# Optimization algorithms for graph layout problems

Jintong Ren

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

université angers

# THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE ANGERS
COMUE UNIVERSITÉ BRETAGNE LOIRE

ÉCOLE DOCTORALE N° 601
*Mathématiques et Sciences et Technologies de l'Information et de la Communication*
Spécialité : *Informatique*

Par

**Jintong REN**

## Optimization algorithms for graph layout problems

**Thèse présentée et soutenue à « LERIA », le « 15 Juillet 2020 »**
**Unité de recherche :**
**Thèse N° :**

### Rapporteurs avant soutenance :

| | |
|---|---|
| M. Nicolas Durand | Professeur à Ecole Nationale de l'Aviation Civile |
| Mme. Yang Wang | Professeur à Northwestern Polytechnical University |

### Composition du Jury :

| | | |
|---|---|---|
| Président : | M. David Lesaint | Professeur à Université d'Angers |
| Examinateurs : | M. Nicolas Durand | Professeur à Ecole Nationale de l'Aviation Civile |
| | M. Jin-Kao Hao | Professeur à Université d'Angers |
| | M. David Lesaint | Professeur à Université d'Angers |
| | M. Eduardo Rodriguez-Tello | Professeur à CINVESTAV, Tamaulipas |
| | Mme. Yang Wang | Professeur à Northwestern Polytechnical University |
| Dir. de thèse : | M. Jin-Kao Hao | Professeur à Université d'Angers |

# Acknowledgment

How time flies! I began my PhD study in Leria four years ago and I have finished all today. From a beginner who knows nothing about the optimization algorithms to a researcher who really enjoys the study of hard problems, I cannot express my thanks to all the people who helped me in just few lines.

I'm extremely grateful to my supervisor Prof. Jin-Kao Hao who provided me with encouragement and patience throughout the duration of my thesis. During the past four years, he gave me valuable guidance and advice not only on the research field but also on the learning attitude. I would like to extend my sincere thanks to my monitoring committee Prof. Eduardo Rodriguez-Tello who I cooperated a lot during the four years. Thanks for his advice on the optimization methods and his kind invitation, it is sure that I will visit the pyramid in Mexico in the future.

I am also grateful to the jury members. Thanks for Prof. David Lesaint who manages my thesis defense and gave me much valuable advice in the future research. I'd also like to extend my gratitude to my referees Prof. Nicolas Durand and Prof. Yang Wang who reviewed my dissertation during this special period of Covid-19. I was quite enjoying the discussion with them and their detailed and difficult questions inspired me to do more theoretical work in the future.

I'd like to acknowledge the assistance of our technicians Eric Girardeau and Jean-Mathieu Chantrein, our nice secretaries Catherine Pawlonski and Christine Bardarine and all the other lab members. Thanks to them, I have enjoyed my life in Angers and had much fun in the daily life.

I am particularly grateful to my parents and my girlfriend. Thanks for all their love and selfless support! Without them, this work would not be possible. I also thank my Chinese friends Zequn Wei, Zhi Lu, Qing Zhou, Yinuo Li, Pengfei He and Zuocheng Li in Angers for their accompanies.

This research has been financially supported by China Scholarship Council (CSC).

# Table of Contents

# General Introduction

## Context

Graph layout problems are a particular class of combinatorial optimization problems. Graph layout problems aim to find a linear layout of a given graph with respect to predefined optimization criteria. There are many problems which can be formulated as graph layout problems: bandwidth problem, cutwidth problem, sum cut problem, minimum linear arrangement problem and so on ([DPS02]). In this thesis, we focus on two problems: the cyclic bandwidth problem (CBP) [LVW84] and the minimum linear arrangement problem (MinLA) [Har64]. As a variant of the bandwidth problem, CBP is to find an arrangement on a cycle to minimize the longest cyclic distance. MinLA is a min-sum problem whose goal is to find a linear arrangement of an input graph to minimize the sum of all edges. The study of these two problems is meaningful for many fields such as very large-scale integration (VLSI) circuit design [BT84; Har64], bioinformatics [DPS02] and code designs [Chu88]. Given their importance, the research on these problems becomes more and more intense in recent years.

However, both problems are known to be $\mathcal{NP}$-hard [Har82; Lin94]. It is unlikely to find the global optimal solution in a polynomial time unless $P = NP$. Even for small instances, exact algorithms would consume much time to find the optimal solution. But for most of their applications, feasible solutions of good quality are sufficient. Therefore, it is meaningful to develop effective heuristic and metaheuristic algorithms to find such solutions in a reasonable time. This thesis aims to develop effective heuristic algorithms for these problems. To assess the proposed algorithms, extensive experiments are carried out over available benchmarks and comparisons with state-of-the-art methods in the literature are shown. Moreover, we also investigate the key components of the proposed algorithms to reveal their influences on the performances of the algorithms.

## Objectives

The main objectives of this thesis are summarized as follows:

— Understand deeply CBP and MinLA and analyse the main difficulty of developing heuristic algorithms for these problems.
— Explore effective and meaningful neighborhoods to operate efficiently in the local search procedure.
— Design problem-specific evaluation functions to help distinguish the solutions which have same objective value.
— Develop effective perturbation strategies to help escape from the local optimum traps.
— Investigate different recombination operators for the memetic algorithm (MA) to find out the feature for a suitable crossover.
— Evaluate the proposed algorithms over well-known benchmark instances and carry out a comprehensive comparison with the state-of-the-art algorithms.

# Contributions

The main contributions of this thesis are summarized below:

— For CBP, we achieved the following results:
  — Firstly, we proposed an iterated three-phase search approach ($ITPS$). The 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. This work has been published in *IEEE Access*.
  — Secondly, we proposed a new iterated local search algorithm ($NILS$). 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 from local optimum traps and explore unvisited areas. We present computational results on 113 benchmark instances from 8 different families, and show performances that are never achieved by current best algorithms in the literature. This study was presented in a paper submitted to *Knowledge-Based Systems*.
  — Thirdly, we investigated five classical permutation crossovers within a basic memetic algorithm integrating a simple descent local search procedure. We

studied the correlation between algorithmic performances and population diversity measured by the average population distance and entropy. This work has been selected by the conference Artificial Evolution 2019 and published in *Lecture Notes in Computer Science*.

— For MinLA, we studied a set based neighborhood heuristic algorithm under the framework of two phase iterated local search. The main contribution is the introduction of a set based neighborhood with a decomposition method in the descent phase. We compared the proposed algorithm with a traditional neighborhood heuristic algorithm, and computational results show that the set based neighborhood performs better than the traditional neighborhood in the local search phase.

# Organization

The thesis is organized in the following way:

— In the first chapter, after introducing CBP and MinLA, we present related applications and a brief overview of existing algorithms. The test benchmark and the computational assessments are shown at the end of the chapter.

— In the second chapter, we study the cyclic bandwidth problem and make a brief review of the existing algorithms. Then, we introduce the iterated three-phase search ($ITPS$) approach. We present experimental results of the proposed algorithm, as well as comparisons with state-of-the-art algorithms.

— In the third chapter, we propose another heuristic algorithm for CBP: the new iterated local search algorithm ($NILS$). After introducing the simple but powerful neighborhood, the directed perturbation phase and special strong perturbation phases are presented. Then, experimental results of $NILS$ and the state-of-the-art algorithms including $ITPS$ and $TScb$ are shown over benchmark instances.

— In the fourth chapter, we focus on the study of different recombination operators for CBP. The general outline of the memetic algorithm for CBP is first described and 5 different crossovers are presented. Afterwards, computational results of the memetic algorithm with different crossovers are presented. Moreover, we investigate the correlation between the performance and the diversity of population.

— In the last chapter, we consider the minimum linear arrangement problem. We present the set based neighborhood with the decomposition method under the framework of two phase iterated heuristic algorithm. Comparative results with the

algorithm using the traditional 2-flip neighborhood as well as the state-of-the-art algorithm are presented.

Part I

# Introduction

Chapter 1

# Introduction

## 1.1 Graph layout problems

Graph layout problems (GLP) are a class of combinatorial optimization problems. Starting from the bandwidth minimization problem (BMP) [Har64] in the 1960s, GLP are to find a linear layout of an input graph in such a way that a certain objective function is optimized. Because of the strong application background, there are many theoretical and practical studies on GLP in the past decades. In this thesis, we mainly focus on two GLP: the cyclic bandwidth problem (CBP) and the minimum linear arrangement problem (MinLA).

— **The cyclic bandwidth problem (CBP)**.
Let $G(V, E)$ be a finite undirected graph of order $n$ and $C_n$ a cycle graph where $V$ represents the set of vertices and $E$ depicts the set of edges. Given a bijection $\varphi : V \to V$ which represents an embedding (also called a labeling) of $G$ in $C_n$, the cyclic bandwidth (the cost) for $G$ with respect to $\varphi$ is defined as:

$$\mathrm{Cb}(G, \varphi) = \max_{(u,v)\in E}\{|\varphi(u) - \varphi(v)|_n\}, \tag{1.1}$$

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 $\varphi$ can be computed as follows:

$$\mathrm{Cb}(u, \varphi) = \max_{v\in A(u)}\{|\varphi(u) - \varphi(v)|_n\}\,. \tag{1.2}$$

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

$$\varphi^* = \arg\min_{\varphi\in\Omega}\{\mathrm{Cb}(G, \varphi)\}, \tag{1.3}$$

where $\Omega$ represents the set of all possible embeddings. The embedding $\varphi^*$ satisfying this condition in Equation 1.3 is called an optimal or exact embedding of the given graph.

— **The minimum linear arrangement problem (MinLA)**.
Let $G(V, E)$ be a finite undirected graph, where $V$ ($|V| = n$) represents the set of vertices and $E$ depicts the set of edges. Given a mapping $\varphi : V \to \{1, 2, ...n\}$ which represents a linear arrangement $\varphi$, the sum of edge length (the cost) for $G$

with respect to $\varphi$ is defined as:

$$S_{LA}(G, \varphi) = \sum_{(u,v)\in E} |\varphi(u) - \varphi(v)| \tag{1.4}$$

The objective is to find an arrangement $\varphi^*$ whose the sum of total edge length $S_{LA}(G, \varphi^*)$ is minimal.

CBP and MinLA are both $\mathcal{NP}$-hard problems and their decision problems are known to be $\mathcal{NP}$-complete [Har82; Lin94].

## 1.2 Applications

CBP has many applications in the real world. One of them is to design a ring interconnection network [LVW84] for a couple of computers to ensure every message to be sent at its destination in less than certain steps. Another application is the VLSI design [BT84]. In the circuit layout domain, the maximum delay determines the clock-period of the system in the circuit. To increase the speed of the chip, it is essential to decrease the maximum delay by producing a layout with the longest edge as short as possible. Also, CBP is applied in data structure representations [RS78]. The replacement of the logical data structure with the physical storage structure is unavoidable step in the algorithm implementation procedure. This encoding of data structure could be seen as a match between a “logical" guest structure and a “physical" host structure. An optimal solution for an input graph will offer a good match for the encoding which will improve the efficiency of the computers. Moreover, CBP is involved in the interconnection networks for parallel computer systems [Hro+92].

MinLA is a min-sum problem whose objective is to find a linear arrangement to minimize the sum of edge length for an input graph. It is widely used in many fields. First of all, the optimal solution of MinLA can derive the lower bound and upper bound for the bipartite crossing number problem [Sha+00]. Concerning the design of the error-correcting codes [Har64], the application of MinLA will help minimize the average absolute error in message transmission. Also, in the field of VLSI, the vertices represent the pin on the chip while the edges depict the wire between the pin. The cost of the arrangement stands for the total wire length [AH73]. In addition, MinLA has other applications in biological fields, graph drawing, software diagram layout and job scheduling.

## 1.3 Algorithms of the cyclic bandwidth problem and the minimum linear arrangement problem

The existing algorithms for CBP and MinLA fall into two categories: the exact algorithms and the heuristic algorithms.

For CBP, the majority of the studies focused on theoretical work: getting the exact value for certain family of graphs [LSC97; LSC02; Lin97] and finding lower bounds for the general graphs [CLS08; KNS11; Zho00]. There are few practical algorithms to solve CBP. To our knowledge, there is only one exact algorithm, i.e. a branch and bound algorithm in [RRR12] which can solve small problem instances. Concerning the heuristic algorithms, a heuristic algorithm based on tabu search was presented in [Rod+15] to handle large and general instances.

For MinLA, there exist some exact methods to get the optimal solution for some special families of graphs such as trees, rooted trees, hypercubes, meshes, outerplanar graphs, and others ([DPS02]). Besides that, several heuristic algorithms were developed such as the spectral sequencing method (SSQ) [JM92], improved frontal increase minimization (IFIM) [Mca99], multi-scale algorithm (MS) [KH02], algebraic multigrid scheme (AMG) [SRB04], simulated annealing (SA) [Pet03a][Pet03b][RHT08a], population-based algorithms [RHT06] [Por05][SS09] and variable neighborhood search [MUP16].

The detailed review of previous work will be made for each considered problem in the following chapters.

## 1.4 Algorithm assessment

The assessment of the performance of an algorithm is based on experimental results over benchmark instances. In this section, we will make a brief introduction of the benchmark instances and some indicators describing the algorithm performances.

### 1.4.1 Benchmarks

For CBP, the benchmark graphs are organized in two different groups.

— **Standard Graphs.** The first group 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 optimal solutions for these graphs are known, the reader is referred to [Rod+15] for the details. Therefore, attaining the optimal solutions for these instances is an important factor to evaluate the performance of algorithms.

— **Harwell-Boeing Graphs.** The second group contains 28 graphs from the *Harwell-Boeing Sparse Matrix Collection.*[1] These instances were directly constructed from sparse adjacency matrices produced in practical and engineering real world applications. Their order fluctuates in the interval $9 \leq |V| \leq 715$ and their size is in the range $46 \leq |E| \leq 3720$. The optimal solutions for 7 small graphs are already known, while for the remaining 21 graphs lower and upper bounds can be calculated according to [Lin97].

For MinLA, the benchmark instances were introduced in [Pet03b]. There are 21 graphs which are classified into 5 groups.[2].

— **Random Graphs.** This group consists of 5 graphs including 4 random graphs and a random geometric graph. The number of vertices of these graphs is 1000 and the number of edges is between 4974 and 49820.

— **Regular Graphs.** There are 3 graphs in this group: a complete binary tree with 10 levels, a 10-dimension hypercube and a 33×33 mesh graph. It is noticed that the optimal solutions of these graphs are known.

— **FE Graphs.** This group includes 3 graphs from finite element discretization. The numbers of vertices of these graphs are 4720, 4253 and 9800 respectively and the numbers of edges are 13722, 12289 and 28989 respectively.

— **VLSI Graphs.** This is a set of 5 graphs from the VLSI design. Their order fluctuates $828 \leq |V| \leq 1366$ and their size are in the range $1749 \leq |E| \leq 2915$.

— **GD Graphs.** It is composed of 5 graphs from graph drawing competitions. Four of them are in small size which have less than 180 vertices and 228 edges. And the other has 1096 vertices and 1676 edges.

### 1.4.2 Metaheuristic algorithms evaluation

To evaluate the performances of heuristic algorithms, the general method is to compare the results of different algorithms under the same environment over the same benchmark

1. They are downloadable at http://math.nist.gov/MatrixMarket/data/Harwell-Boeing
2. They are downloadable on https://www.tamps.cinvestav.mx/~ertello/minla.php

set. Due to the stochastic nature of heuristic algorithms, a common practice is to run the heuristic algorithm multiple times with different seeds (for example, we run our algorithms 50 times independently for each instance). Then we collect the results of each execution and establish the comparison between the proposed algorithms and the state-of-the-art algorithms. Normally, we consider the best objective value found, the average objective value for each instance, the running time to get the best value and the deviation of the solution quality. In this thesis, we also used other indicators such as the overall relative root mean square error to describe the overall performance of the algorithm. The statistical significance test is implemented to verify whether the differences are statistically significant.

To shed light on the key components and reveal their influences to the performance of the proposed algorithm, additional control experiments are needed. Usually, we create several variants of the proposed algorithm by disabling or replacing some components. The variants will follow the same experiment setting of the proposed algorithm over the benchmark set. This method is widely used in the following chapters and we will make specific presentation for each algorithm.

Part II

# Contribution

Chapter 2

# An iterated three-phase search approach for solving the cyclic bandwidth problem

---

In this chapter, we introduce an iterated three-phase search approach which 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. The content of this chapter has been published in *IEEE Access*.

## 2.1 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 [LVW84], 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 CBP is known to be $\mathcal{NP}$-complete [Lin94]. Also, it has some other important applications in VLSI design [BT84], data structure representations [RS78], code designs [Chu88] and parallel computer systems [Hro+92].

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

$$\mathrm{Cb}(G, \varphi) = \max_{(u,v)\in E} \{|\varphi(u) - \varphi(v)|_n\}, \tag{2.1}$$

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 $\varphi$ can be computed as follows:

$$\mathrm{Cb}(u, \varphi) = \max_{v\in A(u)} \{|\varphi(u) - \varphi(v)|_n\}. \tag{2.2}$$

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

$$\varphi^* = \arg\min_{\varphi\in\Omega} \{\mathrm{Cb}(G, \varphi)\}, \tag{2.3}$$

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

Table 2.1 – Table of cyclic distances for all the edges of graph $G$ depicted in Fig. 2.1.

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

Fig. 2.1 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. 2.2, 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 2.1). 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 (2.1), 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. 2.2 indicated in red.

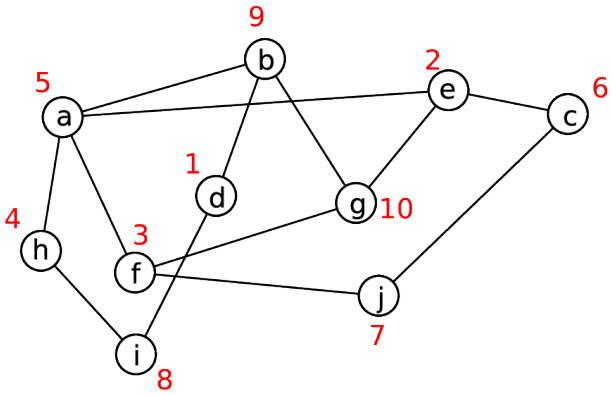



Figure 2.1 – The vertices are named from $a$ to $j$ and a labeling is represented by the red numbers from 1 to 10.

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 [YZ95], it was shown that for every unit interval graph, there exists a simultaneously optimal labeling for several labeling problems including CBP. The study of [Hro+92] 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 [LSC97; LSC02; Lin97] 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 [Zho00], 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 CBP in terms of some distance- and degree-related parameters.

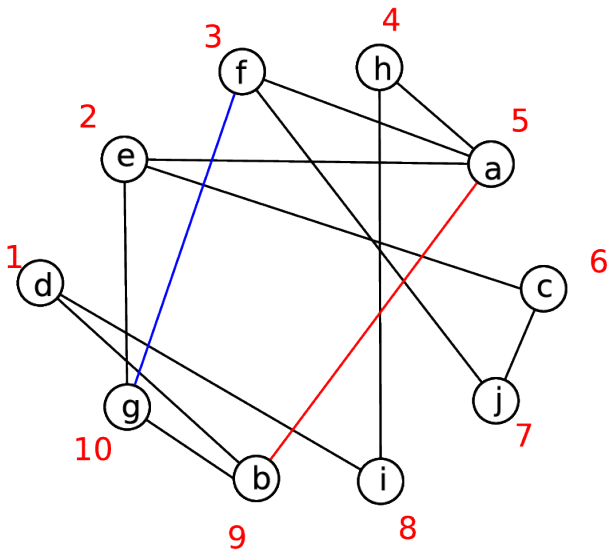



Figure 2.2 – The graph $G$ of Fig. 2.1 with its vertices $a$ to $j$ reordered clockwise on a cycle according to the label numbers 1 to 10 (in red).

The work of [CLS08] 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 [KNS11] 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 CBP. To our knowledge, there are only three published algorithms on solving CBP. In [RRR12], 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 meta-heuristic (named $TScb$) was presented in [Rod+15]. The authors also adapted a highly effective simulated annealing designed for the related Bandwidth Minimization Problem (BMP) [Har64] 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 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., [Mla+10; Piñ+04; RHT08b; Tor+15]), effective algorithms dedicated to 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 CBP. The algorithm is characterized by the following features. First, the algorithm is composed of three key search components: a double neighbor descent 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 chapter is organized as follows: Section 2.2 first introduces the main scheme of the proposed algorithm. Then, the implementation details of the neighbor-based descent procedure as well as the responsive threshold-based search method are presented. In Section 2.3 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, $TScb$ [Rod+15]. Section 2.4 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 in Section 2.5.

## 2.2 Iterated three-phase search for CBP

### 2.2.1 Main scheme

The proposed $ITPS$ algorithm was inspired by the three-phase approach presented in [FH15]. Even if the work of [FH15] 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 [ZHG16]. In this work, we adapted this three-phase approach to 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 $\Omega$ 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 $\varphi$ 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 $\Omega$, $ITPS$ 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.

The pseudo-code of the $ITPS$ algorithm is presented in Algorithm 1. It starts with a randomly generated solution $\varphi$. Then the algorithm enters the main 'while' loop (lines

**Algorithm 1** $ITPS$ algorithm for CBP

1: **Input**: Finite undirected graph $G(V, E)$, neighborhoods $N_1$ and $N_2$, extended evaluation function $f_e$, search depth $\delta$ 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: $\quad NonImp \leftarrow 0$
7: $\quad$ **while** $NonImp < \delta$ **do**
8: $\quad\quad (\varphi, \varphi^*) \leftarrow DNDS(\varphi, \varphi^*, N_1, N_2)$ // Section 2.2.3
9: $\quad\quad (\varphi, \varphi^*) \leftarrow RTBS(\varphi, \varphi^*, N_1, N_2)$ // Section 2.2.4
10: $\quad\quad NonImp \leftarrow NonImp + 1$
11: $\quad$ **end while**
12: $\quad \varphi \leftarrow Purturbation(\varphi)$ // Section 2.2.5
13: **end while**
14: **return** $\varphi^*$

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 2.2.3) 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 2.2.4) to discover additional local optima of better quality from the incumbent solution (line 9, Alg. 1). These two phases are repeated $\delta$ 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 2.2.5) 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, $\varphi^*$ 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 $\varphi^*$ is returned.

### 2.2.2 Extended evaluation function

A notable feature of 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 (2.1). 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 $\mathrm{Cb}(G, \varphi)$:

$$NumE(\mathrm{Cb}(G, \varphi)) = \sum_{(u,v)\in E} X_{uv}\,, \tag{2.4}$$

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:

$$f_e(\varphi) = \mathrm{Cb}(G, \varphi) + \frac{NumE(\mathrm{Cb}(G, \varphi))}{|E|}\,. \tag{2.5}$$

As we show below, this evaluation function is able to distinguish the solutions that under the conventional evaluation function presented in (2.1) have the same objective value. An analysis of the influence of the new evaluation function $f_e$ is provided in Section 2.4.1.

Figure 2.3 shows an example of the extend evaluation function $f_e$ applied to two solu-

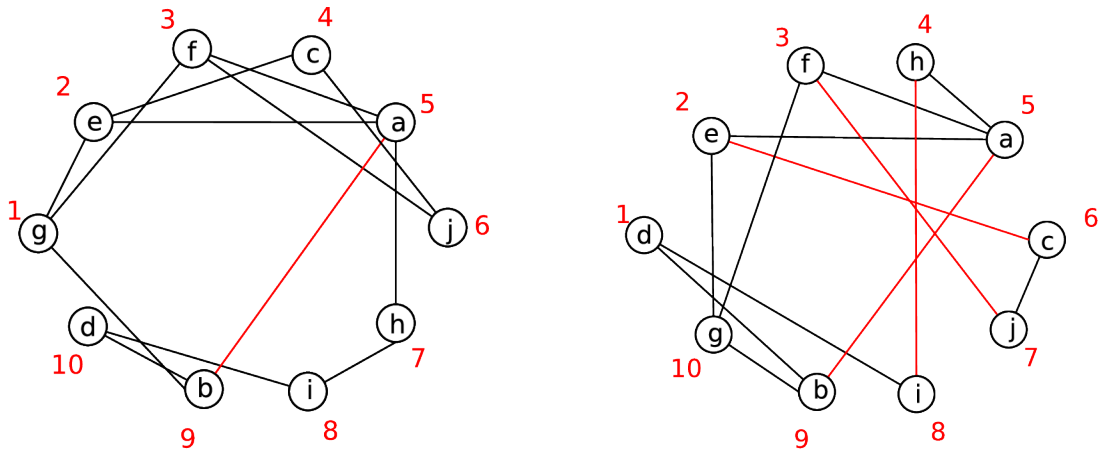



Figure 2.3 – An illustration of the extended evaluation function $f_e$ applied to two different embeddings. (a) $\varphi_1$. (b) $\varphi_2$. Both embeddings have the same cost (cyclic bandwidth) under the conventional evaluation function (2.1). However, the new function $f_e$ discriminates these embeddings by assigning to them two different values $f_e(\varphi_1) = 4 + 1/13 = 4.0769$ and $f_e(\varphi_2) = 4 + 3/13 = 4.2307$.

tions 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 $\varphi_1$ is considered to be "better" than the embedding $\varphi_2$. This is reasonable, because one notices, from Fig. 2.3(a), that for reducing the cost value $\mathrm{Cb}(G, \varphi)$ of the embedding $\varphi_1$ it is necessary to deal with only one edge (marked in red), while for embedding $\varphi_2$, depicted in Fig. 2.3(b), there are four edges (marked in red) that should be considered. Thus, it is easier to operate with $\varphi_1$ than with $\varphi_2$ to reduce the cyclic bandwidth of $G$.

### 2.2.3 First phase - Double neighborhood descent search

To explore the given search space, we first apply the double neighborhood descent 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

**Algorithm 2** Double neighborhood descent search

1: **Input**: input solution $\varphi$, best optimum found $\varphi^*$, neighborhoods $N_1$ and $N_2$, evaluation function $f_e$, maximum non-improving limit $L_d$, and best neighbor move strategy probability $\rho_{best}$
2: **Output**:last local optimum $\varphi$, best optimum found $\varphi^*$
3: $NonImpCounter \leftarrow 0$
4: $Improving \leftarrow True$
5: **while** $NonImpCounter < L_d$ **do**
6:  **if** $Improving$ **then**
7:  $N \leftarrow N_1$
8:  **else**
9:  $N \leftarrow N_2$
10:  **end if**
11:  $CLst \leftarrow \emptyset$
12:  **for** each $\varphi' \in N(\varphi)$ **do**
13:  **if** $f_e(\varphi') \leq f_e(\varphi)$ **then**
14:  $CLst \leftarrow CLst \cup \{\varphi'\}$
15:  **end if**
16:  **end for**
17:  **if** $CLst \neq \emptyset$ **then**
18:  **if** $rand(0,1) < \rho_{best}$ **then**
19:  $\varphi \leftarrow BestSol(CLst)$ // With probability $\rho_{best}$
20:  **else**
21:  $\varphi \leftarrow RandomSol(CLst)$
22:  **end if**
23:  $Improving \leftarrow True$
24:  **else**
25:  $Improving \leftarrow False$
26:  **end if**
27:  **if** $f_e(\varphi) < f_e(\varphi^*)$ **then**
28:  $NonImpCounter \leftarrow 0$
29:  $\varphi^* \leftarrow \varphi$
30:  **else**
31:  $NonImpCounter \leftarrow NonImpCounter + 1$
32:  **end if**
33: **end while**
34: **return** $\varphi$, $\varphi^*$

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 $\rho_{best}$ (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 $\varphi^*$ is updated each time a better solution is attained. The DNDS process terminates if the best-found solution $\varphi^*$ 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.

### Neighborhoods

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

$$\begin{aligned} N_1(\varphi) = \{&\varphi' = \varphi \oplus swap(u, v) : u \in C(\varphi), \\ &v \in S(u), swap(u, v) \notin TL\}, \end{aligned} \tag{2.6}$$

where $\varphi' = \varphi \oplus swap(u, v)$ denotes the neighbor solution obtained by applying $swap(u, v)$ to transform $\varphi$, $TL$ is the so-called tabu list that records the swaps that were recently performed (see Section 2.2.3). 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 $u$ [Rod+15].

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

Figure 2.4 depicts an illustrative example of the neighborhood $N_1(\varphi)$. It presents an embedding $\varphi$ containing a critical vertex $c \in C(\varphi)$ (marked in red), which has the label 6

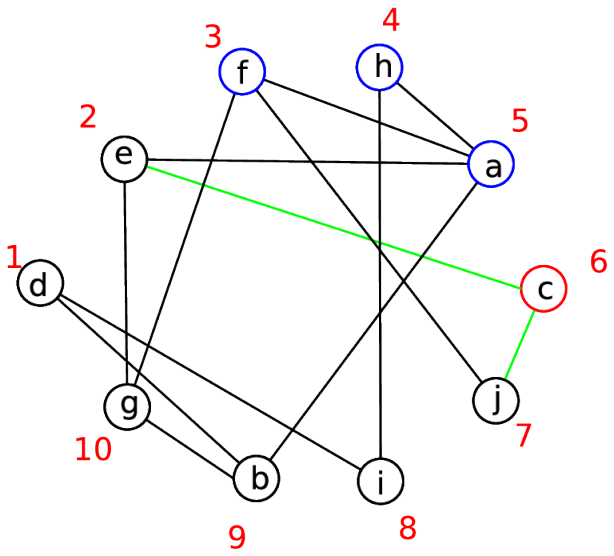



Figure 2.4 – A simple illustration of the neighborhood $N_1(\varphi)$. The embedding $\varphi$ containing a critical vertex $c \in C(\varphi)$ (marked in red), as well as the set $S(c) = \{a, h, f\}$ of suitable vertices eligible to be swapped with vertex $c$ (highlighted in blue) are depicted.

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:

$$\begin{aligned} N_2(\varphi) = \{&\varphi' = \varphi \oplus swap(u, v) : u \in C(\varphi), \\ &v \in R_\gamma(u), swap(u, v) \notin TL\}, \end{aligned} \tag{2.7}$$

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 $\varphi$, 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 $\gamma$ 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 $\varphi'$ according to the evaluation function $f_e$. Let $\varphi'$ be an embedding obtained by swapping $u$ and $v$ in $\varphi$. 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 $\mathcal{O}((|A(u)| + |A(v)|) * n^2)$.

### 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 [GL97]. The idea is to keep track of the performed swaps and forbid the reverse swap operations during the next $\tau$ iterations ($\tau$ 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 [GBF11]. As shown in other studies [LH16; WH13], this technique proves to be robust and effective in different settings and was also used in [Rod+15] for 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.

**Discussions**

Like [Rod+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 $\text{Cb}(w, \varphi) \geq \alpha * \text{Cb}(G, \varphi)$ ($\alpha$ is a prefixed parameter between 0 and 1) used in [Rod+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 [Rod+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 [Rod+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 [Rod+15] uses tabu search which may accept worsening solution transitions.

### 2.2.4 Second phase - Responsive threshold-based search

As explained in Section 2.2.3, 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 [Due93; DS90] 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 $\rho_{N_1}$. 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

**Algorithm 3** Responsive threshold-based descent procedures

**Input**: input solution $\varphi$, best found solution $\varphi^*$, neighborhoods $N_1$ and $N_2$, evaluation function $f_e$, maximum non-improving limit $L_t$, neighborhood $N_1$ application probability $\rho_{N_1}$, and best neighbor move strategy probability $\rho_{best}$
**Output**: best found solution $\varphi^*$, last solution $\varphi$
$NonImpCounterT \leftarrow 0$
**while** $NonImpCounterT < L_t$ **do**
  $T \leftarrow Threshold(\varphi)$
  **if** $rand(0,1) < \rho_{N_1}$ **then**
    $N \leftarrow N_1$ // With probability $\rho_{N_1}$
  **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 $\rho_{best}$
    **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** $\varphi$, $\varphi^*$

solution which is selected by examining the neighborhoods $N_1$ and $N_2$. The former is applied with probability $\rho_{N_1}$, 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 $\rho_{best}$, 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). The best solution found $\varphi^*$ during the search is updated each time a better solution is discovered (lines 24-29, Alg. 3). If $\varphi^*$ 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 can not 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 ($T - \mathrm{Cb}(G, \varphi) \gg 0$), 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 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 [CH15] 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.

### 2.2.5 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 $\varphi$ 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. 2.5(a) (solution before the $ShiftInsert$ operation) and Fig. 2.5(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 2.2 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 2.4.3, we investigate the usefulness of the *Shift-Insert*-based perturbation.

Table 2.2 – Changes of the cyclic distances associated to the edges impacted by the $ShiftInsert$ operation when applied over the solution depicted in Fig. 2.5.

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

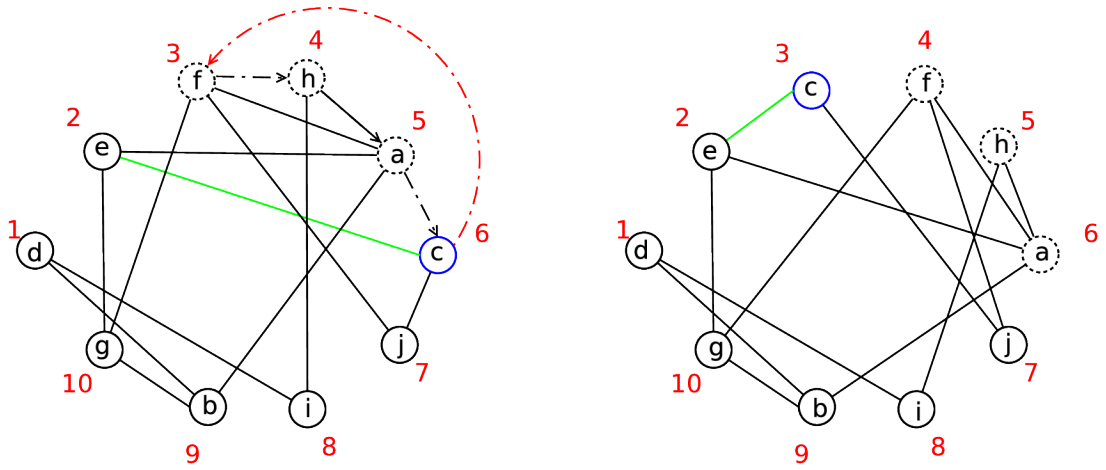



Figure 2.5 – An illustrative example of the *Shift-Insert*-based perturbation. (a) Solution $\varphi$ before applying the $ShiftInsert$ perturbation. (b) Solution $\varphi$ after applying the perturbation $ShiftInsert(c, e)$.

## 2.3 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) [Rod+15].

### 2.3.1 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.

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 [Rod+15] is available at: https://www.tamps.cinvestav.mx/~ertello/cbmp.php

The test-suite used for the experiments presented in this work is composed of 113 topologically diverse graphs [3] previously tested in the literature [Rod+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 [Rod+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.* [Dua+11] and Lozano *et al.* [Loz+12] as benchmark instances for the related antibandwidth problem [LVW84] and employed in [Rod+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 [Rod+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 [Rod+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 ($\geq 0$) 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. 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.

3. Available at https://www.tamps.cinvestav.mx/~ertello/cbmp.php
4. http://math.nist.gov/MatrixMarket/data/Harwell-Boeing

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.

### 2.3.2 Determination of the input parameter values for ITPS

The proposed $ITPS$ algorithm, like most meta-heuristic algoithms, 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 [Lop+16], which is one of a number of automatized parameter tuning tools such as ParamILS [Hut+09] and GGA++ [Ans+15]. 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 2.3.1 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 ($\delta$ 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 ($L_d$, $\rho_{best}$, $\rho_{N_1}$, $L_t$, and $\gamma$). To ensure a fair comparison between our $ITPS$ algorithm and the $TScb$ method, the same cutoff time limit reported in [Rod+15] was adopted (i.e., $T_{max} = 600$ seconds). Table 2.3 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 2.3.1. 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, cater-*

Table 2.3 – Parameters to be tunned with *irace* for the $ITPS$ algorithm.

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

Table 2.4 – Final values found by *irace* after the parameter calibration experiments.

| Instance group | $\delta$ | $L_d$ | $\rho_{best}$ | $L_t$ | $\rho_{N_1}$ | $\gamma$ |
|---|---|---|---|---|---|---|
| 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 |

*pillar29, 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 CBP literature [Rod+15]. The final values returned by these parameter calibration experiments, for each group of instances, are summarized in Table 2.4.

### 2.3.3 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 $TScb$ [Rod+15], which is the current best-performing CBP reference method. These experiments were carried out using the experimental conditions presented in Section 2.3.1, and the parameter setting determined in Section 2.3.2.

The computational results of this experiment are summarized in Table 2.5 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 reach its best solution

Table 2.5 – 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 |

(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 $TScb$ [Rod+15]. The detailed instance-by-instance results from this experiment are provided in Tables 6.1 and 6.2 listed in the Appendix 6.1.

Table 2.5 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 hyper-*

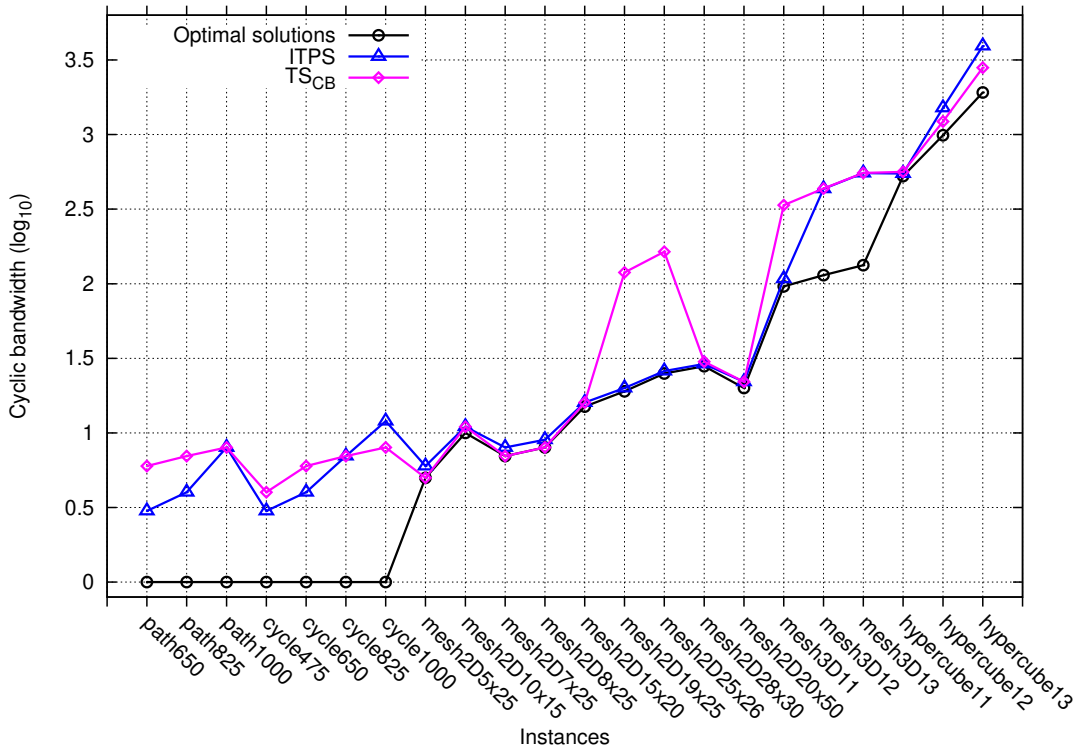
*cubes*; indicating that this type of graphs is still an open challenge for metaheuristic algorithms. The columns listing the O-RMSE values disclose that in average $ITPS$ presents a slightly higher deviation with respect to the known optimal costs than $TScb$ (1.56 vs. 1.27), notwithstanding $ITPS$ showed to be more effective for finding global optimal embeddings. By inspecting the row allocated for the Harwell-Boeing graphs of Table 2.5, 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 thought 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.

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 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 2.6). For the remaining 7 instances (6.19%) $TScb$ still offers the best-known results.

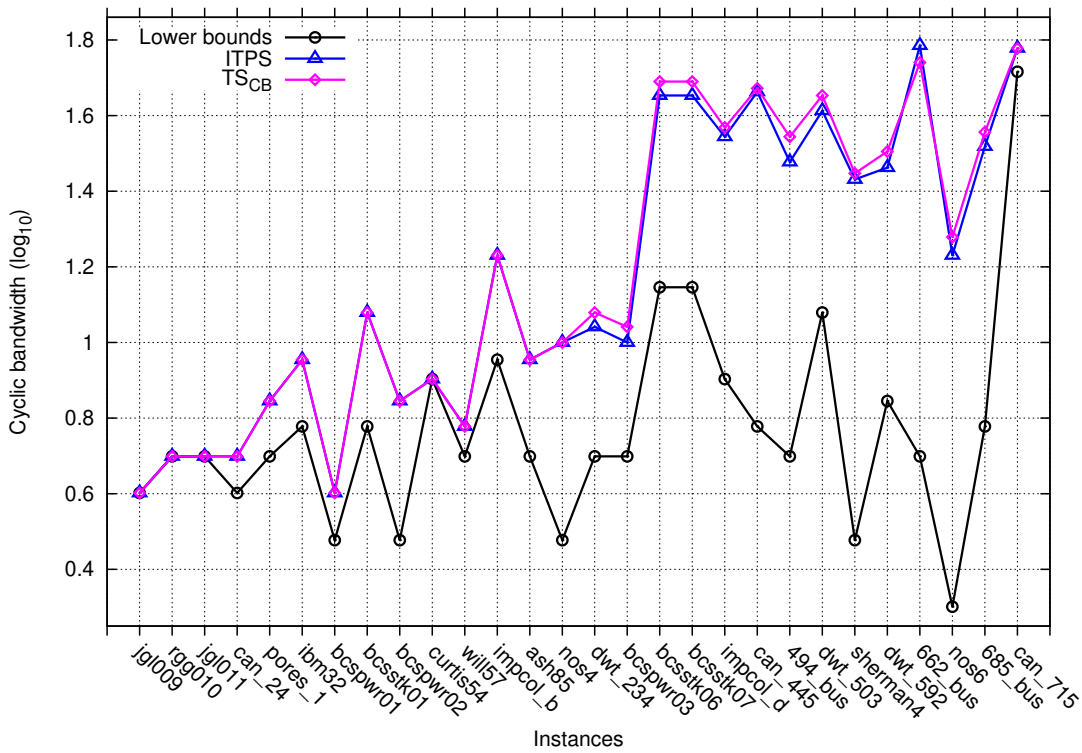
Finally, the statistical analysis carried out for this experiment, and presented in the last two columns of Tables 6.1 and 6.2, allows us to verify that a statistically significant performance amelioration was achieved by $ITPS$ with respect to $TScb$ on 37 instances (32.74% of the graphs). Nevertheless, $ITPS$ 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.

## 2.4 Analysis

We present additional experiments to investigate the key components of the $ITPS$ algorithm: a) the extended evaluation function ($f_e$) of Section 2.2.2, b) the responsive threshold-based search (RTBS) method of Section 2.2.4, and c) the *Shift-Insert*-based perturbation mechanism of Section 2.2.5. For these experiments, we adopted the same subset of 20 representative graphs (14 standard topology graphs and 6 Harwell-Boeing




**(a)**




**(b)**

Figure 2.6 – 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 [Lin97].

graphs) that were used for parameter tuning in Section 2.3.2.

### 2.4.1 Influence of the extended evaluation function

As we pointed out in Section 2.2.2, the objective function of 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 [PA12; Sta92], on which identifying a promising search direction may become difficult for local search methods [Mar+11; MAK07]. 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 CBP. Table 2.6 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 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 2.3.1 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 $\star$ 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 2.6, 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.

Table 2.6 – 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 |

## 2.4.2 Influence of the responsive threshold-based search

In our $ITPS$ algorithm, the first phase employs a double neighborhood descent search procedure (DNDS) to explore embeddings of increasing quality until a local optimal solution is reached. To escape from the basin of attraction [PA12], $ITPS$ 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 the full $ITPS$ method. Table 2.7 presents the results from this comparison, using the same column organization previously described for Table 2.6.

It is clear, from Table 2.7, 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.

### 2.4.3 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 2.8, which has the same column headings defined for Table 2.6.

As shown in Table 2.8, 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.

Table 2.7 – 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 |

## 2.5 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-descent 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.

We have assessed the proposed algorithm on two groups of 113 benchmark graphs from

Table 2.8 – 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 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.

In the next chapter, we will carry on studying CBP and propose a new iterated local search algorithm.

Chapter 3

# A NEW ITERATED LOCAL SEARCH ALGORITHM FOR THE CYCLIC BANDWIDTH PROBLEM

In this chapter, We will present another effective heuristic algorithm based on the general iterated local search framework 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 from 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. The content of this chapter has been submitted to *Knowledge-Based Systems*.

## 3.1 Introduction

In this work, we aim to advance the state-of-the-art of solving CBP. 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 $TScb$ and $ITPS$, the proposed algorithm is simpler (e.g., it uses only one neighborhood against 2 for $TScb$ and $ITPS$) and requires fewer parameters (4 against 8 for $TScb$ 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 $TScb$ and $ITPS$. Our experiments indicate that the proposed

algorithm dominates the reference algorithms and achieves a performance that has never been reported in CBP literature.

The remainder of the chapter is organized as follows. In Section 3.2, we present the main framework of the proposed algorithm and its key components. In Section 3.3, we show the computational results on the benchmark instances and comparisons with reference results in the literature. In Section 3.4, we report additional experiments to investigate the influences of main algorithmic components on the global performance of the algorithm. Conclusions are drawn in Section 3.5.

## 3.2 New iterated local search algorithm

Iterated local search [LMS03] is a general search framework with numerous successful application examples (e.g., [FH17; GJL09; MK19; ZH17]. 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 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 4. The algorithm starts with a random initialization solution $\varphi$. The inner 'while' loop iteratively performs the dedicated tabu search procedure (Section 3.2.1), followed by the directed perturbation procedure (Section 3.2.2). At each iteration, the input solution is first improved by DTS which is based on the neighborhood $N_f$ (Section 3.2.1) 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 $\varphi^*$ is returned.

**Algorithm 4** New iterated local search algorithm for CBP

**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 $L_3$, controlling percent $\alpha$ and cutoff time limit $T_{max}$
**Output**: The best solution found $\varphi^*$
$\varphi \leftarrow InitialSolution()$
$\varphi^* \leftarrow \varphi$
**while** the cutoff time limit $T_{max}$ is not reached **do**
  $Count \leftarrow 0$
  **while** $Count < L_3$ **do**
    $(\varphi, \varphi^*) \leftarrow DTS(\varphi, \varphi^*, N_f, f_e, L_1)$ // Local optimization with dedicated tabu search, Section. 3.2.1
    $(\varphi, \varphi^*) \leftarrow Directed\_Perturb(\varphi, \varphi^*, f_e, L_2)$ // Directed perturbation, Section. 3.2.2
    $Count \leftarrow Count + 1$
  **end while**
  $\varphi \leftarrow Strong\_Perturb(\varphi, \alpha)$ // Strong perturbation, Section 3.2.3
**end while**
**return** $\varphi^*$

To assess the quality of a candidate solution $\varphi$, the algorithm adopts the extended evaluation function $f_e(\varphi)$ introduced in [RHR19], which is defined as follows.

$$f_e(\varphi) = \mathrm{Cb}(G, \varphi) + \frac{Z(\mathrm{Cb}(G, \varphi))}{|E|} \tag{3.1}$$

where $Z(\mathrm{Cb}(G, \varphi)) = \sum_{\{u,v\} \in E} I_{uv}$ represents the number of edges whose cyclic distances equal $\mathrm{Cb}(G, \varphi)$, and the indicator variable $I_{uv} = 1$ if $|\varphi(u) - \varphi(v)|_n = \mathrm{Cb}(G, \varphi)$, 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.

### 3.2.1 Dedicated tabu search

The dedicated tabu search (DTS) procedure (Algorithm 5) is designed to exploit candidate solutions with the help of the neighborhood $N_f$ (see below). DTS starts with an

**Algorithm 5** 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** $\varphi$, $\varphi^*$

input solution $\varphi$ 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. 5), 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. 5). After each solution transition, the performed swap operation is recorded in the so-called tabu list [GL97] 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 $\varphi$ 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 [Rod+15], we constraint the candidate vertex $u$ to a specific subset of *critical* vertices $C(\varphi)$ defined as follows.

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

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 $u$ [Rod+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 $\varphi$.

Given a solution $\varphi$ 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 $\varphi$ 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. 5, 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 [RHR19; Rod+15], which fixes $tl$ according to a periodic step function.

Fig. 3.1 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.

### 3.2.2 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 from the trap, we apply a directed perturbation procedure (depicted in Algorithm 6), which relies on a *randomized* version of the *ShiftInsert* operator [RHR19]. Our *RandomizedShiftInsert* operator works as follows. First, we identify an edge $e = \{x, y\}$ with the largest cyclic distance (i.e., $\text{Cb}(G, \varphi)$). Then, one endpoint of the edge is chosen (say $x$) and used to perform $\beta$ (a random number between 1 and $\text{Cb}(G, \varphi)$) 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.

In the example shown in Fig. 3.2(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) in a clockwise direction, leading to the solution shown in Fig. 3.2(b).

We investigate the degree of influence of the directed perturbation procedure over the search performance of the proposed *NILS* algorithm in Section 3.4.

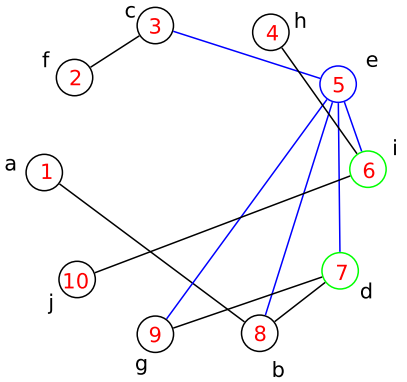



Figure 3.1 – Illustration for solution transformation: a graph with its labeling $\varphi$, critical set $C(\varphi) = \{e, i, g, j\}$ and set $S(e)$ for the first critical vertex $e$.

**Algorithm 6** Directed perturbation

**Input**: input solution $\varphi$, best solution $\varphi^*$, and perturbation strength $L_2$
**Output**: perturbed solution $\varphi$, updated best solution $\varphi^*$
$Counter \leftarrow 0$
**while** $Counter < L_2$ **do**
  $\varphi \leftarrow RandomizedShiftInsert(\varphi)$
  $Counter \leftarrow Counter + 1$
  **if** $f_e(\varphi) < f_e(\varphi^*)$ **then**
    $\varphi^* \leftarrow \varphi$
  **end if**
**end while**
**return** $\varphi$, $\varphi^*$

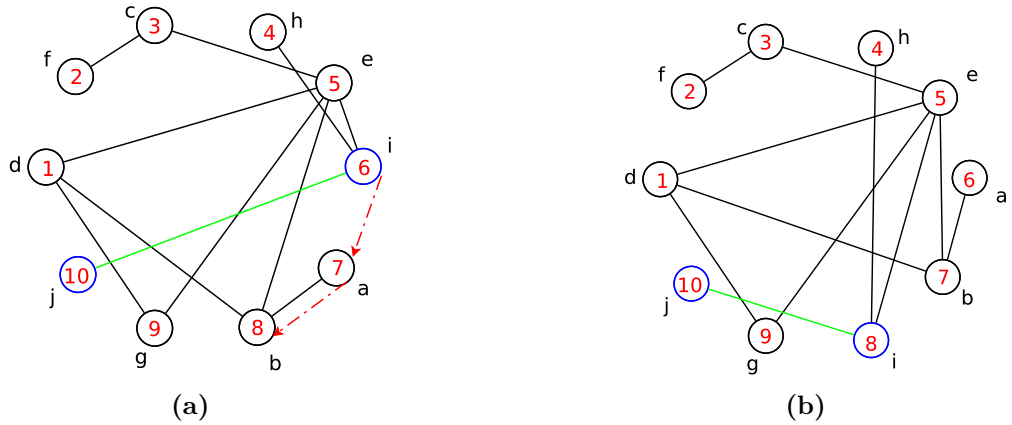



Figure 3.2 – Illustration of the *RandomizedShiftInsert* operator: (a) The cycle graph before the operation, (b) The cycle graph after the operation (i.e., $swap(i, a)$ followed by $swap(i, b)$).

### 3.2.3 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: $C_R(\varphi) = \{w \in V : \mathrm{Cb}(w, \varphi) \leq \alpha \cdot \mathrm{Cb}(G, \varphi)\}$ where $\alpha \in [0, 1]$ is a controlling parameter. Thus, $C_R(\varphi)$ is composed of vertices with a cyclic bandwidth up to $\alpha \cdot \mathrm{Cb}(G, \varphi)$. 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 [Lin97]. 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. 3.3. Given a graph $G(V, E)$ ($|V| = 10$, $L_B = 3$), the objective value of the solution in Fig. 3.3(a) is 4 (Cb$(G, \varphi) = 4$). For the destruction step, if we set $\alpha$ 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. 3.3(b). For the greedy reconstruction step, we starting from a random vertex $u \in V \setminus C_R(\varphi) = \{d, e, i, j, g\}$ (say $d$ in Fig. 3.3(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. 3.3(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. 3.3(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. 3.3(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 3.4.

### 3.2.4 Relations with previous studies

*NILS* distinguishes itself from two previous algorithms *TScb* [Rod+15] and *ITPS* [RHR19] by the following features. First, unlike [RHR19; Rod+15], the dedicated tabu search procedure of *NILS* relies on a single neighborhood while both *TScb* and *ITPS* explore two neighborhoods in a probabilistic way. As such, the key optimization compo-

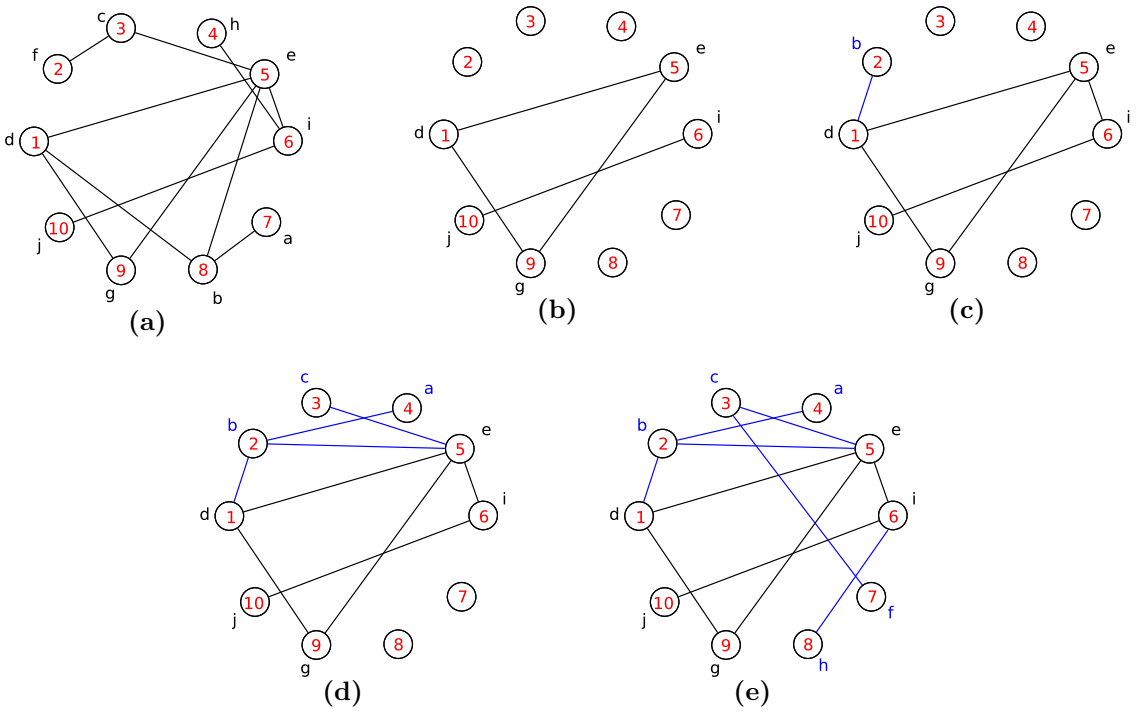



Figure 3.3 – Illustration of the strong perturbation procedure using destruction and reconstruction on a graph with $\mathrm{Cb}(G, \varphi) = 4$, analytical lower bound $L_B$=3 and controlling parameter $\alpha = 0.8$. (a) input solution; (b) partial solution after removing 5 vertices of $C_R$; (c) beginning of solution reconstruction from vertex $d$; (d) reconstruction in progress; (e) completion of the reconstruction.

nent 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 $TScb$ and 9 for $ITPS$), making it much easier to use and analyze.

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.

## 3.3 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: $TScb$ [Rod+15] and $ITPS$ [RHR19].

### 3.3.1 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.* [Rod+15], and latter by Ren *et al.* in [RHR19]. 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 [Rod+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 already known, while for the remaining 21 graphs lower and upper bounds can be calculated according to [Lin97].

The performance assessment of the three analyzed algorithms was done using the same comparison metrics previously employed in [Rod+15] and [RHR19], 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

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

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 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 $TScb$ and $ITPS$ methods are publicly available (see [RHR19]), 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.

### 3.3.2 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 [Lop+16] 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 [RHR19; Rod+15], 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*).

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

Table 3.1 – Description and ranges for the input parameters of the $NILS$ algorithm automatically tuned with *irace* [Lop+16].

| 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} |
| $\alpha$ | Controlling percent | Real | (0.0, 1.0) |

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 [RHR19; Rod+15]. The other four input parameters of $NILS$ are listed in Table 3.1, 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.

### 3.3.3 Comparisons with state-of-the-art algorithms

This section is devoted to present the results obtained from the experimental performance assessment among the two best performing algorithms in CBP literature (i.e., $TScb$ [Rod+15] and $ITPS$ [RHR19]) and our $NILS$ algorithm. This analysis was carried out under the experimental conditions previously detailed in Section 3.3.1.

Table 3.2 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 6.3 and 6.4 provided in Appendix 6.2.

From Table 3.2, 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, $TScb$ could not ensure optimal solutions for any subset of instances.

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

Table 3.2 – Summary of the experimental performance comparison among the two reference methods in CBP literature (i.e., *TScb* [Rod+15] and *ITPS* [RHR19]) and the *NILS* algorithm over 113 benchmark instances: 85 standard graphs with known optimal solutions, and 28 Harwell-Boeing instances.

| | | *TScb* | | | | *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 3.3 – Summary of the statistical signification analysis from the comparison among the two reference methods in CBP literature (i.e., *TScb* [Rod+15] and *ITPS* [RHR19]) and the *NILS* algorithm over 113 benchmark instances: 85 standard graphs with known optimal solutions, and 28 Harwell-Boeing instances.

| | | *NILS / TScb* | | | *NILS / ITPS* | | |
|---|---|---|---|---|---|---|---|
| Graph type | Num. | + | ⋆ | − | + | ⋆ | − |
| *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 |

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 $TScb$ 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 $TScb$ (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 $TScb$ (584.21 vs. 546.23), but $NILS$ produced much better solutions than $TScb$.

An in-depth statistical significance analysis, using the methodology described in Section 3.3.1, was performed for validating the experimental results produced in our performance comparisons. This analysis, presented in Table 3.3, and detailed in the last four columns of Tables 6.3 and 6.4, revealed that $NILS$ was able to statistically outperform $TScb$ 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.

## 3.4 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 3.3.1 and reported their computational results in Tables 3.4 and 3.5 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 3.2. The last three columns (Statistics) present the statistical results obtained by using the methodology detailed in Section 3.3.1.

Table 3.4 – 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 | $+$ | $\star$ | $-$ |
| *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 |

Table 3.5 – 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 |

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 3.4 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*).

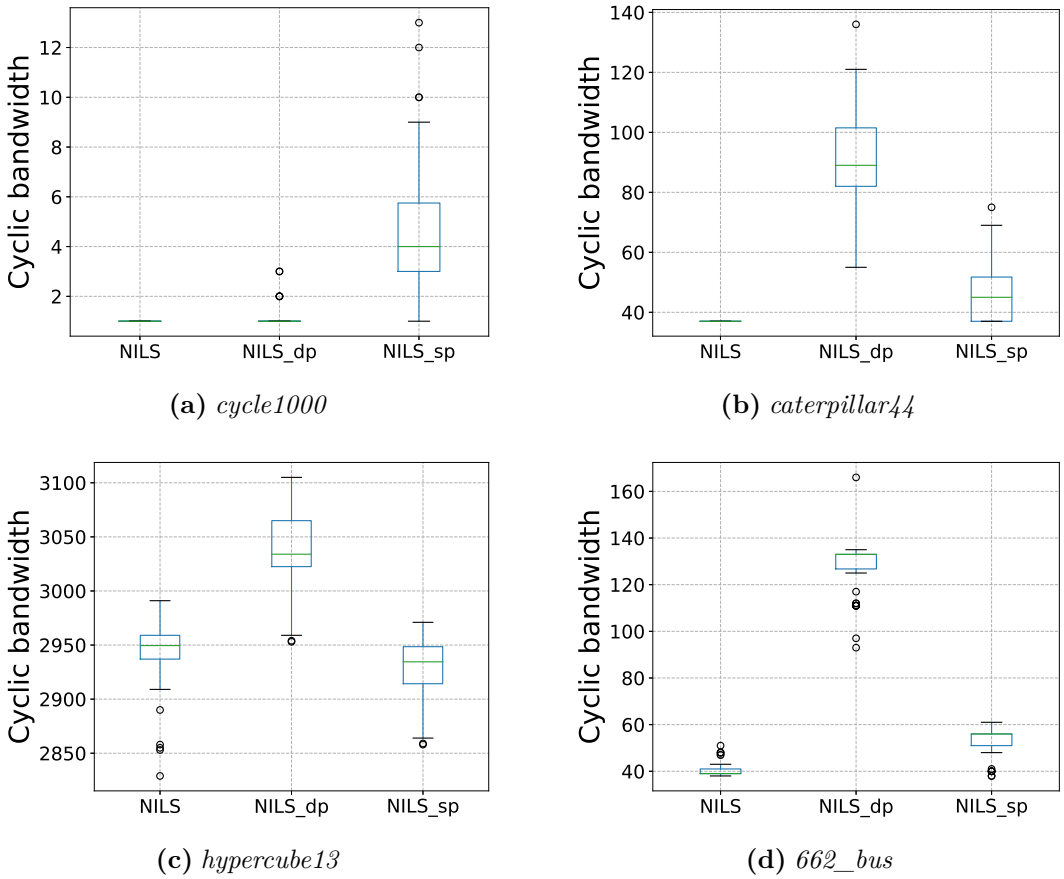
Similarly, the results of Table 3.5 disclose that the strong perturbation also impacts

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.

In regard to 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. 3.4 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. 3.4(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*. 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. 3.4(b)-3.4(d). It is worth noting that in Fig. 3.4(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 3.5 (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. 3.4, which means that the directed perturbation and strong perturbation




**(a)** *cycle1000* **(b)** *caterpillar44*

**(c)** *hypercube13* **(d)** *662_bus*

Figure 3.4 – 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.

play complementary roles in $NILS$.

## 3.5 Conclusions

The $NILS$ algorithm presented in this chapter enriches the practical solution toolbox for effectively solving CBP. 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 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 [RHR19; Rod+15]. 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.

In the next chapter, we will commit a study of recombination operators for CBP under the framework of memetic algorithm.

Chapter 4

# A STUDY OF RECOMBINATION OPERATORS FOR THE CYCLIC BANDWIDTH PROBLEM

In this chapter, a study of CBP with the paradigm of memetic algorithms is presented. To find out how to choose or design a suitable recombination operator for the problem, we study five classical permutation crossovers within a basic memetic algorithm integrating a simple descent local search procedure. We investigate the correlation between algorithmic performances and population diversity measured by the average population distance and entropy. This work invites more research to improve the two key components of the memetic algorithm: reinforcement of the local search and design of a meaningful recombination operator suitable for the problem. The content of this chapter has been published in *Lecture Notes in Computer Science*.

## 4.1 Introduction

As a well-known meta-heuristic framework [KS05; MC03], memetic algorithms (MAs) have been widely used to solve a large number of NP-hard problems [CH16; JHH14; LH16; WH13; ZHG18]. For permutation problems, MAs have also reported good performances for the Traveling Salesman Problem (TSP) [FM96; MF01], the Quadratic Assignment Problem [BH15], and other bandwidth problems [BS11; RB11].

Despite the theoretical and practical relevance of CBP, few studies can be found in the literature for solving the problem. A branch and bound algorithm was presented [RRR12] to handle small graphs ($n < 40$). A tabu search algorithm was proposed [Rod+15] to deal with standard and random graphs with 8 to 8192 vertices. Very recently, an iterated three-phase search approach [RHR19] was introduced and improved a number of previous best results reported in [Rod+15]. To our knowledge, the memetic approach has never

been experimented to solve CBP in the literature, though MAs have been applied to other labeling problems such as the cyclic bandwidth sum problem [RNL18] and the antibandwidth problem [RB11]. This work fills the gap by investigating the memetic approach for CBP. In particular, we focus on the role of the recombination or crossover (used interchangeably in this paper) component and study the contributions of five permutation recombination operators which are conveniently applicable to CBP. To highlight the impacts of the studied recombination operators, we base our study on a canonical memetic algorithm which combines a recombination operator for solution generation and a simple descent local search for solution improvement.

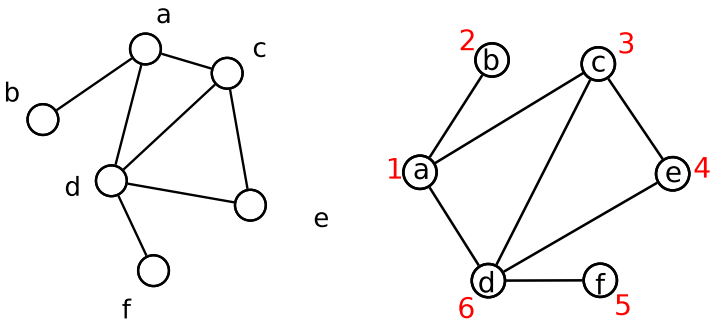
## 4.2 Memetic Algorithm for CBP

### 4.2.1 Search space, representation, fitness function

Given a graph $G = (V, E)$ of order $|V| = n$ and a cycle graph $C_n$, the search space $\Omega$ for CBP is composed of all possible embeddings (labellings, solutions or arrangements) of $G$ in $C_n$, $\varphi : V \to V$. Considering the symmetry characteristic of solutions, there exist $(n-1)!/2$ possible embeddings for $G$ [Rod+15].

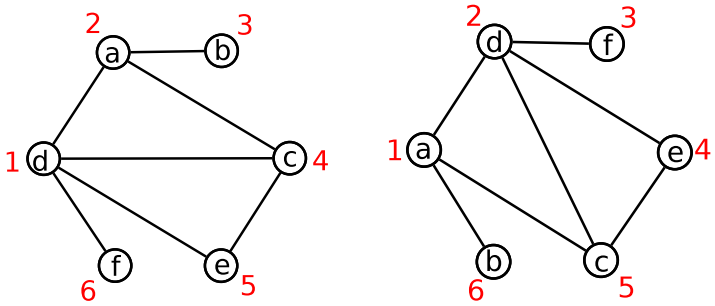
Figure 4.1 shows a graph with 6 vertices named from 'a' to 'f' (Fig. 4.1(a)). According to Equation (2.1), the objective value of Fig. 4.1(b) is 3 (decided by the longest edge 'dc' in the example). An embedding arranged in a cycle graph (Fig. 4.1(b)) where the numbers in red are the labels assigned to the vertices, and two embeddings where the vertices are rearranged in the cycle graph in clockwise direction (Fig. 4.1(c)) and in anticlockwise direction (Fig. 4.1(d)). Notice that the relative position of each pair of nodes in Fig. 4.1(b)-4.1(d) is not changed. So according to Equation (2.1), these three embeddings have the same objective value, and in fact they correspond to the same solution.

In practice, we represent an embedding $\varphi$ by permutations $l = \{1, 2, \ldots, n\}$ such that the $i$-th element $l[i]$ denotes the label assigned to vertex $i$ of $V$. Another representation of an embedding is proposed in [RHT08b], which maps a permutation $\varphi$ to an array $\gamma$ indexed by the labels. The $i$-th value of $\gamma[i]$ indicates the vertex whose label is $i$. We illustrate these representations with an example. For the embedding of Fig. 4.1(b), we have $\varphi$=(1 2 3 6 4 5) for the vertices from 'a' to 'f', and the corresponding $\gamma$ representation is $\gamma$=(a b c e f d). In our algorithm, the $\varphi$ representation is used in the local search procedure, because it eases the implementation of the *swap* operation, while the $\gamma$ representation is




**(a)** Original graph $G$

**(b)** An embedding on cycle graph




**(c)** Clockwise rotation

**(d)** Anticlockwise rotation

Figure 4.1 – Illustration of a graph (a) with an embedding (b) and two equivalent symmetric embeddings (c) and (d)

adopted for the recombination operators, as well as the distance calculation presented in Section 4.5. The fitness of a candidate embedding $\varphi$ in the search space is evaluated by Equation (2.1).

### 4.2.2 General procedure

The studied MA follows the general MA framework in discrete optimization [Hao12]. Staring with an initial population (Section 4.2.3), it alternates between a local search procedure (Section 4.2.4) and a recombination procedure (Section 4.2.5). The pseudo-code of the proposed MA is presented in Algorithm 7. The algorithm first fills the population $P$ with $|P|$ local optimal solutions provided by the local search procedure and then performs a series of generations. At each generation, two parent solutions $\varphi_F$ and $\varphi_M$ are selected at random from the population and are recombined to generate an offspring solution $\varphi_C$. Then, the local search is used to improve the offspring solution to attain a new local optimal solution. Finally, the improved solution is used to update the population (Section 4.2.6). This process is repeated until a fixed number of generations ($MaxGene$) is reached.

### 4.2.3 Initialization

In the initialization procedure ($Ini\_Population$), $|P|$ embeddings are generated randomly and independently at first. And then each embedding is improved by the local search procedure of Section 4.2.4 to attain a local optimum (lines 5-10, Alg. 7). The best solution $\varphi^*$ in $P$ is also recorded, which is updated during the subsequent search, each time an improved best solution is encountered.

### 4.2.4 Local search

Local search (LS) is an important component of the memetic algorithm, which aims to improve the input solution by searching a given neighborhood. In this work, we apply a simple Descent Local Search (DLS) in order to highlight the contributions of the recombination component.

DLS adopts the swap-based neighborhood of [Rod+15], where a neighboring solution of a given solution $\varphi$ is obtained by simply swapping the labels of two vertices of $\varphi$. To specify the neighborhood, we first define, for a vertex $u$, its cyclic bandwidth $\mathrm{Cb}(u, \varphi)$

**Algorithm 7** Pseudo-code of general procedure

**Input**: Finite undirected graph $G(V, E)$, fitness function Cb, fixed size of population $|P|$ and maximum generations $MaxGene$
**Output**: The best solution found $\varphi^*$
$P = \{\varphi^1, \varphi^2, ...\varphi^{|P|}\} \leftarrow Init\_Population()$
$\varphi^* \leftarrow Best(P)$
**for** $i = 1$ to $|P|$ **do**
  $\varphi^i \leftarrow Local\_Search(\varphi^i)$
  **if** $\mathsf{Cb}(G, \varphi^i) < \mathsf{Cb}(G, \varphi^*)$ **then**
    $\varphi^* \leftarrow \varphi^i$
  **end if**
**end for**
**for** $j = 1$ to $MaxGene$ **do**
  $\varphi_F, \varphi_M \leftarrow Parent\_Selection(P)$
  $\varphi_C \leftarrow Recombination\_Sol(\varphi_F, \varphi_M)$
  $\varphi_C \leftarrow Local\_Search(\varphi_C)$
  **if** $\mathsf{Cb}(G, \varphi_C) < \mathsf{Cb}(G, \varphi^*)$ **then**
    $\varphi^* \leftarrow \varphi_C$
  **end if**
  $P \leftarrow Update\_Pop(\varphi_C, P)$
  $j \leftarrow j + 1$
**end for**
**return** $\varphi^*$

with respect to the embedding $\varphi$ as follows:

$$\mathrm{Cb}(u, \varphi) = \max_{v \in A(u)} \{|l(u) - l(v)|_n\}, \tag{4.1}$$

where $A(u)$ denotes the set of vertices adjacent to $u$ of cardinality $deg(u)$. Then the set of critical vertices is given by:

$$C(\varphi) = \{u \in V : \mathrm{Cb}(u, \varphi) = \mathrm{Cb}(G, \varphi)\}. \tag{4.2}$$

The neighborhood is defined as follows:

$$N(\varphi) = \{\varphi' = \varphi \oplus swap(u, v) : u \in C(\varphi), v \in V\}. \tag{4.3}$$

DLS starts with an input embedding, then it iteratively visits a series of neighboring solutions of increasing quality according to the given neighborhood. At each iteration, only solutions with a better objective value are considered and the best one is used to replace the incumbent solution. If there exist multiple best solutions, the first one encountered is adopted. We repeat this process until no better solution exists in the neighborhood. In this case, DLS attains a local optimum and the procedure of recombination is triggered to escape from the local optimum.

### 4.2.5 Recombination

Recombination is another important part of the MA, which aims to generate new diversified and potentially improving solutions. In our case, only one offspring solution is generated at each generation by each recombination application. In Section 4.3, we present five permutation recombination operators applied to CBP.

### 4.2.6 Updating population

Each new offspring solution improved by the local search procedure is used to update the population. In the proposed MA, we apply a simple strategy: we insert the new offspring into $P$, and remove the "worst" solution in terms of the objective value.

## 4.3 Recombination operators

There are several recombination operators that are already applied to permutation problems [Dav85; FM96; GL+85; OSH; Sys91]. We consider five crossover operators introduced below. It is worth noting that all the recombination operations work with the $\gamma$ representation mentioned in Section 4.2.1.

### 4.3.1 Order crossover

The Order Crossover operator (OX) [Dav85] generates an offspring solution with a substring of one parent solution and conserves the relative order of the numbers of the other parent solution. Let's consider an example with two parent solutions $\varphi_F$=(1 2 3 4 5 6 7 8) and $\varphi_M$=(2 4 6 8 7 5 3 1) (each number here denotes the index of a node). Given two random cut points (in this case, the first cut point is between second and third positions and the second cut point is between fifth and sixth positions, i.e., $\varphi_F$=(1 2 | 3 4 5 | 6 7 8) and $\varphi_M$=(2 4 | 6 8 7 | 5 3 1), two offspring solutions first inherit the substring between the two cut points: $\varphi_{C1}$=(+ + | 3 4 5 | + + +) and $\varphi_{C2}$=(+ + | 6 8 7 | + + +). Then, we copy the permutation starting from the second cut point of $\varphi_M$ to the end, as well as from the beginning to the second cut point: (5 3 1 2 4 6 8 7). At last, the new obtained permutation is used to insert into $\varphi_{C1}$ from the second cut point. The repeated numbers are skipped and we get $\varphi_{C1}$=(8 7 | 3 4 5 | 1 2 6). The same operations are performed on $\varphi_{C2}$ with $\varphi_F$ to get $\varphi_{C2}$=(4 5 | 6 8 7 | 1 2 3).

### 4.3.2 Order-based crossover

The Order-based Crossover operator (OX2) [Sys91] is a modified version of OX. Instead of choosing two cut points, OX2 chooses several random positions of one parent solution, and then the order of the selected positions is imposed on the other parent solution. For instance, we have two parent solutions $\varphi_F$=(1 2 3 4 5 6 7 8) and $\varphi_M$=(2 4 6 8 7 5 3 1), and the second, third and sixth positions are picked for $\varphi_M$. So the order of "4 6 5" is kept. For solution $\varphi_F$, we remove the corresponding numbers of these positions to get (1 2 3 + + + 7 8). Then we insert the numbers in the order "4 6 5" into $\varphi_F$ and we get the offspring solution $\varphi_{C1}$=(1 2 3 4 6 5 7 8). The same operation can be performed for $\varphi_M$ to obtain the other offspring solution $\varphi_{C2}$=(2 4 3 8 7 5 6 1).

### 4.3.3 Cycle crossover

The Cycle Crossover operator (CX) [OSH] seeks a way to preserve the common information in both parent solutions. Two new offspring solutions $\varphi_{C1}$ and $\varphi_{C2}$ are created from two parents $\varphi_F$ and $\varphi_M$ where the number of each position in $\varphi_{C1}$ and $\varphi_{C2}$ is decided by the number of the corresponding position of one parent. For example, we consider two parent solutions $\varphi_F$=(1 2 3 4 5 6 7 8) and $\varphi_M$=(2 4 6 8 7 5 3 1). Firstly, the number of the first position of $\varphi_{C1}$ could be 1 or 2, Supposing that we pick 1 here (1 + + + + + + +). Then, the number of the eighth position could not be 1 because it is already assigned to the first position, hence we allocate it with a number from $\varphi_F$ to get (1 + + + + + + 8). After that, we find the position of $\varphi_M$ whose number is 8 and assign the number of $\varphi_F$ to the corresponding position of $\varphi_{C1}$. We repeat the same operation and find that the forth and the second number of $\varphi_{C1}$ come from $\varphi_F$, which leads to (1 2 + 4 + + + 8). For the rest of the positions, we fill them with the numbers from $\varphi_M$ to obtain a complete offspring solution $\varphi_{C1}$=(1 2 6 4 7 5 3 8). Similarly, we could get the other offspring solution $\varphi_{C2}$=(2 4 3 8 5 6 7 1).

### 4.3.4 Partially mapped crossover

The Partially Mapped Crossover operator (PMX) [GL+85] passes the absolute position information from the parent solutions to the offspring solutions. An offspring solution gets a substring from one parent and its remaining positions take the values of the other parent. For example, we consider again $\varphi_F$=(1 2 3 4 5 6 7 8) and $\varphi_M$=(2 4 6 8 7 5 3 1). At the beginning, two random cut points are chosen for both parent solutions: $\varphi_F$=(1 2 3 | 4 5 6 | 7 8) and $\varphi_M$=(2 4 6 | 8 7 5 | 3 1). Then we pass the information between the two cut points to the offspring solutions: $\varphi_{C1}$=(+ + + | 4 5 6 | + +) and $\varphi_{C2}$=(+ + + | 8 7 5 | + +). Also, we get the mapping for the substrings between the two cut points: 4↔8, 5↔7, 6↔5. After that, the other positions of the offspring solutions are filled with the other parent solution, hence we get $\varphi_{C1}$=(2 4 6 | 4 5 6 | 3 1) and $\varphi_{C2}$ =(1 2 3 | 8 7 5 | 7 8). For the duplicate labels in the solution, we use the mapping of substrings to replace the repeated numbers. In this case, 5↔7 and 6↔5 result in 6↔7. Therefore, the offspring solutions are $\varphi_{C1}$=(2 8 7 | 4 5 6 | 3 1) and $\varphi_{C2}$=(1 2 3 | 8 7 5 | 6 4).

### 4.3.5 Distance preserved crossover

The Distance Preserved Crossover operator (DPX) [FM96], designed for solving the Traveling Salesman Problem (TSP), aims to produce an offspring solution which has the same distance to each of its parents. It is noteworthy that the distance here is the distance based on the common connections between two solutions, instead of the Hamming distance. We come back to this issue in Section 4.5. For DPX, we firstly delete the uncommon connections of two neighboring numbers for both parent solutions. Then, the parent solutions are separated into different substrings. Finally, we reconnect all the substrings without using any of the connections which are contained in only one of the parent solutions. For more detailed explanations and examples, please refer to [FM96].

## 4.4 Experimental results

### 4.4.1 Instances and settings

In this section, we report experimental results of the MA using the 5 different recombination operators introduced in Section 4.3. The study was based on 20 representative graphs with 59 to 2048 vertices, selected from a test-suite of 113 benchmark instances (https://www.tamps.cinvestav.mx/~ertello/cbmp.php). 14 of the chosen graphs are standard graphs covering 7 dissimilar categories (path, cycle, complete tree, 2-dimension mesh, 3-dimension mesh, caterpillar and hypercube) and the other 6 graphs (called Harwell-Boeing graphs) come from real-world scientific and engineering applications and are part of the Harwell-Boeing Sparse Matrix Collection. Considering the stochastic nature of the algorithm, each instance was independently solved 50 times under the environment of Linux using an Intel Xeon E5-2695 2.1 GHz CPU and 2GB RAM. Each execution was limited to 20000 generations ($MaxGene = 20000$) and the population size $|P|$ was set to 20.

### 4.4.2 Computational results

Table 4.1 outlines the computational results of our MA variants with the 5 different recombination operators. The columns“Best" and “Avg" list the best and average objective values found. According to the definition introduced in Section 4.1, a smaller objective value indicates a better result. Table 4.1 shows that the algorithm with OX2 obtains

| | CX | | DPX | | OX | | OX2 | | PMX | |
|---|---|---|---|---|---|---|---|---|---|---|
| Graph | Best | Avg | Best | Avg | Best | Avg | Best | Avg | Best | Avg |
| nos6 | 327 | 331.28 | 327 | 329.74 | 266 | 287.98 | **216** | **227.84** | 327 | 331.98 |
| path1000 | 461 | 475.42 | 462 | 474.02 | 254 | 301.04 | **226** | **247.54** | 468 | 482.68 |
| nos4 | 44 | 46.12 | 43 | 45.24 | 32 | 39.32 | **28** | **34.48** | 42 | 45.78 |
| tree10x2 | 39 | 42.72 | 35 | 40.72 | **28** | 32.50 | **28** | **29.26** | 36 | 41.56 |
| cycle1000 | 457 | 476.66 | 466 | 473.38 | 252 | 296.98 | **226** | **246.94** | 459 | 480.86 |
| mesh2D8x25 | 88 | 93.04 | 89 | 91.82 | 59 | 75.18 | **57** | **62.94** | 87 | 93.28 |
| caterpillar29 | 203 | 211.48 | 203 | 208.70 | 138 | 162.98 | **100** | **127.32** | 198 | 210.14 |
| mesh3D6 | 102 | 103.88 | 101 | 102.96 | 86 | 93.08 | **73** | **78.26** | 102 | 104.28 |
| hypercube11 | 1022 | 1022.76 | 1022 | 1022.14 | 1019 | 1021.26 | **952** | **1010.48** | 1022 | 1022.54 |
| cycle475 | 200 | 215.16 | 206 | 213.36 | 105 | 128.36 | **99** | **110.76** | 192 | 217.30 |
| mesh2D28x30 | 409 | 413.40 | 410 | 412.06 | 336 | 371.76 | **270** | **287.46** | 406 | 414.06 |
| mesh3D11 | 660 | 662.04 | 660 | 661.28 | 625 | 650.30 | **507** | **522.82** | 660 | 662.40 |
| can___715 | 354 | 355.80 | 355 | 355.14 | 347 | 353.92 | **293** | **316.70** | 354 | 355.74 |
| impcol_b | 28 | 28.46 | 27 | 27.96 | 25 | 27.22 | **20** | **26.72** | 28 | 28.00 |
| path475 | 202 | 214.50 | 206 | 212.86 | 112 | 132.24 | **102** | **112.94** | 189 | 217.56 |
| 494_bus | 220 | 230.76 | 222 | 228.72 | 135 | 165.74 | **128** | **138.62** | 216 | 233.38 |
| tree21x2 | 199 | 212.08 | 203 | 208.96 | 139 | 171.34 | **124** | **140.84** | 200 | 210.68 |
| caterpillar44 | 481 | 493.28 | 479 | 491.24 | 340 | 400.78 | **281** | **321.70** | 480 | 495.60 |
| impcol_d | 207 | 209.60 | 207 | 208.80 | 190 | 202.98 | **159** | **169.74** | 208 | 209.80 |
| tree2x9 | 475 | 489.08 | 478 | 485.86 | 296 | 330.14 | **257** | **276.60** | 472 | 491.84 |
| Average | 308.90 | 316.38 | 310.50 | 315.75 | 239.20 | 262.26 | **207.30** | **224.50** | 307.30 | 317.47 |
| p-value | 6.71e-14 | | | | | | | | | |

Table 4.1 – Experimental results of MA using 5 different recombination operators.

the best results not only in terms of “Best" but also in terms of “Avg" over the 20 test instances. From the average values listed in the last row, we find that OX2 is a much more suitable operator than the other operators for CBP. Also, the non-parametric Friedman test on the 5 groups of best results leads to a $p$-value=6.71e-14 $< 0.05$, confirming that there exists a statistically significant difference among the compared results.

Table 4.2 reports the comparative results between the best MA with OX2 (called $MA_{OX2}$) and $TScb$, which is the state-of-art algorithm for CBP presented in [Rod+15]. Table 4.2 shows the same information as in Table 4.1, except for the column “CC" which represents the difference between the best values found by $TScb$ and $MA_{OX2}$. A negative “CC" indicates a worse result of $MA_{OX2}$ compared to $TScb$. It is clear that for the 20 test graphs, $MA_{OX2}$ does not compete well with $TScb$. Indeed, $TScb$ is a powerful iterated tabu search algorithm which uses three dedicated neighborhoods to effectively explore the search space. Also, the Wilcoxon signed-rank test with the two groups of best values leads to a $p$-value=1.31e-4 $< 0.05$, confirming the statistical significance between the compared results. This comparison tends to indicate that in practice, it is not enough for the MA to apply a recombination operator and a simple local search. In addition to a suitable recombination operator, the MA needs a powerful local optimization procedure to ensure an effective exploitation.

| Graph | $MA_{OX2}$ | | $TScb$ | | CC |
|---|---|---|---|---|---|
| | Best | Avg | Best | Avg | |
| nos6 | 216 | 227.84 | 22 | 23.50 | -194 |
| path1000 | 226 | 247.54 | 8 | 8.90 | -218 |
| nos4 | 28 | 34.48 | 10 | 10.00 | -18 |
| tree10x2 | 28 | 29.26 | 28 | 28.00 | 0 |
| cycle1000 | 226 | 246.94 | 8 | 8.50 | -218 |
| mesh2D8x25 | 57 | 62.94 | 8 | 8.20 | -49 |
| caterpillar29 | 100 | 127.32 | 24 | 25.80 | -76 |
| mesh3D6 | 73 | 78.26 | 31 | 31.00 | -42 |
| hypercube11 | 952 | 1010.48 | 570 | 582.20 | -382 |
| cycle475 | 99 | 110.76 | 5 | 5.80 | -94 |
| mesh2D28x30 | 270 | 287.46 | 30 | 174.00 | -240 |
| mesh3D11 | 507 | 522.82 | 336 | 336.80 | -171 |
| can___715 | 293 | 316.70 | 60 | 65.80 | -233 |
| impcol_b | 20 | 26.72 | 17 | 17.00 | -3 |
| path475 | 102 | 112.94 | 5 | 5.60 | -97 |
| 494_bus | 128 | 138.62 | 46 | 56.10 | -82 |
| tree21x2 | 124 | 140.84 | 116 | 116.00 | -8 |
| caterpillar44 | 281 | 321.70 | 39 | 54.00 | -242 |
| impcol_d | 159 | 169.74 | 38 | 43.10 | -121 |
| tree2x9 | 257 | 276.60 | 63 | 64.20 | -194 |
| Average | 207.30 | 224.50 | **73.20** | **83.23** | |
| p-value | 1.31e-4 | | | | |

Table 4.2 – Comparison between $MA_{OX2}$ and $TScb$ [Rod+15].

## 4.5 Understanding the performance differences of the compared crossovers

In Section 4.4, we observe that OX2 excels compared to the other crossover operators. In this section, we investigate the reasons why OX2 has a better performance than the other crossovers. For this, we follow [WLH10] and study the evolution of the population diversity. To this end, we consider two diversity indicators: average solution distance $D_{avg}(P)$ and population entropy $E_p(P)$.

### 4.5.1 Distance and population entropy

We first introduce the average solution distance $D_{avg}(P)$ of the population.

$$D_{avg}(P) = \frac{2}{|P|(|P|-1)} \sum_{i=1}^{|P|} \sum_{j=i+1}^{|P|} d_{ij} \tag{4.4}$$

where $d_{ij}$ is the distance between two solutions $\gamma_i$ and $\gamma_j$ of $P$, which is defined as the number of the adjacent connections that are contained in $\gamma_i$ but not in $\gamma_j$. For example, given two solutions $\gamma_1$={h a b d e f c g} and $\gamma_2$={b a c h g d f e}. The set of adjacent

connections is {ha, ab, bd, de, ef, fc, cg, gh} for $\gamma_1$ and {ba, ac, ch, hg, gd, df, fe, eb} for $\gamma_2$. The common adjacent connections are {ab, ef, gh} (ba and ab are the same connections). The distance $d_{12}$ equals thus 8-3=5. This distance is used in [FM96] to deal with TSP whose solutions have the symmetry feature. As shown in Fig. 4.1, CBP solutions have the feature of symmetry, so the use of this distance measure is very important for CBP.

Another indicator to describe the population diversity is the population entropy $E_p(P)$ [FF96].

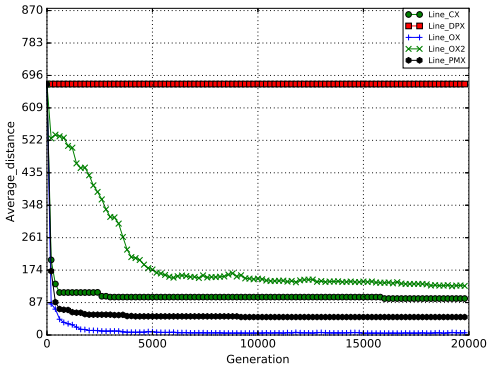
$$E_p(P) = \frac{-\sum_{i=1}^{n}\sum_{j=1}^{n}\left(\frac{n_{ij}}{|P|}\right)\log\left(\frac{n_{ij}}{|P|}\right)}{n\log n} \tag{4.5}$$

where $n_{ij}$ represents the number of times that variable $i$ is set to value $j$ in all solutions in $P$. One notices that $E_p(P)$ varies in the interval [0,1]. When $E_p(P)$ equals 0, all the solutions of $P$ are identical. A large $E_p(P)$ value indicates a more diverse population.

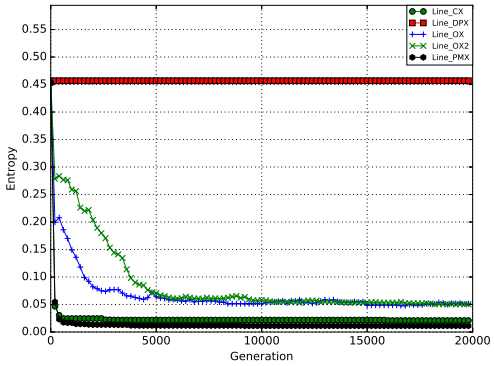
The instance 'nos6' is a representative large graph with 675 nodes from practical application and rather difficult, so we use it as a working example. Figure 4.2 shows the average distance, average entropy and average best objective value found in 50 independent executions over the graph 'nos6'. Under 5000 generations, the population of the MA with OX2 has a high average distance and entropy, leading to much better solutions. From generations 5000 to 20000, the entropy is identical to that of OX, and the best average objective found stops decreasing. These observations remain valid for all test graphs except the graph 'impcol_b' (even if the MA with OX2 does not have a large population distance and entropy, it gets good results comparing to others). Therefore, for the operators CX, OX, OX2 and PMX, a higher entropy and average distance of population leads to a good quality solution. However, what is surprising is that the average distance and entropy with DPX always stay at a high level for all test graphs, yet the quality of solutions found is not as good as that of the other operators. To shed light on this behavior, we show a deeper analysis of the interaction between the crossover mechanism and the characteristics of problem in the next section.

### 4.5.2 Interaction between crossover and problem characteristics

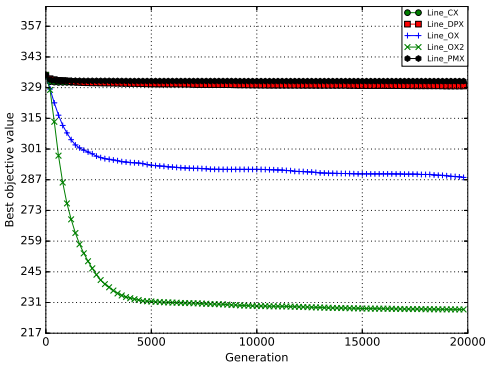
In Section 4.5.1, we find that the recombination operator with a higher entropy and average distance of the population generally helps to find solutions of good quality. However, the DPX operator fails to reach good solutions even if the entropy and average distance of population under the MA with DPX always stay at a high level. From Figure




**(a)** Average distance of the population in 20000 generations




**(b)** Entropy of the population in 20000 generations




**(c)** Best objective value found in 20000 generations

Figure 4.2 – Distance and population entropy applied to the instance nos6.

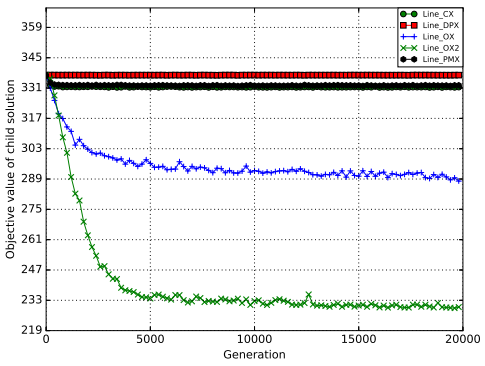



Figure 4.3 – Average objective value of the child solution over 50 independent executions.

4.3, which presents the average objective value of the offspring solutions of instance nos6 using the average data of 50 independent executions, we find that DPX does not generate high quality offspring solutions during the search.

To understand why DPX does not help the MA to find good quality solutions, we first recall that DPX is designed for TSP, which is a quite different problem compared to CBP considered in this work. In [Boe95], it is observed that for TSP, the average distance between local optima is similar to the average distance between a local optimum and the global optimum and common substrings in the local optima also appear in the global optimum. DPX explores this particular feature of TSP and is thus suitable to TSP. However, CBP has a totally different objective function and does not share the above characteristic.

Indeed, to calculate the objective value of a solution of TSP, we only need to consider, for each vertex, its two linked edges and sum up the edge distances of the tour. In this case, solution sub-tours (substrings) are clearly a key component which characterizes the solutions. Yet in a solution of CBP, we need to consider for each vertex all the edges linked to the vertex in the graph, such that the objective value (see Equation (2.1)) relies on the largest cyclic bandwidth. In the case of CBP, the key point is the relative position for the pairs of nodes which are linked by an edge. Therefore given that TSP and CBP have very different characteristics, a good crossover operator designed for TSP (in our case, DPX) may fail when it is applied to CBP.

This inspires us that the choice and design of recombination operators are not only relied on the entropy and average distance of population, but also on the characteristics of the considered problem.

## 4.6 Conclusions

In this chapter, we have investigated the memetic framework for solving the NP-hard Cyclic Bandwidth problem. We have compared five permutation recombination operators (CX, OX, OX2, PMX and DPX) within a basic memetic algorithm which uses a simple descent procedure for local optimization. The experimental results indicate that OX2 achieves the best performance for the test instances. We have studied the population diversity measured by the average distance and entropy of the MA variants using different recombination operators. We have also explored the correlation between the population diversity and the performance of the studies MA variants. This study indicates

that the basic memetic algorithm combining an existing recombination operator and a simple descent local search procedure is not competitive compared to the state-of-the-art algorithms. Additional (preliminary) experiments with MAs using an enforced local optimization procedure (such as the powerful local search algorithms presented in [RHR19; Rod+15]) have not led to more convincing results. Meanwhile, given the excellent performances achieved by MAs on many difficult optimization problems, this work invites more research effort on seeking meaningful recombination operators suitable for CBP. It is then expected that a MA integrating such a recombination operator and a powerful local optimization procedure would achieve state-of-the-art performances.

In the next chapter, we will consider the minimum linear arrangement problem which is also a graph layout problem. We propose a set based neighborhood heuristic algorithm to solve it.

Chapter 5

# A SET BASED NEIGHBORHOOD HEURISTIC ALGORITHM FOR SOLVING THE MINIMUM LINEAR ARRANGEMENT PROBLEM

---

In this chapter, we investigate a set based neighborhood heuristic algorithm under the framework of iterated local search for MinLA. The algorithm consists of two phases: a descent phase exploring two neighborhoods (a median based neighborhood and a set based neighborhood with a decomposition method to reduce the computational complexity) as well a perturbation phase. Experimental results show that the proposed set based neighborhood is more effective than the traditional 2-flip neighborhood.

## 5.1 Introduction

The minimum linear arrangement problem (MinLA) is a well-known labelling problem, first introduced in [Har64] to minimize the average absolute errors in designing error-correcting codes. Afterwards, people found other important applications in VLSI layout, biological applications, graph drawing, software diagram layout and job scheduling [DPS02; LW99]. It is proven to be a $\mathcal{NP}$-hard problem in [Har82].

Its mathematical model could be defined as follows. Let $G(V, E)$ be a finite undirected graph, where $V$ ($|V| = n$) represents the set of vertices and $E$ depicts the set of edges. Given a mapping $\varphi : V \rightarrow \{1, 2, ...n\}$ which represents a linear arrangement $\varphi$, the sum of edge length (the cost) for $G$ with respect to $\varphi$ is defined as:

$$S_{LA}(G, \varphi) = \sum_{(u,v)\in E} |\varphi(u) - \varphi(v)| \tag{5.1}$$

The goal is to find an arrangement $\varphi^*$ whose sum of edge length $S_{LA}(G, \varphi^*)$ is minimum.

There exist some exact methods to obtain the optimal solutions for special families of graphs such as trees, rooted trees, hypercubes, meshes, outerplanar graphs, and others ([DPS02]). Besides that, many heuristic algorithms are developed to solve MinLA like the spectral sequencing method (SSQ) [JM92], improved frontal increase minimization (IFIM) [Mca99], multi-scale algorithm (MS) [KH02], algebraic multigrid scheme (AMG) [SRB04], simulated annealing (SA) [Pet03a] [Pet03b] [RHT08a], population-based algorithms [RHT06] [Por05] [SS09] and variable neighborhood search [MUP16]. A detail review is made in Section 5.2.

Generally, neighborhoods could be classified in two types: parameter-based neighborhoods and set based neighborhoods according to [GR17]. A parameter-based neighborhood is for example a 2-flip neighborhood in some heuristic algorithms of MinLA. Using the best-improvement strategy, these algorithms should enumerate all available configurations by swapping the labels of two vertices and choose one of them to replace the current solution. On the contrary, a set based neighborhood is generated by restricting enumerating on some set $S$ which is composed of partial decision variables. A special characterization of set based neighborhood is that we can identify the best solution in a single step in a polynomial time for an exponentially large number of solutions. Based on this idea, a class of set based neighborhoods is introduced in [GR17]. The authors proposed a class of set based neighborhoods on the vertex set $S$ which is constructed by a maximal independent set (MIS) of the input graph that could be obtained by a simple greedy heuristic algorithm. The authors also estimated that these set based neighborhoods are more powerful than other neighborhoods because of the combinatorial leverage and did not verify the effectiveness of the set based neighborhood for MinLA in a heuristic algorithm. So the objective of this work is to investigate the effectiveness of the set based neighborhood compared to the traditional 2-flip neighborhood in computational experiments.

The proposed algorithm is realized by a two phase iterated heuristic: a descent phase and a perturbation phase. In the descent phase, the algorithm alternates a dedicated neighborhood (median based neighborhood) to explore the search space efficiently and a set based neighborhood with a decomposition method to search the solutions in a larger area. In the perturbation phase, we employ a rotation operation to help escape from local optimum. The results show that the set based neighborhood is more effective than the traditional 2-filp neighborhood for improving the solution quality.

This chapter is organized as follows. The previous relevant work is briefly reviewed in

Section 5.2. Then we introduce our algorithm in Section 5.3. In Section 5.4, computational results on the benchmark instances and comparisons with reference results in the literature are presented. After that, we describe additional experiments to investigate the influences of main algorithmic components on the global performance of the algorithm in Section 5.5. In the last, conclusions and perspectives are given in Section 5.6.

## 5.2 Previous work

There has been extensive theoretical and practical researches in recent decades.

Many exact algorithms are developed to solve special graphs like undirected tree [Shi79], 1/3 balanced decomposition trees of bounded degree graphs [Bar+04], chord graphs [RH08] and incomplete hypercubes [Mil+15]. There are also some work focusing on lower bounds of general graphs. For example, using polyhedral approach, a cutting plane algorithm is presented in [Ama+08]. The results show that the proposed algorithm can yield competitive lower bounds in a reasonable time. In 2011, a linear-programming based approach to compute lower bounds of the benchmark instances is proposed in [CLS11]. It is the first time to show that the best known solutions are not far from the lower bounds for most benchmark instances. Also, some other theoretical work [Sha+00] has found that the optimal solution of MinLA can derive the upper bounds and the lower bounds for the bipartite crossing number problem. Recently, [AMS11] shows that MinLA has no polynomial time approximation scheme, unless NP-complete problems can be solved in randomized subexponential time.

For the practical work, much effort has been put into developing effective heuristic algorithms.

In [JM92], the spectral sequencing method (SSQ) was introduced by using eigenvectors corresponding to the second smallest laplace eigenvalue of a graph. The results showed that this method displayed good behavior in most cases. Later, Mcallistar [Mca99] introduced a heuristic method to construct the sequence order of labels based on the degrees and previous labeled vertices. This method performed better than the previous method not only in quality of solutions but also in execution time. It is worth noting that, many researches made use of the above two methods as the initialization in their algorithms.

In 2002, a linear-time algorithm based on the multi-scale (MS) paradigm was devised in [KH02]. The multi-scale techniques convert a high-dimensional problem in an iterative fashion into ones of increasingly lower dimensions, via a process called coarsening. As the

scale of lower dimension problems is small and easy-solving, they can be solved exactly. Then the solution is progressively projected back to high dimension problem until the original problem is returned. The experimental results of this approach were comparable with the best known results and the running time is much more attractive. Another multi-scale algorithm based on algebraic multigrid scheme (AMG) was presented in [SRB04]. Compared with MS, AMG uses weighted aggregation other than strict aggregation in coarsening scheme. The authors declared that AMG can get better results than MS in linear time.

The simulated annealing method was much studied in the last decade. In [Pet03a] and [Pet03b], Petit proposed two heuristic algorithms combining SSQ and simulated annealing (SA) to solve MinLA on a set of large graphs. The two algorithms compared favorably with other heuristic algorithms in the solution quality and running time. The author has also presented the benchmark set which is employed in our work and introduced in Section 5.4.1.

A genetic hill-climbing algorithm was presented in [Por05]. It used a local search procedure to improve the solution iteratively and two crossover operations to make the diversification. In the local search, the algorithm randomly chooses one solution $\varphi_n$ in the neighborhood to replace the current solution $\varphi$ if $\varphi_n$ is not worse than $\varphi$. The improving procedure ends with a maximum non-improving limit $n \log_{10} n$ ($n$ is the number of vertices in the instance.). The experiments revealed that randomly taking one solution from the neighborhood is better than fully examining the neighborhood in terms of the search efficiency. Two different crossover operators were adopted to provide a weak and a strong operation respectively. The results showed that it is comparable with other SA algorithms on most instances.

After that, Rodriguez-Tello *et al.* proposed a memetic algorithm [RHT06] incorporating a specialized crossover operator, a local search operator based on SA methodology and an initialization procedure by Mcallistar method [Mca99]. The results of this algorithm were superior than above algorithms presented. Later in [RHT08a], a two-stage simulated annealing algorithm ($TSSA$) was developed to resolve MinLA effectively. The algorithm operated in two steps: initializing the solution with Mcallistar method [Mca99] and improving the solution iteratively by simulated annealing based on exchanging two labels. $TSSA$ alternated the search with two different neighboring functions with a pre-fixed probability. An extended evaluation function was also employed to differentiate the solutions with same objective value. The results showed that $TSSA$ outperforms the pre-

vious algorithms. As the computational results were superior than the other algorithms, we take $TSSA$ as reference algorithm in this work.

In 2009, Sharma *et al.* proposed a hybrid approach [SS09] which incorporates SA under the framework of evolutionary algorithm. It utilised the features of level structure, Depth First Search (DFS), Frontal Increase Minimisation (FIM) method and Spectral Sequencing (SS) of the graphs in the procedure of initialization. The proposed technique produced results that are well comparable with the existing approaches known for their good results.

Two years later, a new algorithm incorporating scatter search and path relinking (SSPR) to solve MinLA was proposed in [Glo+11]. The proposed algorithm includes three parts: the diversification generation method, the improvement method and the combination method. Based on the Mcallistar method in [Mca99], SSPR introduced a probability-choosing strategy to increase the diversification of the population. In the improvement procedure, SSPR used the ejection chain to enlarge the neighborhood. Finally, SSPR employed a path relinking methodology to help escape from local optimum. The experimental results showed that SSPR could be comparable with $TSSA$ in most cases under the 1000 seconds of cutoff time.

[MUP16] offered a variable neighborhood search (VNS) algorithm to solve MinLA. The authors used a sequential variable neighborhood descent based on the swapping and rotation operation. When there is no improving solutions in the neighborhoods, the algorithm enters the perturbation phase to get out of the local optimum. The two procedures are repeated until the running time reaches 2000 seconds. The computational results showed that the proposed VNS is comparable with $TSSA$ in some cases.

## 5.3 Set based neighborhood heuristic algorithm

Iterated local search is an effective framework and it is widely applied in many other NP-hard problems. As is shown in Algo 8, the proposed set based neighborhood heuristic algorithm ($SBNH$) starts by the initialization phase (line 3 in Algo 8) by the Mcallsitar method [Mca99] (See Section 5.3.2). As well, the best found solution $\varphi^*$ and the recorded solution $\varphi_b$ are set as the initialization solution $\varphi$ (lines 4-5 in Algo 8). Then $SBNH$ gets into the descent phase (lines 7-11 in Algo 8) alternating between exploring the median based neighborhood (See Section 5.3.3) and the set based neighborhood (See Section 5.3.3) until there is no improving solutions in any of the neighborhoods. After

that, the perturbation phase (See Section 5.3.4) is triggered to help escape from the local optimum (line 12 in Algo 8). The two phases repeat until the cutoff time $T_{max}$ is reached.

**Algorithm 8** Set based neighborhood heuristic algorithm for MinLA

1: **Input**: Finite undirected graph $G(V, E)$, objective function $f(\varphi)$, repeat times $MaxiR$, pattern size $SizeP$, perturbation strength $StrenP$ and cutoff time limit $T_{max}$
2: **Output**: The best solution found $\varphi^*$
3: $\varphi \leftarrow InitialMcallistar()$
4: $\varphi^* \leftarrow \varphi$
5: $\varphi_b \leftarrow \varphi$
6: **while** the cutoff time limit $T_{max}$ is not reached **do**
7:  **while** $f(\varphi^*) < f(\varphi_b)$ **do**
8:  $\varphi_b \leftarrow \varphi^*$
9:  $(\varphi, \varphi^*) \leftarrow LS_s(\varphi, \varphi^*)$
10:  $(\varphi, \varphi^*) \leftarrow LS_{SB}(\varphi, \varphi^*, MaxiR, SizeP)$
11:  **end while**
12:  $\varphi \leftarrow Perturb(\varphi, StrenP)$
13: **end while**
14: **return** $\varphi^*$

### 5.3.1 Representation and evaluation function

As MinLA is a typical labeling problem, we use a permutation $l$ to represent the solution, where $l(i)$ represents the label assigned to vertex $i$. This representation is used in the descent phase and perturbation phase. For the initialization procedure (See Section 5.3.2), we use another permutation $t$ where $t(i)$ denotes the vertex assigned to label $i$. We employ the original fitness function in Equation 5.1 other than the extended evaluation function introduced in [RHT08b].

### 5.3.2 Initialization

A well devised initialization procedure is helpful to promote the searching efficiency. Our algorithm $SBNH$ employs the Mcallistar method [Mca99] to obtain an initial solution. Mcallistar method uses a greedy heuristic to choose vertices to the labels from 1 to $n$. First of all, this heuristic method randomly chooses a vertex for label 1. Then it follows a frontal increase minimization (FIM) strategy to fill the label from 2 to n with the unplaced vertices. To well describe the mechanism for choosing a vertex for label i, we first give some preliminary definitions: let $P_i$ ($|P_i| = i - 1$) represent the set of

vertices placed from label 1 to label $i-1$, $U_i$ depicts the set of unplaced vertices and $F_i = \{u \in U_i : v \in P_i, (u,v) \in E\}$ denotes the set of unplaced vertices which have adjacent vertices in $P_i$. The basic idea is to choose the vertex from $F_i$ which has fewest adjacent vertices in the set $U_i - F_i$. To realize that, Mcallistar method introduces an index $tf(u) = deg(u) - 2 * tp(u)$ for each vertex $u \in F_i$ to describe its connection with the set $U_i - F_i$, where $deg(u)$ denotes the degree for the vertex $u$, $tp(u)$ depicts the number of placed vertices that is adjacent to $u$.

$$tp(u) = |\{v \in P_i, (u,v) \in E\}| \tag{5.2}$$

At each step, the vertex $u \in F_i$ with smallest $tf(u)$ is chosen for the label $i$. The whole procedure repeats until all the vertices are set.

### 5.3.3 Descent phase

In the descent phase, we operate the descent local search in two different neighborhoods. The first neighborhood (See Section 5.3.3) is a median based neighborhood from [RHT08b]. Because of its simple structure and meaningful design, it is helpful to explore the search space to reach a local best solution quickly. Another neighborhood is a set based neighborhood (See Section 5.3.3) to advance the search when there is no improving solutions in the median based neighborhood.

#### Median based neighborhood

We first explore the median based neighborhood in the descent phase. As is shown in Algo 9, a descent local search $LS_s(\varphi, \varphi^*)$ operates over the median based neighborhood $N_s(u, \varphi)$. For each vertex $u \in V$, we evaluate all its neighboring solutions and choose the best one $\varphi'$ (line 7 in Algo 9) to replace the current solution $\varphi$ if $\varphi'$ is better than $\varphi$ (lines 8-10 in Algo 9). Then the best found solution $\varphi^*$ is updated if an improved solution is met (lines 11-13 in Algo 9). This local search ends when there exist no improving solutions in the neighborhood and the current solution $\varphi$ and the best found solution $\varphi^*$ are returned (line 16 in Algo 9).

The proposed median based neighborhood $N_s(u, \varphi)$ is introduced in [RHT08b]. It is defined as follows. Given a vertex $u \in V$ and its adjacent vertices set $A(u) = \{v_1, v_2, v_3 ... v_{deg(u)}\}$, we could sort their labels $p_i \in P(u) = \{\varphi(v), v \in A(u)\}$ in an ascending order: $p_1 < p_2 <$

**Algorithm 9** Descent local search $LS_s(\varphi, \varphi^*)$

**Input**: Finite undirected graph $G(V, E)$, objective function $f(\varphi)$, the current solution $\varphi$ and the best solution found $\varphi^*$
**Output**: The current solution $\varphi$ and the best solution found $\varphi^*$
$\varphi_t \leftarrow \varphi$
**while** $f(\varphi) < f(\varphi_t)$ **do**
  $\varphi_t \leftarrow \varphi$
  **for** each $u \in V$ **do**
    $\varphi' \leftarrow$ Best of $N_s(u, \varphi)$
    **if** $f(\varphi') < f(\varphi)$ **then**
      $\varphi \leftarrow \varphi'$
    **end if**
    **if** $f(\varphi') < f(\varphi^*)$ **then**
      $\varphi^* \leftarrow \varphi'$
    **end if**
  **end for**
**end while**
**return** $\varphi$, $\varphi^*$

$... < p_{deg(u)}$. The definition of median label $Med(u)$ of $P(u)$ is given by:

$$Med(u) = \begin{cases} p_{(deg(u)+1)/2} & \text{if deg(u) is odd,} \\ \frac{1}{2}(p_{deg(u)/2} + p_{(1+deg(u)/2)}) & \text{if deg(u) is even.} \end{cases} \tag{5.3}$$

The set of suitable vertices of vertex $u$ is defined as: $S(u) = \{v \in V, Med(u) - 2 \leq \varphi(v) \leq Med(u) + 2\}$. Then the proposed median based neighborhood for the vertex $u$ with respect to $\varphi$ could be expressed as follows:

$$N_s(u, \varphi) = \{\varphi' = \varphi \oplus swap(u, v) : v \in S(u)\}. \tag{5.4}$$

where the operation $swap(u, v)$ is to swap the labels of verties $u$ and $v$. To improve the computational efficiency, we calculate only the changing part from solution $\varphi$ to $\varphi'$. For each swap operation $swap(u, v)$, only the edges related to vertex $u$ and $v$ are changed. Therefore the complexity of evaluating each neighboring solution $\varphi'$ is $O(deg(u) + deg(v)) < O(D + D) = O(D)$, where $D$ is the maximal degree of the vertex $u \in V$. As the size of set $S(u)$ equals to 5, the complexity for each vertex $u$ is $O(5\dot{D}) = O(D)$. That means that the first local search procedure is fast to carry out and we could rapidly find out the local optimum by this effective neighborhood. After that, the search explores the

set based neighborhood to continue the descent search in a larger area.

### Set based neighborhood

In [GR17], the authors proposed a class of set based neighborhoods for MinLA without presenting practical validations. The set based neighborhood is described as follows. Given a undirected graph $G(V, E)$, the set based neighborhood $N_{SB}(N_0, \varphi)$ is generated by making a full arrangement of the labels in the labeling set $L_0$, where $L_0 = \{\varphi(u), u \in N_0\}$ and $N_0$ is the maximal independent set. For example, we are given a 8 vertices graph $V = \{a, b, c, d, e, f, g, h\}$ and a solution $\varphi = \{2, 6, 4, 1, 5, 8, 3, 7\}$ ($\varphi(a) = 2$, $\varphi(b) = 6$ and so on.), the MIS $N_0 = \{a, c, h\}$ and its labeling set $L_0 = \{2, 4, 7\}$. Therefore we have 3!=6 arrangements in the set based neighborhood $N_{SB}(N_0, \varphi)$ with respect to the solution $\varphi$ and the MIS $N_0$: $\{2, 6, 4, 1, 5, 8, 3, 7\}$, $\{2, 6, 7, 1, 5, 8, 3, 4\}$, $\{4, 6, 2, 1, 5, 8, 3, 7\}$, $\{4, 6, 7, 1, 5, 8, 3, 2\}$, $\{7, 6, 2, 1, 5, 8, 3, 4\}$, and $\{7, 6, 4, 1, 5, 8, 3, 2\}$.

There are $n_0!$ solutions in the set based neighborhood, where $n_0$ is the size of the MIS $N_0$. Even the set based neighborhood is $n_0!$ large, we could identify the best solution in a polynomial time by transforming finding the best solution in the neighborhood into a minimum-weight perfect matching problem (MWPMP) [CR99]. It is worth noting that, identifying the MIS of a graph is also a $\mathcal{NP}$-hard problem, we employ a heuristic method in Algo 10 to find out the MIS for the input graph. Using this method, we could find out the MIS quickly.

**Algorithm 10** A Heuristic to identify the maximum independent set $MIS(G)$

1: **Input**: Finite undirected graph $G(V, E)$
2: **Output**: The maximum independent set $N_0$
3: $N_0 \leftarrow \emptyset$
4: $N_r \leftarrow V$
5: **while** $N_r \neq \emptyset$ **do**
6: select a random vertex $u \in N_r$
7: $N_0 \leftarrow N_0 \cup u$
8: $N_r \leftarrow N_r \setminus (A(u) \cup \{u\})$ // $N(u)$ depicts the set of adjacent vertices of $u$
9: **end while**
10: **return** $N_0$

To create a MWPMP, we start with generating a MIS by Algo 10. For a given undirected graph $G(V, E)$, the MIS algorithm (Algo 10) firstly initializes the set $N_0$ as an empty set and the set $N_r$ as $V$ (lines 3-4 in Algo 10). Then we enter the loop to update

$N_0$ selecting a random vertex $u$ from $N_r$ (lines 6-7 in Algo 10) and update the $N_r$ by deleting the vertex $u$, as well as its adjacent vertices (line 8 in Algo 10). The cycle ends when the set $N_r$ gets empty. And the MIS $N_0 \subset E$ is returned. With the independent vertex set $N_0 \subset V$ and its labeling set $L_0$, a MWPMP is created on the graph $G_0 = (N_0, L_0, E_0)$ (See Figure 5.1), where $E_0 = \{(p, q) : p \in N_0, q \in L_0\}$ and the weight for each edge $w(p, q)$ is defined by the follows:

$$w(p, q) = \sum_{k \in N(p)} |\varphi(k) - q| \tag{5.5}$$

MWPMP is much studied in the past and solved in a polynomial time $O(n_0^4)$ [Ger95].

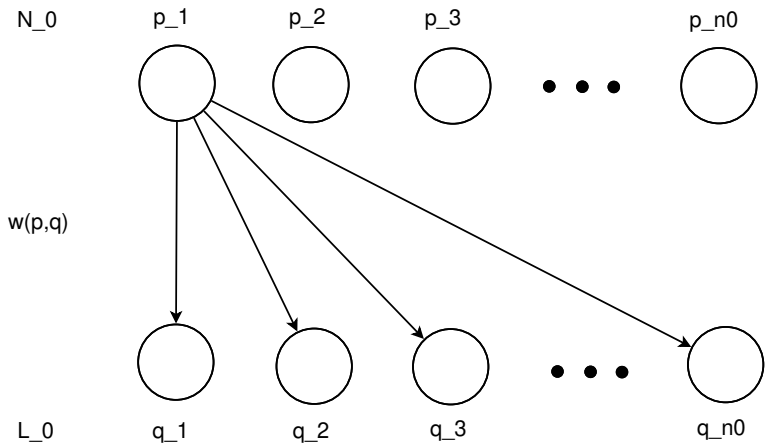



Figure 5.1 – Transforming finding the best solution in the set based neighborhood into MWPMP.

However, the size of $N_0$ is rather large when the instance is in large scale. For example, there is an instance named "gd96a" which has 1096 vertices and 1676 edges, and its maximum independent set $N_0$ generated by Algo 10 normally has over 450 vertices which leads to lots of time consuming for each iteration. For this reason, we increase the searching efficiency of the local search $LS_{SB}(\varphi, \varphi^*, MaxiR, SizeP)$ by employing a decomposition method [MRW18] which is successfully applied to solve the quadratic assignment problem (QAP) [Law63]. The main idea of the decomposition method is to divide the set of decision variables into several no-overlapping subsets and then enumerate all the configurations generating by the subsets. To apply the set based neighborhood, we divide the maximal independent set $N_0$ into several subsets and identify the best solution in the neighborhood generated by each subset with the method in [Ger95].

The pseudo code of the local search $LS_{SB}(\varphi, \varphi^*, MaxiR, SizeP)$ is presented in Algo 11. The counter $Cnt$ is initialized as 0 (line 3 in Algo 11). After the recorded solution $\varphi_t$ is set

as $\varphi$ (line 5 in Algo 11), we generate the maximum independent set $N_0$ by Algo 10 (line 6 in Algo 11) and divide vertex set $N_0$ into different subsets $\{P_1, P_2, ...P_m\}$ where each subset is an independent vertex set which consists of $SizeP$ vertices (line 7 in Algo 11), where $SizeP$ is an input parameter to control the size of each subset. In the inner loop, we replace the current solution $\varphi$ with the best solution in the set based neighborhood relative to the subset $P_i$ which is achieved by the solver [Ger95] (line 10 in Algo 11). The best found solution $\varphi^*$ gets updated if a better solution is obtained in terms of objective value (lines 11-13 in Algo 11). After visiting all the subsets $P_i$ $(i = 1, 2, ...m)$, the counter $Cnt$ is set to 0 if the solution $\varphi$ is not improved, otherwise $Cnt$ is added 1 (lines 16-21 in Algo 11). The outer loop ends if the current solution $\varphi$ has not been improved for $MaxiR$ successive times. Finally the current solution $\varphi$ and the best found solution $\varphi^*$ are returned (line 22 in Algo 11).

**Algorithm 11** Descent local search $LS_{RD}(\varphi, \varphi^*, MaxiR, SizeP)$

1: **Input**: Finite undirected graph $G(V, E)$, objective function $f(\varphi)$, repeating times $MaxiR$, pattern size $SizeP$, the current solution $\varphi$ and the best solution found $\varphi^*$
2: **Output**: The current solution $\varphi$ and the best solution found $\varphi^*$
3: $Cnt \leftarrow 0$
4: **while** $Cnt < MaxiR$ **do**
5: $\quad \varphi_t \leftarrow \varphi$
6: $\quad N_0 \leftarrow MIS(G)$
7: $\quad \{P_1, P_2, ...P_m\} \leftarrow DivideN(N_0, SizeP)$
8: $\quad i \leftarrow 1$
9: $\quad$ **repeat**
10: $\quad\quad \varphi \leftarrow MWPMPSolver(N_{SB}(P_i, \varphi))$
11: $\quad\quad$ **if** $f(\varphi) < f(\varphi^*)$ **then**
12: $\quad\quad\quad \varphi^* \leftarrow \varphi$
13: $\quad\quad$ **end if**
14: $\quad\quad i \leftarrow i + 1$
15: $\quad$ **until** i>m
16: $\quad$ **if** $f(\varphi) < f(\varphi_t)$ **then**
17: $\quad\quad Cnt \leftarrow 0$
18: $\quad$ **else**
19: $\quad\quad Cnt \leftarrow Cnt + 1$
20: $\quad$ **end if**
21: **end while**
22: **return** $\varphi$, $\varphi^*$

With the decomposition method above, we could reduce the time complexity from $O(n_0^4)$ to $O(n_0 * SizeP^3)$ and regulate the computational complexity by the input pa-

rameter $SizeP$. It is worth noting that the processes of different subsets $P_i \subset N_0$ are independent, which provides some possibility to optimise the computational speed by parallel computing in the future.

### 5.3.4 Perturbation phase

When the descent search stops, the perturbation phase is triggered to help escape from the local optimum. The pseudo-code is presented in Algo 12. We first initialize the counter $Count$ as 0 (line 3 in Algo 12). Then we randomly choose a vertex $u \in V$ and find its adjacent vertex $v \in A(u)$ which is farthest to $u$ based on the label (lines 5-6 in Algo 12):

$$v = \arg\max_{v \in A(u)} |\varphi(u) - \varphi(v)|. \tag{5.6}$$

Then $Step$ is randomly decided as a positive integer less than $|\varphi(u) - \varphi(v)|$ (line 7 in Algo 12). A rotation operation is carried out with a probability $p$ and the inverse direction with $1-p$ (lines 8-13 in Algo 12). The perturbation procedure finishes after $StrenP$ times operations and the perturbed solution $\varphi$ is returned.

**Algorithm 12** Perturbation phase $Perturb(\varphi, StrenP)$

1: **Input**: Finite undirected graph $G(V, E)$, perturbation strength $StrenP$ and the current solution $\varphi$
2: **Output**: The current solution $\varphi$
3: $Count \leftarrow 0$
4: **while** $Count < StrenP$ **do**
5: $u \leftarrow RandomChoose(V)$
6: $v \leftarrow FindFarNode(u, \varphi)$
7: $Step \leftarrow Rand(|\varphi(u) - \varphi(v)|)$
8: $\rho \leftarrow Randouble(0, 1)$
9: **if** $\rho < 0.5$ **then**
10: $\varphi \leftarrow Rotation(u, v, Step, \varphi)$
11: **else**
12: $\varphi \leftarrow Rotation(v, u, Step, \varphi)$
13: **end if**
14: $Count \leftarrow Count + 1$
15: **end while**
16: **return** $\varphi$

## 5.4 Experimental results

In this section, we present the experimental results of the proposed $SBNH$, a 2-flip neighborhood heuristic algorithm and the state-of-the-art algorithm $TSSA$. We first introduce the experimental settings and the instances in Section 5.4.1. Then we present the method to determine the input parameters for the proposed $SBNH$ in Section 5.4.2. Because the objective of this work is to reveal the effectiveness of the set based neighborhood, we make comparisons between the proposed algorithm to a 2-flip neighborhood heuristic algorithm in the following paragraph, as well as the state-of-the-art $TSSA$ in Section 5.4.3.

The proposed $SBNH$ and the 2-flip neighborhood heuristic algorithm ($2FNH$) share the same perturbation phase. The main difference between them is the neighborhoods in the descent phase. $SBNH$ employs the median based neighborhood $N_s$ presented in Section 5.3.3 and the set based neighborhood $N_{SB}$ introduced in Section 5.3.3 while the $2FNH$ integrate $N_s$ and a 2-filp neighborhood $N_t$ which is defined as follows:

$$N_t(u, \varphi) = \{\varphi' = \varphi \oplus swap(u, v) : u \in V, v \in V\}. \tag{5.7}$$

### 5.4.1 Instances and settings

The experimentation of this work was carried out on a set of 21 graphs [1] which are introduced in [Pet03b]. Table 5.1 summarizes the detailed information of each instance. Besides the basic information of the graph (graph name, vertex, edge and family), the lower bounds are given by a linear programming method in [CLS11] in column "Lower Bound" and the best results found (See column "EveBest") are also listed [2] with the gap to the lower bound in the last column (See column "Gap%"). The set of benchmark graphs consists of 4 random graphs using different probability $p$ to generate edges, one random geometric graph with a neighborhood radius $r = 0.075$, three regular graphs with known optimal values (marked in "*"), three graphs from finite element discretization (FE), five graphs from very-large-scale integration application (VLSI) and five graphs from graph drawing competitions (GD). The number of vertices varies in the range between 62 and 9800 while the number of edges is between 125 and 49820.

The $SBNH$ and the compared algorithm ($2FNH$) are coded in C/C++ and compiled

1. https://www.tamps.cinvestav.mx/ ertello/minla.php
2. https://www.tamps.cinvestav.mx/ ertello/minla.php

| Graph | Vertex | Edge | Family | Lower Bound | EveBest | Gap% |
|---|---|---|---|---|---|---|
| randomA1 | 1000 | 4974 | p=0.01 | 140634 | 866968 | 83.8 |
| randomA2 | 1000 | 24738 | p=0.05 | 4429294 | 6522206 | 32.1 |
| randomA3 | 1000 | 49820 | p=0.1 | 11463259 | 14194583 | 19.2 |
| randomA4 | 1000 | 8177 | p=0.0164 | 601130 | 1717176 | 65.0 |
| randomG4 | 1000 | 8173 | r=0.075 | 39972 | 140211 | 71.5 |
| bintree10 | 1023 | 1022 | 10-bintree | 3696* | 3696 | 0.0 |
| hc10 | 1024 | 5120 | 10-hypercube | 523776* | 523776 | 0.0 |
| mesh33x33 | 1089 | 2112 | 33x33-mesh | 31680* | 31729 | 1.5 |
| 3elt | 4720 | 13722 | FE | 44785 | 357329 | 87.5 |
| airfoil1 | 4253 | 12289 | FE | 40221 | 272931 | 85.3 |
| whitaker3 | 9800 | 28989 | FE | 144854 | 1143645 | 87.3 |
| c1y | 828 | 1749 | VLSI | 59971 | 62230 | 3.6 |
| c2y | 980 | 2102 | VLSI | 76253 | 78757 | 3.2 |
| c3y | 1327 | 2844 | VLSI | 113801 | 123145 | 7.6 |
| c4y | 1366 | 2915 | VLSI | 106942 | 114936 | 7.0 |
| c5y | 1202 | 2557 | VLSI | 88741 | 96850 | 8.4 |
| gd95c | 62 | 144 | GD | 443 | 506 | 12.5 |
| gd96a | 1096 | 1676 | GD | 77860 | 95242 | 18.3 |
| gd96b | 111 | 193 | GD | 1281 | 1416 | 9.5 |
| gd96c | 65 | 125 | GD | 402 | 519 | 22.5 |
| gd96d | 180 | 228 | GD | 2021 | 2391 | 15.5 |

Table 5.1 – Benchmark graphs with the lower bounds and best found results. (known optimal values are marked by *)

in gcc 4.4.7 using the optimization flag -O3. Considering the stochastic nature of these algorithms, each instance is executed independently 10 times, employing different random seeds, with a cutoff time of 3600 seconds. And all experiments are conducted on Linux system with a 2.5GHz Intel-E5-2670 CPU and 1GB RAM.

### 5.4.2 Determination of the input parameters

There are four parameters to be determined: the maximum repeating times $MaxiR$, the subset size $SizeP$, the perturbation strength $StrenP$ and the cutoff time $T_{max}$. The perturbation strength $StrenP$ is empirically fixed as 50 and the cutoff time is set as 3600 seconds. To decide the other two parameters, we have chosen 4 difficult and representative instances from the set of 21 instances: randomA4, whitaker3, c3y and gd96a.

We test 19 different combinations of ($MaxiR$,$SizeP$) for the proposed algorithm $SBNH$. For each combination, 10 independent executions are carried out over each instance using a cutoff time of 3600 seconds. To find out a good combination of parameters, we illustrate in Fig. 5.2 a curve of algorithm performance over 4 representative graphs respecting to 19 different combinations of parameters. In Fig 5.2, the y-axis represents the average best objective values of 10 executions over 4 representative graphs while the x-axis depicts the 19 different combinations of parameters. The curve in Fig 5.2 shows that for a

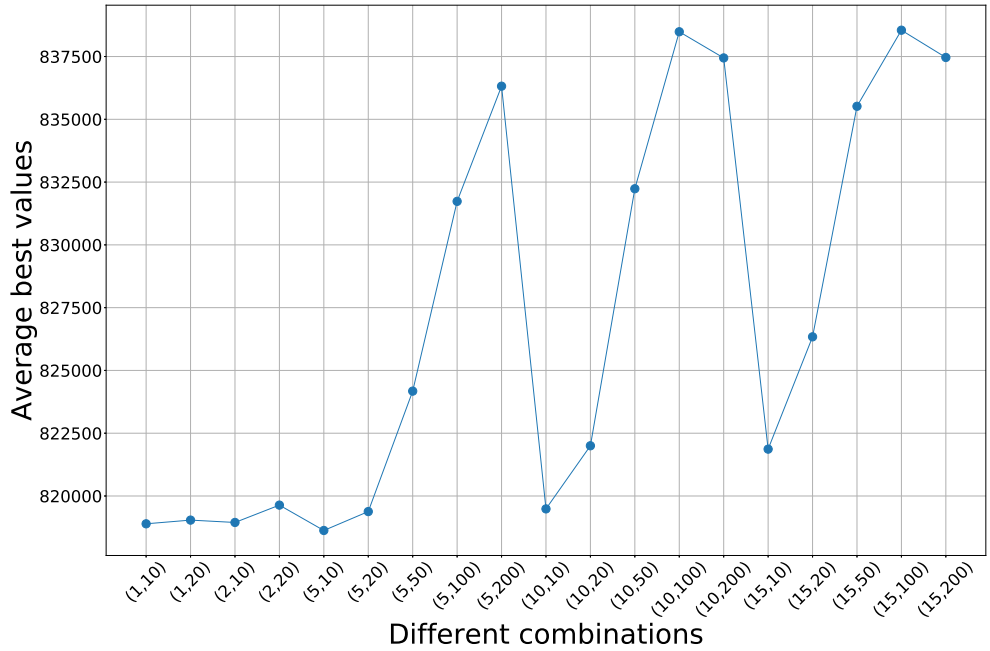



Figure 5.2 – The average best objective values of 10 executions over 4 representative graphs (randomA4, whitaker3, c3y and gd96a) according to the different combinations of ($MaxiR$,$SizeP$).

fixed $MaxiR$, the average result gets worse $SizeP$ increases; and the combination of input parameter (5,10) has a better result compared to others. For this reason, the parameter values for obtaining a good performance of $SBNH$ are: $MaxiR = 5$, $SizeP = 10$ and $StrenP = 50$.

### 5.4.3 Comparison with the other neighborhood algorithm

This section presents the results obtained from the experimental performance assessment of the reference algorithm $2FNH$ and our $SBNH$ algorithm. This analysis was carried out under the experimental conditions in Section 5.4.1.

Table 5.2 summarizes the results provided by this computational experiment organized by instance (See column Graph). The column "EveBest" depicts the best found values for each instance. Three columns are used to register the results produced by $SBNH$ and $2FNH$ including the best objective value found (column "Best"), the average value over 10 executions (column "Average") and the average time to find the best objective

| Graph | EveBest | *SBNH* | | | 2FNH | | | Δ |
|---|---|---|---|---|---|---|---|---|
| | | Best | Average | Time_avg | Best | Average | Time_avg | |
| randomA1 | 866968 | 870256 | 876713.2 | 3564.77 | 871288 | 878371.9 | 3550.27 | 1032 |
| randomA2 | 6522206 | 6534039 | 6557104.5 | 3402.40 | 6538720 | 6564729.4 | 3553.62 | 4681 |
| randomA3 | 14194583 | 14229092 | 14251947.4 | 2833.83 | 14225595 | 14252412.4 | 3321.05 | -3497 |
| randomA4 | 1717176 | 1727320 | 1737251.6 | 3566.52 | 1733244 | 1739208.9 | 3017.24 | 5924 |
| randomG4 | 140211 | 153391 | 175262.7 | 3271.34 | 161528 | 182245.7 | 2457.31 | 8137 |
| bintree10 | 3696 | 4549 | 4839.0 | 3481.58 | 6422 | 6966.5 | 3572.09 | 1873 |
| hc10 | 523776 | 523776 | 523776.0 | 0.00 | 523776 | 523776.0 | 0.00 | 0 |
| mesh33x33 | 31729 | 32652 | 33175.6 | 554.82 | 34037 | 37377.7 | 3213.47 | 1385 |
| 3elt | 357329 | 439848 | 539694.5 | 3191.96 | 478596 | 595144.1 | 3462.15 | 38748 |
| airfoil1 | 272931 | 352140 | 414330.3 | 3581.20 | 386693 | 480375.1 | 3567.86 | 34553 |
| whitaker3 | 1143645 | 1313695 | 1804234.3 | 3441.36 | 1314614 | 1806072.4 | 3227.80 | 919 |
| c1y | 62230 | 62881 | 64333.2 | 3189.73 | 63295 | 64621.1 | 3385.05 | 414 |
| c2y | 78757 | 80030 | 81630.9 | 1001.85 | 80696 | 82676.7 | 3494.54 | 666 |
| c3y | 123145 | 132057 | 137119.8 | 2835.83 | 136138 | 140975.1 | 3532.81 | 4081 |
| c4y | 114936 | 118271 | 122969.7 | 3293.06 | 121109 | 127815.9 | 3594.47 | 2838 |
| c5y | 96850 | 98528 | 104605.8 | 3460.93 | 103877 | 109673.6 | 3560.74 | 5349 |
| gd95c | 506 | 506 | 506.0 | 2.83 | 506 | 506.0 | 2.98 | 0 |
| gd96a | 95242 | 101435 | 104079.8 | 3174.46 | 103718 | 107436.7 | 3588.03 | 2283 |
| gd96b | 1416 | 1416 | 1429.8 | 278.87 | 1416 | 1431.2 | 2.22 | 0 |
| gd96c | 519 | 519 | 519.0 | 2.94 | 519 | 519.0 | 4.82 | 0 |
| gd96d | 2391 | 2391 | 2425.0 | 3070.59 | 2409 | 2416.6 | 1370.54 | 18 |
| Average | 1254773.43 | 1275180.57 | | | 1280390.29 | | | |
| Wilcoxon | | 1.64e-3 | | | | | | |

Table 5.2 – The experimental results between the $SBNH$ and the heuristic algorithm $2FNH$ over the 21 instances where each instance is executed 10 times independently.

value (column "Time.avg"). The last column ("$\Delta$") records the the gap between the best objective value obtained by $2FNH$ and $SBNH$ respectively. If the result is positive, it means that $SBNH$ performs better. Otherwise $2FNH$ works better.

Obviously, the proposed algorithm presents a better performance than $2FNH$ in most instances concerning the best objective values found and the average values of 10 independent executions. From the column $\Delta$, we find that $SBNH$ obtains better results concerning the best objective values found over 16 instances, and the compared algorithms achieve the same best objective value over 4 instances while $SBNH$ performs worse only on one instance"randomA3". There is only one instance "gd96d" where $2FNH$ get a better average value than $SBNH$. The average best objective value of all the instances also shows that $SBNH$ achieves a smaller objective value respecting to the overall performance in the last row of the Table 5.2. The Wilcoxon signed rank test with the two groups of best objective values leads to a $p$-value=1.64e-3 $< 0.05$, confirming the statistical significance between the compared results. It is worth noting that both $SBNH$ and $2FNH$ can find the optimal values for the instance "hc10" thanks to the initialization using the Mcallistar method. To sum up, $SBNH$ achieves a better performance than $2FNH$ for the most of cases in the benchmark. The set based neighborhood is proved more efficient than the traditional 2-flip neighborhood.

We also make a rough comparison with the state-of-the-art $TSSA$ [RHT08b] in Table 5.3. The experimental results of $TSSA$ in the Table 5.3 are from the reference article [RHT08b]. The authors executed each instance 10 independent times over 2GHz CPU and 1 GB Ram with a less than 3600 seconds running time. The information listed in Table 5.3 is shown employing the same column headings as those used in Table 5.2.

Indeed, $TSSA$ is a powerful two stage simulated annealing algorithm which uses a more diversified mechanism to effectively explore the search space. Also, the Wilcoxon signed rank test with the two groups of best objective values leads to a $p$-value=4.38e-4 $< 0.05$, confirming the statistical significance between the compared results. This comparison shows that, it is not enough for the $SBNH$ to employ descent local search and a simple perturbation operator. To develop an effective heuristic algorithm, we need a more efficient mechanism to offer the diversification during the search process.

| Graph | EveBest | *SBNH* | | | TSSA | | | Δ |
|---|---|---|---|---|---|---|---|---|
| | | Best | Average | Time_avg | Best | Average | Time_avg | |
| randomA1 | 866968 | 870256 | 876713.2 | 3564.77 | 866968 | 866975.4 | 86.50 | -3288 |
| randomA2 | 6522206 | 6534039 | 6557104.5 | 3402.40 | 6522206 | 6522221.6 | 181.00 | -11833 |
| randomA3 | 14194583 | 14229092 | 14251947.4 | 2833.83 | 14194583 | 14194585.5 | 279.10 | -34509 |
| randomA4 | 1717176 | 1727320 | 1737251.6 | 3566.52 | 1717176 | 1717179.6 | 90.00 | -10144 |
| randomG4 | 140211 | 153391 | 175262.7 | 3271.34 | 140596 | 140597.0 | 76.50 | -12795 |
| bintree10 | 3696 | 4549 | 4839.0 | 3481.58 | 3696 | 3697.1 | 38.80 | -853 |
| hc10 | 523776 | 523776 | 523776.0 | 0.00 | 523776 | 523776.0 | 1.20 | 0 |
| mesh33x33 | 31729 | 32652 | 33175.6 | 554.82 | 31856 | 31904.2 | 89.90 | -796 |
| 3elt | 357329 | 439848 | 539694.5 | 3191.96 | 359151 | 359176.0 | 1030.80 | -80697 |
| airfoil1 | 272931 | 352140 | 414330.3 | 3581.20 | 276381 | 276866.5 | 982.10 | -75759 |
| whitaker3 | 1143645 | 1313695 | 1804234.3 | 3441.36 | 1143645 | 1145304.7 | 3330.10 | -170050 |
| c1y | 62230 | 62881 | 64333.2 | 3189.73 | 62230 | 62234.4 | 32.80 | -651 |
| c2y | 78757 | 80030 | 81630.9 | 1001.85 | 78757 | 78810.8 | 46.70 | -1273 |
| c3y | 123145 | 132057 | 137119.8 | 2835.83 | 123145 | 123151.1 | 93.30 | -8912 |
| c4y | 114936 | 118271 | 122969.7 | 3293.06 | 114936 | 114971.6 | 88.10 | -3335 |
| c5y | 96850 | 98528 | 104605.8 | 3460.93 | 96850 | 96877.2 | 69.20 | -1678 |
| gd95c | 506 | 506 | 506.0 | 2.83 | 506 | 506.1 | 2.10 | 0 |
| gd96a | 95242 | 101435 | 104079.8 | 3174.46 | 95263 | 95277.9 | 61.00 | -6172 |
| gd96b | 1416 | 1416 | 1429.8 | 278.87 | 1416 | 1417.8 | 2.70 | 0 |
| gd96c | 519 | 519 | 519.0 | 2.94 | 519 | 519.1 | 2.80 | 0 |
| gd96d | 2391 | 2391 | 2425.0 | 3070.59 | 2391 | 2394.2 | 5.70 | 0 |
| Average | 1254773.43 | 1275180.57 | | | 1255049.86 | | | |
| Wilcoxon | | 4.38e-4 | | | | | | |

Table 5.3 – The experimental results between the *SBNH* and the state-of-the-art *TSSA* over the 21 instances.

## 5.5 Investigations of the key components

There are two important parts in the proposed algorithm: the median based neighborhood and the decomposition method in the set based neighborhood. In this section, we investigate the influence of these key components on the performances of the algorithm. For this reason, we create two variants of $SBNH$: $SBNH_1$ by disabling the median based neighborhood and $SBNH_2$ by disabling the decomposition method. We ran both variants to solve the 21 benchmark instances according to the condition specified in Section 5.4.1 and reported their computational results in Tables 5.4 and 5.5 together with those produced by $SBNH$.

In these tables, the information of the compared algorithms is shown employing the same column headings as those used in Table 5.2. The last row gives the statistical significance by using the Wilcoxon signed rank test.

### 5.5.1 Influence of the median based neighborhood

The median based neighborhood $N_s$ is employed in our $SBNH$ in 5.3.3 because of its simple structure and effectiveness. As the size of $N_s$ is small, we could use it to reach the local optima rapidly. Moreover, the median based neighborhood $N_s$ includes the configurations by swapping the labels of adjacent vertices. On the contrary, the set based neighborhood $N_{SB}$ is generated by the maximal independent set where the vertices are non-adjacent. This means that the label swapping is never performed between the adjacent vertices if we use only the set based neighborhood in the descent phase, which possibly leads to bad performance even for some simple instances. For this reason, the employment of the $N_s$ in the descent phase could help break the ties of the non-adjacent vertices.

Table 5.4 gives the experimental results of $SBNH$ and the first variant $SBNH_1$. By observing the column $\Delta$, we could find that $SBNH$ obtains better results than $SBNH_1$ in most instances (13 of 21) and it cannot compete with $SBNH_1$ in 5 cases. Considering the overall performance (See the last row), $SBNH$ has a smaller average best objective values (1275180.57 VS 1276873.62) than $SBNH_1$. The wilcoxon singed-test shows that there is no statistical difference ($p$-value=0.112>0.05) between the two groups of data. It is worth noting that, $SBNH_1$ takes longer time for exploring some small instances ("gd95c", "gd96b" and "gd96c") to obtain the equal or worse results in $SBNH$. This proves that the existence of median based neighborhood $N_s$ could break the ties between

| Graph | EveBest | $SBNH$ | | | $SBNH_1$ | | | Δ |
|---|---|---|---|---|---|---|---|---|
| | | Best | Average | Time_avg | Best | Average | Time_avg | |
| randomA1 | 866968 | 870256 | 876713.2 | 3564.77 | 873461 | 879244.2 | 3311.39 | 3205 |
| randomA2 | 6522206 | 6534039 | 6557104.5 | 3402.40 | 6530284 | 6549459.1 | 3037.99 | -3755 |
| randomA3 | 14194583 | 14229092 | 14251947.4 | 2833.83 | 14218935 | 14244294 | 3352.21 | -10157 |
| randomA4 | 1717176 | 1727320 | 1737251.6 | 3566.52 | 1730676 | 1735348.4 | 3405.07 | 3356 |
| randomG4 | 140211 | 153391 | 175262.7 | 3271.34 | 155781 | 176752.6 | 3575.49 | 2390 |
| bintree10 | 3696 | 4549 | 4839.0 | 3481.58 | 4359 | 4638.7 | 3549.90 | -190 |
| hc10 | 523776 | 523776 | 523776.0 | 0.00 | 523776 | 523776.0 | 0.00 | 0 |
| mesh33x33 | 31729 | 32652 | 33175.6 | 554.82 | 32505 | 33460.9 | 3473.58 | -147 |
| 3elt | 357329 | 439848 | 539694.5 | 3191.96 | 456670 | 574510.5 | 3596.32 | 16822 |
| airfoil1 | 272931 | 352140 | 414330.3 | 3581.20 | 367439 | 451462.8 | 3593.64 | 15299 |
| whitaker3 | 1143645 | 1313695 | 1804234.3 | 3441.36 | 1316291 | 1808216.7 | 17.48 | 2596 |
| c1y | 62230 | 62881 | 64333.2 | 3189.73 | 63647 | 65119.3 | 3537.59 | 766 |
| c2y | 78757 | 80030 | 81630.9 | 1001.85 | 80091 | 82074.7 | 3102.06 | 61 |
| c3y | 123145 | 132057 | 137119.8 | 2835.83 | 133481 | 138543.9 | 3385.57 | 1424 |
| c4y | 114936 | 118271 | 122969.7 | 3293.06 | 118271 | 124149.5 | 2229.08 | 0 |
| c5y | 96850 | 98528 | 104605.8 | 3460.93 | 103475 | 108408.6 | 3579.65 | 4947 |
| gd95c | 506 | 506 | 506.0 | 2.83 | 516 | 516.6 | 406.88 | 10 |
| gd96a | 95242 | 101435 | 104079.8 | 3174.46 | 100339 | 102359.9 | 3531.20 | -1096 |
| gd96b | 1416 | 1416 | 1429.8 | 278.87 | 1424 | 1428.6 | 2739.46 | 8 |
| gd96c | 519 | 519 | 519.0 | 2.94 | 519 | 519.7 | 1865.27 | 0 |
| gd96d | 2391 | 2391 | 2425.0 | 3070.59 | 2406 | 2410.5 | 2667.44 | 15 |
| Average | 1254773.43 | 1275180.57 | | | 1276873.62 | | | |
| Wilcoxon | | 0.112 | | | | | | |

Table 5.4 – The experimental results between the $SBNH$ and the variant algorithm $SBNH_1$ by disabling the median based neighborhood over the 21 instances where each instance is executed independently 10 times.

the non-adjacent vertices to help explore the search space more efficiently.

As a matter of fact, the $N_s$ is not the most suitable supplementary neighborhood because its existence makes some instances get worse results. It encourages us to investigate other meaningful and powerful k-filp neighborhood in the future.

### 5.5.2 Influence of the decomposition method

As we presented in Section 5.3.3, the set based neighborhood $N_{SB}$ is introduced in our $SBNH$ because of its advantage: identify the best solution of the $n_0!$ solutions in a polynomial time $O(n_0^4)$ [3] by transforming the neighborhood-searching problem into a minimum-weight perfect matching problem. In order to reduce the computational complexity, we introduce a decomposition method to split the maximum independent set into several parts. To reveal the influence of decomposition method, we create $SBNH_2$ by disabling the decomposition part.

From the column "$\Delta$" in Table 5.5, one observes that $SBNH_2$ is not able to compare favorably with $SBNH$ in almost all instances (17 of 21). Especially, $SBNH_2$ costs more time to get worse results than $SBNH$ in 16 instances. This shows that the decomposition method is significant in increasing the search speed and in improving the solution quality. The Wilcoxon signed rank test with the two groups of best objective values gets a $p$-value=2.93e-4 $< 0.05$, confirming the statistical significance between the compared results.

## 5.6 Conclusions and perspectives

In this chapter, we have proposed a set based neighborhood heuristic algorithm for solving the NP-hard Minimum Linear Arrangement Problem. The proposed algorithm explores the search space in two phases: a descent phase integrating a median based neighborhood and a set based neighborhood with a decomposition method as well as a perturbation phase to help the search escape from the local optimum. The experimental results indicate that the set based neighborhood is more efficient than traditional 2-filp swapping neighborhood in a two phase iterated heuristic framework. We have also investigated the influence of the key components to the algorithm performance. The result shows that the median based neighborhood is important to break the ties of non-adjacent vertices for offering the different neighborhood. Also, the decomposition method is neces-

3. $n_0$ is the size of maximum independent set $N_0$

| Graph | EveBest | $SBNH$ | | | $SBNH_2$ | | | $\Delta$ |
|---|---|---|---|---|---|---|---|---|
| | | Best | Average | Time_avg | Best | Average | Time_avg | |
| randomA1 | 866968 | 870256 | 876713.2 | 3564.77 | 917200 | 928703.0 | 3601.46 | 46944 |
| randomA2 | 6522206 | 6534039 | 6557104.5 | 3402.40 | 6572228 | 6603694.2 | 3599.85 | 38189 |
| randomA3 | 14194583 | 14229092 | 14251947.4 | 2833.83 | 14246002 | 14263037.0 | 3150.59 | 16910 |
| randomA4 | 1717176 | 1727320 | 1737251.6 | 3566.52 | 1775997 | 1791516.0 | 3594.77 | 48677 |
| randomG4 | 140211 | 153391 | 175262.7 | 3271.34 | 161301 | 186372.1 | 3580.75 | 7910 |
| bintree10 | 3696 | 4549 | 4839.0 | 3481.58 | 18180 | 20007.8 | 3454.21 | 13631 |
| hc10 | 523776 | 523776 | 523776.0 | 0.00 | 523776 | 523776.0 | 0.00 | 0 |
| mesh33x33 | 31729 | 32652 | 33175.6 | 554.82 | 34796 | 38617.0 | 3540.56 | 2144 |
| 3elt | 357329 | 439848 | 539694.5 | 3191.96 | 478295 | 595772.6 | 3639.66 | 38447 |
| airfoil1 | 272931 | 352140 | 414330.3 | 3581.20 | 386962 | 480843.1 | 3652.37 | 34822 |
| whitaker3 | 1143645 | 1313695 | 1804234.3 | 3441.36 | 1314658 | 1806279.4 | 25.00 | 963 |
| c1y | 62230 | 62881 | 64333.2 | 3189.73 | 67288 | 67858.0 | 3530.72 | 4407 |
| c2y | 78757 | 80030 | 81630.9 | 1001.85 | 85891 | 89265.8 | 3573.86 | 5861 |
| c3y | 123145 | 132057 | 137119.8 | 2835.83 | 141733 | 148150.3 | 3600.45 | 9676 |
| c4y | 114936 | 118271 | 122969.7 | 3293.06 | 127165 | 135225.4 | 3599.35 | 8894 |
| c5y | 96850 | 98528 | 104605.8 | 3460.93 | 109496 | 114252.6 | 3481.73 | 10968 |
| gd95c | 506 | 506 | 506.0 | 2.83 | 506 | 506.0 | 18.18 | 0 |
| gd96a | 95242 | 101435 | 104079.8 | 3174.46 | 116333 | 123452.4 | 3339.32 | 14898 |
| gd96b | 1416 | 1416 | 1429.8 | 278.87 | 1416 | 1431.2 | 102.60 | 0 |
| gd96c | 519 | 519 | 519.0 | 2.94 | 519 | 519.0 | 4.92 | 0 |
| gd96d | 2391 | 2391 | 2425.0 | 3070.59 | 2413 | 2501.2 | 2785.77 | 22 |
| Average | 1254773.43 | 1275180.57 | | | 1289626.429 | | | |
| Wilcoxon | | 2.93e-4 | | | | | | |

Table 5.5 – The experimental results between the $SBNH$ and the variant algorithm $SBNH_2$ by disabling the decomposition method over the 21 instances where each instance is executed independently 10 times.

sary to accelerate the search process and reduce the computational complexity. This study also indicates that the descent local search combining a simple perturbation method is not enough to obtain the better results than state-of-the-art algorithms. The more effort should be put into finding a more powerful diversification method. Moreover, it is possible to develop some parallel computing algorithms for the decomposition method in the future.

Part III

# Conclusion

# CONCLUSIONS

This thesis concerns two NP-hard graph layout problems: the cyclic bandwidth problem (CBP) and the Minimum Linear Arrangement Problem (MinLA). There exist numerous applications in the real world like very-large-scale-integration layout(VLSI), data structure representations, code design, graph drawing, software diagram layout and job scheduling and interconnection networks for parallel computer systems. Much effort has been put into theoretical and practical studies in the literature. In this thesis, we present several heuristic algorithms to solve these problems.

In Chapter 2, after studying the solution distribution of CBP, we find that the existence of many plateaus in the search space leads to the difficulty in identifying better solutions and escaping from the local optimum. For this reason, we introduced an enriched evaluation function to discriminate the solutions with same objective value and proposed an iterated three-phase search algorithm ($ITPS$) integrating a double neighbor-descent phase, a threshold-based search phase and a special perturbation phase to ensure a highly effective examination of the search space. To assess the performance of the proposed $ITPS$, we have carried out intensive experiments over 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. Compared with the results of the best reference algorithm in the literature, the proposed algorithm shows a very competitive performance. Concerning the 85 standard graphs, our $ITPS$ could improve the best known results of 19 instances and match the best known results of 60 instances. As to the 28 *Harwell-Boeing* graphs, 12 new records are discovered and 15 best known results are matched by the proposed algorithm. Moreover, additional experiments reveal that the key composing ingredients of the algorithm including: the extended evaluation function, the threshold-based search and the shift-insert-based perturbation strategy play an important role in algorithm performance.

In Chapter 3, we presented another effective heuristic algorithm $NILS$ based on the general iterated local search framework. The proposed algorithm employs a simple but powerful local optimization procedure to discover high-quality solutions in a particular search area. $NILS$ also integrates two complementary perturbation strategies: a directed

perturbation and a strong perturbation to help escape from local optimum and explore unvisited areas. Especially, the strong perturbation is originally composed of a destruction step and a reconstruction step to bring the search to a distant and new region. Intensive experiments on the same 113 benchmark instances are carried out and $NILS$ shows the performances that are never achieved by current best algorithms in the literature. 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 established 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. We have also investigated the influence of the directed perturbation and strong perturbation to the performance of the proposed algorithm. The experimental results show that the existence of them gives a positive influence to $NILS$.

In Chapter 4, we studied the memetic framework for solving CBP. Five different permutation crossovers (CX, OX, OX2, PMX and DPX) are compared under the framework of a memetic algorithm (MA) using a simple descent procedure to commit the local optimization. The experimental results show that the variant of MA with recombination operator OX2 achieved the best performance for the tested instances in terms of the best objective value found and the average objective value. Then we conducted an investigation on the population diversity measured by the average distance and entropy of the MA variants using different recombination operators. In addition, we have explored the correlation between the population diversity and the performance of the studied MA variants. The analysis shows that a higher entropy and average distance of the population generally helps to find solutions of good quality. This study also indicates that the basic memetic algorithm combining an existing recombination operator and a simple descent local search procedure is not competitive compared to the state-of-the-art CBP algorithms. Meanwhile, given the excellent performances achieved by MAs on many difficult optimization problems, this work invites more research effort on seeking meaningful recombination operators suitable for CBP. It is then expected that a MA integrating such a recombination operator and a powerful local optimization procedure would achieve state-of-the-art performances.

In Chapter 5, we proposed a set-based neighborhood heuristic algorithm ($SBNH$) for solving MinLA. The proposed algorithm is realized in two phases: a descent phase

as a local optimization procedure integrating a median based neighborhood and a set based neighborhood using a decomposition method as well as a perturbation phase to help escape from the local optimum. The highlight of this work is that we introduced a set based neighborhood where we can identify the best solution in exponentially large number of solutions within a polynomial time by transforming finding the best solution in the neighborhood into a minimum weight perfect matching problem. We also employed a decomposition method to accelerate the calculating by splitting the relative maximal independent set into multiple subsets. To verify the effectiveness of the set based neighborhood, we created a $SBNH$ variant ($2FNH$) which replaces the set based neighborhood with a traditional 2-flip neighborhood. Intensive experiments on the 21 benchmark instances were carried out and the computational results indicate that the set based neighborhood performs more efficient than traditional 2-filp neighborhood under the framework of the iterated local search. We also investigated the influence of the key components to the algorithm performance. The result shows that the median based neighborhood is important to break the ties of non-adjacent vertices for offering different neighborhoods. Also, the decomposition method is necessary to accelerate the search process and reduce the computational complexity. This study also indicates that the descent local search combining a simple perturbation method is not enough to obtain the better results than state-of-the-art algorithms. More efforts should be put into finding a more powerful diversification method.

# Perspectives

In this thesis, we mainly focus on developing effective heuristic algorithms and explore meaningful components of the algorithms for the graph layout problems. For the future, the study can be extended in the following directions.

Firstly, it is worth studying other forms of extended evaluation functions. After thorough research on CBP, we find that the extended evaluation function can offer a positive influence to the performance by guiding the search during the local optimization. However, the extended evaluation function is normally designed for the specified problem and it is not able to be adjusted with the change of the search space. For this reason, we are willing to advance the study in introducing machine learning and neural network technology to create an auto-design and adaptive evaluation function for the combinatorial optimization problem under consideration.

Secondly, we would like to develop hybrid algorithms combining heuristic algorithms and some exact methods. Heuristic algorithms could find sub-optimal solutions in a reasonable time while exact algorithms can ensure a global optimal solution of an instance. It seems that heuristic algorithms and exact methods are two non-crossing lines to solve the problem independently. Indeed, there exist some exact methods like branch and bound method [RRR12] to solve CBP but only for small instances. We have already developed powerful heuristic algorithms for CBP. Hence, it is possible to develop a hybrid algorithm for solving CBP more efficiently.

Finally, we will extend the set based neighborhood method to other graph layout problems like Cyclic Bandwidth Sum Problem (CBSP) [JHa01]. With the use of the set based neighborhood, it is unnecessary to enumerate configurations in the neighborhood and we can identify the best configuration in a single step by means of a polynomial-time algorithm, even though the number of the configurations in the neighborhood is exponentially large. Also, we will try to combine the decomposition method with parallel computing technology to improve the computational efficiency.

# List of Figures

# List of Tables

Part IV

# Appendix

Chapter 6

# Appendix

## 6.1 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 [Rod+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 6.1, whereas the results for the group of 28 graphs from real-world applications with unknown optima are listed in Table 6.2. 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 ($Cb_b$), average (Avg. $Cb$) and standard deviation ($Dev.$) of the cyclic bandwidth cost 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 ($Cb_b$) 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 2.3.1 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 $\star$ indicate that no significant difference exists between the analyzed algorithms.

## 6.2 Detailed performance evaluation of $NILS$ with respect to the CBP state-of-the-art algorithms

In this appendix, we present the results of a detailed performance evaluation which considers $NILS$ and two state-of-the-art algorithms: $TScb$ [Rod+15] and $ITPS$ [RHR19].

Tables 6.3 and 6.4 summarize these results on an instance-by-instance basis. The former is composed of a set of 85 standard graphs with known optimal solutions while the latter consists of a set of 28 graphs arising from real-world engineering applications. These tables list in column 1 to 3 the instance name, its order ($|V|$) and size ($|E|$). The optimal solution cost ($Cb^*$), reported from the literature [Chu88; Hro+92; Lin94; Smi95], is registered in Table 6.3; while the theoretical lower ($L_B$) and upper ($U_B$) bounds for the instances listed in Table 6.4 (see columns 4 and 5) were computed according to the formulas $L_B = \lceil \Delta(G)/2 \rceil$ and $U_B = \lfloor |V|/2 \rfloor$, where $\Delta(G)$ denotes the maximum degree of the graph $G$ [Lin97]. The rest of the columns in these tables are dedicated to present, for each algorithm considered in this comparison, the best ($Cb_b$), average (Avg. $Cb$) and standard deviation ($Dev.$) of the cyclic bandwidth cost attained in 50 independent executions, the computational time used up to produce 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 $NILS$ first against $TScb$ [Rod+15], and then versus $ITPS$ [RHR19] was executed. The corresponding 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 $+$ is used to identify the cases where $NILS$ offers a significant better performance than the state-of-the-art algorithms. On the contrary, if $NILS$ is defeated by them the cell is marked with the symbol $-$, while a $\star$ is used to record those cases where it was not possible to conclude a statistical significant difference between the compared solution methods.

Table 6.1 – 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* | $Cb_b$ | Avg. Cb | $Dev.$ | Avg. $T_{best}$ | $D$ | $Cb_b$ | 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 | + |

*continued on the next page ...*

Table 6.1 – – continued from previous page

| | | | | *TScb* | | | | | *ITPS* | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Graph | $\|V\|$ | $\|E\|$ | Cb* | $Cb_b$ | Avg. Cb | *Dev.* | Avg. $T_{best}$ | *D* | $Cb_b$ | 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 |

Table 6.2 – 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* | $Cb_b$ | Avg. $Cb$ | *Dev.* | Avg. $T_{best}$ | $D$ | $Cb_b$ | 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 |

Table 6.3 – Detailed performance assessment of the $NILS$ algorithm with respect to two state-of-the-art methods: $TScb$ [Rod+15] and $ITPS$ [RHR19]. 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).

| | | | | $TScb$ | | | | | | $ITPS$ | | | | | $NILS$ | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 | ⋆ | 1.00E+00 | ⋆ |
| 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 | ⋆ | 1.00E+00 | ⋆ |
| 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 | ⋆ | 1.00E+00 | ⋆ |
| 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 | ⋆ | 1.00E+00 | ⋆ |
| 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 | ⋆ | 1.00E+00 | ⋆ |
| 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 | ⋆ | 1.00E+00 | ⋆ |
| 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 | ⋆ | 1.00E+00 | ⋆ |
| 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 | ⋆ |
| 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 | ⋆ |
| 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 | ⋆ |
| 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 | ⋆ | 1.00E+00 | ⋆ |
| 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 | ⋆ | 1.00E+00 | ⋆ |
| 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 | ⋆ | 1.00E+00 | ⋆ |
| 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 | ⋆ | 1.00E+00 | ⋆ |
| 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 | ⋆ | 1.00E+00 | ⋆ |
| 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 | ⋆ | 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 | ⋆ | 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 | ⋆ | 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 | ⋆ | 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 | ⋆ | 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 | + |

Continued on next page ...

Table 6.3 – Continued from previous page

| | | | | *TScb* | | | | | *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 | $SS_2$ |
| 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 | + |
| 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 | + |

Continued on next page ...

Table 6.3 – Continued from previous page

| | | | | *TScb* | | | | | *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 | $SS_2$ |
| 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 | ⋆ |
| 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 | ⋆ | 1.00E+00 | ⋆ |
| 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 | ⋆ |
| 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 | ⋆ | 1.00E+00 | ⋆ |
| 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 | ⋆ | 1.00E+00 | ⋆ |
| 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 | ⋆ | 1.00E+00 | ⋆ |
| 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 | ⋆ |
| 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 | ⋆ | 1.00E+00 | ⋆ |
| 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 | ⋆ | 1.00E+00 | ⋆ |
| 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 | ⋆ | 1.00E+00 | ⋆ |
| 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 | ⋆ | 1.00E+00 | ⋆ |
| 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 | ⋆ | 1.00E+00 | ⋆ |
| 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 | ⋆ | 1.00E+00 | ⋆ |
| 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 | ⋆ | 1.00E+00 | ⋆ |
| 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 | ⋆ | 1.00E+00 | ⋆ |
| 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 | ⋆ | 1.00E+00 | ⋆ |
| 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 | ⋆ |
| 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 | ⋆ |
| 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 | ⋆ |
| 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 | + |

Continued on next page ...

Table 6.3 – Continued from previous page

| | | | | *TScb* | | | | | *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 | $SS_2$ |
| 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 6.4 – Detailed performance assessment of the $NILS$ algorithm with respect to two state-of-the-art methods: $TScb$ [Rod+15] and $ITPS$ [RHR19]. 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.

| | | | Bounds | | | $TScb$ | | | | | | $ITPS$ | | | | | | $NILS$ | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 | | | | |

# List of Publications

## Published/accepted papers

— Jintong Ren, Jin-Kao Hao, Eduardo Rodriguez-Tello. An iterated three-phase search approach for solving the Cyclic Bandwidth Problem. IEEE Access 7: 98436-98452, 2019.

— Jintong Ren, Jin-Kao Hao, Eduardo Rodriguez-Tello. A study of recombination operators for the Cyclic Bandwidth Problem. In Idoumghar et al. (Eds.) Revised selected papers, 14th International Conference on Artificial Evolution (EA 2019), France, October 29-30, 2019, Lecture Notes in Computer Science 12052: 177-191, 2020.

## Submitted papers

— Jintong Ren, Jin-Kao Hao, Eduardo Rodriguez-Tello, Liwen Li, Kun He. A new iterated local search algorithm for the cyclic bandwidth problem. Knowledge Based Systems. Submitted Feb 2020, revised in April 2020.

# Reference


[AH73] D. Adolphson and T.C. Hu, « Optimal linear ordering », *in*: *SIAM Journal on Applied Mathematics* 25.*3* (1973), pp. 403–423 (cit. on p. 17).

[Ama+08] A.R.S. Amaral, A. Caprara, A.N. Letchford, and J.J. Salazar-Gonzalez, « A New Lower Bound for the Minimum Linear Arrangement of a Graph », *in*: *Electronic Notes in Discrete Mathematics* 30 (Feb. 2008), pp. 87–92 (cit. on p. 91).

[AMS11] C. Ambühl, M. Mastrolilli, and O. Svensson, « Inapproximability Results for Maximum Edge Biclique, Minimum Linear Arrangement, and Sparsest Cut », *in*: *SIAM Journal on Computing* 40.*2* (Jan. 2011), pp. 567–596 (cit. on p. 91).

[Ans+15] C. Ansótegui, Y. Malitsky, H. Samulowitz, M. Sellmann, and K. Tierney, « Model-based genetic algorithms for algorithm configuration », *in*: *Proceedings of the 24th International Conference on Artificial Intelligence*, AAAI Press, 2015, pp. 733–739 (cit. on p. 41).

[BS11] R. Bansal and K. Srivastava, « A memetic algorithm for the cyclic antibandwidth maximization problem », *in*: *Soft Computing* 15.*2* (2011), pp. 397–412 (cit. on p. 73).

[Bar+04] R. Bar-Yehuda, G. Even, J. Feldman, and J. Naor, « Computing an Optimal Orientation of a Balanced Decomposition Tree for Linear Arrangement Problems », *in*: *Graph Algorithms and Applications 2*, WORLD SCIENTIFIC, May 2004, pp. 387–413 (cit. on p. 91).

[BH15] U. Benlic and J.K. Hao, « Memetic search for the quadratic assignment problem », *in*: *Expert Systems with Applications* 42.*1* (2015), pp. 584–595 (cit. on p. 73).

[BT84] S.N. Bhatt and L.F. Thomson, « A framework for solving VLSI graph layout problems », *in*: *Journal of Computer and System Sciences* 28.*2* (1984), pp. 300–343 (cit. on pp. 9, 17, 24).

[Boe95] K.D. Boese, *Cost versus distance in the traveling salesman problem*, UCLA Computer Science Department Los Angeles, 1995, pp. 109–133 (cit. on p. 86).

[CLS11] A. Caprara, A.N. Letchford, and J.J. Salazar-González, « Decorous lower bounds for minimum linear arrangement », *in*: *INFORMS Journal on Computing* 23.*1* (Dec. 2011), pp. 26–40 (cit. on pp. 91, 101).

[CLS08] W.H. Chan, P.C.B. Lam, and W.C. Shiu, « Cyclic bandwidth with an edge added », *in*: *Discrete Applied Mathematics* 156.*1* (2008), pp. 131–137 (cit. on pp. 18, 26).

[CH15] Y. Chen and J.K. Hao, « Iterated responsive threshold search for the quadratic multiple knapsack problem », *in*: *Annals of Operations Research* 226.*1* (Mar. 2015), pp. 101–131 (cit. on p. 37).

[CH16] Y. Chen and J.K. Hao, « Memetic search for the generalized quadratic multiple knapsack problem », *in*: *IEEE Transactions on Evolutionary Computation* 20.*6* (2016), pp. 908–923 (cit. on p. 73).

[Chu88] F.R.K. Chung, « Labelings of graphs », *in*: *Selected topics in graph theory volume 3*, ed. by L.W. Beineke and R.J. Wilson, London, England: Academic Press, 1988, chap. 7, pp. 151–168 (cit. on pp. 9, 24, 124).

[CR99] W. Cook and A. Rohe, « Computing minimum-weight perfect matchings », *in*: *INFORMS journal on computing* 11.*2* (1999), pp. 138–148 (cit. on p. 97).

[Dav85] L. Davis, « Applying adaptive algorithms to epistatic domains. », *in*: *IJCAI*, vol. 85, 1985, pp. 162–164 (cit. on p. 79).

[DPS02] J. Dıéaz, J. Petit, and M. Serna, « A Survey of Graph Layout Problems », *in*: *ACM Comput. Surv.* 34.*3* (Sept. 2002), pp. 313–356 (cit. on pp. 9, 18, 89, 90).

[Dua+11] A. Duarte, R. Martí, M.G.C. Resende, and R.M.A. Silva, « GRASP with path relinking heuristics for the antibandwidth problem », *in*: *Networks* 58.*3* (Oct. 2011), pp. 171–189 (cit. on p. 40).

[Due93] G. Dueck, « New optimization heuristics: The Great Deluge Algorithm and the Record-to-Record Travel », *in*: *Journal of Computational Physics* 104.*1* (Jan. 1993), pp. 86–92 (cit. on p. 35).

[DS90] G. Dueck and T. Scheuer, « Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing », *in*: *Journal of Computational Physics* 90.*1* (Sept. 1990), pp. 161–175 (cit. on p. 35).

[FF96] J. Ferland and C. Fleurent, « Object-oriented implementation of heuristic search methods for graph coloring, maximum clique, and satisfiability », *in*: *DIMACS Ser. Discrete Math.* 26 (1996), pp. 619–652 (cit. on p. 84).

[FM96] B. Freisleben and P. Merz, « A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems », *in*: *Proceedings of IEEE international conference on evolutionary computation*, IEEE, 1996, pp. 616–621 (cit. on pp. 73, 79, 81, 84).

[FH15] Z.H. Fu and J.K. Hao, « A three-phase search approach for the quadratic minimum spanning tree problem », *in*: *Engineering Applications of Artificial Intelligence* 46 (2015), pp. 113–130 (cit. on p. 28).

[FH17] Z.H. Fu and J.K. Hao, « Knowledge-guided local search for the prize-collecting Steiner tree problem in graphs », *in*: *Knowledge-Based Systems* 128 (2017), pp. 78–92 (cit. on p. 54).

[GBF11] P. Galinier, Z. Boujbel, and M.C. Fernandes, « An efficient memetic algorithm for the graph partitioning problem », *in*: *Annals of Operations Research* 191.*1* (2011), pp. 1–22 (cit. on p. 34).

[Ger95] A.M.H. Gerards, « Chapter 3 Matching », *in*: *Network Models*, vol. 7, Handbooks in Operations Research and Management Science, Elsevier, 1995, pp. 135–224 (cit. on pp. 98, 99).

[GL97] F. Glover and M. Laguna, *Tabu Search*, Kluwer Academic Publishers, 1997 (cit. on pp. 34, 56).

[Glo+11] F. Glover, R. Martıé, J.J. Pantrigo, A. Duarte, and V. Campos, « Scatter Search and Path Relinking: A Tutorial on the Linear Arrangement Problem », *in*: *Int. J. Swarm. Intell. Res.* 2.*2* (Apr. 2011), pp. 1–21 (cit. on p. 93).

[GR17] F. Glover and C. Rego, « New relationships for multi-neighborhood search for the minimum linear arrangement problem », *in*: *Journal of Discrete Algorithms* 46-47 (Sept. 2017), pp. 16–24 (cit. on pp. 90, 97).

[GL+85] D.E. Goldberg, R. Lingle, et al., « Alleles, loci, and the traveling salesman problem », *in*: *Proceedings of an international conference on genetic algorithms and their applications*, vol. 154, Lawrence Erlbaum, Hillsdale, NJ, 1985, pp. 154–159 (cit. on pp. 79, 80).

[GJL09] A. Grosso, A. Jamali, and M. Locatelli, « Finding maximin latin hypercube designs by iterated local search heuristics », *in*: *European Journal of Operational Research* 197.*2* (2009), pp. 541–547 (cit. on p. 54).

[Hao12] J.K. Hao, « Memetic algorithms in discrete optimization », *in*: *Handbook of memetic algorithms*, Springer, 2012, pp. 73–94 (cit. on p. 76).

[Har64] L.H. Harper, « Optimal Assignment of Numbers to Vertices », *in*: *Journal of SIAM* 12.*1* (1964), pp. 131–135 (cit. on pp. 9, 16, 17, 26, 89).

[Har82] J. Hartmanis, « Computers and intractability: a guide to the theory of NP-completeness (michael r. garey and david s. johnson) », *in*: *Siam Review* 24.*1* (1982), p. 90 (cit. on pp. 9, 17, 89).

[Hro+92] J. Hromkovič, V. Müller, O. Sýkora, and I. Vrťo, « On embedding interconnection networks into rings of processors », *in*: *Lecture Notes in Computer Science* 605 (1992), pp. 51–62 (cit. on pp. 17, 24, 25, 124).

[Hut+09] F. Hutter, H.H. Hoos, K. Leyton-Brown, and T. Stützle, « ParamILS: an automatic algorithm configuration framework », *in*: *Journal of Artificial Intelligence Research* 36 (2009), pp. 267–306 (cit. on p. 41).

[JHa01] J.Hao, « Cyclic bandwidth sum of graphs », *in*: *Applied Mathematics-A Journal of Chinese Universities* 16.*2* (2001), pp. 115–121 (cit. on p. 116).

[JHH14] Y. Jin, J.K. Hao, and J.P. Hamiez, « A memetic algorithm for the minimum sum coloring problem », *in*: *Computers & Operations Research* 43 (2014), pp. 318–327 (cit. on p. 73).

[JM92] M. Juvan and B. Mohar, « Optimal linear labelings and eigenvalues of graphs », *in*: *Discrete Applied Mathematics* 36.*2* (Apr. 1992), pp. 153–168 (cit. on pp. 18, 90, 91).

[KNS11] E. De Klerk, M.E. Nagy, and R. Sotirov, « On semidefinite programming bounds for graph bandwidth », *in*: *Optimization Methods and Software* 28 (2011), pp. 485–500 (cit. on pp. 18, 26).

[KH02] Y. Koren and D. Harel, « A Multi-scale Algorithm for the Linear Arrangement Problem », *in*: Springer, Berlin, Heidelberg, 2002, pp. 296–309 (cit. on pp. 18, 90, 91).

[KS05] N. Krasnogor and J. Smith, « A tutorial for competent memetic algorithms: model, taxonomy, and design issues », *in*: *IEEE Transactions on Evolutionary Computation* 9.*5* (2005), pp. 474–488 (cit. on p. 73).

[LH16] X. Lai and J.K. Hao, « A tabu search based memetic algorithm for the max-mean dispersion problem », *in*: *Computers & Operations Research* 72 (2016), pp. 118–127 (cit. on pp. 34, 73).

[LW99] Y.L. Lai and K. Williams, « A survey of solved problems and applications on bandwidth, edgesum, and profile of graphs », *in*: *Journal of graph theory* 31.*2* (1999), pp. 75–94 (cit. on p. 89).

[LSC97] P.C.B. Lam, W.C. Shiu, and W.H. Chan, « On bandwidth and cyclic bandwidth of graphs », *in*: *Ars Combinatoria* 47.*3* (1997), pp. 147–152 (cit. on pp. 18, 25).

[LSC02] P.C.B. Lam, W.C. Shiu, and W.H. Chan, « Characterization of graphs with equal bandwidth and cyclic bandwidth », *in*: *Discrete Mathematics* 242.*1* (2002), pp. 283–289 (cit. on pp. 18, 25).

[Law63] E.L. Lawler, « The Quadratic Assignment Problem », *in*: *Manage. Sci.* 9.*4* (1963), pp. 586–599 (cit. on p. 98).

[LVW84] J. Leung, O. Vornberger, and J. Witthoff, « On some variants of the bandwidth minimization problem », *in*: *SIAM Journal on Computing* 13.*3* (1984), pp. 650–667 (cit. on pp. 9, 17, 24, 40).

[Lin94] Y. Lin, « The cyclic bandwith problem », *in*: *Journal of Systems Science and Complexity* 7.*3* (1994), pp. 282–288 (cit. on pp. 9, 17, 24, 124).

[Lin97] Y. Lin, « Minimum bandwidth problem for embedding graphs in cycles », *in*: *Networks* 29.*3* (May 1997), pp. 135–140 (cit. on pp. 18, 19, 25, 45, 60, 62, 124).

[Lop+16] M. Lopez-Ibañez, J. Dubois-Lacoste, L.P. Caceres, T. Stutzle, and M. Birattari, « The irace package: Iterated Racing for Automatic Algorithm Configuration », *in*: *Operations Research Perspectives* 3 (2016), pp. 43–58 (cit. on pp. 41, 63, 64).

[LMS03] H.R. Lourenço, O.C. Martin, and T. Stützle, « Iterated local search », *in*: *Handbook of metaheuristics*, Springer, 2003, pp. 320–353 (cit. on p. 54).

[Loz+12] M. Lozano, A. Duarte, F. Gortázar, and R. Martí, « Variable neighborhood search with ejection chains for the antibandwidth problem », *in*: *Journal of Heuristics* 18.*6* (Dec. 2012), pp. 919–938 (cit. on p. 40).

[Mar+11] M.E. Marmion, C. Dhaenens, L. Jourdan, A. Liefooghe, and S. Verel, « On the Neutrality of Flowshop Scheduling Fitness Landscapes », *in*: *Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers*, ed. by C. A. Coello Coello, vol. 6683, Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 238–252 (cit. on p. 46).

[Mca99] A.J. Mcallister, *A new heuristic algorithm for the linear arrangement problem*, tech. rep. TR-99-126a, Faculty of Computer Science, University of New Brunswick, 1999 (cit. on pp. 18, 90–94).

[MK19] D. Meignan and S. Knust, « A neutrality-based iterated local search for shift scheduling optimization and interactive reoptimization », *in*: *European Journal of Operational Research* 279.*2* (2019), pp. 320–334 (cit. on p. 54).

[MF01] P. Merz and B. Freisleben, « Memetic algorithms for the traveling salesman problem », *in*: *Complex Systems* 13.*4* (2001), pp. 297–346 (cit. on p. 73).

[MAK07] W. Michiels, E. Aarts, and J. Korst, *Theoretical Aspects of Local Search*, 1st ed., Monographs in Theoretical Computer Science. An EATCS Series, Springer, 2007 (cit. on p. 46).

[MRW18] K. Mihic, K. Ryan, and A. Wood, « Randomized decomposition solver with the quadratic assignment problem as a case study », *in*: *INFORMS Journal on Computing* 30.*2* (2018), pp. 295–308 (cit. on p. 98).

[Mil+15] M. Miller, R.S. Rajan, N. Parthiban, and I. Rajasingh, « Minimum Linear Arrangement of Incomplete Hypercubes », *in*: *The Computer Journal* 58.*2* (Feb. 2015), pp. 331–337 (cit. on p. 91).

[MUP16] N. Mladenović, D. Urošević, and D. Pérez-Brito, « Variable neighborhood search for minimum linear arrangement problem », *in*: *Yugoslav Journal of Operations Research* 26.*1* (2016), pp. 3–16 (cit. on pp. 18, 90, 93).

[Mla+10] N. Mladenovic, D. Urosevic, D. Pérez-Brito, and C.G. Garcıéa-González, « Variable neighbourhood search for bandwidth reduction », *in*: *European Journal of Operational Research* 200.*1* (2010), pp. 14–27 (cit. on p. 27).

[MC03] P. Moscato and C. Cotta, « A gentle introduction to memetic algorithms », *in*: *Handbook of metaheuristics*, Springer, 2003, pp. 105–144 (cit. on p. 73).

[OSH] I.M. Oliver, D.J. Smith, and J.R.C. Holland, « A Study of Permutation Crossover Operators on the Traveling Salesman Problem. In John J. Grefenstette », *in*: *Genetic Algorithms and their Applications: Proc. of the Second International Conf*, pp. 224–230 (cit. on pp. 79, 80).

[Pet03a] J. Petit, « Combining spectral sequencing and parallel simulated annealing for the MINLA problem », *in*: *Parallel Processing Letters* 13.*1* (Mar. 2003), pp. 77–91 (cit. on pp. 18, 90, 92).

[Pet03b] J. Petit, « Experiments on the minimum linear arrangement problem », *in*: *Journal of Experimental Algorithmics* 8 (Jan. 2003), p. 3 (cit. on pp. 18, 19, 90, 92, 101).

[Piñ+04] E. Piñana, I. Plana, V. Campos, and R. Martıé, « GRASP and Path Relinking for the Matrix Bandwidth Minimization », *in*: *European Journal of Operational Research* 153 (2004), pp. 200–210 (cit. on p. 27).

[PA12] E. Pitzer and M. Affenzeller, « A Comprehensive Survey on Fitness Landscape Analysis », *in*: *Recent Advances in Intelligent Engineering Systems*, ed. by J. Fodor, R. Klempous, and C. P. Suárez-Araujo, vol. 378, Studies in Computational Intelligence, Springer, 2012, chap. 8, pp. 161–191 (cit. on pp. 46, 47).

[Por05] T. Poranen, « A genetic hillclimbing algorithm for the optimal linear arrangement problem », *in*: *Fundamenta Informaticae* 68.*4* (2005), pp. 333–356 (cit. on pp. 18, 90, 92).

[RHR19] J. Ren, J.K. Hao, and E. Rodriguez-Tello, « An Iterated Three-Phase Search Approach for Solving the Cyclic Bandwidth Problem », *in*: *IEEE Access* 7 (2019), pp. 98436–98452 (cit. on pp. 55, 57, 58, 60, 62–64, 66, 71, 73, 87, 123, 124, 128, 132).

[RB11] E. Rodriguez-Tello and L.C. Betancourt, « An improved memetic algorithm for the antibandwidth problem », *in*: *International Conference on Artificial Evolution (Evolution Artificielle)*, Springer, 2011, pp. 121–132 (cit. on pp. 73, 74).

[RHT06] E. Rodriguez-Tello, J.K. Hao, and J. Torres-Jimenez, « Memetic Algorithms for the MinLA Problem », *in*: *Artificial Evolution*, ed. by El-Ghazali Talbi, Pierre Liardet, Pierre Collet, Evelyne Lutton, and Marc Schoenauer, Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 73–84 (cit. on pp. 18, 90, 92).

[RHT08a] E. Rodriguez-Tello, J.K. Hao, and J. Torres-Jimenez, « An effective two-stage simulated annealing algorithm for the minimum linear arrangement problem », *in*: *Computers & Operations Research* 35.*10* (Oct. 2008), pp. 3331–3346 (cit. on pp. 18, 90, 92).

[RHT08b] E. Rodriguez-Tello, J.K. Hao, and J. Torres-Jimenez, « An Improved Simulated Annealing Algorithm for Bandwidth Minimization », *in*: *European Journal of Operational Research* 185.*3* (2008), pp. 1319–1335 (cit. on pp. 27, 74, 94, 95, 105).

[RNL18] E. Rodriguez-Tello, V. Narvaez-Teran, and F. Lardeux, « Comparative study of different memetic algorithm configurations for the cyclic bandwidth sum problem », *in*: *International Conference on Parallel Problem Solving from Nature*, Springer, 2018, pp. 82–94 (cit. on p. 74).

[Rod+15] E. Rodriguez-Tello, H. Romero-Monsivais, G. Ramirez-Torres, and F. Lardeux, « Tabu search for the cyclic bandwidth problem », *in*: *Computers & Operations Research* 57 (2015), pp. 17–32 (cit. on pp. 18, 19, 26, 27, 32, 34, 35, 39–43, 57, 60, 62–64, 66, 71, 73, 74, 76, 82, 83, 87, 123, 124, 128, 132).

[RRR12] H. Romero-Monsivais, E. Rodriguez-Tello, and G. Ramírez, « A New Branch and Bound Algorithm for the Cyclic Bandwidth Problem », *in*: *Lecture Notes in Artificial Intelligence* 7630 (2012), pp. 139–150 (cit. on pp. 18, 26, 73, 116).

[RS78] A.L. Rosenberg and L. Snyder, « Bounds on the costs of data encodings », *in*: *Theory of Computing Systems* 12.*1* (1978), pp. 9–39 (cit. on pp. 17, 24).

[RH08] H. Rostami and J. Habibi, « Minimum linear arrangement of Chord graphs », *in*: *Applied Mathematics and Computation* 203.*1* (Sept. 2008), pp. 358–367 (cit. on p. 91).

[SRB04] I. Safro, D. Ron, and A. Brandt, « Graph minimum linear arrangement by multilevel weighted edge contractions », *in*: *Journal of Algorithms* 60.*1* (July 2004), pp. 24–41 (cit. on pp. 18, 90, 92).

[Sha+00] F. Shahrokhl, O. Sykora, L.A. Szekely, and I. Vrto, « On bipartite drawings and the linear arrangement problem », *in*: *SIAM Journal on Computing* 30.*6* (2000), pp. 1773–1789 (cit. on pp. 17, 91).

[SS09] R. Sharma and K. Srivastava, « A new hybrid Evolutionary Algorithm for the MinLA problem », *in*: *International Journal of Operational Research* 5.*2* (2009), pp. 229–249 (cit. on pp. 18, 90, 93).

[Shi79] Y. Shiloach, « A minimum linear arrangement algorithm for undirected trees », *in*: *SIAM Journal on Computing* 8.*1* (1979), pp. 15–32 (cit. on p. 91).

[Smi95] L. Smithline, « Bandwidth of the complete k-ary tree », *in*: *Discrete Mathematics* 142.*1* (1995), pp. 203–212 (cit. on p. 124).

[Sta92] P.F. Stadler, « Correlation in Landscapes of Combinatorial Optimization Problems », *in*: *Europhysics Letters* 20.*6* (Nov. 1992), pp. 479–482 (cit. on p. 46).

[Sys91] G. Syswerda, « Scheduling Optimization Using Genetic Algorithms », *in*: *Handbook of Genetic Algorithms* (1991), pp. 322–349 (cit. on p. 79).

[Tor+15] 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 », *in*: *Information Sciences* 303 (2015), pp. 33–49 (cit. on p. 27).

[WLH10] Y. Wang, Z. Lü, and J.K. Hao, « A study of multi-parent crossover operators in a memetic algorithm », *in*: *International Conference on Parallel Problem Solving from Nature*, Springer, 2010, pp. 556–565 (cit. on p. 83).

[WH13] Q. Wu and J.K. Hao, « Memetic search for the max-bisection problem », *in*: *Computers & Operations Research* 40.*1* (2013), pp. 166–179 (cit. on pp. 34, 73).

[YZ95] J. Yuan and S. Zhou, « Optimal labelling of unit interval graphs », *in*: *Applied Mathematics* 10.*3* (1995), pp. 337–344 (cit. on p. 25).

[Zho00] S. Zhou, « Bounding the bandwidths for graphs », *in*: *Theoretical Computer Science* 249.*2* (2000), pp. 357–368 (cit. on pp. 18, 25).

[ZH17] Y. Zhou and J.K. Hao, « An iterated local search algorithm for the minimum differential dispersion problem », *in*: *Knowledge-Based Systems* 125 (2017), pp. 26–38 (cit. on p. 54).

[ZHG18] Y. Zhou, J.K. Hao, and F. Glover, « Memetic search for identifying critical nodes in sparse graphs », *in*: *IEEE transactions on cybernetics* 49.*10* (2018), pp. 3699–3712 (cit. on p. 73).

[ZHG16] Y. Zhou, J.K. Hao, and A. Goëffon, « A three-phased local search approach for the clique partitioning problem », *in*: *Journal of Combinatorial Optimization* 32.*2* (Aug. 2016), pp. 469–491 (cit. on p. 28).

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

université angers

**Titre :** Algorithmes d'optimisation pour deux problèmes de disposition des graphes

**Mot clés :** Problèmes de disposition des graphes, Recherche locale, Métaheuristiques, Expérimentations.

**Résumé :** Cette thèse considère deux problèmes de disposition des graphes : le problème de la bande passante cyclique (CBP) et le problème de l'agencement linéaire minimum (MinLA). Le CBP est une extension naturelle du problème de minimisation de la bande passante (BMP) et le MinLA est un problème de somme minimale. Ces problèmes sont largement appliqués dans la vie réelle. Puisqu'ils sont $\mathcal{NP}$-difficile, il est difficile de les résoudre dans le cas général. Par conséquent, cette thèse est consacrée au développement d'algorithmes heuristiques efficaces pour faire face à ces problèmes. Plus précisément, nous introduisons deux algorithmes de recherche locale itétée, un algorithme mémétique avec différents opérateurs de recombinaison pour le CBP et une heuristique de voisinage basée sur un ensemble pour résoudre le MinLA. On montre expérimentalement que pour le CBP, les deux algorithmes de recherche locale itéré pouvaient concurrencer favorablement les méthodes de l'état de l'art, le croisement approprié est identifié pour l'algorithme mémétique. On montre également que pour le MinLA, l'heuristique de voisinage basée sur l'ensemble s'est avérée plus efficace que des algorithmes avec voisinage traditionnel à 2-flip.

**Title:** Optimization algorithms for graph layout problems

**Keywords:** Graph layout problems, Local search, Metaheuristics, Computational experiments.

**Abstract:** This thesis considers two graph layout problems: the cyclic bandwidth problem (CBP) and the minimum linear arrangement problem (MinLA). The CBP is a natural extension of the bandwidth minimization problem (BMP) and the MinLA is a min-sum problem. These problems are widely applied in the real life. Since they are $\mathcal{NP}$-hard problems, it is computational difficult to solve them in the general case. Therefore, this thesis is dedicated to developing effective heuristic algorithms to deal with these challenging problems. Specifically, we introduce two iterated local search algorithms, a memetic algorithm with different recombination operators for the CBP and a set based neighborhood heuristic algorithm to solve the MinLA. The two iterated local search algorithms are experimentally demonstrated to be able to compete favourably with state-of-the-art methods, the feature of a suitable crossover for the memetic algorithm is identified for the CBP and the set based neighborhood heuristic algorithm is proven to be more efficient than the traditional 2-flip neighborhood algorithm.