A new iterated local search algorithm for the cyclic bandwidth problem

Jintong Ren[a], Jin-Kao Hao[a,b,*] Eduardo Rodriguez-Tello[c], Liwen Li[d], Kun He[d]

[a]*LERIA, Université d'Angers, 2 boulevard Lavoisier, 49045 Angers, France*

[b]*Institut Universitaire de France, 1 rue Descartes, 75231 Paris, France*

[c]*Cinvestav Tamaulipas, Km. 5.5 Carretera Victoria - Soto La Marina, 87130 Victoria Tamps., Mexico*

[d]*Huazhong University of Science and Technology, 1037 Luoyu Road. Wuhan, Hubei 430074, China*

Knowledge-Based Systems, June 2020

https://doi.org/10.1016/j.knosys.2020.106136

Abstract

The Cyclic Bandwidth Problem is an important graph labeling problem with numerous applications. This work aims to advance the state-of-the-art of practically solving this computationally challenging problem. We present an effective heuristic algorithm based on the general iterated local search framework and integrating dedicated search components. Specifically, the algorithm relies on a simple, yet powerful local optimization procedure reinforced by two complementary perturbation strategies. The local optimization procedure discovers high-quality solutions in a particular search zone while the perturbation strategies help the search to escape local optimum traps and explore unvisited areas. We present intensive computational results on 113 benchmark instances from 8 different families, and show performances that are never achieved by current best algorithms in the literature.

Key words: Heuristic; computational methods; cyclic bandwidth minimization; graph labeling; combinatorial optimization.

* Corresponding author.
Email address: `jin-kao.hao@univ-angers.fr` (Jin-Kao Hao).

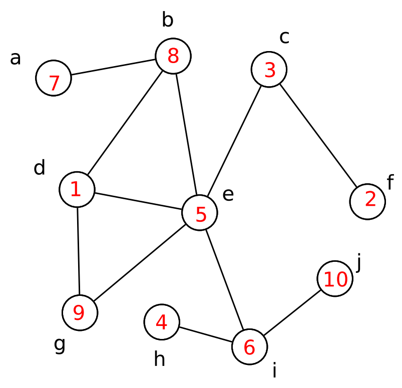
1 Introduction

Let $G(V, E)$ be a finite undirected graph with vertex set V ($|V| = n$), edge set E ($|E| = m$) and C_n a cycle graph. An embedding φ (also called a labeling or an arrangement) of G in C_n is a one-to-one mapping from V to V. The cyclic bandwidth of φ for G is given by

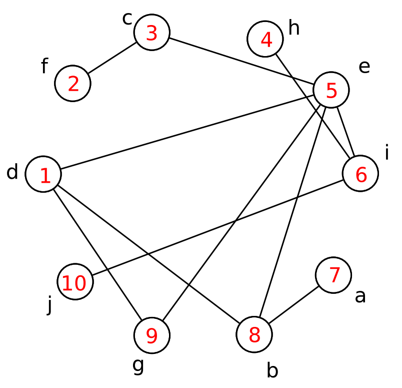
$$C_B(G, \varphi) = \max_{\{u,v\} \in E} \{|\varphi(u) - \varphi(v)|_n\}, \tag{1}$$

where $\varphi(u)$ represents the label assigned to vertex u and $|x|_n = \min\{|x|, n - |x|\}$ is the cyclic distance.

The Cyclic Bandwidth Problem (CBP) is then to find a labeling $\varphi^* \in \Omega$ which minimizes the cyclic bandwidth of a given graph, where Ω is the set of all possible labelings. See Fig. 1 for an illustrative example.



(a) Graph G with $n = 10$



(b) Cycle graph C_{10}

Fig. 1. An illustrative example: (a) graph G ($n = 10$) with its vertices named by a to j and a labeling (labels named by 1 to 10); (b) embedding to cycle graph C_{10} by reordering all the vertices on a cycle according to their labels in the clockwise direction. The cyclic bandwidth of the shown embedding $C_B(G, \varphi)$ equals 4, which is defined by the edges $\{d, e\}$, $\{e, g\}$ and $\{i, j\}$. One observes that the cyclic bandwidth corresponds to the minimum steps needed to go from one endpoint to the other endpoint of these edges either in a clockwise or counterclockwise direction on the cycle graph.

First introduced to formulate a design problem in the area of ring interconnection networks [1], the CBP has also been found to be a relevant model for a number of additional applications including VLSI design [2], data structure representations [3], code designs [4] and parallel computer systems [5]. In terms of computational complexity, the decision version of the CBP is $\mathcal{NP}$-complete [6]. Consequently, the CBP is computationally challenging for solution methods.

Given the relevance of the problem, a number of studies have been proposed in the literature. A majority of early studies are of theoretical nature and focused on finding exact cyclic bandwidths for special graphs or determining lower bounds for general graphs. For example, in [5], the relationship between the bandwidth $B_P(G)$ and cyclic bandwidth $B_C(G)$ for a graph G was identified: $B_P(G) \geq B_C(G) \geq \frac{1}{2}B_P(G)$. The authors of [7] showed that for seven graph labeling problems including the CBP, there exists a labeling that is simultaneously optimal for every unit interval graph. More investigations [8–10] were carried out to identify two extreme cases to obtain exact cyclic bandwidths for some special graphs. An exact algorithm [11] used the branch and bound method to solve small graphs with up to 40 vertices. The study of [12] was devoted to a systematic method to calculate lower bounds for $B_P(G)$ and $B_C(G)$ according to distance and degree-related parameters. In [13], the authors proposed a method to obtain sharp upper bounds of a graph by adding a new edge. Based on semidefinite programming (SDP) relaxations of the quadratic assignment problem, better lower bounds of $B_P(G)$ and $B_C(G)$ were introduced in [14].

Besides these theoretical studies, practical solution methods based on metaheuritics began to appear in recent years. To our knowledge, there are three such algorithms in the literature. In [15], the authors proposed the first tabu search algorithm (TS_{CB}) and compared it with a simulated annealing algorithm adapted from an algorithm designed for the related Bandwidth Minimization Problem [16]. Computational results confirmed the value of TS_{CB} on a set of benchmark instances. Recently, a three-phase heuristic algorithm called $ITPS$ was presented in [17], which improved several best known results in the literature. Very recently, the population-based evolutionary approach was investigated in [18], where five classical *permutation* crossovers (order crossover, order-based crossover, cycle crossover, partially mapped crossover, distance preserved crossover) [19] were compared within a hybrid genetic algorithm combining such a crossover and a descent search. This study found that the order-based crossover performs the best among the five compared crossovers. However, these hybrid genetic algorithms do not compete well with the best performing CBP algorithms. Indeed, the experimental results reported in the above studies showed that $ITPS$ [17] and TS_{CB} [15] represent the state-of-the-art for solving the CBP. Meanwhile, these two algorithms are complementary because they performed the best on different benchmark instances.

In this work, we aim to enrich the solution toolbox for effectively solving the cyclic bandwidth problem. For this, we investigate a new iterated local search ($NILS$) algorithm which distinguishes itself by two original features. First, we devise a new and effective strategy to explore candidate neighbor solutions generated by the conventional swap operator. Second, we employ two perturbation procedures with different intensities to better diversify the search. Compared to the two existing heuristic algorithms TS_{CB} and $ITPS$, the proposed algorithm is simpler (e.g., it uses only one neighborhood against 2 for TS_{CB} and $ITPS$) and requires fewer parameters (4 against 8 for TS_{CB} and 9 for $ITPS$), making it much easier to use.

To assess the performance of the proposed algorithm, we show computational results on the set of 113 well-known benchmark instances in the literature and make comparisons with the results of TS_{CB} and $ITPS$. Our experiments indicate that the proposed algorithm dominates the reference algorithms and achieves a performance that has never been reported in the CBP literature.

The remainder of the paper is organized as follows. In Section 2, we present the main algorithm and its key components. In Section 3, we show the computational results on the benchmark instances and comparisons with reference results in the literature. In Section 4, we report additional experiments to investigate the influences of main algorithmic components on the global performance of the algorithm. Conclusions are drawn in Section 5.

2 New iterated local search algorithm

Iterated local search [20] is a general search framework with numerous successful application examples (e.g., [21–24]. The basic idea of this approach is to use a local optimization procedure to find local optima and a perturbation procedure to move away from each local optimum discovered. The new iterated local search algorithm (*NILS*) presented in this work for the CBP follows this general approach and relies on three key components specially designed for this problem: a dedicated tabu search procedure (DTS) with a specific neighborhood exploration strategy, a directed perturbation procedure (Directed_Pertub) with a randomized shift-insert operator and a strong perturbation procedure with a destruction-reconstruction heuristic (Strong_Pertub). The algorithm employs the dedicated tabu search procedure to attain high-quality local optimal solutions and probes additional nearby local optimal solutions with the help of the directed perturbation procedure. To better diversify its search, the algorithm uses the strong perturbation procedure to displace the process to more distant unexplored regions. These three procedures are iterated to ensure a large exploitation and exploration of the whole search space.

The pseudo-code of the *NILS* algorithm is presented in Algorithm 1. The algorithm starts with a random initialization solution φ. The inner 'while' loop iteratively performs the dedicated tabu search procedure (Section 2.1), followed by the directed perturbation procedure (Section 2.2). At each iteration, the input solution is first improved by DTS which is based on the neighborhood N_f (Section 2.1) and the evaluation function f_e (See below). When DTS stagnates, Directed_Pertub is used to modify the incumbent solution to provide a new input solution for the next round of DTS. The process of DTS and Directed_Pertub is repeated L_3 times (L_3 is a parameter called exploration limit). When the inner 'while' loop terminates, we consider that the search has sufficiently examined the current and close regions. As a result, we heavily alter the incumbent solution with the strong perturbation procedure to move the search to a far and away region, then the 'DTS-Directed_Pertub' process is triggered to explore new local optimal solutions. The whole algorithm is repeated until a given cut off time limit T_{max} is reached, and the best solution found φ^* is returned.

Algorithm 1 New iterated local search algorithm for the CBP

```
1: **Input**: Finite undirected graph $G(V,E)$, neighborhood $N_f$, evaluation function $f_e$,
   tabu search depth $L_1$, directed perturbation strength $L_2$, exploration limit $L_3$, con-
   trolling percent $\alpha$ and cutoff time limit $T_{max}$
2: **Output**: The best solution found $\varphi^*$
3: $\varphi \leftarrow InitialSolution()$
4: $\varphi^* \leftarrow \varphi$
5: **while** the cutoff time limit $T_{max}$ is not reached **do**
6:    $Count \leftarrow 0$
7:    **while** $Count < L_3$ **do**
8:       $(\varphi, \varphi^*) \leftarrow DTS(\varphi, \varphi^*, N_f, f_e, L_1)$    // Local optimization with dedicated
         tabu search, Section. 2.1
9:       $(\varphi, \varphi^*) \leftarrow Directed\_Perturb(\varphi, \varphi^*, f_e, L_2)$    // Directed perturbation,
         Section. 2.2
10:      $Count \leftarrow Count + 1$
11:   **end while**
12:   $\varphi \leftarrow Strong\_Perturb(\varphi, \alpha)$    // Strong perturbation, Section 2.3
13: **end while**
14: **return** $\varphi^*$
```

To assess the quality of a candidate solution φ, the algorithm adopts the extended evaluation function $f_e(\varphi)$ introduced in [17], which is defined as follows.

$$f_e(\varphi) = C_B(G,\varphi) + \frac{Z(C_B(G,\varphi))}{|E|} \tag{2}$$

where $Z(C_B(G,\varphi)) = \sum_{\{u,v\}\in E} I_{uv}$ represents the number of edges whose cyclic distances equal $C_B(G,\varphi)$, and the indicator variable $I_{uv} = 1$ if $|\varphi(u)-\varphi(v)|_n = C_B(G,\varphi)$, and $I_{uv} = 0$ otherwise. The second term of $f_e(\varphi)$ in the range $(0,1]$ is used to distinguish solutions with the same cyclic bandwidth.

Algorithm 2 New tabu search phase

```
1: Input: input solution $\varphi$, best solution $\varphi^*$, neighborhood $N_f$, evaluation function $f_e$
   and tabu search depth $L_1$
2: Output: improved solution $\varphi$, updated best solution $\varphi^*$
3: $l \leftarrow 0$                                    // Counter of non-improving iterations
4: $\varphi' \leftarrow \varphi$                                    // Copy of the current solution
5: $\varphi_b \leftarrow \varphi$                                    // Local best solution
6: $\varphi_{ib} \leftarrow \varphi$                                    // Best solution in inner loop
7: while $l < L_1$ do
8:    $C(\varphi') \leftarrow CriticalSet(\varphi')$                    // Identify the critical vertices
9:    for Each $u \in C(\varphi')$ do
10:       $\varphi \leftarrow FindBestNeighbor(N_f(\varphi, u))$    // Choose a best neighbor solution
11:       $Update\ tabu\ list$
12:       if $f_e(\varphi) < f_e(\varphi_{ib})$ then
13:          $\varphi_{ib} \leftarrow \varphi$
14:       end if
15:    end for
16:    $\varphi' \leftarrow \varphi$
17:    if $f_e(\varphi_{ib}) < f_e(\varphi_b)$ then
18:       $l \leftarrow 0$
19:       $\varphi_b \leftarrow \varphi_{ib}$
20:    else
21:       $l \leftarrow l + 1$
22:    end if
23:    if $f_e(\varphi_b) < f_e(\varphi^*)$ then
24:       $\varphi^* \leftarrow \varphi_b$
25:    end if
26: end while
27: return $\varphi$, $\varphi^*$
```

2.1 Dedicated tabu search

The dedicated tabu search (DTS) procedure (Algorithm 2) is designed to exploit candidate solutions with the help of the neighborhood N_f (see below). DTS starts with an input solution φ and iteratively makes transitions from the current solution to a neighbor solution. At each iteration of the outer 'while' loop, DTS first identifies the critical vertices relative to the current solution (line 8, Alg. 2), and then for each critical vertex, swaps the label of this vertex against the label of another specifically selected vertex to generate a neighbor solution (lines 9-15, Alg. 2). After each solution transition, the performed swap operation is recorded in the so-called tabu list [25] to avoid revisiting the replaced solution. Once all the critical vertices are examined, operations are performed to update the counter of non-improving iterations, local best solution found during DTS and global best solution. DTS terminates when the local best solution cannot be improved for L_1 consecutive iterations.

To transform the incumbent solution, DTS uses the conventional swap operator which operates on specifically identified vertices. Let φ be the current solution, and $\varphi \oplus swap(u,v)$ be the neighbor solution obtained by exchanging the labels of vertices u and v. Like [15], we constraint the candidate vertex u to a specific subset of *critical* vertices $C(\varphi)$ defined as follows.

Let $C_B(u,\varphi) = \max_{v \in A(u)}\{|\varphi(u) - \varphi(v)|_n\}$ ($A(u)$ is the set of adjacent vertices of u) be the cyclic bandwidth of vertex u with respect to φ. Then the critical vertex set $C(\varphi)$ is given by $C(\varphi) = \{w \in V : C_B(w,\varphi) = C_B(G,\varphi)\}$.

Now for a given critical vertex $u \in C(\varphi)$, let $mid(u)$ denote the middle point of the shortest path in the cycle graph C_n containing all the vertices adjacent to u [15]. Then we define $S(u) \subseteq V$ to be the set of vertices which are closer than $u \in C(\varphi)$ to the middle point $mid(u)$ or equal to $mid(u)$, i.e., $S(u) = \{v \in V : |mid(u) - \varphi(v)|_n \leq |mid(u) - \varphi(u)|_n\}$.

It is worth noting that $S(u)$ is related not only to the critical vertex u but also to the labeling φ.

Given a solution φ and a critical vertex $u \in C(\varphi)$, we use $N_f(\varphi,u)$ to denote the set of solutions that can be obtained by swapping u and a vertex in $S(u)$.

Then, based on $C(\varphi)$ and $S(\cdot)$, DTS applies at each iteration the *swap* operator to transform φ to a new (neighbor) solution. For a vertex $u \in C(\varphi)$, the associated $S(u)$ is identified and the best eligible $swap(u,v)$ ($v \in S(u)$) is applied (see Alg. 2, line 10) to obtain a new incumbent solution (a swap is eligible if it is not forbidden by the tabu list or if it leads to the best solution found so far). Then the performed $swap(u,v)$ is added in the tabu list and the reverse operation $swap(v,u)$ will not be allowed for the next tl iterations (tl is called tabu tenure). In this work, we adopt the dynamic tabu tenure method used in [15, 17], which fixes tl according to a periodic step function.

Fig. 2 provides a simple illustration of solution transformation. According to the definition of set $S(u)$ above, we identify the critical set $C(\varphi) = \{e, i, g, j\}$. Then the swap operation is applied to a vertex $u \in C(\varphi)$ with a suitable vertex of $S(u)$. For instance, starting from the critical vertex e, the middle point $mid(e)$ is recognized as i with label 6. Then, the distance between e and i is 1 and $S(e) = \{i, d\}$. So for the critical vertex e, there are two possible swaps: $swap(e,i)$ and $swap(e,d)$. Since $swap(e,d)$ generates a better solution than $swap(e,i)$ does, it is applied to obtain the new incumbent solution. Note that when one examines next critical vertex, its $S(\cdot)$ will be defined relative to the new solution. After all the critical vertices are examined, DTS terminates its current iteration and starts its next iteration with a new critical set.

2.2 Directed perturbation with randomized shift-insert

When DTS stops, the search is considered to be trapped in a local optimum (it is stagnating since it cannot improve its best solution during L_1 iterations). To escape the trap, we apply a directed perturbation procedure (depicted in Algorithm 3), which relies on a *randomized* version of the *ShiftInsert* operator [17]. Our *RandomizedShiftInsert* operator works as follows. First, we identify an edge $e = \{x, y\}$ with the largest cyclic distance (i.e., $C_B(G, \varphi)$). Then, one endpoint of the edge is chosen (say x) and used to perform β (a random number between 1 and $C_B(G, \varphi)$) chained swaps where each swap involves x and the next vertex in the direction of decreasing the cyclic distance of edge e. Based on this operator, the directed perturbation procedure modifies the input solution by applying L_2 times the *RandomizedShiftInsert* operator. This perturbation procedure has the desirable property that it changes the input solution without deteriorating too much of its quality.

Algorithm 3 Directed perturbation

```
1: Input: input solution $\varphi$, best solution $\varphi^*$, and perturbation strength $L_2$
2: Output: perturbed solution $\varphi$, updated best solution $\varphi^*$
3: $Counter \leftarrow 0$
4: while $Counter < L_2$ do
5:     $\varphi \leftarrow RandomizedShiftInsert(\varphi)$
6:     $Counter \leftarrow Counter + 1$
7:     if $f_e(\varphi) < f_e(\varphi^*)$ then
8:         $\varphi^* \leftarrow \varphi$
9:     end if
10: end while
11: return $\varphi$, $\varphi^*$
```

In the example shown in Fig. 3(a), the edge with the largest cyclic distance is $\{i, j\}$ indicated in green. The *RandomizedShiftInsert* operator uses i as the starting vertex to perform 2 swaps (2 is randomly determined from 1 and 4)

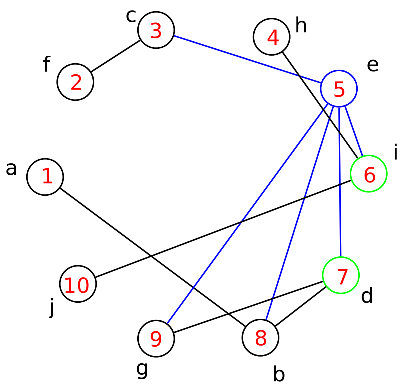


Fig. 2. Illustration for solution transformation: a graph with its labeling φ, critical set $C(\varphi) = \{e, i, g, j\}$ and set $S(e)$ for the first critical vertex e.

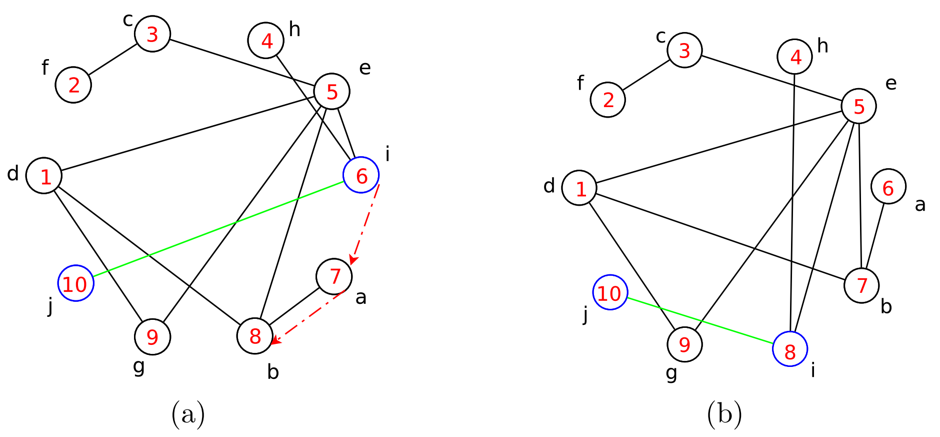


Fig. 3. Illustration of the *RandomizedShiftInsert* operator: (a) The cycle graph before the operation, (b) The cycle graph after the operation (i.e., $swap(i, a)$ followed by $swap(i, b)$).

in a clockwise direction, leading to the solution shown in Fig. 3(b).

We investigate the degree of influence of the directed perturbation procedure over the search performance of the proposed *NILS* algorithm in Section 4.

2.3 Strong perturbation with destruction-reconstruction

When the process of DTS and directed perturbation stops, the search is considered to be trapped in a deep local optimum. To enable the algorithm to continue its search, we introduce a strong perturbation to definitely bring the search to a distant new region. The core idea is to move uncritical vertices to get closer to the critical vertices. For this purpose, the strong perturbation performs two steps: erase the labels of some specifically selected vertices (destruction step) and then re-assign new labels to them according to a greedy strategy (reconstruction step).

To destruct a solution, we first identify the set of vertices C_R whose labels will be removed: $C_R(\varphi) = \{w \in V : C_B(w, \varphi) \leq \alpha \cdot C_B(G, \varphi)\}$ where $\alpha \in [0, 1]$ is a controlling parameter. Thus, $C_R(\varphi)$ is composed of vertices with a cyclic bandwidth up to $\alpha \cdot C_B(G, \varphi)$. Then, we use $Ł_a$ to collect the labels freed by the vertices of $C_R(\varphi)$: $Ł_a = \{l(w) : w \in C_R(\varphi)\}$.

To reconstruct the solution, we re-assign the labels of $Ł_a$ to the vertices of $C_R(\varphi)$ with a greedy heuristic. Starting from a random node $u \in V \setminus C_R(\varphi)$, we employ a breadth first search to traverse the whole graph. In order to select a label from $Ł_a$ for each vertex $v \in C_R(\varphi) \cap A(u)$ ($A(u)$ is the set of adjacent vertices of u), we first identify the set of labels $L_{in}(u)$ whose cyclic distances to $l(u)$ are no more than L_B: $L_{in}(u) = \{l_e : |l(u) - l_e|_n \leq L_B, l_e \in Ł_a\}$ where

L_B is the analytical lower bound of the graph according to [8]. If $L_{in}(u)$ is not empty, a random label from $L_{in}(u)$ is selected and assigned to v. Otherwise, a random label from $Ł_a \setminus L_{in}(u)$ is assigned to v. Then, the same operation is performed on each vertex $v \in A(u)$. The entire reconstruction step finishes when all vertices in $C_R(\varphi)$ are re-assigned labels.

An illustrative example is shown in Fig. 4. Given a graph $G(V, E)$ ($|V| = 10$, $L_B = 3$), the objective value of the solution in Fig. 4(a) is 4 ($C_B(G, \varphi) = 4$). For the destruction step, if we set α to be 0.8, we get $C_R(\varphi) = \{a, b, c, f, h\}$ and $Ł_a = \{2, 3, 4, 7, 8\}$; while the partial solution after removing the vertices in $C_R(\varphi)$ is showed in Fig. 4(b). For the greedy reconstruction step, we starting from a random vertex $u \in V \backslash C_R(\varphi) = \{d, e, i, j, g\}$ (say d in Fig. 4(c)), we first allocate labels to vertices $v \in C_R(\varphi) \cap A(d) = \{b\}$. According to the description above, $L_{in}(d) = \{2, 3, 4, 8\}$ (labels 9 and 10 are already assigned to vertices). A random label (2 in Fig. 4(c)) is chosen from $\{2, 3, 4, 8\}$ to be assigned to vertex b. Once all the adjacent vertices of d ($\{b, g, e\}$) are successfully re-assigned, they will go through the same operation iteratively following the principle of the breadth first search. And vertices c and a are re-assigned labels 3 and 4 respectively in Fig. 4(d). When we consider allocating labels to the adjacent vertices of c, $L_{in}(c)$ is empty, so we choose a label from $Ł_a \setminus L_{in}(c) = \{7, 8\}$ (7 in our case) for vertex f. We repeat the above operation until each vertex in $C_R(\varphi)$ receives a label. And the solution in Fig. 4(e) with a cyclic bandwidth of 4 is returned as the output of the strong perturbation procedure.

The impact of the strong perturbation procedure, introduced here, on the behavior of the $NILS$ algorithm is investigated in Section 4.

2.4 Relations with previous studies

$NILS$ distinguishes itself from two previous algorithms TS_{CB} [15] and $ITPS$ [17] by the following features. First, unlike [15, 17], the dedicated tabu search procedure of $NILS$ relies on a single neighborhood while both TS_{CB} and $ITPS$ explore two neighborhoods in a probabilistic way. As such, the key optimization component of our algorithm is simpler and more focused while making its search more effective and more computationally efficient. Second, $NILS$ employs two perturbation strategies which are different from the previous studies. The directed perturbation with the randomized shift-insert operation favors the generation of more diverse solutions, while the destruction-reconstruction based strong perturbation provides a complementary and guided strategy to bring the search to new promising search regions. Last but not least, the $NILS$ algorithm requires much fewer parameters (4 against 8 for TS_{CB} and 9 for $ITPS$), making it much easier to use and analyze.

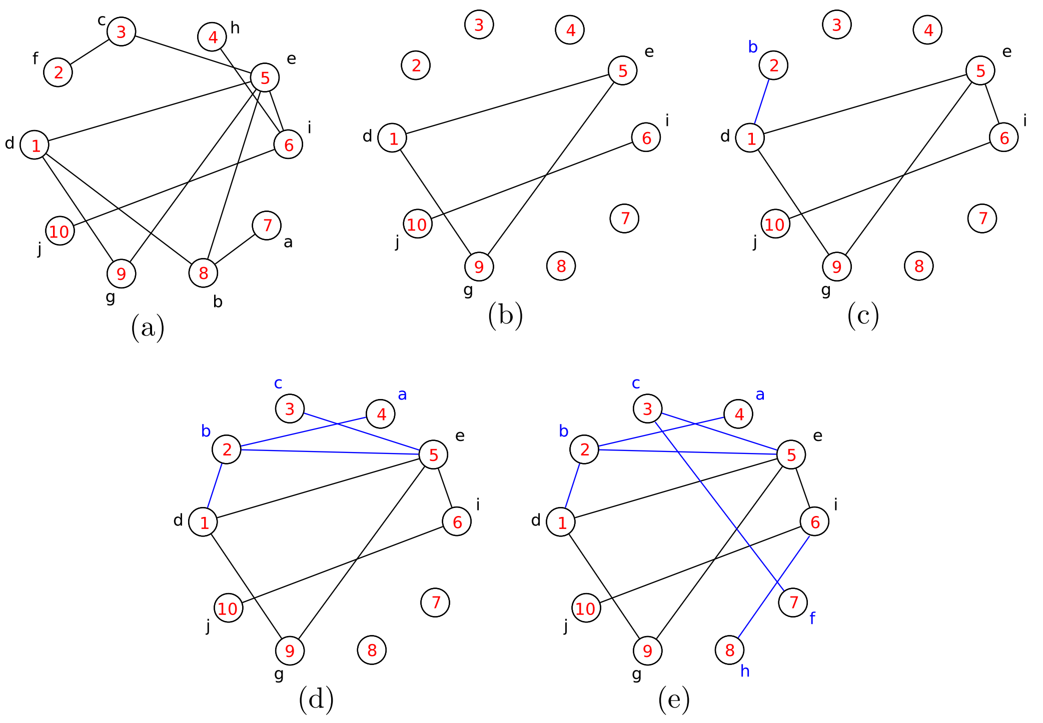


Fig. 4. Illustration of the strong perturbation procedure using destruction and reconstruction on a graph with $C_B(G, \varphi) = 4$, analytical lower bound L_B=3 and controlling parameter $\alpha = 0.8$. (a) input solution; (b) partial solution after removing 5 vertices of C_R; (c) beginning of solution reconstruction from vertex d; (d) reconstruction in progress; (e) completion of the reconstruction.

As we show in the next section on computational experiments, the $NILS$ algorithm integrating these specific features performs extremely well on the set of 113 well-known CBP benchmark instances.

3 Experimental results

This section starts presenting the experimental conditions under which the empirical comparisons were carried out. It continues by giving details about the methodology used to identify the most appropriate combination of input parameter values for the proposed $NILS$ algorithm. This section concludes by providing an in-depth comparative analysis which considers the proposed $NILS$ algorithm and two solution approaches which are currently considered as the reference methods in the state-of-the-art: TS_{CB} [15] and $ITPS$ [17].

3.1 Experimental setup

The experimentation of this work was carried out on 113 graphs which were previously employed to assess the performance of the state-of-the-art algorithms reported by Rodriguez-Tello *et al.* [15], and latter by Ren *et al.* in [17]. These graphs are organized in two different groups. The first one is made up of 85 graphs belonging to 7 different families of standard graphs (paths, cycles, two dimensional meshes, three dimensional meshes, complete r-level k-ary trees, caterpillars and r-dimensional hypercubes). Their order $|V|$ varies in the range from 9 to 8192, while their size $|E|$ goes from 8 to 53248. The values of the optimal solutions for these graphs are known, the reader is referred to [15] for consulting the details. Therefore, attaining the optimal solutions for these instances is an important factor to evaluate the performance of algorithms. The second group contains 28 graphs coming from the *Harwell-Boeing Sparse Matrix Collection* [1] . These instances were directly constructed from sparse adjacency matrices produced in practical and engineer real world applications. Their order fluctuates in the interval $9 \leq |V| \leq 715$ and their size are in the range $46 \leq |E| \leq 3720$. The optimal solutions for 7 small graphs are known, while for the remaining 21 graphs lower and upper bounds can be calculated according to [8].

The performance assessment of the three analyzed algorithms was done using the same comparison metrics previously employed in [15] and [17], i.e., the best cyclic bandwidth attained for each instance (smaller values are preferred), the computation time expended in seconds, the relative root mean square error (RMSE) and the overall relative root mean square error (O-RMSE). The RMSE indicator permits to evaluate the performance of an algorithm over an individual benchmark instance, while the O-RMSE indicator computes average RMSE values over a whole set of test instances.

In order to further analyze the behavior of the three compared algorithms, a statistical significance analysis was carried out. It starts by evaluating, through the use of Shapiro-Wilk test, the normality of data distributions. Bartlett's test is then implemented to determine whether the variances of the normally distributed data is homogeneous or not. In case variance homogeneity is confirmed, ANOVA test is applied; on the contrary Welch's t parametric tests are executed. When facing non-normal data Kruskal-Wallis test is carried out. The significance level consistently considered in all the cases is 0.05. Concretely, we made this analysis by comparing a pair of different algorithm implementations, say A and B (denoted as A/B). Three different outcomes, represented respectively as $+$, $-$, and $\star$, can be obtained: 1) algorithm A offers a significant better performance than B; 2) B significantly outperforms A (i.e., A is

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

defeated by B); and 3) it was not possible to conclude a statistical significant difference between the compared methods.

The proposed $NILS$ algorithm was coded using the C++ programming language[2]. Given that the source codes of the TS_{CB} and $ITPS$ methods are publicly available (see [17]), the three analyzed algorithms were compiled in gcc 4.4.7 using the optimization flag -O3. These three algorithms were independently executed 50 times, using different random seeds, over each test instance and with a maximum running time of 600 seconds.

3.2 Tuning of parameters

In order to automatically determine the most suitable combination of input parameter values for the proposed $NILS$ algorithm, we have decided to employ *I/F-Race*, an iterated procedure based on the use of racing and Friedman's non-parametric two-way analysis of variances by ranks. It is part of the popular *irace* package [26, 27] for automatic parameter configuration.

For this tuning experiment, the *irace* parameter controlling the maximum number of runs of the algorithm being tuned (tuning budget) was fixed to 2000. Then, a subset of 10 instances, identified as challenging for the state-of-the-art algorithms [15, 17], was selected and consistently used. This subset includes certain Harwell-Boeing instances (*bcsstk06, 494_bus, dwt_592, 662_bus, 685_bus, can_715*), as well as some graphs from different standard topologies (*path1000, cycle1000, mesh2D20x50, mesh3D13, tree2x9, caterpillar44, hypercube11*).

Our $NILS$ algorithm requires to define five different input parameters before start working. The first one is the cutoff time T_{max}. It was fixed to 600 seconds for all the experiments presented in this work, which is the same value employed by the state-of-the-art algorithms [15, 17]. The other four input parameters of $NILS$ are listed in Table 1, along with their description, type, and range of possible values.

After the execution of our automatized tuning experiments, the parameter values for obtaining the best performance of $NILS$ identified by *irace* are: $L_1 = 100$, $L_2 = 20$, $L_3 = 2000$, and $\alpha = 0.84$. Hence, these values are consistently employed along the whole experimentation reported in the following.

[2] The source code of our $NILS$ algorithm will be available at: `https://github.com/thetopjiji/NILS`

Table 1
Description and ranges for the input parameters of the *NILS* algorithm automatically tuned with *irace* [26].

Parameter	Description	Type	Range/Values
L_1	Tabu search depth	Categorical	{1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 1500, 2000, 3000, 5000, 10000, 20000}
L_2	Directed perturbation strength	Categorical	{1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 1500, 2000, 3000, 5000, 10000, 20000}
L_3	Exploration limit	Categorical	{1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 1500, 2000, 3000, 5000, 10000, 20000}
α	Controlling percent	Real	(0.0, 1.0)

3.3 Comparisons with state-of-the-art algorithms

This section presents a performance assessment of our *NILS* algorithm compared to the two best performing algorithms in the CBP literature (i.e., TS_{CB} [15] and *ITPS* [17]). We ignore the recent hybrid genetic algorithms studied in [18], because their results are dominated by those of these two reference algorithms. This assessment was carried out under the experimental conditions previously detailed in Section 3.1.

Table 2 summarizes the results provided by this computational experiment organized by instance subsets (see column 1). The first seven subsets correspond to standard graph topologies, whereas the last one is composed of graphs coming directly from engineering real world problems. Column 2 (Num.) shows the number of benchmark instances in each subset. Four columns are employed to register the results (averaged over all the graphs in a subset) produced by each compared algorithm: the best cyclic bandwidth reached (Avg. Cb_b), the computational time (in seconds) expended to attain this objective cost (Avg. T_b), the overall relative root mean square error (O-RMSE), and the success percentage for finding the optimal (or best-known) solutions (% Best). Detailed results for each of the 113 benchmark instances used in this experiment are shown in Tables A.1 and A.2 provided in Appendix A.

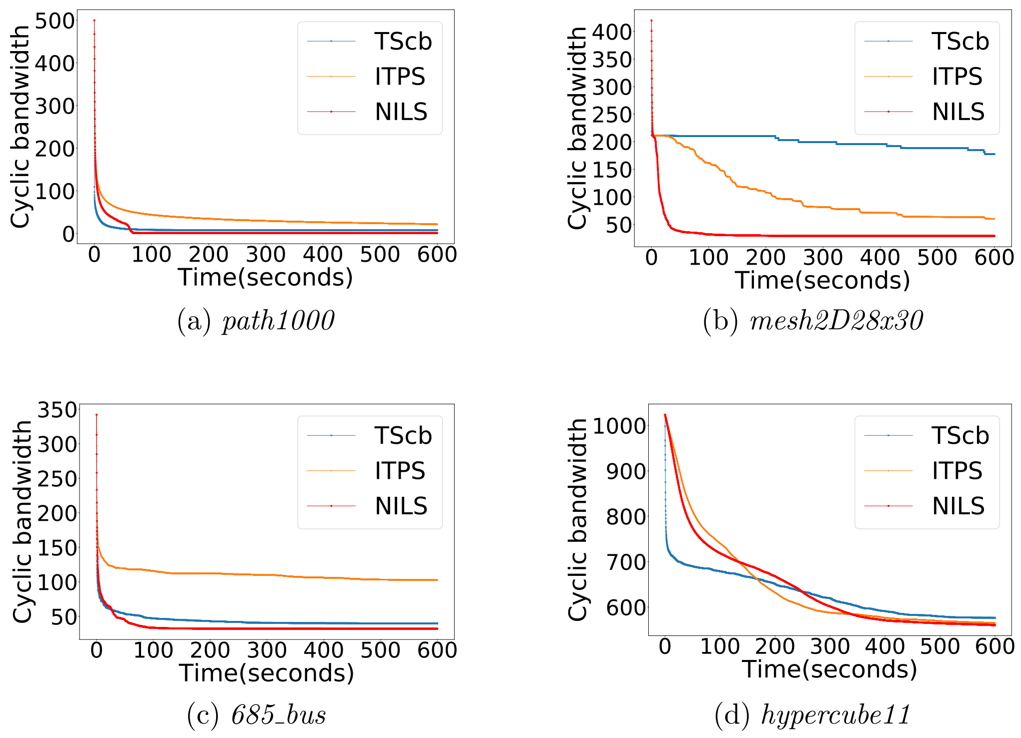
From Table 2, one observes that our *NILS* algorithm has reached better average best cyclic bandwidth values (See column Avg. Cb_b) than the two state-of-the-art algorithms for all the 8 subsets of instances tested. Indeed, *NILS* was able to attain new best-known results for 18 standard graphs and for 4 Harwell-Boeing instances, respectively. For the remaining 91 benchmark graphs it matches the best recorded results in the literature. We remark that for the first 6 graph types *NILS* attained the optimal solution values (see column % Best) for each of its runs, while *ITPS* could only do this for the subsets *tree* and *caterpillar*. In contrast, TS_{CB} could not ensure optimal solutions for any subset of instances.

It is worth noting that the three large instances in the subset *mesh3D* (3-dimensional meshes) and the three instances of the *hypercube* subset (r-dimensional hypercubes) are among the most difficult benchmarks. To illustrate this, consider the graph *mesh3D13* (with 2197 vertices and 6084 edges) for which neither TS_{CB}, nor $ITPS$ can get the optimal objective value of 133 (553 and 551, respectively). Nevertheless, $NILS$ is able to find the optimal solution for this graph, which represents an important improvement in solution quality with respect to that furnished by $ITPS$ and TS_{CB} (75.86% and 75.95%). It proves the effectiveness of $NILS$ for solving challenging instances.

Concerning the O-RMSE values scored by the three compared algorithms, our $NILS$ algorithm reports the ideal value of zero for 5 subsets (*path, cycle, mesh2D, tree* and *caterpillar*). On the other hand, $ITPS$ did it only for one subset (*tree*) and TS_{CB} for none of them. This means that our algorithm is more robust than the two reference algorithms, considering it achieved the optimal solution at every execution for all the graphs in most of the subsets. For the two remaining subsets of instances (*mesh3D* and *hypercube*), $NILS$ also achieved lower O-RMSE values (0.36 and 0.26) than those scored by TS_{CB} (1.47 and 0.34) and $ITPS$ (1.39 and 0.59). Moreover, the average computational time expended by $NILS$ to attain these solutions (see column Avg. T_b) is largely reduced with respect to that consumed by the competing algorithms. An exception is the case of the *hypercube* subset, where the computational effort needed by $NILS$ is 6.50% higher than that of TS_{CB} (584.21 vs. 546.23), but $NILS$ produced much better solutions than TS_{CB}.

An in-depth statistical significance analysis, using the methodology described in Section 3.1, was performed for validating the experimental results produced in our performance comparisons. This analysis, presented in Table 3, and detailed in the last four columns of Tables A.1 and A.2, revealed that $NILS$ was able to statistically outperform TS_{CB} and $ITPS$ in 51.33% and 44.25% of the 113 tested instances (58 and 50 graphs, respectively). For the remaining benchmark instances, it was not possible to identify a statistical difference in performance between $NILS$ and the state-of-the-art algorithms.

If we check the detailed results of Tables A-1 et A-2 in the Appendix, we can make some general comments about the behaviors of the three algorithms with respect to the size (complexity) of the benchmark graphs. First, we observe that within each of the 8 graph families, larger graphs with more vertices and edges are usually more difficult to solve for all algorithms. This is especially true for the largest instances such as *path*, *cycle*, *mesh2D*, *mesh3D* as well as large instances of the Harwell-Boeing family. Second, between $ITPS$ and TS_{CB}, $ITPS$ reached an equal or a better performance for most graphs except some *cycle*, *mesh2* and *hypercube* graphs, while TS_{CB} was more successful on some large graphs. Third, our $NILS$ algorithm performed remarkably well on almost all graphs compared to the reference algorithms both in terms of



(a) *path1000* (b) *mesh2D28x30*

(c) *685_bus* (d) *hypercube11*

Fig. 5. Convergence charts (running profiles) of TS_{CB}, $ITPS$ and $NILS$ for solving four representative difficult instances (*path*, *mesh2D28x30*, *685_bus* and *hypercube11*). The results were obtained from 50 independent executions of each compared algorithm.

best solutions found and computational efficiency. This is particularly the case for the instances which are difficult for the reference algorithms such as the largest *path*, *cycle*, *mesh2D*, *mesh3D* and *hypercube* graphs.

Finally, to study the behaviors of the three compared algorithms throughout the execution, we performed an additional experiment to obtain the convergence charts (running profiles) of the algorithms on four representative and difficult instances: two standard graphs (*path1000* and *mesh2D28x30*) and two Harwell-Boeing graphs (*685_bus* and *hypercube11*). For this experiment, we ran each algorithm 50 times to solve each instance with the time limit of 600 seconds and recorded the best objective values during the executions. Fig. 5 shows the corresponding convergence charts that indicate how the average best objective values found by each algorithm (y-axis) evolves as a function of the running time of the algorithm (x-axis). We observe that even if all the algorithms are able to improve the solution quality quickly at the beginning of their search, our $NILS$ algorithm has a better behavior on the long term. Indeed, when the reference algorithms began to slow down their improvement or even stagnate on their best solution after some 100 seconds, our $NILS$ algorithm continued its search to find still better solutions. This experiment shed light on why $NILS$ competes highly favorably with the reference algorithms.

Table 2
Summary of the experimental performance comparison among the two reference methods in the CBP literature (i.e., TS_{CB} [15] and $ITPS$ [17]) and the $NILS$ algorithm over 113 benchmark instances: 85 standard graphs with known optimal solutions, and 28 Harwell-Boeing instances.

		TS_{CB}				$ITPS$				$NILS$			
Graph type	Num.	Avg. Cb_b	Avg. T_b	O-RMSE	% Best	Avg. Cb_b	Avg. T_b	O-RMSE	% Best	Avg. Cb_b	Avg. T_b	O-RMSE	% Best
path	15	2.53	131.85	2.01	66.67	1.87	158.22	3.01	80.00	1.00	6.24	0.00	100.00
cycle	15	2.40	40.71	1.82	73.33	2.60	162.75	4.22	73.33	1.00	9.38	0.00	100.00
mesh2D	15	27.67	144.52	1.88	66.67	12.07	112.86	0.44	40.00	11.40	10.45	0.00	100.00
mesh3D	10	180.30	328.32	1.47	30.00	140.50	266.65	1.39	70.00	64.50	132.87	0.36	100.00
tree	12	55.08	75.90	0.02	91.67	54.67	23.36	0.00	100.00	54.67	1.52	0.00	100.00
caterpillar	15	15.13	75.31	0.07	93.33	15.07	60.54	0.07	100.00	15.07	18.07	0.00	100.00
hypercube	3	1551.67	546.23	0.34	0.00	2017.33	591.41	0.59	0.00	1492.00	584.21	0.26	0.00
Harwell-Boeing	28	22.21	112.70	2.65	28.57	23.50	141.24	3.90	28.57	20.39	40.69	2.15	28.57
Win/Match/Fail		26/87/0											

Table 3
Summary of the statistical signification analysis from the comparison among the two reference methods in the CBP literature (i.e., TS_{CB} [15] and $ITPS$ [17]) and the $NILS$ algorithm over 113 benchmark instances: 85 standard graphs with known optimal solutions, and 28 Harwell-Boeing instances.

		$NILS$ / TS_{CB}			$NILS$ / $ITPS$		
Graph type	Num.	$+$	$\star$	$-$	$+$	$\star$	$-$
path	15	8	7	0	5	10	0
cycle	15	5	10	0	10	5	0
mesh2D	15	8	7	0	10	5	0
mesh3D	10	10	0	0	7	3	0
tree	12	4	8	0	1	11	0
caterpillar	15	6	9	0	3	12	0
hypercube	3	2	1	0	3	0	0
Harwell-Boeing	28	15	13	0	11	17	0
Total	185	58	55	0	50	63	0

Table 4
Summary of comparative results between $NILS$ and its $NILS\_dp$ variant (i.e., without the directed perturbation component) on the 8 families of 113 benchmark instances.

		$NILS\_dp$				$NILS$				Statistics		
Graph type	Num.	Avg. Cb_b	Avg. T_b	O-RMSE	% Best	Avg. Cb_b	Avg. T_b	O-RMSE	% Best	+	⋆	−
path	15	1.00	9.48	0.00	100.00	1.00	6.24	0.00	100.00	0	15	0
cycle	15	1.00	14.63	0.38	100.00	1.00	9.38	0.00	100.00	2	13	0
mesh2D	15	58.73	98.56	2.94	66.67	11.40	10.45	0.00	100.00	9	6	0
mesh3D	10	208.20	257.35	1.69	0.00	64.50	132.87	0.36	100.00	10	0	0
tree	12	54.92	66.68	0.02	91.67	54.67	1.52	0.00	100.00	3	9	0
caterpillar	15	17.73	174.91	0.36	73.33	15.07	18.07	0.00	100.00	8	7	0
hypercube	3	1586.00	550.01	0.34	0.00	1492.00	584.21	0.26	0.00	3	0	0
Harwell-Boeing	28	41.00	125.37	10.04	28.57	20.39	40.69	2.15	28.57	15	13	0
Total	113									50	63	0

4 Analysis of the two perturbation strategies

The $NILS$ algorithm applies two perturbation strategies to achieve diversification effects of different intensities: directed perturbation with the randomized shift-insert operation and strong perturbation using a destruction-reconstruction process. In this section, we investigate the influence of these perturbation strategies on the performances of the algorithm. For this purpose, we created two $NILS$ variants: $NILS\_dp$ by disabling the *directed perturbation* component of $NILS$ and $NILS\_sp$ by disabling the destruction-reconstruction based *strong perturbation*. We ran both variants to solve the 113 benchmark instances according to the condition specified in Section 3.1 and reported their computational results in Tables 4 and 5 together with those produced by $NILS$.

In these tables, the information of the compared algorithms is shown employing the same column headings as those used in Table 2. The last three columns (Statistics) present the statistical results obtained by using the methodology detailed in Section 3.1.

From these tables, we observe that removing any of these perturbation strategies greatly deteriorates the performance of the $NILS$ algorithm.

Specifically, the results of Table 4 show that the directed perturbation is important for 7 out of 8 families of instances in terms of most performance indicators. Without the directed perturbation, the algorithm leads to worse results in terms of best and average objective values while its performance is less stable. Globally, the statistical analysis indicates that for 50 instances (44.25%), the directed perturbation plays a significant and positive role. This is particular the case for instances belonging to three families (*mesh2D, mesh3D,* and *Harwell-Boeing*).

Table 5
Summary of comparative results between $NILS$ and its $NILS\_sp$ variant (i.e., without the strong perturbation component) on the 8 families of 113 benchmark instances.

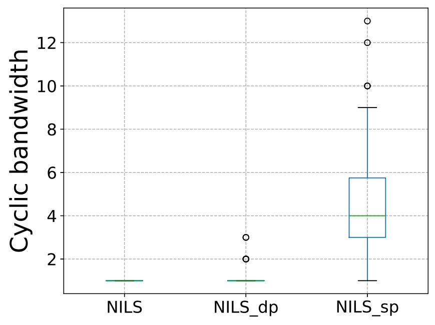
		$NILS\_sp$				$NILS$				Statistics		
Graph type	Num.	Avg. Cb_b	Avg. T_b	O-RMSE	% Best	Avg. Cb_b	Avg. T_b	O-RMSE	% Best	$+$	$\star$	$-$
path	15	1.00	30.22	0.45	100.00	1.00	6.24	0.00	100.00	10	5	0
cycle	15	1.00	20.50	2.18	100.00	1.00	9.38	0.00	100.00	11	4	0
mesh2D	15	11.40	16.86	0.03	100.00	11.40	10.45	0.00	100.00	1	14	0
mesh3D	10	64.50	136.14	0.57	100.00	64.50	132.87	0.36	100.00	0	10	0
tree	12	54.67	1.70	0.00	100.00	54.67	1.52	0.00	100.00	0	12	0
caterpillar	15	15.07	40.64	0.08	100.00	15.07	18.07	0.00	100.00	4	11	0
hypercube	3	1502.67	536.50	0.25	0.00	1492.00	584.21	0.26	0.00	0	2	1
Harwell-Boeing	28	20.39	49.47	2.53	28.57	20.39	40.69	2.15	28.57	8	20	0
Total	113									34	78	1

Similarly, the results of Table 5 disclose that the strong perturbation also impacts the performance of the $NILS$ algorithm even if the impact is less important compared to that of the directed perturbation. This observation is supported by our statistical assessment, which revealed that a relevant statistical difference in favor of $NILS$ with respect to $NILS\_sp$ exists for only 34 benchmark instances (30.09%). Disabling the strong perturbation in our algorithm leads to a less stable implementation for all the graph families except for the *tree* family (observe column O-RMSE). The benefit of using the strong perturbation is particularly visible on instances of four families (*path, cycle, mesh3D,* and *Harwell-Boeing*). In this sense, the strong perturbation is complementary with respect to the directed perturbation, given that they help to improve the solution of instances from different families.

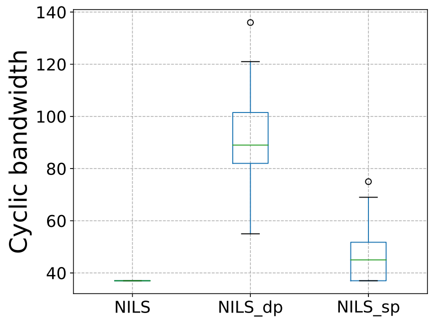
Concerning the average expended computational time, we can observe that both $NILS\_dp$ and $NILS\_sp$ consume more CPU resources than $NILS$ for most of the benchmark instances evaluated. Only in the case of the *hypercube* graphs, $NILS$ makes use of a higher average computational time than the other two reference algorithms. But this is largely compensated by the better quality solutions provided by our $NILS$ algorithm.

To further highlight the benefits of employing the two proposed perturbation strategies, we illustrate in Fig. 6 a detailed comparison between $NILS$ and the two variants $NILS\_dp$ and $NILS\_sp$ on four representative instances (*cycle1000, caterpillar44, hypercube13,* and *662_bus*) from different benchmark families. The plots are based on the results of 50 independent runs of the algorithms.

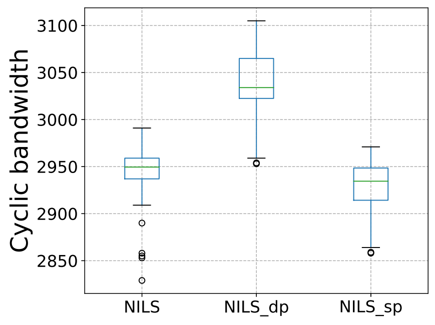
Fig. 6(a) shows that the results of $NILS$ and $NILS\_dp$ share the same median except that there are several outliers for $NILS\_dp$, while $NILS\_sp$ has a worse performance in terms of the median and interquartile range. This indicates the important role of strong perturbation for instance *cycle1000.*



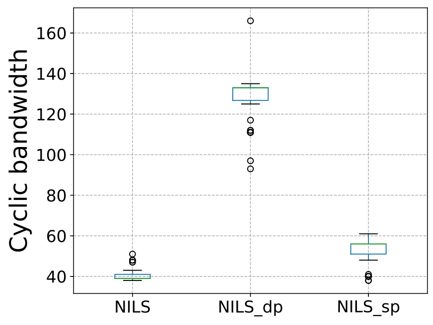
(a) *cycle1000*



(b) *caterpillar44*



(c) *hypercube13*



(d) *662_bus*

Fig. 6. Boxplots depicting the cyclic bandwidth cost reached by $NILS$, $NILS\_dp$ and $NILS\_sp$ when used for solving four representative instances from the subsets *cycle, caterpillar, hypercube,* and *Harwell-Boeing*. The results were obtained from 50 independent executions of each compared algorithm.

On the contrary, $NILS\_sp$ performs better than $NILS\_dp$ with smaller medians, tighter interquartile ranges and smaller minimal values for the other 3 instances in Fig. 6(b)-6(d). It is worth noting that in Fig. 6(c), $NILS\_sp$ shows a better performance than $NILS$ with a smaller first quartile, median and third quartile. That explains why there is a statistical difference against $NILS$ for one *hypercube* instance registered in Table 5 (column $-$). However, $NILS$ has obtained smaller outlier values than $NILS\_sp$, which also leads to a better average cyclic cost (1492.00 vs. 1502.67). To sum up, this experiment shows that both $NILS\_dp$ and $NILS\_sp$ report a worse performance than $NILS$ in each representative instance in Fig. 6, which means that the directed perturbation and strong perturbation play complementary roles in $NILS$.

5 Conclusions

The $\mathcal{NP}$-hard cyclic bandwidth problem has a number of relevant applications. The $NILS$ algorithm presented in this work enriches the practical solution toolbox for effectively solving this challenging problem. For the 85 standard instances with known optimal solutions, $NILS$ attains the optimal cyclic bandwidth costs for 82 instances (96.47%) while the two best performing algorithms in the literature only achieve 59 (69.41%) and 63 (74.12%) optimal solutions respectively. Remarkably, our algorithm establish 4 new record results (improved upper bounds) for 4 *Harwell-Boeing* instances. Moreover, the algorithm is highly robust across the instances of most tested families with very different structures and topologies.

Finally, the proposed algorithm has the advantage of requiring fewer parameters compared to the two leading algorithms presented in [15, 17]. As a result, it is easier for the user to apply it to solve new problem instances. Given that the source code of our algorithm will be publicly available, we hope this work will help to better solve some practical cyclic bandwidth problems and contribute to design other more powerful CBP algorithms.

Acknowledgment

We are grateful to the anonymous referees for their helpful comments and suggestions, which helped us to improve the presentation of the work. This work was partially supported by the Franco-Chinese PHC Cooperation Program (No. 41342NC). Support for the first author from the China Scholarship Council (2016-2020, No. 201608070103) and support for the third author from the Mexican Secretariat of Public Education under Grant SEP-Cinvestav (2019-2020, No. 00114) are also acknowledged.

References

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

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

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

[4] F. R. Chung, Labelings of graphs, Selected Topics in Graph Theory 3 (1988) 151–168.

[5] J. Hromkovič, V. Müller, O. Sýkora, I. Vrt'o, On embedding interconnection networks into rings of processors, in: D. Etiemble, J.C. Syre (Eds). International Conference on Parallel Architectures and Languages Europe, Lecture Notes in Computer Science 605 (1992) 51–62, Springer, Berlin, Heidelberg.

[6] Y. Lin, The cyclic bandwidth problem, Systems Science and Mathematical Sciences 7 (3) (1994) 282–288.

[7] J. Yuan, S. Zhou, Optimal labeling of unit interval graphs, Applied Mathematics 10 (1995) 337–344.

[8] Y. Lin, Minimum bandwidth problem for embedding graphs in cycles, Networks: An International Journal 29 (3) (1997) 135–140.

[9] P. C. Lam, W. C. Shiu, W. H. Chan, Characterization of graphs with equal bandwidth and cyclic bandwidth, Discrete Mathematics 242 (1-3) (2002) 283–289.

[10] P. C. B. Lam, W. C. Shiu, W. H. Chan, On bandwidth and cyclic bandwidth of graphs, Ars Combinatoria 47.

[11] H. Romero-Monsivais, E. Rodriguez-Tello, G. Ramírez, A new branch and bound algorithm for the cyclic bandwidth problem, in: Mexican International Conference on Artificial Intelligence, Springer, 2012, pp. 139–150.

[12] S. Zhou, Bounding the bandwidths for graphs, Theoretical Computer Science 249 (2) (2000) 357–368.

[13] W. H. Chan, P. C. Lam, W. C. Shiu, Cyclic bandwidth with an edge added, Discrete Applied Mathematics 156 (1) (2008) 131–137.

[14] E. De Klerk, M. E.-Nagy, R. Sotirov, On semidefinite programming bounds for graph bandwidth, Optimization Methods and Software 28 (3) (2013) 485–500.

[15] E. Rodriguez-Tello, H. Romero-Monsivais, G. Ramirez-Torres, F. Lardeux, Tabu search for the cyclic bandwidth problem, Computers & Operations Research 57 (2015) 17–32.

[16] L. H. Harper, Optimal assignments of numbers to vertices, Journal of the Society for Industrial and Applied Mathematics 12 (1) (1964) 131–135.

[17] J. Ren, J. K. Hao, E. Rodriguez-Tello, An iterated three-phase search approach for solving the cyclic bandwidth problem, IEEE Access 7 (2019) 98436–98452.

[18] J. Ren, J. K. Hao, E. Rodriguez-Tello, A study of recombination operators for the cyclic bandwidth problem, In L. Idoumghar et al (Eds): Selected and revised papers from Artificial Evolution (EA 2019), Lecture Notes in Computer Science 12052 (2020) 1–15.

[19] P. Larrañaga, C.M.H. Kuijpers, R.H. Murga, I. Inza, S. Dizdarevic, Genetic algorithms for the travelling salesman problem: a review of representations and operators, Artificial Intelligence Review 13 (1999) 129–170.

[20] H. R. Lourenço, O. C. Martin, T. Stützle, Iterated local search, in: Handbook of metaheuristics, Springer, 2003, pp. 320–353.

[21] Z. H. Fu, J. K. Hao, Knowledge-guided local search for the prize-collecting steiner tree problem in graphs, Knowledge-Based Systems 128 (2017) 78–92.

[22] A. Grosso, A. R. M. J. U. Jamali, M. Locatelli, Finding maximin latin hypercube designs by Iterated Local Search heuristics, European Journal of Operational Research, 197 (2) (2009) 541–547.

[23] D. Meignan, S. Knust, A neutrality-based iterated local search for shift scheduling optimization and interactive reoptimization, European Journal of Operational Research, 279 (2) (2019) 320–334.

[24] Y. Zhou, J. K. Hao, An iterated local search algorithm for the minimum differential dispersion problem, Knowledge-Based Systems 125 (2017) 26–38.

[25] F. Glover, M. Laguna, Tabu search, Springer Science+Business Media New York, 1997.

[26] M. López-Ibáñez, J. Dubois-Lacoste, T. Stützle, M. Birattari, The irace package, iterated race for automatic algorithm configuration, Tech. Rep. TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium, published in Operations Research Perspectives (2011).

[27] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, T. Stützle, The irace package: Iterated racing for automatic algorithm configuration, Operations Research Perspectives 3 (2016) 43–58.

[28] L. Smithline, Bandwidth of the complete k-ary tree, Discrete Mathematics 142 (1-3) (1995) 203–212.

A Detailed performance evaluation

This appendix presents the detailed results of the proposed $NILS$ algorithm and the two reference algorithms (TS_{CB} [15] and $ITPS$ [17]). Table A.1 shows the results for the 85 standard graphs with known optimal solutions, while Table A.2 concerns the 28 Harwell-Boeing graphs arising from engineering applications. In these tables, columns 1 to 3 indicate the name, number of vertices ($|V|$) and number of edges ($|E|$) of each instance. Column Cb^* shows the known optimal cost from the literature [4–6, 28], while the theoretical lower (L_B) and upper (U_B) bounds for the instances (Table A.2) are computed according to the formulas $L_B = \lceil \Delta(G)/2 \rceil$ and $U_B = \lfloor |V|/2 \rfloor$, where $\Delta(G)$ is the maximum degree of the graph [8]. The remaining columns present, for each algorithm, the best (Cb_b), average (Avg. Cb) and standard deviation ($Dev.$) of the cyclic bandwidth cost attained in 50 independent runs, the computation time needed to reach this cost (Avg. T_b), and the variation (D) between its best result (Cb_b) and the corresponding best-known bound (either Cb^* or L_B depending on the type of graph). A statistical significance analysis comparing $NILS$ against TS_{CB} [15] and $ITPS$ [17] was executed. The resulting p-values (marked as 1 and 2) as well as the final outcome of the statistical comparison are presented in the last four columns. A symbol + or − indicates respectively that $NILS$ offers a significant better or worse performance than the reference algorithms. A $\star$ symbol indicates implies that it is not possible to conclude a statistically significant difference between the compared algorithms.

Table A.1: Detailed performance assessment of the $NILS$ algorithm with respect to two state-of-the-art methods: TS_{CB} [15] and $ITPS$ [17]. It comprises a total of 85 instances with known optimal solution values belonging to 7 different standard topologies (paths, cycles, two dimensional meshes, three dimensional meshes, complete r-level k-ary trees, caterpillars and r-dimensional hypercubes).

				TS_{CB}					$ITPS$					$NILS$								
Graph	$\|V\|$	$\|E\|$	Cb^*	Cb_b	Avg. Cb	Dev.	Avg. T_b	D	Cb_b	Avg. Cb	Dev.	Avg. T_b	D	Cb_b	Avg. Cb	Dev.	Avg. T_b	D	p-value1	SS_1	p-value2	SS_2
path20	20	19	1	1	1.00	0.00	0.15	0	1	1.00	0.00	0.05	0	1	1.00	0.00	0.00	0	1.00E+00	$\star$	1.00E+00	$\star$
path25	25	24	1	1	1.00	0.00	0.58	0	1	1.00	0.00	0.13	0	1	1.00	0.00	0.00	0	1.00E+00	$\star$	1.00E+00	$\star$
path30	30	29	1	1	1.00	0.00	1.62	0	1	1.00	0.00	0.19	0	1	1.00	0.00	0.02	0	1.00E+00	$\star$	1.00E+00	$\star$
path35	35	34	1	1	1.00	0.00	3.61	0	1	1.00	0.00	0.32	0	1	1.00	0.00	0.08	0	1.00E+00	$\star$	1.00E+00	$\star$
path40	40	39	1	1	1.00	0.00	4.28	0	1	1.00	0.00	0.46	0	1	1.00	0.00	0.19	0	1.00E+00	$\star$	1.00E+00	$\star$
path100	100	99	1	1	1.00	0.00	66.42	0	1	1.00	0.00	7.08	0	1	1.00	0.00	0.73	0	1.00E+00	$\star$	1.00E+00	$\star$
path125	125	124	1	1	1.02	0.14	174.03	0	1	1.00	0.00	9.35	0	1	1.00	0.00	1.54	0	3.17E-01	$\star$	1.00E+00	$\star$
path150	150	149	1	1	1.46	0.50	271.56	0	1	1.00	0.00	26.79	0	1	1.00	0.00	1.36	0	5.39E-08	+	1.00E+00	$\star$
path175	175	174	1	1	1.70	0.46	403.93	0	1	1.00	0.00	52.52	0	1	1.00	0.00	1.72	0	2.85E-13	+	1.00E+00	$\star$
path200	200	199	1	1	1.94	0.24	324.70	0	1	1.00	0.00	106.34	0	1	1.00	0.00	1.98	0	7.27E-21	+	1.00E+00	$\star$
path300	300	299	1	2	3.00	0.35	186.25	1	1	1.10	0.36	205.97	0	1	1.00	0.00	4.24	0	3.13E-22	+	4.23E-02	+
path475	475	474	1	5	5.60	0.49	105.16	4	1	2.48	1.25	429.42	0	1	1.00	0.00	8.86	0	5.37E-21	+	8.85E-12	+
path650	650	649	1	6	6.94	0.31	133.60	5	3	6.92	3.64	417.44	2	1	1.00	0.00	15.59	0	2.18E-22	+	2.69E-20	+
path825	825	824	1	7	7.92	0.40	179.72	6	4	13.78	6.26	531.12	3	1	1.00	0.00	23.88	0	5.99E-22	+	3.11E-20	+
path1000	1000	999	1	8	8.90	0.58	122.17	7	9	22.08	5.56	586.12	8	1	1.00	0.00	33.43	0	1.36E-21	+	3.10E-20	+
cycle20	20	20	1	1	1.00	0.00	0.22	0	1	1.00	0.00	0.03	0	1	1.00	0.00	0.01	0	1.00E+00	$\star$	1.00E+00	$\star$
cycle25	25	25	1	1	1.00	0.00	0.60	0	1	1.00	0.00	0.08	0	1	1.00	0.00	0.11	0	1.00E+00	$\star$	1.00E+00	$\star$
cycle30	30	30	1	1	1.00	0.00	0.42	0	1	1.00	0.00	0.16	0	1	1.00	0.00	0.18	0	1.00E+00	$\star$	1.00E+00	$\star$
cycle35	35	35	1	1	1.00	0.00	0.82	0	1	1.00	0.00	0.30	0	1	1.00	0.00	0.39	0	1.00E+00	$\star$	1.00E+00	$\star$
cycle40	40	40	1	1	1.00	0.00	0.81	0	1	1.00	0.00	0.32	0	1	1.00	0.00	0.81	0	1.00E+00	$\star$	1.00E+00	$\star$
cycle100	100	100	1	1	1.00	0.00	2.51	0	1	1.34	0.80	39.73	0	1	1.00	0.00	2.08	0	1.00E+00	$\star$	3.36E-03	+
cycle125	125	125	1	1	1.00	0.00	4.58	0	1	1.46	1.03	30.85	0	1	1.00	0.00	2.37	0	1.00E+00	$\star$	1.78E-03	+
cycle150	150	150	1	1	1.00	0.00	9.15	0	1	1.86	1.28	87.66	0	1	1.00	0.00	3.25	0	1.00E+00	$\star$	7.36E-06	+
cycle175	175	175	1	1	1.00	0.00	12.49	0	1	2.40	1.62	100.70	0	1	1.00	0.00	4.04	0	1.00E+00	$\star$	3.21E-08	+
cycle200	200	200	1	1	1.00	0.00	18.12	0	1	2.48	1.58	125.42	0	1	1.00	0.00	5.24	0	1.00E+00	$\star$	2.26E-09	+
cycle300	300	300	1	1	2.86	0.67	244.25	0	1	3.18	1.93	314.64	0	1	1.00	0.00	9.81	0	5.65E-19	+	3.40E-12	+
cycle475	475	475	1	4	5.56	0.54	33.94	3	3	5.28	2.71	351.38	2	1	1.00	0.00	11.77	0	6.16E-21	+	2.30E-20	+
cycle650	650	650	1	6	7.00	0.40	47.56	5	4	8.08	2.75	403.19	3	1	1.00	0.00	23.27	0	6.02E-22	+	2.88E-20	+

Continued on next page ...

Table A.1 – Continued from previous page

				TS_{CB}					$ITPS$					$NILS$								
Graph	$\|V\|$	$\|E\|$	Cb^*	Cb_b	Avg. Cb	Dev.	Avg. T_b	D	Cb_b	Avg. Cb	Dev.	Avg. T_b	D	Cb_b	Avg. Cb	Dev.	Avg. T_b	D	p-value1	SS_1	p-value2	SS_2
cycle825	825	825	1	7	7.96	0.28	85.64	6	7	14.32	4.65	472.48	6	1	1.00	0.00	30.99	0	1.49E-22	+	3.10E-20	+
cycle1000	1000	1000	1	8	8.76	0.56	149.60	7	14	25.76	7.63	514.27	13	1	1.00	0.00	46.43	0	3.03E-21	+	3.13E-20	+
mesh2D5x4	20	31	4	4	4.00	0.00	3.21	0	4	4.00	0.00	0.06	0	4	4.00	0.00	0.01	0	1.00E+00	⋆	1.00E+00	⋆
mesh2D5x5	25	40	5	5	5.00	0.00	2.83	0	5	5.00	0.00	0.04	0	5	5.00	0.00	0.00	0	1.00E+00	⋆	1.00E+00	⋆
mesh2D5x6	30	49	5	5	5.00	0.00	0.97	0	5	5.00	0.00	0.12	0	5	5.00	0.00	0.01	0	1.00E+00	⋆	1.00E+00	⋆
mesh2D5x7	35	58	5	5	5.00	0.00	1.29	0	5	5.00	0.00	0.16	0	5	5.00	0.00	0.00	0	1.00E+00	⋆	1.00E+00	⋆
mesh2D5x8	40	67	5	5	5.00	0.00	2.05	0	5	5.00	0.00	50.23	0	5	5.00	0.00	0.01	0	1.00E+00	⋆	1.00E+00	⋆
mesh2D10x10	100	180	10	10	10.50	0.51	129.13	0	10	10.74	0.44	215.13	0	10	10.00	0.00	0.05	0	9.22E-09	+	2.44E-14	+
mesh2D5x25	125	220	5	5	5.00	0.00	19.17	0	6	6.00	0.00	1.30	1	5	5.00	0.00	0.58	0	1.00E+00	⋆	2.53E-23	+
mesh2D10x15	150	275	10	10	10.90	0.30	97.97	0	11	11.00	0.00	3.87	1	10	10.00	0.00	0.22	0	2.26E-19	+	2.53E-23	+
mesh2D7x25	175	318	7	7	7.02	0.14	80.22	0	8	8.00	0.00	6.18	1	7	7.00	0.00	1.00	0	3.17E-01	⋆	2.53E-23	+
mesh2D8x25	200	367	8	8	8.12	0.33	86.30	0	9	9.00	0.00	9.72	1	8	8.00	0.00	0.72	0	1.19E-02	+	2.53E-23	+
mesh2D15x20	300	565	15	16	23.08	19.37	136.00	1	16	16.60	0.49	237.19	1	15	15.00	0.00	2.29	0	3.10E-22	+	5.37E-21	+
mesh2D19x25	475	906	19	119	119.96	0.20	499.62	100	20	20.92	0.27	54.90	1	19	19.00	0.00	8.91	0	6.48E-23	+	1.49E-22	+
mesh2D25x26	650	1249	25	164	164.00	0.00	15.98	139	26	27.30	3.32	328.53	1	25	25.00	0.00	31.20	0	2.53E-23	+	3.27E-21	+
mesh2D28x30	840	1622	28	30	184.94	62.74	592.36	2	29	63.32	69.53	404.77	1	28	28.00	0.00	55.00	0	4.40E-22	+	2.26E-20	+
mesh2D20x50	1000	1930	20	22	184.26	101.62	500.71	2	22	39.86	54.26	380.64	2	20	20.00	0.00	56.72	0	3.47E-21	+	1.75E-20	+
mesh3D4	64	300	14	14	15.68	0.71	274.32	0	14	14.00	0.00	18.11	0	14	14.00	0.00	0.45	0	9.45E-18	+	1.00E+00	⋆
mesh3D5	125	540	21	21	23.02	3.37	91.05	0	21	21.00	0.00	60.22	0	21	21.00	0.00	0.52	0	3.62E-16	+	1.00E+00	⋆
mesh3D6	216	882	30	30	32.34	5.79	270.53	0	30	30.00	0.00	33.23	0	30	30.00	0.00	1.95	0	5.65E-19	+	1.00E+00	⋆
mesh3D7	343	1344	40	41	47.14	14.88	277.81	1	40	60.40	22.92	318.54	0	40	40.00	0.00	5.52	0	6.22E-21	+	1.25E-10	+
mesh3D8	512	1344	52	53	114.30	30.51	474.67	1	52	104.36	35.95	172.07	0	52	52.00	0.00	23.46	0	1.26E-20	+	9.08E-14	+
mesh3D9	729	1944	65	68	180.38	22.36	305.51	3	65	157.40	48.71	112.03	0	65	65.00	0.00	71.37	0	4.37E-22	+	5.54E-21	+
mesh3D10	1000	2700	80	252	252.98	0.47	311.48	172	80	216.92	68.70	378.08	0	80	80.02	0.14	191.61	0	2.10E-21	+	1.89E-20	+
mesh3D11	1331	3630	96	336	336.50	0.51	331.17	240	119	325.38	41.94	589.17	23	96	111.44	46.62	287.46	0	8.29E-19	+	4.43E-19	+
mesh3D12	1728	4752	114	435	436.00	0.53	507.51	321	433	433.48	0.50	466.11	319	114	147.96	73.83	340.38	0	9.47E-19	+	1.54E-17	+
mesh3D13	2197	6084	133	553	554.22	0.74	439.16	420	551	553.16	0.91	518.93	418	133	385.58	195.48	405.94	0	8.15E-19	+	1.21E-18	+
tree2x4	31	30	4	4	4.00	0.00	0.96	0	4	4.00	0.00	0.00	0	4	4.00	0.00	0.00	0	1.00E+00	⋆	1.00E+00	⋆
tree3x3	40	39	7	7	7.00	0.00	0.42	0	7	7.00	0.00	0.00	0	7	7.00	0.00	0.00	0	1.00E+00	⋆	1.00E+00	⋆
tree10x2	111	110	28	28	28.00	0.00	0.25	0	28	28.00	0.00	0.00	0	28	28.00	0.00	0.00	0	1.00E+00	⋆	1.00E+00	⋆
tree3x4	121	120	15	15	15.70	0.46	161.53	0	15	15.00	0.00	0.53	0	15	15.00	0.00	0.02	0	2.85E-13	+	1.00E+00	⋆
tree5x3	156	155	26	26	26.00	0.00	10.39	0	26	26.00	0.00	0.06	0	26	26.00	0.00	0.02	0	1.00E+00	⋆	1.00E+00	⋆

Continued on next page ...

Table A.1 – Continued from previous page

				TS_{CB}					$ITPS$					$NILS$								
Graph	$\|V\|$	$\|E\|$	Cb^*	Cb_b	Avg. Cb	Dev.	Avg. T_b	D	Cb_b	Avg. Cb	Dev.	Avg. T_b	D	Cb_b	Avg. Cb	Dev.	Avg. T_b	D	p-value1	SS_1	p-value2	SS_2
tree13x2	183	182	46	46	46.00	0.00	0.31	0	46	46.00	0.00	0.01	0	46	46.00	0.00	0.01	0	1.00E+00	$\star$	1.00E+00	$\star$
tree2x7	255	254	19	19	20.00	0.20	47.33	0	19	19.00	0.00	1.00	0	19	19.00	0.00	0.50	0	2.81E-22	+	1.00E+00	$\star$
tree17x2	307	306	77	77	77.00	0.00	0.54	0	77	77.00	0.00	0.07	0	77	77.00	0.00	0.05	0	1.00E+00	$\star$	1.00E+00	$\star$
tree21x2	463	462	116	116	116.00	0.00	0.80	0	116	116.00	0.00	0.21	0	116	116.00	0.00	0.12	0	1.00E+00	$\star$	1.00E+00	$\star$
tree25x2	651	650	163	163	163.00	0.00	1.08	0	163	163.00	0.00	0.56	0	163	163.00	0.00	0.28	0	1.00E+00	$\star$	1.00E+00	$\star$
tree5x4	781	780	98	98	98.24	0.43	133.63	0	98	98.00	0.00	4.66	0	98	98.00	0.00	0.90	0	2.39E-04	+	1.00E+00	$\star$
tree2x9	1023	1022	57	62	64.16	1.02	553.57	5	57	57.38	0.49	273.19	0	57	57.00	0.00	16.29	0	1.89E-20	+	1.44E-06	+
caterpillar3	9	8	3	3	3.00	0.00	0.00	0	3	3.00	0.00	0.00	0	3	3.00	0.00	0.00	0	1.00E+00	$\star$	1.00E+00	$\star$
caterpillar4	14	13	3	3	3.00	0.00	0.50	0	3	3.00	0.00	0.00	0	3	3.00	0.00	0.00	0	1.00E+00	$\star$	1.00E+00	$\star$
caterpillar5	20	19	4	4	4.00	0.00	0.50	0	4	4.00	0.00	0.00	0	4	4.00	0.00	0.00	0	1.00E+00	$\star$	1.00E+00	$\star$
caterpillar6	27	26	5	5	5.00	0.00	0.61	0	5	5.00	0.00	0.00	0	5	5.00	0.00	0.00	0	1.00E+00	$\star$	1.00E+00	$\star$
caterpillar7	35	34	6	6	6.00	0.00	0.54	0	6	6.00	0.00	0.00	0	6	6.00	0.00	0.00	0	1.00E+00	$\star$	1.00E+00	$\star$
caterpillar13	104	103	10	10	10.00	0.00	23.33	0	10	10.00	0.00	0.42	0	10	10.00	0.00	0.12	0	1.00E+00	$\star$	1.00E+00	$\star$
caterpillar14	119	118	11	11	11.00	0.00	14.83	0	11	11.00	0.00	0.14	0	11	11.00	0.00	0.17	0	1.00E+00	$\star$	1.00E+00	$\star$
caterpillar16	152	151	13	13	13.00	0.00	12.86	0	13	13.00	0.00	0.39	0	13	13.00	0.00	0.45	0	1.00E+00	$\star$	1.00E+00	$\star$
caterpillar17	170	169	14	14	14.00	0.00	15.93	0	14	14.00	0.00	0.62	0	14	14.00	0.00	0.97	0	1.00E+00	$\star$	1.00E+00	$\star$
caterpillar19	209	208	15	15	15.64	0.48	126.78	0	15	15.00	0.00	3.24	0	15	15.00	0.00	2.68	0	8.76E-12	+	1.00E+00	$\star$
caterpillar23	299	298	19	19	19.26	0.49	85.00	0	19	19.00	0.00	7.11	0	19	19.00	0.00	6.04	0	2.41E-04	+	1.00E+00	$\star$
caterpillar29	464	463	24	24	26.20	1.53	167.98	0	24	24.00	0.00	52.22	0	24	24.00	0.00	26.78	0	9.83E-19	+	1.00E+00	$\star$
caterpillar35	665	664	29	29	33.80	3.33	127.57	0	29	32.68	5.85	235.20	0	29	29.00	0.00	50.47	0	1.08E-19	+	7.34E-08	+
caterpillar39	819	818	33	33	39.80	4.38	230.96	0	33	39.08	8.43	242.27	0	33	33.00	0.00	73.56	0	4.30E-19	+	1.71E-06	+
caterpillar44	1034	1033	37	38	49.02	5.63	322.31	1	37	55.26	10.93	366.44	0	37	37.00	0.00	109.86	0	3.12E-20	+	1.57E-18	+
hypercube11	2048	11264	526	562	585.32	11.08	519.82	36	548	565.64	8.35	577.71	22	535	543.70	3.78	559.33	9	6.36E-18	+	3.88E-26	+
hypercube12	4096	24576	988	1235	1356.86	35.18	523.33	247	1551	1580.96	12.64	597.96	563	1112	1179.28	66.86	599.15	124	1.34E-16	+	6.78E-18	+
hypercube13	8192	53248	1912	2858	2937.64	34.94	595.54	946	3953	3968.74	8.56	598.56	2041	2829	2940.94	32.88	594.15	917	2.44E-01	$\star$	6.64E-18	+
Average				92.18	101.40	4.35	137.87	31.54	101.02	109.48	5.25	142.78	40.39	72.99	78.75	4.94	44.25	12.35				

Table A.2
Detailed performance assessment of the $NILS$ algorithm with respect to two state-of-the-art methods: TS_{CB} [15] and $ITPS$ [17]. It comprises a total of 28 Harwell-Boeing instances coming from real world engineering applications whose theoretical lower (L_B) and upper (U_B) bounds are known. Values in bold indicate improved upper bounds.

			Bounds			TS_{CB}					$ITPS$					$NILS$								
Graph	$\|V\|$	$\|E\|$	L_B	U_B	Cb^*	Cb_b	Avg. Cb	Dev.	Avg. T_b	D	Cb_b	Avg. Cb	Dev.	Avg. T_b	D	Cb_b	Avg. Cb	Dev.	Avg. T_b	D	p-value1	SS_1	p-value2	SS_2
jgl009	9	50	4	4	4	4	4.00	0.00	0.00	0	4	4.00	0.00	0.00	0	4	4.00	0.00	0.00	0	1.00E+00	⋆	1.00E+00	⋆
rgg010	10	76	5	5	5	5	5.00	0.00	0.00	0	5	5.00	0.00	0.00	0	5	5.00	0.00	0.00	0	1.00E+00	⋆	1.00E+00	⋆
jgl011	11	76	5	5	5	5	5.00	0.00	0.00	0	5	5.00	0.00	0.00	0	5	5.00	0.00	0.00	0	1.00E+00	⋆	1.00E+00	⋆
can_24	24	92	4	12	5	5	5.00	0.00	0.02	0	5	5.00	0.00	0.18	0	5	5.00	0.00	0.02	0	1.00E+00	⋆	1.00E+00	⋆
pores_1	30	103	5	15	7	7	7.00	0.00	0.24	0	7	7.00	0.00	0.01	0	7	7.00	0.00	0.01	0	1.00E+00	⋆	1.00E+00	⋆
ibm32	32	90	6	16	9	9	9.00	0.00	0.03	0	9	9.00	0.00	0.02	0	9	9.00	0.00	0.01	0	1.00E+00	⋆	1.00E+00	⋆
bcspwr01	39	46	3	19	4	4	4.16	0.37	174.98	0	4	4.16	0.37	71.84	0	4	4.00	0.00	0.32	0	3.35E-03	+	3.35E-03	+
bcsstk01	48	176	6	24		12	12.00	0.00	0.03	6	12	12.00	0.00	0.17	6	12	12.00	0.00	0.02	6	1.00E+00	⋆	1.00E+00	⋆
bcspwr02	49	59	3	24		7	7.00	0.00	0.01	4	7	7.00	0.00	0.04	4	7	7.00	0.00	0.00	4	1.00E+00	⋆	1.00E+00	⋆
curtis54	54	124	8	27		8	8.00	0.00	0.48	0	8	8.00	0.00	0.61	0	8	8.00	0.00	0.13	0	1.00E+00	⋆	1.00E+00	⋆
will57	57	127	5	28		6	6.00	0.00	12.93	1	6	6.00	0.00	0.80	1	6	6.00	0.00	0.20	1	1.00E+00	⋆	1.00E+00	⋆
impcol_b	59	281	9	29		17	17.00	0.00	0.56	8	17	17.00	0.00	0.13	8	17	17.00	0.00	0.02	8	1.00E+00	⋆	1.00E+00	⋆
ash85	85	219	5	42		9	9.00	0.00	54.35	4	9	9.00	0.00	1.10	4	9	9.00	0.00	0.10	4	1.00E+00	⋆	1.00E+00	⋆
nos4	100	247	3	50		10	10.00	0.00	0.97	7	10	10.00	0.00	0.54	7	10	10.00	0.00	0.04	7	1.00E+00	⋆	1.00E+00	⋆
dwt_234	117	162	5	58		11	11.98	0.14	487.74	6	11	11.00	0.00	2.01	6	11	11.00	0.00	0.39	6	1.79E-22	+	1.00E+00	⋆
bcspwr03	118	179	5	59		11	11.00	0.00	12.92	6	10	10.00	0.00	15.39	5	10	10.00	0.00	2.37	5	2.53E-23	+	1.00E+00	⋆
bcsstk06	420	3720	14	210		49	49.74	0.53	246.86	35	45	45.00	0.00	215.88	31	45	45.00	0.00	40.01	31	4.26E-21	+	1.00E+00	⋆
bcsstk07	420	3720	14	210		49	49.74	0.53	247.87	35	45	45.00	0.00	218.09	31	45	45.00	0.00	40.51	31	4.26E-21	+	1.00E+00	⋆
impcol_d	425	1267	8	212		37	38.68	0.51	29.58	29	35	44.18	6.22	91.20	27	35	35.00	0.00	56.08	27	3.03E-21	+	1.07E-19	+
can_445	445	1682	6	222		46	46.96	0.20	24.92	40	46	84.22	36.39	372.86	40	46	46.00	0.00	36.69	40	1.18E-21	+	4.85E-18	+
494_bus	494	586	5	247		35	38.72	1.67	15.23	30	36	51.76	5.16	310.62	31	**28**	28.96	0.28	318.51	23	9.80E-20	+	3.25E-20	+
dwt_503	503	2762	12	251		45	45.90	0.58	191.34	33	41	60.24	8.12	272.11	29	41	41.00	0.00	33.40	29	2.71E-21	+	4.35E-18	+
sherman4	546	1341	3	273		27	28.28	0.54	377.31	24	27	27.62	0.49	187.02	24	27	27.00	0.00	10.39	24	7.44E-20	+	2.57E-11	+
dwt_592	592	2256	7	296		32	32.68	0.55	301.44	25	29	48.76	50.41	544.67	22	29	29.00	0.00	25.66	22	6.02E-21	+	7.58E-20	+
662_bus	662	906	5	331		59	66.76	3.73	243.33	54	75	89.30	6.09	530.51	70	**38**	40.74	3.09	274.47	33	4.22E-18	+	2.33E-18	+
nos6	675	1290	2	337		19	20.32	0.68	249.01	17	17	21.16	5.91	212.15	15	**16**	16.00	0.00	74.95	14	1.09E-20	+	1.11E-20	+
685_bus	685	1282	6	342		34	40.10	3.18	217.07	28	73	102.88	12.14	443.50	67	**32**	32.00	0.00	64.00	26	2.93E-20	+	1.75E-20	+
can_715	715	2975	52	357		60	60.88	0.52	266.46	8	60	183.14	73.88	463.40	8	60	60.00	0.00	161.11	8	9.89E-16	+	2.70E-20	+
Average						22.21	23.21	0.49	112.70	14.29	23.50	33.30	7.33	141.24	15.57	20.39	20.53	0.12	40.69	12.46				