftInsert*(c, e) operation.

The *Shift-Insert*-based perturbation has some interesting features. On the one hand, by displacing a significant number of vertices, this strategy helps to break long standing ties and forces the search to overcome deep local traps. Second, by considering edges whose cyclic distance is equal to the current cyclic bandwidth, this strategy maintains the quality of the transformed solution at a reasonable level and thus avoids searching from a lower quality solution.

When the third phase is triggered, the *Shift-Insert*-based perturbation is applied one time to transform the current solution. The modified solution is then used as the new starting solution of the next round of the ITPS algorithm. In Section IV-C, we investigate the usefulness of the *Shift-Insert*-based perturbation.

III. COMPUTATIONAL EXPERIMENTS

This section is dedicated to an experimental assessment of the proposed ITPS algorithm, the experimental setup, the test-suite, the procedure used to set the parameter values and a performance comparison between ITPS and TScb (the reference state-of-the-art method) [15].

TABLE 2. Changes of the cyclic distances associated to the edges impacted by the *ShiftInsert* operation when applied over the solution depicted in Fig. 5.

(u, v)	(f, g)	(f, j)	(h, i)	(h, a)	(a, f)	(a, e)	(a, b)
$\|\varphi(u) - \varphi(v)\|_n$	3	4	4	1	2	3	4
$\|\varphi(u) - \varphi(v)\|_n$	4	3	3	1	2	4	3
Change	+1	-1	-1	0	0	+1	-1

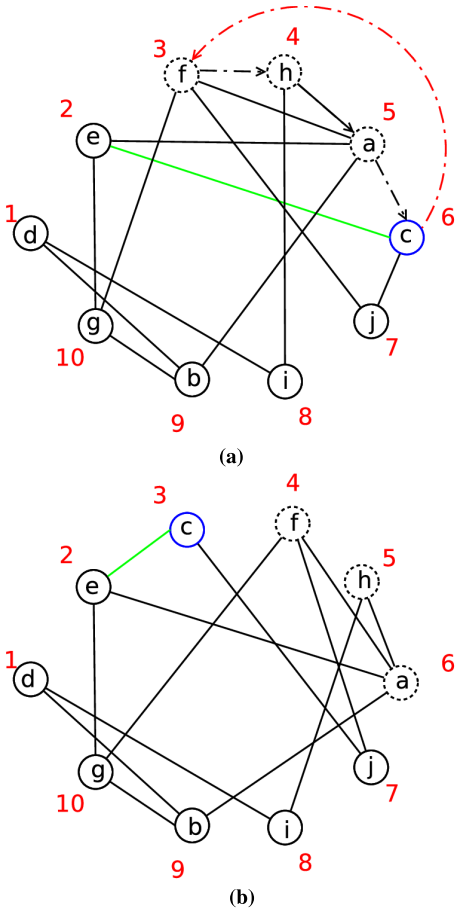

FIGURE 5. An illustrative example of the *Shift-Insert*-based perturbation. (a) Solution φ before applying the *ShiftInsert* perturbation. (b) Solution φ after applying the perturbation *ShiftInsert*(c, e).

A. EXPERIMENTAL SETUP

The ITPS algorithm described in the previous section was coded in the *C++* programming language.[1] We have also the *C* source code of the TScb algorithm.[2] Thus, both algorithms were compiled with *g++* version 4.4.7 using the optimization flag *-O3*. All the experiments presented in this work were run sequentially on the same computational platform with a CPU Intel Xeon X5650 at 2.66 GHz, 2 GB of RAM with Linux operating system. For each benchmark instance a total of 50 independent executions, using different random seeds, of the analyzed algorithms were accomplished due to their stochastic nature.

The test-suite used for the experiments presented in this work is composed of 113 topologically diverse graphs[3] previously tested in the literature [15]. It is divided into two subsets. The first one consists of 85 standard graphs from seven different families (*r-dimensional hypercubes*, *three dimensional meshes*, *complete r level k-ary trees*, *paths, cycles, two dimensional meshes*, and *caterpillars*). These instances have 9 to 8192 vertices and 8 to 53,248 edges. Their optimal values are known, which have been obtained theoretically as indicated in Section 4.3.1. of [15]. One notices that no existing heuristic algorithm is able to attain all the optimal values. The second subset is composed of 28 problem instances, with unknown optimal cost. These instances are from the Harwell-Boeing Sparse Matrix Collection[4] and corresponds to graphs from scientific and engineering practical problems. Most of the graphs in this subset (24 of them) were previously used by Duarte *et al.* [30] and Lozano *et al.* [31] as benchmark instances for the related antibandwidth problem [1] and employed in [15] for the first time as test instances for the cyclic bandwidth problem. The instances in the second subset have a size ranging from 9 to 715 vertices and 46 to 3,720 edges. For a detailed description of this test-suite as well as the current best known results of the benchmark instances, the reader is referred to [15].

For the performance comparison of the analyzed algorithms we employed the criteria commonly used in the literature related with graph embedding algorithms, i.e., the best cyclic bandwidth yielded for each instance (smaller values are better) and the computation time in seconds. Following [15], we applied two other comparison metrics. The first one is the *relative root mean square error* (RMSE), which is computed for each instance t in the test-suite. A smaller RMSE value (≥ 0) indicates a better performance while zero means that the algorithm achieved $\text{Cb}^*(t)$ for each of R runs. To assess the global performance of the studied algorithms, we additionally used the *overall relative root mean square error* (O-RMSE), which averages the RMSE values over the instances of the test-suite.

To analyze the statistical significance of the experimental data produced in this work the following procedure was systematically used. Normality of data distributions was evaluated by using the *Shapiro-Wilk* test. In the case of non-normal data, the nonparametric *Kruskal-Wallis* test was applied.

[1]The source code of our ITPS algorithm is available at: https://github.com/thetopjiji/ITPS

[2]The source code of the TScb algorithm reported in [15] is available at: https://www.tamps.cinvestav.mx/~ertello/cbmp.php

[3]Available at https://www.tamps.cinvestav.mx/~ertello/cbmp.php

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

TABLE 3. Parameters to be tunned with *irace* for the ITPS algorithm.

Parameter	Description	Type	Range/Values
δ	Search depth	Integer	$[1, 10]$
L_d	Maximum non-improving limit	Categorical	$\{5, 10, 20, 50, 100\}$
ρ_{best}	Best neighbor move strategy probability	Real	$[0.00, 1.00]$
L_t	Maximum non-improving limit	Categorical	$\{0.1, 0.5, 0.7, 1.0, 1.5, 2.0, 2.5, 3.0, 4.0\}$
ρ_{N_1}	Neighborhood N_1 application probability	Real	$[0.00, 1.00]$
γ	percentage of vertices employed in neighborhood N_2	Real	$[0.01, 1.00]$

In contrast, when the data follows a normal distribution the homogeneity of the variances across the samples is first verified with the *Bartlett's* test. Then, for homogeneous data the *ANOVA* parametric test is executed, whereas *Welch's t* test is employed in the presence of *heteroskedasticity*. For all these statistical tests a 0.05 significance level was considered.

B. DETERMINATION OF THE INPUT PARAMETER VALUES FOR ITPS

The proposed ITPS algorithm, like most meta-heuristic algorithms, has a number of input parameters. In general, one can tune these parameters on an instance-by-instance basis to identify the best parameter values for each considered problem instance. However, fine-tuning of parameters becomes a tedious task when one wants to solve a large number of instances (like in our case), and moreover, renders it difficult to make fair comparisons with other algorithms. For the purpose of this work, we accomplished the task of tuning parameters of the ITPS algorithm by employing the popular *irace* utility [32], which is one of a number of automatized parameter tuning tools such as ParamILS [33] and GGA++ [34]. This tool uses a (small) training set of instances to determine the most suitable parameter values for the training instances. In our case, we used 20 out of the 113 benchmark instances of Section III-A for the parameter tuning task with *irace* (see below). Finally, we comment that the parameter values obtained by *irace* can be considered to define the default parameter setting of ITPS, though fine-tuning some parameters for a particular instance could enable the algorithm to achieve better results.

There are seven parameters associated with our ITPS algorithm. The first two of them (δ and T_{max}) are directly used by ITPS, while the other five parameters are required by the double neighborhood descent and the responsive threshold-based search procedures (L_d, ρ_{best}, ρ_{N_1}, L_t, and γ). To ensure a fair comparison between our ITPS algorithm and the TScb method, the same cutoff time limit reported in [15] was adopted (i.e., $T_{max} = 600$ seconds). Table 3 presents for each of the six remaining parameters considered in the tuning process its description, its type, and the values provided to configure *irace*.

For our tuning experiment we have selected a subset of 20 graphs from the original test-suite of 113 benchmark instances described in Section III-A. The criteria used to compose this subset was to include large and complex instances covering all graph types present in the original benchmark. We have observed, from our preliminary tuning tests, that the performance of ITPS presented some variations depending on the graph family. For this reason, we have divided the subset of 20 graphs into three groups:

- *path200, path650, path825, path1000, cycle200, cycle300, cycle650, cycle1000, caterpillar29, tree2×9, mesh2D5×25, mesh2D20×50, mesh3D12×12×12*
- *dwt_592, can_715, can_445, 494_bus, 662_bus, 685_bus*
- *hypercube11*

Each group of instances was then used independently for a tuning process. The maximum number of executions (i.e., maximum budget of experiments, *maxExperiments*) of *irace* was fixed to 2,000, where each one of them was limited to 600 seconds as suggested in the CBP literature [15]. The final values returned by these parameter calibration experiments, for each group of instances, are summarized in Table 4.

TABLE 4. Final values found by *irace* after the parameter calibration experiments.

Instance group	δ	L_d	ρ_{best}	L_t	ρ_{N_1}	γ
1	3	50	0.50	1.00	0.03	0.03
2	2	100	0.29	0.10	0.48	0.27
3	3	100	0.10	3.00	0.97	0.03

C. COMPARISON WITH THE STATE-OF-THE-ART ALGORITHM

The comparative experiments presented in this section have as main objective to assess the performance of the proposed ITPS algorithm with respect to TScb [15], which is the current best-performing CBP reference method. These experiments were carried out using the experimental conditions presented in Section III-A, and the parameter setting determined in Section III-B.

The computational results of this experiment are summarized in Table 5 and organized according to the type of the graphs evaluated. Columns 1 and 2 present the graph type and the number of instances of that family. Then, for each compared algorithm and each graph family, we indicate the following average data: the best cyclic bandwidth cost reached (Avg. Cb_{best}), the computation time in seconds needed to

TABLE 5. Summary of the comparison between TScb and ITPS over 113 benchmark instances: 85 standard graphs from 7 different types with known optimal solutions, and 28 Harwell-Boeing instances with unknown optimal cost arising from scientific and engineering practical problems.

		TSCB				ITPS						
Graph type	Num.	Avg. Cb_{best}	Avg. T_{best}	O-RMSE	% Best	Avg. Cb_{best}	Avg. T_{best}	O-RMSE	% Best	I	M	F
path	15	2.53	62.38	1.98	66.67	1.80	148.34	2.78	80.00	4	11	0
cycle	15	2.40	25.15	1.84	73.33	2.47	145.68	4.04	73.33	2	12	1
mesh2D	15	27.73	53.38	1.80	60.00	12.07	76.17	0.43	40.00	3	9	3
mesh3D	10	163.30	177.21	1.46	40.00	139.40	174.73	1.37	70.00	6	4	0
tree	12	55.17	36.39	0.02	91.67	54.67	18.49	0.00	100.00	1	11	0
caterpilar	15	15.20	41.90	0.07	86.67	15.07	67.56	0.07	100.00	2	13	0
hypercube	3	1532.00	497.41	0.34	0.00	1991.67	550.32	0.57	0.00	1	0	2
Harwell-Boeing	28	22.25	97.82	2.64	28.57	21.36	109.64	3.18	28.57	12	15	1
Total	113									31	75	7

reach its best solution (Avg. T_{best}), the overall relative root mean square error (O-RMSE), as well as the percentage of instances for which an algorithm attains the optima (for the standard graphs) or the best-known solutions (for the Harwell-Boeing graphs) (*% Best*). The last three columns list the number of instances for which our ITPS algorithm improved (*I*), matched (*M*) or failed (*F*) to attain the best cyclic bandwidth costs reported by TScb [15]. The detailed instance-by-instance results from this experiment are provided in Tables 9 and 10 listed in the Appendix.

Table 5 shows that for 5 out of the 7 tested families of standard graphs, our ITPS algorithm produced an average best cyclic bandwidth (see column Avg. Cb_{best}) which is considerably lower (better) than that produced by TScb. Two exceptions are the *cycle* graphs and the *hypercubes* for which TScb was able to score a smaller average best cyclic bandwidth than ITPS (2.78% and 30.00% smaller, respectively). As these seven types of graphs have known optimal solutions, it is important to assess if the compared algorithms attain those optimal values. Comparing columns 6 and 10 (% Best) it is easy to see that ITPS found a greater percentage of optimal Cb values than TScb for the following graph types: *paths, three dimensional meshes*, *complete r level k-ary trees*, and *caterpillars*. For the *cycle* graphs, both of the compared algorithms found the same percentage of optimal solutions (73.33%). However, our ITPS algorithm was able to solve to optimality 100% of the *tree* and *caterpillar* graphs.

In contrast, TScb outperformed ITPS in this regard over the *two dimensional meshes*, and both algorithms failed to reach the optimal cost for any of the *r-dimensional hypercubes*; indicating that this type of graphs is still an open challenge for metaheuristic algorithms. The columns listing the O-RMSE values disclose that in average ITPS presents a slightly higher deviation with respect to the known optimal costs than TScb (1.56 vs. 1.27), notwithstanding ITPS showed to be more effective for finding global optimal embeddings. By inspecting the row allocated for the Harwell-Boeing graphs of Table 5, we notice that TScb achieved an average best solution cost (Avg. Cb_{best}) which is 4.18% higher than that produced by ITPS (22.25 vs. 21.36). Even though the two compared algorithms attained the same number of theoretical lower bounds (i.e., % Best equals 28.57%) for this type of graphs, it is clear that TScb is the one providing the smallest O-RMSE value (2.64 vs. 3.18), showing in average a more stable behavior.

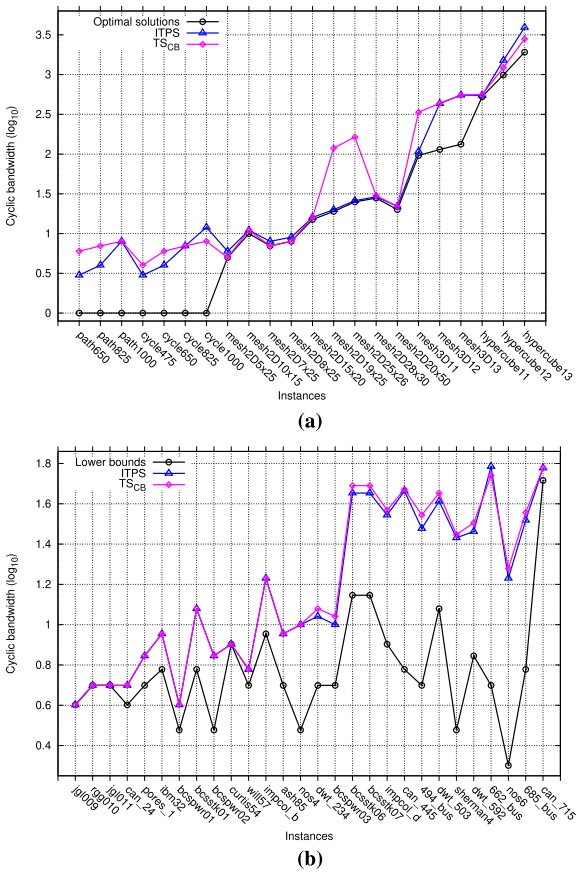

FIGURE 6. Performance evaluation of the best solutions found by the algorithms TScb and ITPS, over a standard test-suite of graphs. (a) Graphs with regular topologies, with respect to the known optimal solutions; the plot includes only the 22 instances whose optimal solutions were not reached by neither of the compared algorithms. (b) Harwell-Boeing instances with unknown optimal cost, with respect to the theoretical lower bounds proposed by Lin [8].

TABLE 6. Performance comparison between *ITPS* and *ITPS$_{nofe}$* over 20 selected graphs.

				ITPS$_{nofe}$					ITPS						
Graph	$\|V\|$	$\|E\|$	Cb*	Cb$_{best}$	Avg. Cb	Dev.	Avg. T_{best}	RMSE	Cb$_{best}$	Avg. Cb	Dev.	Avg. T_{best}	RMSE	p-value	SS
path200	200	199	1	1	1.06	0.24	132.18	0.24	1	1.00	0.00	74.64	0.00	2.3E-20	+
path650	650	649	1	1	3.62	1.10	321.17	2.84	3	6.00	2.97	473.59	5.80	3.2E-18	−
path825	825	824	1	3	6.18	2.50	476.68	5.74	4	12.86	6.08	532.00	13.30	2.2E-17	−
path1000	1000	999	1	4	11.16	5.68	535.62	11.61	8	20.90	5.40	562.32	20.60	6.6E-18	−
cycle200	200	200	1	1	2.34	1.87	83.85	2.28	1	2.34	1.52	71.53	2.01	6.8E-01	⋆
cycle300	300	300	1	1	3.70	2.06	127.81	3.39	1	3.00	1.95	180.95	2.78	7.2E-19	+
cycle650	650	650	1	3	6.22	2.57	323.75	5.81	4	7.50	2.59	469.61	6.99	2.6E-18	−
cycle1000	1000	1000	1	5	14.76	5.29	560.68	14.72	12	24.32	7.67	541.70	24.53	1.2E-17	−
mesh2D5x25	125	220	5	6	6.00	0.00	4.61	0.20	6	6.00	0.00	0.90	0.20	1.8E-20	−
mesh2D20x50	1000	1930	20	23	40.58	11.95	496.91	1.19	22	38.58	54.32	375.06	2.84	4.4E-16	+
mesh3D12	1728	4752	114	435	437.06	1.20	484.29	2.83	108	325.04	0.49	411.07	2.80	1.4E-18	+
tree2x9	1023	1022	57	58	60.94	1.92	429.94	0.08	57	57.34	0.48	215.84	0.01	1.2E-20	+
caterpillar29	494	463	24	24	24.98	2.22	114.65	0.10	24	24.00	0.00	43.14	0.00	9.5E-09	+
hypercube11	2048	11264	526	907	923.84	6.57	593.52	0.76	548	561.46	7.86	457.93	0.07	4.3E-18	+
can_445	445	1682	6	46	64.76	11.46	178.11	9.97	46	59.72	7.63	313.35	9.04	2.7E-20	+
494_bus	494	586	5	54	65.52	4.32	219.85	12.13	30	41.94	6.23	271.86	7.49	9.2E-19	+
dwt_592	592	2256	7	30	32.28	4.14	326.00	3.66	29	36.00	23.80	405.82	5.34	5.7E-17	−
662_bus	662	906	5	95	107.94	6.01	174.85	20.62	61	72.30	4.98	336.13	13.50	3.3E-18	+
685_bus	685	1282	6	99	116.02	4.10	212.21	18.35	33	72.68	12.88	343.03	11.31	5.3E-18	+
can_715	715	2975	52	61	109.22	20.00	110.99	1.16	60	168.12	74.02	231.48	2.64	1.0E-13	−
Average				92.85	101.91	4.76	295.38	5.88	52.90	77.06	11.04	315.60	6.56		
														⋆	1
													Total	+	11
														−	8

From the data generated in this experiment, it is thus possible to conclude that ITPS is certainly a very competitive approach, with respect to the state-of-the-art algorithm TScb, for solving the CBP in the case of graphs with standard topologies, and those coming from practical scientific and engineering problems. In fact, ITPS was able to establish new lower bounds for 31 instances, and to equal the best solution cost reached by TScb for other 75 graphs (see Figure 6). For the remaining 7 instances (6.19%) TScb still offers the best-known results.

Finally, the statistical analysis carried out for this experiment, and presented in the last two columns of Tables 9 and 10, allows us to verify that a statistically significant performance amelioration was achieved by ITPS with respect to TScb on 37 instances (32.74% of the graphs). Nevertheless, ITPS was significantly surpassed by TScb in 24 instances (21.24%). For the remaining 52 graphs (46.02%), a significant difference between the two compared methods could not be concluded. Furthermore, the excellent performance of ITPS was attained by consuming only a slightly higher CPU time than that expended by TScb (in average 161.37 vs. 123.95 seconds), which could be justified by the good final embeddings produced.

IV. ANALYSIS

We present additional experiments to investigate the key components of the ITPS algorithm: a) the extended evaluation function (f_e) of Section II-B, b) the responsive threshold-based search (RTBS) method of Section II-D, and c) the *Shift-Insert*-based perturbation mechanism of Section II-E. For these experiments, we adopted the same subset of 20 representative graphs (14 standard topology graphs and 6 Harwell-Boeing graphs) that were used for parameter tuning in Section III-B.

A. INFLUENCE OF THE EXTENDED EVALUATION FUNCTION

As we pointed out in Section II-B, the objective function of the CBP is unable to establish preferences among different potential embeddings with the same cyclic bandwidth cost. This function could leads to large plateaus in the fitness landscape [35], [36], on which identifying a promising search direction may become difficult for local search methods [37], [38]. This problem could seriously compromise the search efficiency of the search algorithm. The extended evaluation function (f_e) proposed in this work was designed to cope with this delicate problem. To evaluate its impact on the ITPS global performance, we provide a comparison of ITPS with a ITPS variant, named ITPS$_{nofe}$, which only employs the conventional evaluation function of the CBP. Table 6 summarizes the computational results of this comparison. Columns 1 to 3 present for each instance the name, the number of vertices ($|V|$) and edges ($|E|$). For the first

TABLE 7. Performance comparison between *ITPS* and *ITPS*$_{noth}$ over 20 selected graphs.

				ITPS$_{noth}$					ITPS						
Graph	$\|V\|$	$\|E\|$	Cb*	Cb$_{best}$	Avg. Cb	Dev.	Avg. T_{best}	RMSE	Cb$_{best}$	Avg. Cb	Dev.	Avg. T_{best}	RMSE	p-value	SS
path200	200	199	1	1	1.40	0.81	109.81	0.89	1	1.00	0.00	74.64	0.00	2.2E-20	+
path650	650	649	1	5	11.06	2.42	522.89	10.34	3	6.00	2.97	473.59	5.80	3.7E-18	+
path825	825	824	1	10	18.30	3.75	553.66	17.69	4	12.86	6.08	532.00	13.30	3.1E-01	⋆
path1000	1000	999	1	19	28.28	4.13	572.21	27.58	8	20.90	5.40	562.32	20.60	5.1E-18	+
cycle200	200	200	1	1	2.34	1.45	110.52	1.96	1	2.34	1.52	71.53	2.01	9.3E-01	⋆
cycle300	300	300	1	1	3.24	1.36	254.60	2.62	1	3.00	1.95	180.95	2.78	1.6E-18	+
cycle650	650	650	1	6	11.90	2.32	473.54	11.14	4	7.50	2.59	469.61	6.99	5.4E-18	+
cycle1000	1000	1000	1	19	29.14	4.65	567.59	28.51	12	24.32	7.67	541.70	24.53	1.0E-17	+
mesh2D5x25	125	220	5	6	6.00	0.00	14.74	0.20	6	6.00	0.00	0.90	0.20	6.5E-23	−
mesh2D20x50	1000	1930	20	21	43.84	11.90	552.03	1.33	22	38.58	54.32	375.06	2.84	2.6E-05	+
mesh3D12	1728	4752	114	433	433.04	0.20	427.26	2.80	108	325.04	0.49	411.07	2.80	1.6E-18	+
tree2x9	1023	1022	57	57	57.24	0.74	251.39	0.01	57	57.34	0.48	215.84	0.01	8.0E-20	−
caterpillar29	494	463	24	24	27.82	4.78	248.55	0.25	24	24.00	0.00	43.14	0.00	5.2E-13	+
hypercube11	2048	11264	526	662	684.02	5.07	567.99	0.30	548	561.46	7.86	457.93	0.07	4.2E-18	+
can_445	445	1682	6	46	63.48	11.07	147.87	9.75	46	59.72	7.63	313.35	9.04	2.7E-20	+
494_bus	494	586	5	30	57.56	10.17	150.06	10.70	30	41.94	6.23	271.86	7.49	7.8E-19	+
dwt_592	592	2256	7	29	34.12	6.22	397.57	3.97	29	36.00	23.80	405.82	5.34	4.7E-07	−
662_bus	662	906	5	57	90.76	8.63	220.13	17.24	61	72.30	4.98	336.13	13.50	4.0E-18	+
685_bus	685	1282	6	69	96.40	8.48	260.64	15.13	33	72.68	12.88	343.03	11.31	1.2E-17	+
can_715	715	2975	52	60	116.56	31.42	281.04	1.38	60	168.12	74.02	231.48	2.64	1.0E-19	−
Average				77.80	90.83	5.98	334.20	8.19	52.90	77.06	11.04	315.60	6.56		
														⋆	2
													Total	+	14
														−	4

14 instances, column 4 reports the known optimal costs, whereas for the 6 remaining graphs the theoretical lower bounds are listed (Cb*). Next, for each compared algorithm five columns are used to show: the best (Cb$_{best}$), the average (Avg. Cb) and standard deviation (Dev.) of the cyclic bandwidth cost reached over 50 independent executions, the average CPU time in seconds needed for attaining their best solutions (Avg. T_{best}), and the relative root mean square error (RMSE) with respect to the best-known solutions (Cb*) indicated in column 4. The last two columns provide the results of a statistical significance analysis which was executed with the method described in Section III-A over this experimental data. The obtained p-value is presented in column 15. Cells in column 16 (SS) are marked + if a statistically significant difference in favor of ITPS is found over ITPS$_{nofe}$, or − if this difference is against ITPS. Those cells with the ⋆ symbol indicate that no significant difference can be detected between the analyzed algorithms for the corresponding benchmark instance.

By observing the average data presented at the bottom of Table 6, it is possible to identify that the ITPS$_{nofe}$ algorithm, using only the conventional evaluation function, achieved worse values for both the best and average cyclic bandwidth costs (columns Cb$_{best}$ and Avg. Cb) than those of ITPS (92.85 vs. 52.90 and 101.91 vs. 77.06). On the one hand, this confirms the weak discrimination capacity furnished by the conventional evaluation function. On the other hand, it discloses the positive influence of the f_e function in the global performance of ITPS, when it is employed for assessing the quality of the visited potential solutions. The results of our statistical significance analysis indicate that ITPS significantly outperformed ITPS$_{nofe}$ on 11 instances. However, ITPS$_{nofe}$ significantly surpassed ITPS in 8 graphs. It is interesting to remark that 6 of these graphs are paths and cycles of order $n \geq 650$. This suggests that the proposed f_e function has some trouble in discriminating potential solutions for graphs with these specific topologies, but further studies are needed to gain understanding on this behavior.

B. INFLUENCE OF THE RESPONSIVE THRESHOLD-BASED SEARCH

In our ITPS algorithm, the first phase employs a double neighborhood decent search procedure (DNDS) to explore embeddings of increasing quality until a local optimal solution is reached. To escape from the basin of attraction [36], ITPS triggers a second phase using a responsive threshold-based search (RTBS), which accepts neighboring solutions that are not worse than the incumbent solution by more than a given threshold (uphill moves). In this section we evaluate the effect of applying RTBS on the final outcome produced by our iterated three-phase search algorithm. To this end, we completely removed the responsive threshold-based search from our ITPS algorithm; and compared experimentally this algorithmic variant, called ITPS$_{noth}$, with respect to

TABLE 8. Performance comparison between *ITPS* and *ITPS$_{nosi}$* over 20 selected graphs.

				ITPS$_{nosi}$					ITPS						
Graph	$\|V\|$	$\|E\|$	Cb*	Cb$_{best}$	Avg. Cb	Dev.	Avg. T_{best}	RMSE	Cb$_{best}$	Avg. Cb	Dev.	Avg. T_{best}	RMSE	p-value	SS
path200	200	199	1	1	1.02	0.14	82.20	0.14	1	1.00	0.00	74.64	0.00	2.6E-20	+
path650	650	649	1	3	6.26	2.72	504.29	5.91	3	6.00	2.97	473.59	5.80	2.4E-18	+
path825	825	824	1	5	13.38	6.43	521.80	13.92	4	12.86	6.08	532.00	13.30	7.1E-11	+
path1000	1000	999	1	11	21.72	5.56	568.06	21.44	8	20.90	5.40	562.32	20.60	1.2E-143	+
cycle200	200	200	1	1	3.08	1.68	28.25	2.66	1	2.34	1.52	71.53	2.01	2.5E-02	+
cycle300	300	300	1	1	3.48	2.04	140.50	3.20	1	3.00	1.95	180.95	2.78	2.6E-18	+
cycle650	650	650	1	3	8.06	3.01	476.23	7.66	4	7.50	2.59	469.61	6.99	2.8E-20	+
cycle1000	1000	1000	1	8	25.54	8.05	550.77	25.80	12	24.32	7.67	541.70	24.53	3.2E-20	+
mesh2D5x25	125	220	5	6	6.00	0.00	12.64	0.20	6	6.00	0.00	0.90	0.20	7.2E-21	−
mesh2D20x50	1000	1930	20	22	32.56	34.38	384.03	1.81	22	38.58	54.32	375.06	2.84	7.0E-03	−
mesh3D12	1728	4752	114	433	433.52	0.54	387.14	2.80	108	325.04	0.49	411.07	2.80	1.6E-18	+
tree2x9	1023	1022	57	57	57.30	0.46	204.22	0.01	57	57.34	0.48	215.84	0.01	5.8E-21	−
caterpillar29	494	463	24	24	24.00	0.00	70.27	0.00	24	24.00	0.00	43.14	0.00	1.4E-06	−
hypercube11	2048	11264	526	548	564.72	7.88	490.95	0.08	548	561.46	7.86	457.93	0.07	5.2E-18	+
can_445	445	1682	6	149	149.00	0.00	0.74	23.83	46	59.72	7.63	313.35	9.04	2.7E-20	+
494_bus	494	586	5	46	54.74	4.50	268.50	9.99	30	41.94	6.23	271.86	7.49	3.8E-20	+
dwt_592	592	2256	7	198	198.00	0.00	1.87	27.29	29	36.00	23.80	405.82	5.34	2.1E-20	+
662_bus	662	906	5	75	90.60	5.86	284.35	17.16	61	72.30	4.98	336.13	13.50	4.1E-18	+
685_bus	685	1282	6	77	106.54	10.00	200.23	16.84	33	72.68	12.88	343.03	11.31	1.4E-17	+
can_715	715	2975	52	61	235.44	25.17	30.68	3.56	60	168.12	74.02	231.48	2.64	5.3E-19	+
Average				86.45	101.75	5.92	260.39	9.22	52.90	77.06	11.04	315.60	6.56		
														⋆	0
													Total	+	16
														−	4

the full ITPS method. Table 7 presents the results from this comparison, using the same column organization previously described for Table 6.

It is clear, from Table 7, that the inclusion of the responsive threshold-based search in the second phase of the ITPS algorithm enables ITPS to obtain for 15 out of 20 instances better average final results (smaller values in column Avg. Cb) than those produced by the ITPS$_{noth}$ approach, resulting in a smaller O-RMSE value (6.56 vs. 8.59). For some of the analyzed instances (e.g., *mesh3D12* and *hypercube11*), ITPS is even able to attain improvements in the final Cb cost of two orders of magnitude with respect to that produced by ITPS$_{noth}$. Furthermore, from our statistical significance analysis one observes that ITPS, including the RTBS phase, significantly outperformed ITPS$_{noth}$ in 14 instances. It scored significantly worse results than ITPS$_{noth}$ in only 4 graphs. For the *path825* instance, no statistically significant difference is observed between the compared algorithms.

C. INFLUENCE OF THE SHIFT-INSERT-BASED PERTURBATION

After the conclusion of the second phase in our ITPS algorithm, a shift-insert perturbation is applied to the incumbent solution in order to move out search to a distant new region of the search space. As in the two previous sections, we assess the impact of using this shift-insert perturbation on the cost of the final solutions produced by our ITPS algorithm. We prepared an algorithm, named ITPS$_{nosi}$, which excludes the shift-insert perturbation phase. It was then contrasted experimentally against the complete ITPS algorithm. The data produced in this experiment is shown in Table 8, which has the same column headings defined for Table 6.

As shown in Table 8, the algorithm that removed the shift-insert perturbation phase (ITPS$_{nosi}$) was significantly outperformed by the full ITPS version in 16 instances, leading in average to a higher O-RMSE value (9.22 vs. 6.56). It indicates that ITPS$_{nosi}$ presented in average a much higher deviation with respect to the best-known solutions. These observations provide a solid confirmation of the usefulness of applying the third phase, based on the shift-insert perturbation, within our ITPS algorithm.

V. CONCLUSIONS AND FUTURE WORK

Cyclic bandwidth minimization in graphs is a relevant model with a number of significant applications. Given its computational complexity, it is quite challenging to devise solution methods able to solve the problem effectively. In this paper, we have presented an iterated three-phase search algorithm (ITPS) for the problem. The algorithm originally integrates a double neighbor-decent phase, a threshold-based search phase and a special perturbation phase, which are guided by an enriched evaluation function. These different algorithmic components play complementary roles in terms of search intensification and diversification and together ensure a highly effective examination of the search space. This algorithm enriches the solution methods for the cyclic bandwidth problem, which currently remain scarce.

TABLE 9. Detailed performance assessment of the TScb and ITPS algorithms over 85 standard graphs from 7 different families all of them having tight lower bounds.

				TSCB					ITPS						
Graph	$\|V\|$	$\|E\|$	Cb*	Cb_{best}	Avg. Cb	Dev.	Avg. T_{best}	D	Cb_{best}	Avg. Cb	Dev.	Avg. T_{best}	D	p-value	SS
path20	20	19	1	1	1.00	0.00	0.17	0	1	1.00	0.00	0.03	0	1.0E+00	⋆
path25	25	24	1	1	1.00	0.00	0.44	0	1	1.00	0.00	0.08	0	1.0E+00	⋆
path30	30	29	1	1	1.00	0.00	1.25	0	1	1.00	0.00	0.12	0	1.0E+00	⋆
path35	35	34	1	1	1.00	0.00	2.93	0	1	1.00	0.00	0.23	0	1.0E+00	⋆
path40	40	39	1	1	1.00	0.00	4.05	0	1	1.00	0.00	0.28	0	1.0E+00	⋆
path100	100	99	1	1	1.00	0.00	59.82	0	1	1.00	0.00	4.55	0	1.0E+00	⋆
path125	125	124	1	1	1.00	0.00	148.25	0	1	1.00	0.00	6.98	0	1.0E+00	⋆
path150	150	149	1	1	1.34	0.48	190.91	0	1	1.00	0.00	20.55	0	6.7E-06	+
path175	175	174	1	1	1.64	0.48	123.52	0	1	1.00	0.00	38.62	0	8.8E-12	+
path200	200	199	1	1	1.94	0.24	28.63	0	1	1.00	0.00	74.64	0	7.3E-21	+
path300	300	299	1	2	2.96	0.35	46.02	1	1	1.04	0.20	180.24	0	9.8E-22	+
path475	475	474	1	5	5.56	0.50	56.86	4	1	2.34	1.24	330.87	0	2.2E-18	+
path650	650	649	1	6	6.98	0.14	86.92	5	3	6.00	2.97	473.59	2	6.0E-07	+
path825	825	824	1	7	7.92	0.34	66.25	6	4	12.86	6.08	532.00	3	4.5E-10	−
path1000	1000	999	1	8	8.84	0.47	119.71	7	8	20.90	5.40	562.32	7	1.2E-17	−
cycle20	20	20	1	1	1.00	0.00	0.32	0	1	1.00	0.00	0.02	0	1.0E+00	⋆
cycle25	25	25	1	1	1.00	0.00	0.86	0	1	1.00	0.00	0.05	0	1.0E+00	⋆
cycle30	30	30	1	1	1.00	0.00	0.36	0	1	1.00	0.00	0.11	0	1.0E+00	⋆
cycle35	35	35	1	1	1.00	0.00	0.67	0	1	1.00	0.00	0.20	0	1.0E+00	⋆
cycle40	40	40	1	1	1.00	0.00	0.67	0	1	1.00	0.00	0.21	0	1.0E+00	⋆
cycle100	100	100	1	1	1.00	0.00	3.52	0	1	1.34	0.80	25.85	0	3.4E-03	−
cycle125	125	125	1	1	1.00	0.00	4.63	0	1	1.46	1.03	18.36	0	1.8E-03	−
cycle150	150	150	1	1	1.00	0.00	7.86	0	1	1.86	1.28	49.58	0	7.4E-06	−
cycle175	175	175	1	1	1.00	0.00	9.14	0	1	2.44	1.66	62.03	0	3.2E-08	−
cycle200	200	200	1	1	1.00	0.00	21.39	0	1	2.34	1.52	71.53	0	1.3E-08	−
cycle300	300	300	1	1	2.86	0.57	23.82	0	1	3.00	1.95	180.95	0	8.2E-01	⋆
cycle475	475	475	1	4	5.52	0.58	70.24	3	3	5.28	2.71	236.98	2	8.6E-03	+
cycle650	650	650	1	6	7.12	0.56	61.02	5	4	7.50	2.59	469.61	3	9.4E-01	⋆
cycle825	825	825	1	7	8.00	0.40	65.70	6	7	13.72	4.56	528.06	6	3.4E-14	−
cycle1000	1000	1000	1	8	8.88	0.59	107.05	7	12	24.32	7.67	541.70	11	1.2E-18	−
mesh2D5x4	20	31	4	4	4.00	0.00	2.29	0	4	4.00	0.00	0.04	0	1.0E+00	⋆
mesh2D5x5	25	40	5	5	5.00	0.00	2.86	0	5	5.00	0.00	0.02	0	1.0E+00	⋆
mesh2D5x6	30	49	5	5	5.00	0.00	0.86	0	5	5.00	0.00	0.07	0	1.0E+00	⋆
mesh2D5x7	35	58	5	5	5.00	0.00	1.49	0	5	5.00	0.00	0.09	0	1.0E+00	⋆
mesh2D5x8	40	67	5	5	5.00	0.00	1.48	0	5	5.00	0.00	32.58	0	1.0E+00	⋆
mesh2D10x10	100	180	10	10	10.58	0.50	58.15	0	10	10.76	0.43	37.75	0	5.7E-02	⋆
mesh2D5x25	125	220	5	5	5.00	0.00	13.00	0	6	6.00	0.00	0.90	1	2.5E-23	−
mesh2D10x15	150	275	10	11	11.00	0.00	12.02	1	11	11.00	0.00	2.80	1	1.0E+00	⋆
mesh2D7x25	175	318	7	7	7.02	0.14	73.44	0	8	8.00	0.00	4.19	1	1.8E-22	−
mesh2D8x25	200	367	8	8	8.10	0.30	73.37	0	9	9.00	0.00	7.16	1	2.3E-19	−
mesh2D15x20	300	565	15	16	19.66	14.13	117.35	1	16	16.56	0.50	109.68	1	4.0E-04	+
mesh2D19x25	475	906	19	119	119.82	0.39	55.34	100	20	20.92	0.27	31.77	1	3.7E-21	+
mesh2D25x26	650	1249	25	164	164.00	0.00	15.22	139	26	27.22	3.33	239.91	1	4.4E-21	+
mesh2D28x30	840	1622	28	30	142.34	87.31	194.10	2	29	59.76	66.36	300.47	1	3.1E-08	+
mesh2D20x50	1000	1930	20	22	187.06	102.01	179.68	2	22	38.58	54.32	375.06	2	5.3E-09	+
mesh3D4	64	300	14	14	15.70	0.68	47.01	0	14	14.00	0.00	12.30	0	2.7E-18	+
mesh3D5	125	540	21	21	22.76	3.14	111.95	0	21	21.00	0.00	41.29	0	1.5E-14	+
mesh3D6	216	882	30	30	32.34	5.71	84.38	0	30	30.00	0.00	23.21	0	1.1E-19	+
mesh3D7	343	1344	40	40	45.26	12.47	191.09	0	40	55.14	21.44	239.37	0	1.4E-02	−
mesh3D8	512	1344	52	53	114.38	30.21	190.27	1	52	101.32	37.04	107.42	0	2.0E-05	+
mesh3D9	729	1944	65	68	182.44	16.52	150.34	3	65	157.40	48.71	57.20	0	7.1E-19	+
mesh3D10	1000	2700	80	83	249.60	24.05	212.68	3	80	214.02	70.19	155.73	0	3.6E-18	+
mesh3D11	1331	3630	96	336	336.54	0.50	213.90	240	108	325.04	43.33	218.14	12	1.9E-19	+
mesh3D12	1728	4752	114	435	436.26	0.56	252.79	321	433	433.40	0.49	411.07	319	4.3E-19	+
mesh3D13	2197	6084	133	553	554.40	0.67	317.70	420	551	552.68	1.00	481.57	418	1.4E-13	+
tree2x4	31	30	4	4	4.00	0.00	0.86	0	4	4.00	0.00	0.00	0	1.0E+00	⋆
tree3x3	40	39	7	7	7.00	0.00	0.37	0	7	7.00	0.00	0.00	0	1.0E+00	⋆
tree10x2	111	110	28	28	28.00	0.00	0.23	0	28	28.00	0.00	0.00	0	1.0E+00	⋆
tree3x4	121	120	15	15	15.76	0.43	18.60	0	15	15.00	0.00	0.44	0	6.7E-15	+

TABLE 9. *(Continued.)* **Detailed performance assessment of the TScb and ITPS algorithms over 85 standard graphs from 7 different families all of them having tight lower bounds.**

				TSCB					ITPS						
Graph	$\|V\|$	$\|E\|$	Cb*	Cb_{best}	Avg. Cb	Dev.	Avg. T_{best}	D	Cb_{best}	Avg. Cb	Dev.	Avg. T_{best}	D	p-value	SS
tree5x3	156	155	26	26	26.00	0.00	11.83	0	26	26.00	0.00	0.04	0	1.0E+00	⋆
tree13x2	183	182	46	46	46.00	0.00	0.33	0	46	46.00	0.00	0.01	0	1.0E+00	⋆
tree2x7	255	254	19	19	20.12	0.39	75.01	0	19	19.00	0.00	0.83	0	2.5E-21	+
tree17x2	307	306	77	77	77.00	0.00	0.50	0	77	77.00	0.00	0.06	0	1.0E+00	⋆
tree21x2	463	462	116	116	116.00	0.00	0.87	0	116	116.00	0.00	0.18	0	1.0E+00	⋆
tree25x2	651	650	163	163	163.00	0.00	1.02	0	163	163.00	0.00	0.49	0	1.0E+00	⋆
tree5x4	781	780	98	98	98.28	0.45	134.45	0	98	98.00	0.00	3.95	0	6.0E-05	+
tree2x9	1023	1022	57	63	64.30	0.79	192.58	6	57	57.34	0.48	215.84	0	6.9E-19	+
caterpillar3	9	8	3	3	3.00	0.00	0.00	0	3	3.00	0.00	0.00	0	1.0E+00	⋆
caterpillar4	14	13	3	3	3.00	0.00	0.42	0	3	3.00	0.00	0.00	0	1.0E+00	⋆
caterpillar5	20	19	4	4	4.00	0.00	0.49	0	4	4.00	0.00	0.00	0	1.0E+00	⋆
caterpillar6	27	26	5	5	5.00	0.00	0.58	0	5	5.00	0.00	0.00	0	1.0E+00	⋆
caterpillar7	35	34	6	6	6.00	0.00	0.51	0	6	6.00	0.00	0.00	0	1.0E+00	⋆
caterpillar13	104	103	10	10	10.00	0.00	16.33	0	10	10.00	0.00	0.31	0	1.0E+00	⋆
caterpillar14	119	118	11	11	11.00	0.00	11.15	0	11	11.00	0.00	0.11	0	1.0E+00	⋆
caterpillar16	152	151	13	13	13.00	0.00	10.28	0	13	13.00	0.00	0.34	0	1.0E+00	⋆
caterpillar17	170	169	14	14	14.00	0.00	21.29	0	14	14.00	0.00	0.56	0	1.0E+00	⋆
caterpillar19	209	208	15	15	15.68	0.47	41.71	0	15	15.00	0.00	2.76	0	9.2E-13	+
caterpillar23	299	298	19	19	19.32	0.47	54.77	0	19	19.00	0.00	5.97	0	1.4E-05	+
caterpillar29	464	463	24	24	25.64	1.75	79.02	0	24	24.00	0.00	43.14	0	4.4E-13	+
caterpillar35	665	664	29	29	34.46	3.88	154.46	0	29	32.52	5.87	281.87	0	2.0E-05	+
caterpillar39	819	818	33	34	40.08	4.19	121.96	1	33	39.24	8.60	275.49	0	6.9E-03	+
caterpillar44	1034	1033	37	38	49.98	5.59	115.59	1	37	54.10	11.19	402.83	0	1.4E-02	−
hypercube11	2048	11264	526	562	584.64	10.92	472.22	36	548	561.46	7.86	457.93	22	1.1E-20	+
hypercube12	4096	24576	988	1224	1351.12	42.00	503.62	236	1508	1546.32	12.88	596.02	520	6.5E-18	−
hypercube13	8192	53248	1912	2810	2916.70	40.69	516.38	898	3919	3952.02	11.50	597.00	2007	5.1E-79	−
Average				89.52	100.55	4.91	75.80	28.88	100	108.54	5.26	119.84	39.32		
														⋆	38
													Total	+	31
														−	16

We have assessed the proposed algorithm on two groups of 113 benchmark graphs from the literature including 85 standard graphs (e.g., paths, cycles, caterpillars, etc) and 28 Harwell-Boeing graphs which arise from diverse engineering and scientific real-world problems. The computational results are compared with those provided by the best reference algorithm in the literature, showing a very competitive performance. For the 85 standard graphs, the proposed algorithm is able to improve on the best computational results of the reference algorithm for 19 graphs, while matching the best computational results for 60 instances. For the 28 Harwell-Boeing graphs, the proposed algorithm discovers new record results (updated upper bounds) for 12 graphs, while matching the best-known results for 15 instances.

We have performed additional experiments to shed light on the roles of the key composing ingredients of the algorithm including: the extended evaluation function, the threshold-based search and the shift-insert-based perturbation strategy. We have shown that these components contribute positively to the performance of the algorithm.

For future work, it would be useful to study additional search strategies to better solve problem instances for which the proposed algorithm does not perform well (e.g., two dimensional meshes and r-dimensional hypercubes). It would also be useful to investigate hybrid approaches mixing population-based global search and local search. To this end, it is important to identify meaningful "building blocks" in solutions, which can be used to design powerful solution recombination operators. Moreover, it is worth studying other forms of extended evaluation functions to better guide the search process. Finally, there are a number of effective algorithms for other related bandwidth problems. It would be of great interest to study these algorithms with respect to the cyclic bandwidth problem and investigate the possibilities of adapting their key search strategies to design effective algorithms for the cyclic bandwidth problem.

APPENDIX
DETAILED COMPARISON OF THE ITPS AND TScb ALGORITHMS

In this appendix we show detailed results of the proposed ITPS algorithm with respect to the reference TScb method [15] on the two groups of 113 benchmark instances. The results for the group of 85 standard graphs with known optima are presented in Table 9, whereas the results for the group of 28 graphs from real-world applications with unknown optima are listed in Table 10. Columns 1-3 in these tables indicate the graph name, its order ($|V|$) and size ($|E|$). The known optimal values (Cb*) or the theoretical lower (L_B) and upper (U_B) bounds are then listed. The remaining columns show the best (Cb_{best}), average (Avg. Cb) and standard deviation (Dev.) of the cyclic bandwidth cost

TABLE 10. Detailed performance comparison of the TScb and ITPS algorithms over 28 Harwell-Boeing graphs.

			Bounds			TSCB					ITPS						
Graph	$\|V\|$	$\|E\|$	L_B	U_B	Cb*	Cb_{best}	Avg. Cb	Dev.	Avg. T_{best}	D	Cb_{best}	Avg. Cb	Dev.	Avg. T_{best}	D	p-value	SS
jgl009	9	50	4	4	4	4	4.00	0.00	0.00	0	4	4.00	0.00	0.00	0	1.0E+00	⋆
rgg010	10	76	5	5	5	5	5.00	0.00	0.00	0	5	5.00	0.00	0.00	0	1.0E+00	⋆
jgl011	11	76	5	5	5	5	5.00	0.00	0.00	0	5	5.00	0.00	0.00	0	1.0E+00	⋆
can_24	24	92	4	12	5	5	5.00	0.00	0.02	0	5	5.00	0.00	0.47	0	1.0E+00	⋆
pores_1	30	103	5	15	7	7	7.00	0.00	0.15	0	7	7.00	0.00	0.01	0	1.0E+00	⋆
ibm32	32	90	6	16	9	9	9.00	0.00	0.03	0	9	9.00	0.00	0.02	0	1.0E+00	⋆
bcspwr01	39	46	3	19	4	4	4.10	0.30	167.59	0	4	4.00	0.00	2.90	0	2.2E-02	+
bcsstk01	48	176	6	24		12	12.00	0.00	0.03	6	12	12.00	0.00	0.11	6	1.0E+00	⋆
bcspwr02	49	59	3	24		7	7.00	0.00	0.00	4	7	7.00	0.00	0.03	4	1.0E+00	⋆
curtis54	54	124	8	27		8	8.00	0.00	0.55	0	8	8.00	0.00	0.43	0	1.0E+00	⋆
will57	57	127	5	28		6	6.00	0.00	12.80	1	6	6.00	0.00	0.21	1	1.0E+00	⋆
impcol_b	59	281	9	29		17	17.00	0.00	0.47	8	17	17.00	0.00	0.05	8	1.0E+00	⋆
ash85	85	219	5	42		9	9.00	0.00	50.30	4	9	9.00	0.00	0.39	4	1.0E+00	⋆
nos4	100	247	3	50		10	10.00	0.00	0.69	7	10	10.00	0.00	0.41	7	1.0E+00	⋆
dwt_234	117	162	5	58		12	12.00	0.00	19.22	7	11	11.00	0.00	8.74	6	2.5E-23	+
bcspwr03	118	179	5	59		11	11.00	0.00	12.50	6	10	10.00	0.00	2.24	5	2.5E-23	+
bcsstk06	420	3720	14	210		49	49.72	0.57	198.23	35	45	45.00	0.00	200.49	31	6.3E-21	+
bcsstk07	420	3720	14	210		49	49.72	0.61	201.60	35	45	45.00	0.00	204.72	31	7.8E-21	+
impcol_d	425	1267	8	212		37	38.70	0.51	125.60	29	35	39.70	5.17	177.69	27	2.9E-01	⋆
can_445	445	1682	6	222		47	47.00	0.00	83.01	41	46	59.72	7.63	313.35	40	2.3E-12	−
494_bus	494	586	5	247		35	38.50	1.30	287.74	30	30	41.94	6.23	271.86	25	4.1E-02	−
dwt_503	503	2762	12	251		45	46.50	3.73	234.31	33	41	59.00	9.61	116.37	29	1.7E-06	−
sherman4	546	1341	3	273		28	28.18	0.39	180.88	25	27	27.66	0.48	139.71	24	2.6E-07	+
dwt_592	592	2256	7	296		32	32.52	0.54	186.54	25	29	36.00	23.80	405.82	22	1.2E-05	−
662_bus	662	906	5	331		55	66.38	3.98	255.91	50	61	72.30	4.98	336.13	56	3.7E-08	−
nos6	675	1290	2	337		19	20.48	0.54	222.10	17	17	21.88	6.42	313.31	15	8.5E-05	−
685_bus	685	1282	6	342		36	39.78	3.11	303.12	30	33	72.68	12.88	343.03	27	2.6E-15	−
can_715	715	2975	52	357		60	60.86	0.53	195.56	8	60	168.12	74.02	231.48	8	7.8E-15	−
Average						22.25	23.19	0.58	97.82	14.32	21	29.21	5.40	109.64	13.43		
																⋆	14
															Total	+	6
																−	8

reached by each of the compared methods over 50 independent executions, the average computation time in seconds needed to reach their best solution (Avg. T_{best}), and the difference (D) between its best result (Cb_{best}) and the corresponding best-known bound (either Cb* or L_B). A statistical significance analysis was performed for these experiments by using the procedure detailed in Section III-A and the resulting p-values are presented. If a statistically significant difference exists between the results of ITPS and TScb, the corresponding cells in the last column (SS) are marked either + or − depending on whether such a difference is in favor of ITPS or not. Cells marked with the symbol ⋆ indicate that no significant difference exists between the analyzed algorithms.

ACKNOWLEDGMENT

The authors are grateful to our reviewers for their timely and constructive comments that helped us to improve the presentation of the work. Eduardo Rodriguez-Tello would like to thank the high performance computing resources (Neptuno cluster) and the technical assistance provided by CINVESTAV-Tamaulipas.

REFERENCES

[1] J. Y.-T. Leung, O. Vornberger, and J. D. Witthoff, "On some variants of the bandwidth minimization problem," *SIAM J. Comput.*, vol. 13, no. 3, pp. 650–667, Jul. 1984.

[2] Y. Lin, "The cyclic bandwidth problem," *J. Syst. Sci. Complex.*, vol. 7, no. 3, pp. 282–288, 1994.

[3] S. N. Bhatt and F. T. Leighton, "A framework for solving VLSI graph layout problems," *J. Comput. Syst. Sci.*, vol. 28, no. 2, pp. 300–343, 1984.

[4] A. L. Rosenberg and L. Snyder, "Bounds on the costs of data encodings," *Math. Syst. Theory*, vol. 12, no. 1, pp. 9–39, 1978.

[5] F. R. K. Chung, "Labelings of graphs," in *Selected Topics in Graph Theory*, vol. 3, L. W. Beineke and R. J. Wilson, Eds. New York, NY, USA: Academic, 1988, ch. 7, pp. 151–168.

[6] J. Hromkovič, V. Müller, O. Sýkora, and I. Vrt'o, "On embedding interconnection networks into rings of processors," in *PARLE Parallel Architectures and Languages Europe* (Lecture Notes in Computer Science), vol. 605. Berlin, Germany: Springer-Verlag, 1992, pp. 51–62.

[7] Y. Jinjiang and Z. Sanming, "Optimal labelling of unit interval graphs," *Appl. Math.*, vol. 10, no. 3, pp. 337–344, 1995.

[8] Y. Lin, "Minimum bandwidth problem for embedding graphs in cycles," *Network*, vol. 29, no. 3, pp. 135–140, May 1997.

[9] P. C. B. Lam, W. C. Shiu, and W. H. Chan, "Characterization of graphs with equal bandwidth and cyclic bandwidth," *Discrete Math.*, vol. 242, no. 1, pp. 283–289, 2002.

[10] P. C. B. Lam, W. C. Shiu, and W. H. Chan, "On bandwidth and cyclic bandwidth of graphs," *Ars Combinatoria*, vol. 47, no. 3, pp. 147–152, 1997.

[11] S. Zhou, "Bounding the bandwidths for graphs," *Theor. Comput. Sci.*, vol. 249, no. 2, pp. 357–368, 2000.

[12] W. H. Chan, P. C. B. Lam, and W. C. Shiu, "Cyclic bandwidth with an edge added," *Discrete Appl. Math.*, vol. 156, no. 1, pp. 131–137, 2008.

[13] E. de Klerk, M. E-Nagy, and R. Sotirov, "On semidefinite programming bounds for graph bandwidth," *Optim. Methods Softw.*, vol. 28, pp. 485–500, Nov. 2011.

[14] H. Romero-Monsivais, E. Rodriguez-Tello, and G. Ramírez, *A New Branch and Bound Algorithm For the Cyclic Bandwidth Problem*. (Lecture Notes in Artificial Intelligence), vol. 7630. Berlin, Germany: Springer-Verlag, 2012, pp. 139–150.

[15] E. Rodriguez-Tello, H. Romero-Monsivais, G. Ramirez-Torres, and F. Lardeux, "Tabu search for the cyclic bandwidth problem," *Comput. Oper. Res.*, vol. 57, pp. 17–32, May 2015.
[16] L. H. Harper, "Optimal assignments of numbers to vertices," *J. SIAM*, vol. 12, no. 1, pp. 131–135, 1964.
[17] E. Piñana, I. Plana, V. Campos, and R. Martí, "GRASP and path relinking for the matrix bandwidth minimization," *Eur. J. Oper. Res.*, vol. 153, pp. 200–210, Feb. 2004.
[18] E. Rodriguez-Tello, J.-K. Hao, and J. Torres-Jimenez, "An improved simulated annealing algorithm for bandwidth minimization," *Eur. J. Oper. Res.*, vol. 185, no. 3, pp. 1319–1335, 2008.
[19] N. Mladenovic, D. Urosevic, D. Pérez-Brito, and C. G. García-González, "Variable neighbourhood search for bandwidth reduction," *Eur. J. Oper. Res.*, vol. 200, no. 1, pp. 14–27, 2010.
[20] J. Torres-Jimenez, I. Izquierdo-Marquez, A. Garcia-Robledo, A. Gonzalez-Gomez, J. Bernal, and R. N. Kacker, "A dual representation simulated annealing algorithm for the bandwidth minimization problem on graphs," *Inf. Sci.*, vol. 303, pp. 33–49, May 2015.
[21] Z.-H. Fu and J.-K. Hao, "A three-phase search approach for the quadratic minimum spanning tree problem," *Eng. Appl. Artif. Intell.*, vol. 46, pp. 113–130, Nov. 2015.
[22] Y. Zhou, J.-K. Hao, and A. Goëffon, "A three-phased local search approach for the clique partitioning problem," *J. Combinat. Optim.*, vol. 32, no. 2, pp. 469–491, Aug. 2016.
[23] F. Glover and M. Laguna, *Tabu Search*. Norwell, MA, USA: Kluwer Academic, 1997.
[24] P. Galinier, Z. Boujbel, and M. C. Fernandes, "An efficient memetic algorithm for the graph partitioning problem," *Ann. Oper. Res.*, vol. 191, no. 1, pp. 1–22, Nov. 2011.
[25] Q. Wu and J.-K. Hao, "Memetic search for the max-bisection problem," *Comput. Oper. Res.*, vol. 40, no. 1, pp. 166–179, 2013.
[26] X. Lai and J.-K. Hao, "A tabu search based memetic algorithm for the max-mean dispersion problem," *Comput. Oper. Res.*, vol. 72, pp. 118–127, Aug. 2016.
[27] G. Dueck and T. Scheuer, "Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing," *J. Comput. Phys.*, vol. 90, no. 1, pp. 161–175, Sep. 1990.
[28] G. Dueck, "New optimization heuristics: The great deluge algorithm and the record-to-record travel," *J. Comput. Phys.*, vol. 104, no. 1, pp. 86–92, Jan. 1993.
[29] Y. Chen and J.-K. Hao, "Iterated responsive threshold search for the quadratic multiple knapsack problem," *Ann. Oper. Res.*, vol. 226, no. 1, pp. 101–131, Mar. 2015.
[30] A. Duarte, R. Martí, M. G. C. Resende, and R. M. A. Silva, "GRASP with path relinking heuristics for the antibandwidth problem," *Network*, vol. 58, no. 3, pp. 171–189, Oct. 2011.
[31] M. Lozano, A. Duarte, F. Gortázar, and R. Martí, "Variable neighborhood search with ejection chains for the antibandwidth problem," *J. Heuristics*, vol. 18, no. 6, pp. 919–938, Dec. 2012.
[32] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle, "The irace package: Iterated racing for automatic algorithm configuration," *Oper. Res. Perspect.*, vol. 3, pp. 43–58, Jan. 2016.
[33] F. Hutter, H. H. Hoos, L. B. Kevin, and T. Stützle, "ParamILS: An automatic algorithm configuration framework," *J. Artif. Intell. Res.*, vol. 36, no. 1, pp. 267–306, 2009.
[34] C. Ansótegui, Y. Malitsky, H. Samulowitz, M. Sellmann, and K. Tierney, "Model-based genetic algorithms for algorithm configuration," in *Proc. 24th Int. Joint Conf. Artif. Intell.*, Menlo Park, CA, USA, 2015, pp. 733–739.
[35] P. F. Stadler, "Correlation in landscapes of combinatorial optimization problems," *Europhys. Lett.*, vol. 20, no. 6, pp. 479–482, Nov. 1992.
[36] E. Pitzer and M. Affenzeller, "A comprehensive survey on fitness landscape analysis," in *Recent Advances in Intelligent Engineering Systems* (Studies in Computational Intelligence), vol. 378, J. Fodor, R. Klempous, and C. P. Suárez-Araujo, Eds. Berlin, Germany: Springer-Verlag, 2012, ch. 8, pp. 161–191.
[37] W. Michiels, E. Aarts, and J. Korst, *Theoretical Aspects of Local Search* (Monographs in Theoretical Computer Science. An EATCS Series), 1st ed. Berlin, Germany: Springer-Verlag, 2007.
[38] M. Marmion, C. Dhaenens, L. Jourdan, A. Liefooghe, and S. Verel, *On the Neutrality of Flowshop Scheduling Fitness Landscapes* (Lecture Notes in Computer Science), vol. 6683, Berlin, Germany, 2011, pp. 238–252.

JINTONG REN was born in Pizhou, Jiangsu, China, in 1993. He received the B.S. degree in electrical engineering from Northwestern Polytechnical University, China, in 2015, and the M.S. degree in electronics and embedded systems from the Technological University of Belfort-Montbéliard, France, in 2016. He is currently pursuing the Ph.D. degree in computer science with the University of Angers, France.

His research interests include combinatorial optimization, graph embedding problems, and advanced metaheuristics design.

JIN-KAO HAO was born in Hebei, China, in 1961. He received the B.S. degree in computer science from the National University of Defense Technology, China, in 1982, the M.S. degree in computer science from the National Institute of Applied Sciences, Lyon, France, in 1987, the Ph.D. degree in constraint programming from the University of Franche-Comté, France, in 1991, and the Professorship Diploma (Habilitation à Diriger des Recherches) degree from the University of Science and Technology of Montpellier, France, in 1998.

Since 1999, he has been a Full Professor with the LERIA, Université d'Angers, France. He has authored or coauthored more than 250 peer-reviewed publications and co-edited nine books in Springers LNCS series. His research interests include the design of effective algorithms and intelligent computational methods for solving large-scale combinatorial search problems. He is also interested in various application areas, including data science, complex networks, and transportation.

Dr. Hao has been a Senior Fellow of the Institut Universitaire de France, since 2015. He became a Distinguished Professor (Professeur de classe exceptionnelle), in 2010. He has served as an Invited Member of more than 200 program committees of international conferences and is on the Editorial Board of seven international journals.

EDUARDO RODRIGUEZ-TELLO (M'18) was born in Mexico City, Mexico, in 1973. He received the M.S. degree in computer science from ITESM, Cuernavaca, Mexico, in 1999, and the Ph.D. degree in informatics from the University of Angers, France, in 2007.

Since 2008, he has been an Associate Professor (CINVESTAV-3B Researcher) with the CINVESTAV—Tamaulipas, Ciudad Victoria, Mexico. He has authored or coauthored a book and over 40 technical papers and book chapters. His publications currently report over 540 citations in Google Scholar with an H-index of 12. His current research interests include evolutionary computation as well as the design and implementation of effective metaheuristic algorithms for solving large-scale combinatorial optimization problems arising in various application areas, such as bioinformatics, graph theory, and software engineering.