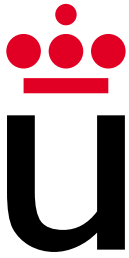


Universidad
Rey Juan Carlos

TESIS DOCTORAL

Heuristic optimization of graph embedding problems in circular layouts

Autor:

Sergio Cavero Díaz

Directores:

Abraham Duarte Muñoz

Eduardo García Pardo

Programa de Doctorado en Tecnologías de la Información y las Comunicaciones

Escuela Internacional de Doctorado

2023

This Doctoral Thesis has been partially supported by the Ministerio de Ciencia, Innovación y Universidades (Grant Ref. PGC2018-095322-B-C22, PID2021-125709OA-C22, FPU19/04098, and EST22/00444) and by the Comunidad de Madrid and the Fondo Europeo de Desarrollo Regional (Grant Ref. P2018/TCS-4566).

El Dr. D. Abraham Duarte Muñoz, Profesor Catedrático de Universidad del Departamento de Informática y Estadística, y el Dr. D. Eduardo García Pardo, Profesor Titular de Universidad del Departamento de Informática y Estadística de la Universidad Rey Juan Carlos, directores de la Tesis Doctoral *Heuristic optimization of graph embedding problems in circular layouts* realizada por el doctorando D. Sergio Cavero Díaz,

HACEN CONSTAR:

que esta Tesis Doctoral reúne los requisitos necesarios para su defensa y aprobación.

En Móstoles, Marzo de 2023,

Dr. D. Abraham Duarte Muñoz

Dr. D. Eduardo García Pardo

FIRMADO POR	FECHA FIRMA
DUARTE MUÑOZ ABRAHAM - DIRECTOR DE LA ESCUELA SUPERIOR DE INGENIERÍA INFORMÁTICA	27-02-2023 15:46:21
GARCIA PARDO EDUARDO	27-02-2023 16:44:44

A mis padres y a mis abuelos.
Esta Tesis es el resultado de vuestro trabajo.

Nōlī turbāre circulōs meōs!

Archimedes

Acknowledgments

The achievement of this thesis would not have been possible without the valuable help and support of my family, friends, and advisors. I would like to take this opportunity to thank everyone who has contributed to my journey of research and learning.

First and foremost, I would like to express my deepest gratitude to my supervisors, Eduardo and Abraham. Eduardo has given me guidance and support beyond this thesis. It seems like only yesterday that you were preparing me to present my bachelor's s thesis. Abraham, thank you for trusting me without even knowing me. I am very grateful for your teachings and advice, but above all for being able to enjoy this predoctoral scholarship with you.

Without a doubt, I have really enjoyed these years as a doctoral student. I could not be more fortunate to have shared an office with the best colleagues and friends I could have in this "fun run". There has been no lack of laugh since the first day we met. It has been a pleasure to have had the opportunity to work with you, guys.

I would like to thank all those people who have made my stay in the United States possible, especially Manuel Laguna. This experience would not have been the same without Manuel Laguna and his family, Rafael Martí and his family, Marco Better and his family and especially the Thoresen family who made me feel at home. I miss you all.

Finally, to all those people who were already there before I started this Thesis, especially my family. Thank you, mom and dad, for supporting me and for teaching me how to play this wonderful board game that is life. Thanks to María, for being by my side in each one of my crazy adventures.

Abstract

Optimization is a discipline that addresses the search for the best possible solution, called the optimal solution, to a problem mathematically modeled. These problems can be classified according to its computational complexity. Problems belonging to the NP-hard class are too complex to be solved with an exact algorithm in a reasonable amount of time. Alternatively, these problems can be approached through approximate techniques that allow finding good quality solutions, although not necessarily optimal, in a reasonable amount of computing time. Among these techniques, heuristic and metaheuristic algorithms stand out, since they have been proven as useful tools in solving high-complexity real problems.

In this Doctoral Thesis, heuristic and metaheuristic algorithms are proposed for the resolution of four Graph Layout Problems (GLP). Specifically, the problems studied belong to the NP-hard class and can be framed as combinatorial optimization problems. GLPs aim to find the best possible assignment of the vertices of an input graph to the vertices of a host graph, optimizing a certain objective function. More specifically, this Doctoral Thesis focuses on the study of GLPs in which the embedding is done in circular layouts. This family of problems is of great interest due to the variety of practical applications they have. In this research, a methodology for addressing GLPs is proposed. Specifically, it starts from the study of each problem, and then proposes heuristic and metaheuristic algorithms for tackling the problem. After a preliminary experimentation, the algorithmic proposal is compared to the existing methods in the state of the art. This methodology has been successfully applied to the studied problems, resulting in various scientific publications that compile the main findings of the research carried out.

Resumen

La optimización es una disciplina que aborda la búsqueda de la mejor solución posible, denominada solución óptima, a problemas modelados matemáticamente. Los problemas pueden ser clasificados según su complejidad computacional. Concretamente, los pertenecientes a la clase NP-difícil son demasiado complejos como para ser resueltos con un algoritmo exacto en un tiempo asumible. Alternativamente, estos problemas pueden ser abordados mediante técnicas aproximadas que permiten encontrar soluciones de calidad, aunque no necesariamente óptimas, en tiempos razonables. Entre estas técnicas se destacan los algoritmos heurísticos y metaheurísticos que se han consolidado como herramientas de gran utilidad en la resolución de problemas reales de alta complejidad.

En esta Tesis Doctoral se proponen algoritmos heurísticos y metaheurísticos para la resolución de cuatro problemas de embebido de grafos (GLP, del inglés, *Graph Layout Problems*). Concretamente, los problemas estudiados pertenecen a la clase NP-difícil y pueden ser enmarcados como problemas de optimización combinatoria. Los GLP tienen como objetivo buscar la mejor asignación posible de los vértices de un grafo de entrada a los vértices de un grafo huésped, optimizando una determinada función objetivo. Más concretamente, esta Tesis Doctoral se centra en el estudio de varios GLP cuyo embebido se realiza en estructuras cíclicas. Esta familia de problemas es de gran interés por la variedad de aplicaciones prácticas que tienen. En esta investigación, además, se propone una metodología para abordar los GLP. Esta metodología parte del estudio de cada problema para, a continuación, proponer algoritmos heurísticos y metaheurísticos para el mismo. Tras una experimentación preliminar, la propuesta algorítmica es comparada con los métodos existentes en el estado del arte. Esta metodología ha sido aplicada con éxito para los problemas estudiados, resultando en diversas publicaciones científicas que recogen las principales contribuciones de la investigación realizada.

Contents

List of tables

List of figures

List of acronyms

2DBMP	Two-Dimensional Bandwidth Minimization Problem
ABP	Antibandwidth Minimization Problem
ABC	Artificial Bee Colony
ACO	Ant Colony Optimization
BFS	Breadth First Search
BMP	Bandwidth Minimization Problem
BVNS	Basic Variable Neighborhood Search
CGLP	Circular Graph Layout Problem
CAB	Cyclic Antibandwidth Problem
CBS	Cyclic Bandwidth Sum Problem
CBMP	Cyclic Bandwidth Minimization Problem
CCMP	Cyclic Cutwidth Minimization Problem
CMP	Cutwidth Minimization Problem
CSP	Constraint Satisfaction Problem
EDA	Estimation of Distribution Algorithms
EP	Evolutionary Programming
FLP	Facility Layout Problem
FPTAS	Fully Polynomial-Time Approximation Scheme
GA	Genetic Algorithm
GLP	Graph Layout Problem
GLS	Guided Local Aearch
GP	Genetic Programming
GRAFO	Group for Research in Algorithms For Optimization
GRASP	Greedy Randomized Adaptive Search Procedure
GVNS	General Variable Neighborhood Search
IG	Iterated Greedy

ILS	Iterated Local Search
JCR	Journal Citation Reports
MA	Memetic Algorithm
MinLA	Minimum Linear Arrangement Problem
PSO	Particle Swarm Optimization
PTAS	Polynomial-Time Approximation Scheme
PVNS	Parallel Variable Neighborhood Search
RVNS	Reduced Variable Neighborhood Search
SA	Simulated Annealing
SJR	Scimago Journal & Country Rank
SS	Scatter Search
TS	Tabu Search
URJC	Universidad Rey Juan Carlos
VFS	Variable Formulation Search
VLSI	Very Large Scale Integration
VND	Variable Neighborhood Descent
VNS	Variable Neighborhood Search

Part I

PhD Dissertation

Chapter 1

Introduction

The Doctoral Thesis presented in this document is framed within the discipline of heuristic optimization. The first chapter defines the key terms and basic concepts necessary to contextualize this dissertation in the area of optimization. In addition, different strategies to address optimization problems are discussed. Next, a particular family of combinatorial optimization problems is described, the Graph Layout Problems. This chapter concludes by formulating the hypothesis and objectives of this Doctoral Thesis and the research method used.

1.1 Optimization

Intuitively, we face optimization problems in almost any decision we make in our daily lives. For instance, when we want to purchase something at the best price, we explore different alternatives and weigh their pros and cons. When we desire to travel from one location to another, we opt for the quickest or most cost-effective route. When we want to manage our time, we rank the most relevant or pressing activities. All these are scenarios in which we aim to maximize or minimize the value of a particular magnitude, subject to certain conditions.

Optimization, from a more technical point of view, is a field of research concerned with the search for the best possible solution to a mathematically modeled problem. For this problem, it is defined as one or more mathematical functions (called objective functions)

and a set of constraints that the solution must satisfy. A solution is represented by the values assigned to a set of variables. The quality of a solution is determined by evaluating the objective function associated to the values of the variables. Furthermore, if a solution satisfies the given constraints, it is called a feasible solution.

In the following section, an optimization problem is mathematically defined, and the most relevant methods used to tackle these problems are listed: exact methods, approximate methods, and heuristic and metaheuristic methods.

1.1.1 Optimization problems

An optimization problem may be defined by the couple (S, f), where S represents the set of feasible solutions, commonly denoted as the search space or the solution space, and $f : S \rightarrow \mathbb{R}$ is the objective function to optimize, that is, to minimize or maximize. The objective function assigns a real number to each solution in the solution space, $s \in S$, which represents the quality of the solution. Therefore, the objective function f provides a clear way to determine whether one solution is better than others. Finally, the solution to a minimization optimization problem consists of finding a solution that minimizes the function f. In mathematical terms:

$$s^\star \in S : \forall s \in S, f(s^\star) \leq f(s). \tag{1.1}$$

The solution to an optimization problem is also known as the global optimum. Therefore, the main goal when solving optimization problems is to find the global optimal solution $s^\star$. Note that the above definition can be particularized for maximization problems by simply changing "$\leq$" to "$\geq$".

Figure 1.1 illustrates a possible graphical representation of a minimization optimization problem, as well as the most relevant concepts presented so far. Specifically, in this figure, the search space of an optimization problem is represented by the x-axis, which also contains all feasible solutions to the problem. The y-axis represents the value of the objective function associated with each of the solutions. As an example, two solutions s and $s^\star$ are illustrated, where $s^\star$ is also the optimal solution to the problem since there is no solution with a lower value of the objective function in the search space.

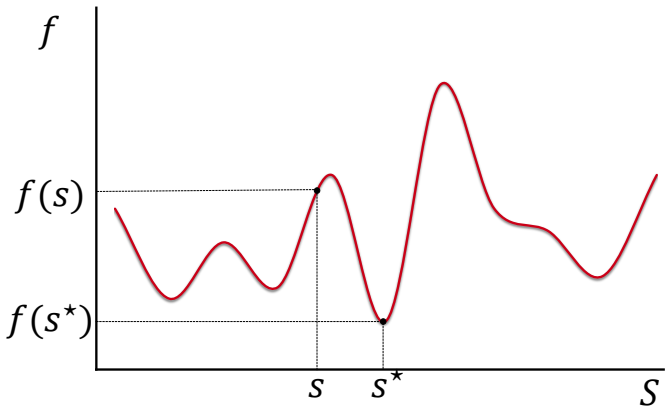


Figure 1.1 Graphical representation of an optimization problem where the x-axis represents the search space S, and the y-axis represents the objective function value of each solution $s, s^\star \in S$.

Different families of optimization problems are used to formulate and solve problems in many domains. Optimization problems naturally fall into two categories with respect to the variables used to represent a solution: those with continuous variables and those with discrete variables, often referred to as combinatorial. Continuous optimization problems are those that search for a set of real numbers or a function; combinatorial optimization problems are characterized by discrete decision variables and a finite search space [188].

Optimization problems are also classified according to their complexity. The complexity of a problem is usually measured with the complexity of the best algorithm known to solve it. A problem can be considered as *tractable* or *uncomplicated* if a polynomial-time algorithm is able to solve it. Otherwise, a problem is considered *intractable* or *complicated* if the algorithm that solves it runs in non-polynomial time [238]. Generally, problems are classified into one of the following categories or classes according to their complexity: P, NP, NP-complete, and NP-hard.

A problem belongs to the complexity class P if there is a deterministic algorithm that can solve it in polynomial time relative to the size of the input. For the vast majority of combinatorial optimization problems, no algorithm is known to obtain an optimal solution in polynomial time. Consequently, they cannot be solved in a reasonable time. This type of problem is called NP. As is shown in Figure 1.2, the set of problems belonging to the class P is a subset of NP problems. Another subset of NP problems is the NP-complete set. For the

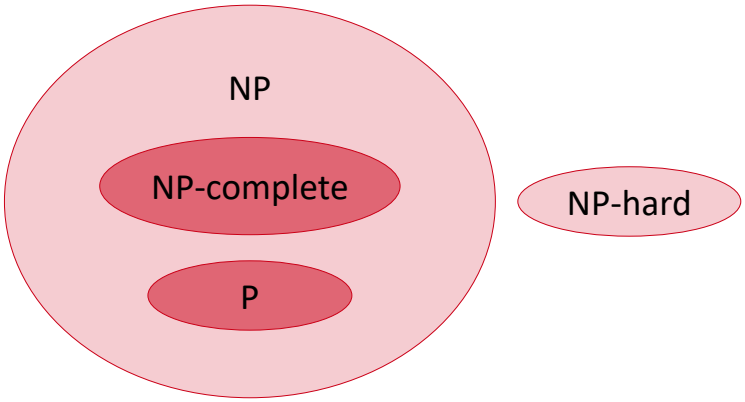


Figure 1.2 Complexity classes of decision problems.

problems belonging to the complexity class NP-Complete, there is no polynomial algorithm able to solve them (or at least it has not been found yet), but there does exist a polynomial algorithm that can determine whether a solution solves that problem or not. These problems are the most complex in the NP class. In addition, they present the peculiarity that if only one of the problems of this class were solvable by means of a polynomial algorithm, all of them would be polynomially solvable. Finally, the NP-hard class encompasses all those problems that do not have a polynomial algorithm to verify whether a given solution solves the problem. The relationship between these four complexity classes is depicted in Figure 1.2.

1.1.2 Optimization methods

Depending on the complexity of an optimization problem, it may be solved using an exact or approximate method. Exact methods are those that guarantee the optimality of the solution found, while the approximate methods generate high-quality solutions, but there is no guarantee of finding a global optimal solution. Additionally, this last group of approximate methods is divided into approximation algorithms, heuristic algorithms, and metaheuristic algorithms. This section collects some of the most relevant exact and approximate methods for dealing with optimization problems. Specifically, the section is organized into four blocks: exact techniques, approximation techniques, heuristic techniques, and metaheuristic techniques.

Exact techniques

Although this Doctoral Thesis does not focus on problem-solving by means of exact algorithms, it is worth highlighting some of the most important exact techniques, such as dynamic programming, the branch and bound family, or constraint programming.

Dynamic programming consists of recursively splitting a problem into simpler subproblems. This stepwise optimization method is the result of a series of partial decisions. The procedure avoids the total enumeration of the search space by removing partial decision series that will not lead to optimal solutions [17, 238].

The branch and bound and similar algorithms (branch and bound, branch and cut, branch and price, or $A^\star$, among others) are based on the dynamic exploration of a solution tree for the optimization problem considered. The root node of the tree represents the problem to be solved and its associated search space. Leaf nodes represent a solution to the problem, and internal nodes are sub-problems of the total solution space. The pruning of the search tree is based on a bounding function that prunes subtrees that do not contain any optimal solution [143, 184, 238].

Constraint programming is a programming paradigm in which the relationships between variables are expressed in terms of constraints. Optimization problems in constraint programming are modeled using a set of variables linked by a set of constraints. Variables take their value in a finite integer domain, while constraints can be expressed symbolically or mathematically [214, 238].

However, most of today's relevant and practical problems do not have an exact algorithm with polynomial complexity that provides optimal solutions in a reasonable amount of time, since the solution space is immense. For this reason, approximate algorithms, such as approximation or heuristic algorithms, play a fundamental role.

Approximation techniques

Approximation algorithms, unlike heuristic algorithms, guarantee the bound of the solution obtained with respect to the global optimum [94, 115]. This Doctoral Thesis also does not focus on proposing approximation algorithms, but it is worthwhile to present some of the most relevant ones.

An ε-approximation algorithm for an optimization problem is a polynomial-time algorithm that for any input instance of the problem produces a solution whose objective function value is within a factor of ε of the value of the optimal solution [252]. In mathematical terms:

$$\begin{aligned} s &\leq \varepsilon \cdot s^{\star} \quad \text{if } \varepsilon > 1 \\ \varepsilon \cdot s^{\star} &\leq s \quad \text{if } \varepsilon < 1 \end{aligned} \tag{1.2}$$

where s is the solution produced and $s^{\star}$ is the global optimal solution, and the factor ε determines the relative performance guarantee [238].

Given the definition of an ε-approximation algorithm, a problem is in the Polynomial-Time Approximation Scheme (PTAS) class if there exists a polynomial-time $(1+\varepsilon)$-approximation algorithm for any fixed $\varepsilon > 0$. Similarly, a problem is in the Fully Polynomial-Time Approximation Scheme (FPTAS) class if there exists a polynomial-time $(1+\varepsilon)$-approximation algorithm in terms of both input size and $1/\varepsilon$ for any fixed $\varepsilon > 0$ [238].

In general, the aim of designing a problem approximation algorithm is to find a tight worst-case bound. However, approximation algorithms are specific to the target optimization problem (i.e., they are problem dependent). This characteristic limits its applicability. Moreover, today's problems tend to be difficult because of the high cardinality of the solution space, the presence of many constraints, or the existence of more than one objective to be optimized simultaneously. In fact, for the problems of most interest, these classical techniques rarely find solutions close to the near-optimal solutions in a reasonable amount of time. In this situation, heuristic and metaheuristic algorithms manage to find a quality solution in a reasonable time.

Heuristic techniques

The word "heuristic" has an etymological origin from the Greek word *euriskein* which comes from *eureka*, a word that means "finding" or "encountering". For example, the

Merriam-Webster Collegiate Dictionary[1] gives the following definition of the word "heuristic": "involving or serving as an aid in learning, discovery, or problem-solving using experimental and especially trial-and-error methods" [113]. Similarly, the dictionary of the Real Academia Española[2] defines the word "heuristics" as the technique of inquiry and discovery, and the process of searching and researching historical documents or sources [114].

From a scientific point of view, the term "heuristics" was first used by G. Polya to express the rules by which humans managed common knowledge. Later, S.H. Zanakis defined it as follows [260]:

> "*Simple procedures, often guided by common sense, that are meant to provide good but not necessarily optimal solutions to difficult problems, easily and quickly*".

Although the coining of the word "heuristics" may be considered recent, people have been using heuristic techniques since its very beginnings. In fact, in the chapter "A History of Metaheuristics" in the recognized book "Handbook of Heuristics", K. Sörensen *et al.* state that when studying the history of heuristics with an open mind, it is easy to realize that people used heuristic procedures long before the term existed [232]. Furthermore, in the same chapter, the authors put forward that it is not until 1940 that the first formal studies on heuristics appear. Since then, the literature in this field has been flooded with a wealth of new techniques and heuristic procedures to solve optimization problems.

Given the diversity of heuristic algorithms proposed over time, and that they are generally designed for specific problems, classifying them is a complicated and challenging task. However, there is a clear division between those procedures that aim to construct a solution to a given problem, generally from scratch; and those search procedures that start from a given solution and try to improve it [227, 261]. In this Doctoral Thesis, heuristic methods are the key component for the resolution of the optimization problems studied. The most

[1] Merriam-Webster, Inc. is a well-known U.S. reference book publisher, usually recognized for its dictionaries.

[2] Real Academia Española, more commonly known as RAE, is a cultural institution dedicated to linguistic regularization among the Spanish-speaking world.

widely used techniques are listed below.

Constructive heuristics are usually the fastest approximate methods. They deal with the generation of solutions starting, generally, from an empty partial solution and ending with a feasible complete solution after adding step-by-step individual components to it (e.g., vertices, edges, variables). Among the heuristic construction algorithms, greedy algorithms emerge above others due to their ability to find good quality solutions [227]. A baseline greedy construction adds at each step a solution component for which the value of a defined heuristic function is the best [235]. However, as will be seen in further detail, the diversity of the solutions produced is also relevant for exploring the solution space in depth. Therefore, in cases where greedy construction is fully deterministic, additional randomization of the construction process may be adequate. Based on these ideas, researchers propose semi-greedy heuristics [107] or the randomization of the procedure [72].

Repeated construction of solutions makes sense when the construction is not completely deterministic and, therefore, the generation of multiple solutions allows exploring different points in the search space. Moreover, this strategy could be exploited when some knowledge gained from previous solutions could be used to generate the following solutions. However, the construction of multiples is not recommended when generation is relatively time-consuming, especially when initial construction steps require a lot of computation compared to later construction steps [235].

Generally, the solution of the constructive procedures mentioned above is used as a starting point for improvement methods such as local search [171, 261]. The local search heuristic starts with a feasible solution and at each iteration tries to replace it with a better solution from a reduced set of solutions in the solution space. This subset of solutions is known as a "neighborhood" and the replacement of the solution is denoted as a "move". Therefore, a neighborhood is made up of all those solutions that can be reached by performing a specific move (or a set of moves).

From a mathematical perspective, a neighborhood can be understood or defined by a function N such that $N : S \rightarrow P(S)$, where S denotes the finite set of solutions and $P(S)$ represents the set of all subsets of solutions of S. For each solution $s \in S$, $N(s)$ provides all neighboring solutions of s, that is, all solutions that can be reached from s by a move [171].

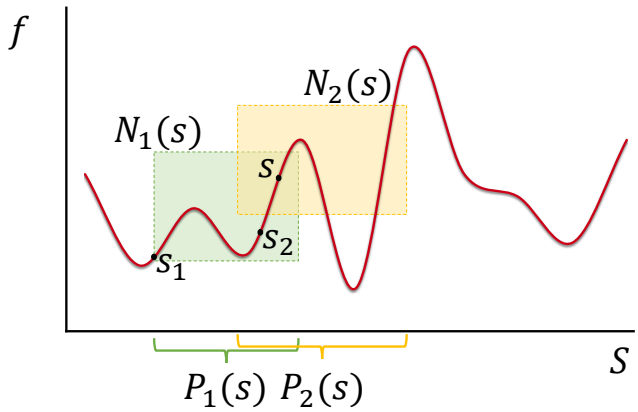


Figure 1.3 Graphical representation of two possible neighborhoods (N_1, and N_2) of a solution s, a solution obtained with a "first improvement" strategy (s_2) and another one obtained with a "best improvement strategy" (s_1).

The exploration of a neighborhood has been extensively studied and can be detrimental to the performance of algorithms, especially in cases where exploration is costly. Among the exploration strategies, the first and best improvement strategies stand out in the literature. If a local search follows the "best improvement strategy", it selects, at each iteration, the best possible move that produces an improvement in the current neighborhood; otherwise, in the "first improvement strategy", it selects the first solution that improves the current solution.

Based on the example of the minimization optimization problem presented above, two possible neighborhoods (N_1 and N_2) for the solution s are represented in Figure 1.3. Each neighborhood is formed by a subset of solutions of the solution space (P_1 and P_2 respectively). When exploring the neighborhood N_1 (for example), either of the two strategies mentioned above can be followed: "best" or "first improvement". On the one hand, following a "best improvement" strategy would result in the exploration of all solutions of $P_1(s)$ and a move would be made to the best of them all, in this case, s_1. On the other hand, if a "first improvement" strategy was followed, the first solution that improves the quality of the solution is chosen, such as s_2.

Having defined the main concepts of the local search heuristic, some potential problems that can be found when implementing it must be discussed. Among these problems, the most relevant in the context of this Doctoral Thesis are presented below.

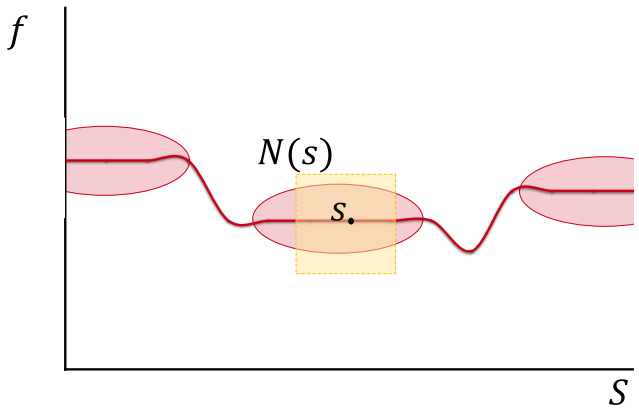


Figure 1.4 Representation of a solution space with three flat landscapes. For example, considering the solution s and its neighborhood $N(s)$, all solutions in $N(s)$ have the same objective function value as s.

The first unavoidable problem is easily getting stuck in local minima or maxima. Local search algorithms often get stuck in local optima, which are suboptimal solutions that cannot be improved by making any changes within the immediate neighborhood of that solution. This issue arises particularly when the objective function fails to clearly differentiate between the quality of candidate solutions. As a consequence, some problems present plateaus or valleys in the fitness landscape, which can obscure distinctions in fitness levels among solutions. This situation is also known as flat landscapes [16, 207, 210]. A graphical representation of a solution space with flat landscapes is depicted in Figure 1.4. In this Doctoral Thesis, we observed that flat landscapes are likely to occur in optimization problems formulated as max-min (or min-max) problems, where the objective function consists of maximizing a minimum value (or minimizing a maximum value). The solution proposed to deal with this situation in the problems studied is presented in Section 3.4.2.

Flat landscapes are one of the reasons for the slow convergence of the proposed algorithms. However, slow convergence may also be due to other reasons such as large search space, complex and time-consuming computation of the objective function, or complex constraints to be satisfied.

When addressing large neighborhoods, the termination criteria could become a relevant issue to consider. In addition, detecting when it is time to terminate an improvement process to avoid over-exploring a region of the search space is a very effective strategy when

combined with metaheuristic procedures, such as multistart metaheuristics, which will be described in the following. Other approaches propose reduction techniques to decrease the number of solutions in a neighborhood. In Section 3.4.3 some strategies are collected in order to reduce the size of the neighborhoods proposed in this research.

The objective function is a key component of local search algorithms since it is calculated multiple times to evaluate the solutions within a neighborhood. In some cases, evaluating the objective function requires a long calculation time, therefore, researchers have devised different approaches to address this crucial issue. For example, using approximations of the original objective functions or other simpler heuristics with similar characteristics. In this dissertation, this problem is faced through an efficient or intelligent evaluation of the objective function, avoiding the complete recalculation after a move (see Section 3.4.1).

An additional aspect to take into account when applying local search strategies is the influence or sensitivity of the initial solution. In some cases, the quality of the final solution found by a local search algorithm can be highly dependent on the initial solution used as a starting point. This means that only a few regions are explored, and it is necessary to introduce some diversity in the generated solutions.

In this sense, local search algorithms must be able to balance the need to explore new solutions with the need to exploit the current best solution. If the algorithm focuses too much on exploitation, it may get easily stuck in a local optimum, while if it focuses too much on exploration, it may waste time exploring bad-quality solutions. This difficulty is closely related to the definition of a suitable neighborhood. In some cases, it may be difficult to define an appropriate neighborhood for a given problem, which can make it difficult to use local search algorithms effectively.

Metaheuristic techiques

Frequently, heuristic methods, such as constructive procedures or local search procedures, are problem-dependent; that is, they are defined for a given problem to exploit problem-dependent information. On the contrary, metaheuristics are problem-independent techniques that can be applied to any problem. Metaheuristics, besides being of general applicability, solve several of the problems of the heuristics presented above.

The term “metaheuristic” is formed by the Greek prefix *meta* (beyond in the sense of

high-level) with heuristic and is credited to F. Glover [88]. From an algorithmic point of view, the term is used in two ways. The first way uses it to refer to a high-level framework, a set of strategies that combine to develop a complete optimization algorithm. The second meaning of "metaheuristic" denotes a specific implementation of an algorithm based on such a framework (or on a combination of concepts from different frameworks) designed to find a solution to a specific optimization problem. In this Doctoral Thesis, the term "metaheuristic framework" refers to the first meaning, while the "metaheuristic algorithm" is associated with the second sense of the word "metaheuristic". Therefore, taking into account the two possible meanings of the word, in this document, the following definition proposed by K. Sörensen and F. Glover is adopted [231]:

> "*A metaheuristic is a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms. The term is also used to refer to a problem-specific implementation of a heuristic optimization algorithm according to the guidelines expressed in such a framework.*"

Despite the wide variety of metaheuristics that have been proposed [233], three main ways of classifying them by the manner they manipulate solutions to produce effective solutions can be found in the literature: constructing a solution by its constituent parts (denoted as constructing metaheuristics); making small changes (local search metaheuristics); and combining solutions into new ones (population-based metaheuristics) [92, 231].

Based on the previous classification and moving from the first greedy heuristic proposed for a combinatorial optimization problem in 1971 by J. Edmonds [66] and the first local search algorithm proposed within the simplex algorithm by G. Dantzig in 1947 [47], Figure 1.5 collects some of the most relevant metaheuristics organized by date of publication (y-axis) and by origin or main inspiration (x-axis).

Specifically, the following metaheuristics are depicted: Artificial Bee Colony (ABC) [225, 258], Ant Colony Optimization (ACO) [55, 56], Estimation of Distribution Algorithms (EDA) [10, 142], Evolutionary Programming (EP) [76, 256], Genetic Algorithm (GA) [116, 51], Guided Local Aearch (GLS) [247, 248], Genetic Programming (GP) [138, 139], Greedy Randomized Adaptive Search Procedure (GRASP) [71, 72], Iterated Greedy

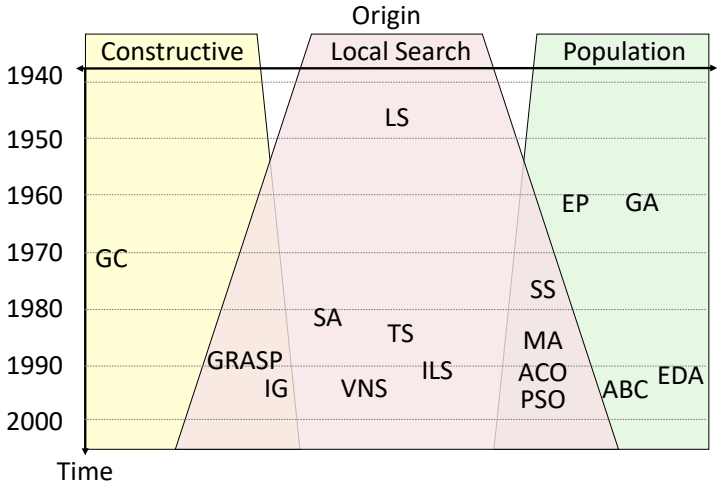


Figure 1.5 Origin and inspiration of the most relevant heuristic and metaheuristic algorithms, ordered chronologically according to their publication date and classified by type of strategy.

(IG) [123, 216], Iterated Local Search (ILS)[155, 160], Memetic Algorithm (MA) [179, 180], Particle Swarm Optimization (PSO) [131, 195], Simulated Annealing (SA) [134, 244], Scatter Search (SS) [87, 91], Tabu Search (TS) [87, 90] and Variable Neighborhood Search (VNS) [174, 175].

Another possible organization of metaheuristics is based on their inspiration or origin. Specifically, there is a clear distinction between nature-inspired and non-nature-inspired metaheuristics. For example, GA and EP have their origin in biology; ABC, ACO, and PSO in different species or animals; or SA in physics. On the other hand, metaheuristics such as VNS or IG have no natural inspiration.

Additionally, researchers also distinguish metaheuristics that make use of memory during the search. The most recognized memory-based metaheuristic is TS [87, 90]. Figure 1.6 illustrates in a simple way the search process guided by a tabu memory. In Figure 1.6(a), starting from the solution s, an improvement is made to the solution s_1 from among the solutions of its neighborhood $N(s)$. After the move, the previous neighborhood is marked as tabu, colored gray in Figure 1.6(b). Next, in Figure 1.6(b), from the solution s_1, any solution of $N(s_1)$ could be explored as long as it is not marked as tabu. In this case, only

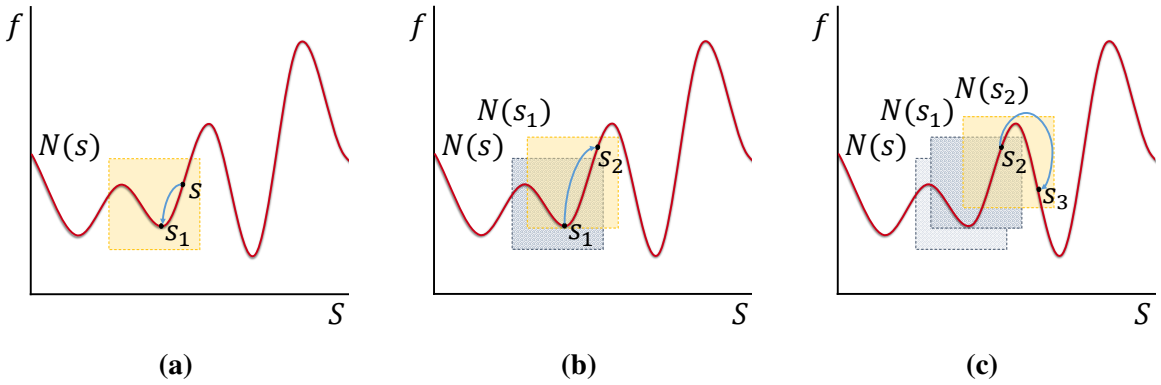


Figure 1.6 Local search procedure based on tabu memory. (a) Improving move from s to $s_1 \in N(s)$. (b) $N(s)$ is marked as tabu and a worsening move is made from s_1 to $s_2 \in N(s_1) \setminus N(s)$. (c) $N(s_1)$ is marked as tabu and an improving move is made from s_2 to $s_3 \in N(s_2) \setminus \{N(s) \cup N(s_1)\}$.

a worsening move is possible, for example, to solution s_2. This new move adds new solutions to the tabu memory. Finally, in Figure 1.6(c) a move is made to one of the solutions of $N(s_2)$, in this case to a better-quality solution. This example shows how metaheuristics can cope with one of the main problems of local search and other heuristic algorithms: local optimal solutions.

Most metaheuristic algorithms operate on a single neighborhood structure. In other words, the search space topology does not change during the course of the algorithm, such as TS. Other metaheuristics, such as VNS or, IG use a set of dynamic structures that allow diversification of the search by switching between different neighborhoods. An example of such a metaheuristic is shown in Figure 1.7. Specifically, in Figure 1.7(a) a move is made from s to the best solution of $N_1(s)$, s_1. Then, in Figure 1.7(b) there is no possible move from s_1 to a better solution in $N_1(s_1)$. Therefore, the neighborhood is changed, from N_1 to N_2. Now, it is possible to move to a better solution, such as s_2.

Researchers also find a clear division between deterministic metaheuristics, those that take deterministic decisions (e.g., IG or TS); and stochastic metaheuristics, where some random rules are applied (e.g., SA, GA or VNS). For example, the VNS metaheuristic performs random movements, known as “shake” [174, 175], to move from a local optimum solution to other regions of the solution space. This perturbation movement is represented

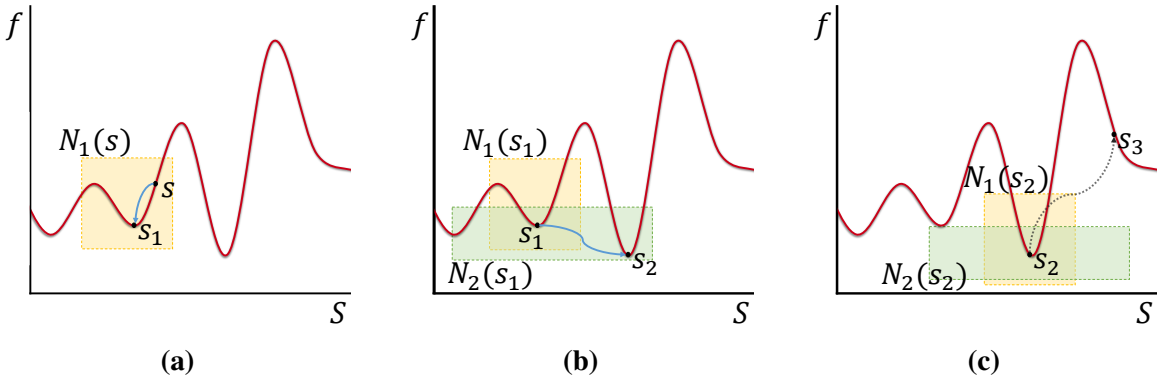


Figure 1.7 Example of a metaheuristics based on multiple neighborhood exploration. (a) Improving move from s to $s_1 \in N_1(s)$. (b) Improving move from s_1 to $s_2 \in N_2(s_1)$. (c) Random shake move from s_2 to s_3.

in Figure 1.7(c) with a gray dashed line from s_2 to s_3. Over time, researchers have proposed more advanced shake or perturbation moves that follow greedy criteria or explore unfeasible solutions. See, for example, [100] or [218].

The perturbation movement aforementioned is an example of a technique that encourages diversification of the exploration of the solution space. Diversification and, therefore, intensification are two opposing criteria that researchers must take into account when implementing a metaheuristic framework. In diversification, unexplored regions must be visited to ensure that all regions of the search space are explored equally, and the search is not limited to a reduced number of regions. In intensification, promising regions are explored more thoroughly in the hope of finding better solutions. Extreme search algorithms in terms of exploration are those based on randomness, such as GRASP [71, 72], Reduced Variable Neighborhood Search (RVNS) [100, 175] or multistart procedures [163, 166]. Similarly, extreme search algorithms based on intensification strategies are Variable Neighborhood Descent (VND) [60, 100] or ILS [155, 160], among others.

Metaheuristic algorithms, in the same way as heuristic algorithms, present certain limitations. D.H. Wolpert and W.G. Macready demonstrated with the well-known "No Free Lunch Theorem" that, on average, no metaheuristic is better than a completely random search [254, 255]. This is due to the fact that there is insufficient corroboration evidence that metaheuristics converge to a local optimum and that a metaheuristic may be efficient

for one set of problems, but not for others.

In this section, two types of methods for dealing with optimization problems have been studied: exact and approximate algorithms. So far, both algorithms have been described independently. However, nowadays, more and more researchers are proposing algorithms that combine ideas from both approaches. Here, matheuristic algorithms emerge. As its name suggests, a matheuristic is the hybridization of mathematical programming with metaheuristics [75]. The availability of commercial software such as Gurobi [97] or CPLEX [122] has created new opportunities in heuristic design, generating this new class of algorithms that combine classical heuristic and metaheuristic schemes with mixed integer linear programming strategies.

1.2 Graph Layout Problems

Graphs are used to represent a significant number of real-life and diverse applications only by connecting points (vertices or nodes) by lines (edges or arcs). For example, in computer science, graphs are employed for the representation of networks of communication, organization of data, the flow of computation, computational devices, etc. In linguistics, graphs are mainly used to analyze language tree grammars and lexical semantic networks [226]. In physics and chemistry, graph theory is used to study molecules by its representation through 3D structures [215]. Statistical physics also uses graphs to represent local connections between interacting parts of a system, as well as the dynamics of a physical process on such systems [85]. In social sciences, some of its most interesting applications are focused on rumor spreading or influence analysis [20]. In the field of biology, graphs have been widely used to represent biomolecules such as genes or proteins [193]. Finally, from a more general point of view, graphs have been used for the design of maps, to define the hierarchy of the company, or to determine the distribution of stores or products in a shopping center [253]. Figure 1.8 illustrates examples of real systems that can be intuitively modeled as graphs.

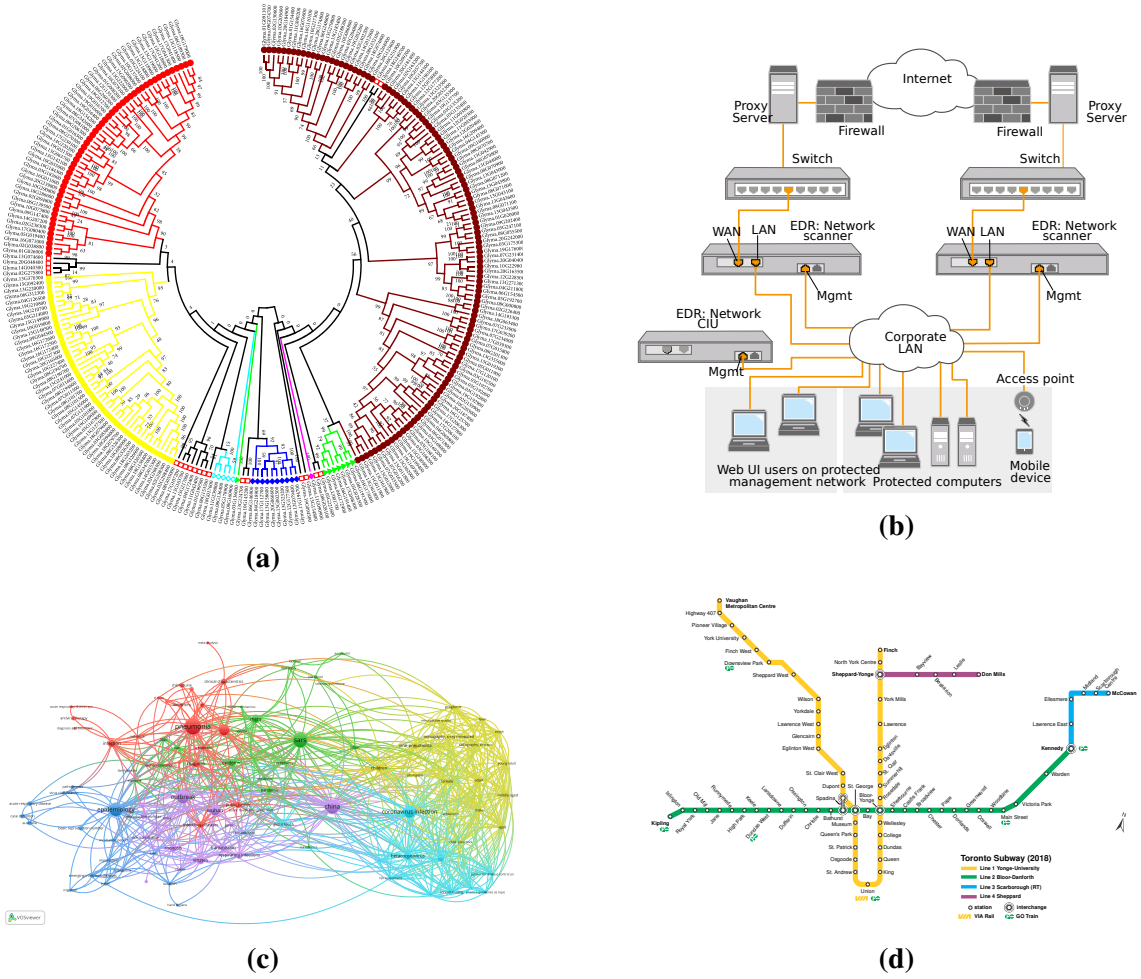
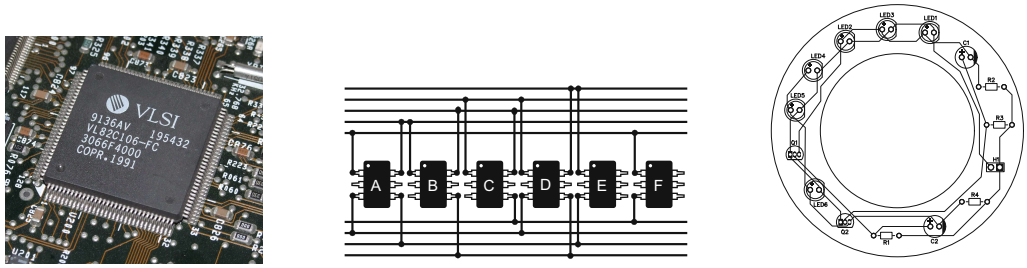


Figure 1.8 Example of graphical representations of real-world systems that can be intuitively modeled as graphs. (a) Phylogenetic relationship of ABC transporter gene family proteins in soybean [173]. (b) Diagram of the architecture of a telecommunications network [24]. (c) Keyword Co-Occurrence Network Graph for the Overall Research Field on COVID-19 up to March 16th, 2020 [5]. (d) Toronto subway map in 2018.

However, the purpose of using graphs is not only to represent real-world systems in order to visualize them but also to model the input of an optimization problem that can be solved computationally by a computer. Therefore, in recent years, there has been a growing interest in studying Graph Layout Problems (GLPs) in the context of Very Large Scale Integration (VLSI) design, which is considered the origin of this family of problems [44,



(a) **(b)** **(c)**

Figure 1.9 (a) A VLSI VL82C106 Super I/O chip [45]. (b) Graph representation of a circuit design with six modules and its corresponding connections [192]. (c) Graph representation of a ring circuit design with 15 modules and its corresponding connections [110].

65]. Figure 1.9(a) shows an example of a VLSI chip. A VLSI circuit can be modeled using a graph, where the edges represent the wires and the vertices represent the modules (see Figures 1.9(b) and 1.9(c)). Then, given a set of modules, designers try to place them on a board in both a non-overlapping manner and wiring together the related terminals on the different modules. Therefore, the objective is to reduce the total number of tracks needed to wire the circuit and, consequently, to decrease the space needed to build it.

After the first motivation, based on the design of VLSI, GLPs has emerged as a family of combinatorial optimization problems whose main objective is to project or embed a graph into another graph, denoted as host graph. This projection has also been denoted as layout [63, 191], labeling [42, 130], numbering [41, 191], arrangement [194, 206] or ordering [25, 81]. Moreover, GLPs have been used to model many other real-life problems and applications such as information retrieval, numerical analysis, biology, and graph theory, among others.

Next, the definition of several graph layout problems and associated concepts are presented. The formulated basic notation is used as a unique framework to define, in general, most of the problems that belong to the GLP family.

1.2.1 Definitions and notation

The main element of every GLP is a graph. In order to define a graph and its related notation, it is presented the standard definition commonly used in Computer Science.

Let $G = (V_G, E_G)$ be an input graph where the set of vertices is denoted as V_G, and its edge set as E_G. Similarly, let $H = (V_H, E_H)$ be a host graph, where the set of vertices is denoted as V_H and its edge set as E_H. Specifically, the most commonly used host graphs are those with a well-known topology in graph theory, such as a path, a cycle, a grid (or lattice), a tree, a hypercube, or a toroid, among others. Some of them are formally defined next.

The notation (u, v) stands for the undirected edge that connects the vertices u and v. The degree of a vertex u on a graph G is denoted as, $deg(u)$ and the maximum and minimum degree of G are defined as $\Delta(G)$ and $\delta(G)$, respectively. Finally, a path of a graph G (similarly with H), denoted as $p(u, v)$, is a sequence of edges. More formally: $p(u, v) = \{(u, u_1), (u_1, u_2), \ldots, (u_{i-1}, u_i), (u_i, v)\}$, where $(u, u_1), (u_1, u_2), \ldots, (u_{i-1}, u_i), (u_i, v) \in E_G$.

Figure 1.10(a) shows an example of an input graph G, with $V_G = \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}, \mathrm{E}\}$ and $E_G = \{(\mathrm{A}, \mathrm{B}), (\mathrm{A}, \mathrm{C}), (\mathrm{A}, \mathrm{E}), (\mathrm{B}, \mathrm{C}), (\mathrm{C}, \mathrm{D}), (\mathrm{C}, \mathrm{E}), (\mathrm{D}, \mathrm{E})\}$. Figures 1.10(b), 1.10(c) and 1.10(d) show an example of a path, cycle, and grid host graph, respectively. In particular, the path host graph, denoted as H_P, depicted in 1.10(b) is made of $V_{H_P} = \{1, 2, 3, 4, 5\}$ and $E_{H_P} = \{(1, 2), (2, 3), (3, 4), (4, 5)\}$. Similarly, the cycle host graph, denoted as H_C, depicted in 1.10(c) is made of $V_{H_C} = \{1, 2, 3, 4, 5\}$ and $E_{H_C} = \{(1, 2), (2, 3), (3, 4)(4, 5), (5, 1)\}$. Finally, the grid host graph, denoted as H_G, depicted in 1.10(d) is made of $V_{H_G} = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ and $E_{H_C} = \{(1, 2), (1, 4), (2, 3), (2, 5), (3, 6), (4, 5), (4, 7), (5, 6), (5, 8), (6, 9), (7, 8), (8, 9)\}$.

Given an input graph G and a host graph H, an embedding or projection of G in H is defined through the definition of two mathematical functions, usually denoted as φ and ψ.

The first of the functions mentioned, φ, assigns each vertex of the input graph to a vertex of the host graph. In mathematical terms, let φ be an injective function such that:

$$\varphi : V_G \to V_H, \forall\, u \in V_G \, \exists!\, v \in V_H \mid \varphi(u) = v. \tag{1.3}$$

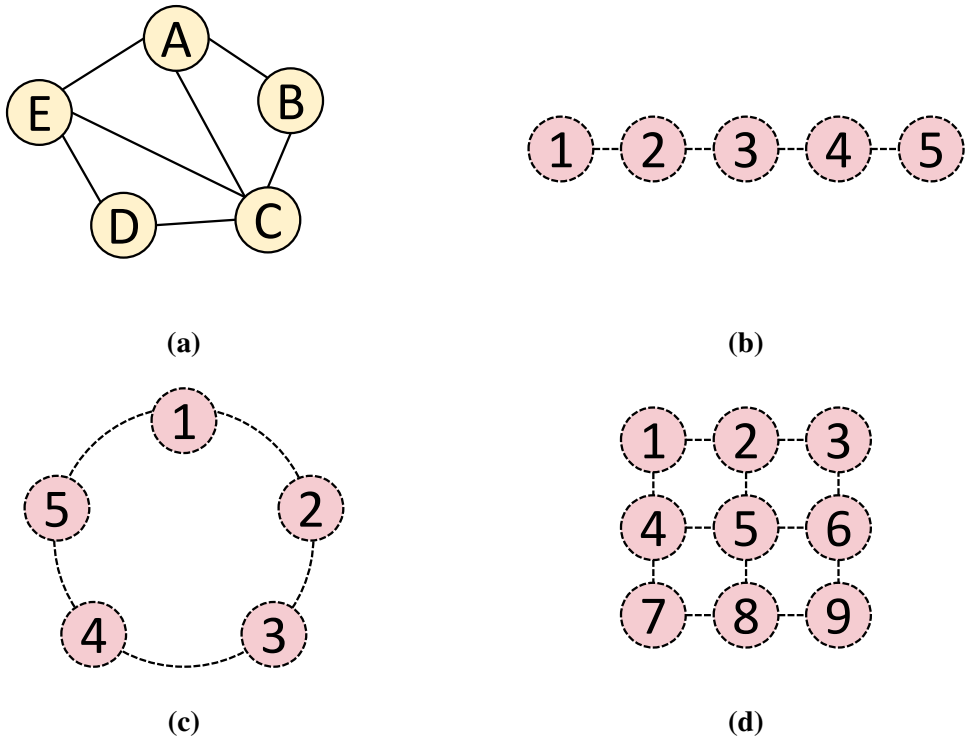


Figure 1.10 Some graphs of the most used host graphs in GLPs. In particular, (a) depicts an input graph, (b) shows a path host graph and, (c) presents a cycle host graph and (d) illustrates a grid host graph.

The second function, ψ, is an injective function that maps each edge $(u,v) \in E_G$ to a set of paths in H whose endpoints are $\varphi(u)$ and $\varphi(v)$. More formally,

$$\psi : E_G \to P_H, \forall\,(u,v) \in E_G\, \exists\, p(\varphi(u),\varphi(v)) \in P_H \text{ with } \varphi(u),\varphi(v) \in V_G. \tag{1.4}$$

For some specific GLPs the function ψ is slightly modified to assign to an edge (u,v) $\in E_G$, the path with the smallest possible cardinality between $\varphi(u)$ and $\varphi(v)$. Formally, this variant of the function ψ is defined in Equation 1.5.

$$\psi((u,v)) = \underset{p(w,z)\,\in\,P_H}{\arg\min}\ \{|p(\varphi(u),\varphi(v))|\}, \tag{1.5}$$

where the operator $|\cdot|$ computes the cardinality of a path, that is, the number of edges that constitute it.

Given the input graph G and the host graphs H_P, H_C and H_G depicted in Figure 1.10, now we define a possible mapping or embedding of G in each of the host graphs. Notice that, in these figures, the host graph is shown in dashed black lines and the input graph in gray solid lines. Specifically, Figure 1.11(a) shows an embedding of G in H_P using the tuple of functions (φ_P, ψ_P). Similarly, Figure 1.11(b) illustrates an embedding of G in H_C through (φ_C, ψ_C). Finally, Figure 1.11(c) shows an embedding of G in H_G through (φ_G, ψ_G). The definition of φ function in any of the three consider embeddings is equivalent: $\varphi_P = \varphi_C = \varphi_G = \{\varphi(\mathrm{A}) = 1, \varphi(\mathrm{B}) = 2, \varphi(\mathrm{C}) = 2, \varphi(\mathrm{D}) = 3, \varphi(\mathrm{E}) = 4\}$. However, the definition of ψ depends on the paths that can be found in the host graph. In Table 1.1 the paths assigned to each edge of the input graph are defined, considering the definition of the ψ function in Equation 1.5. As it can be observed, for the edges (A,E) and (C,E) there are two possible paths that can be assigned through the ψ function when the host graph is a grid. Note that this situation would also occur when the host graph is a cycle.

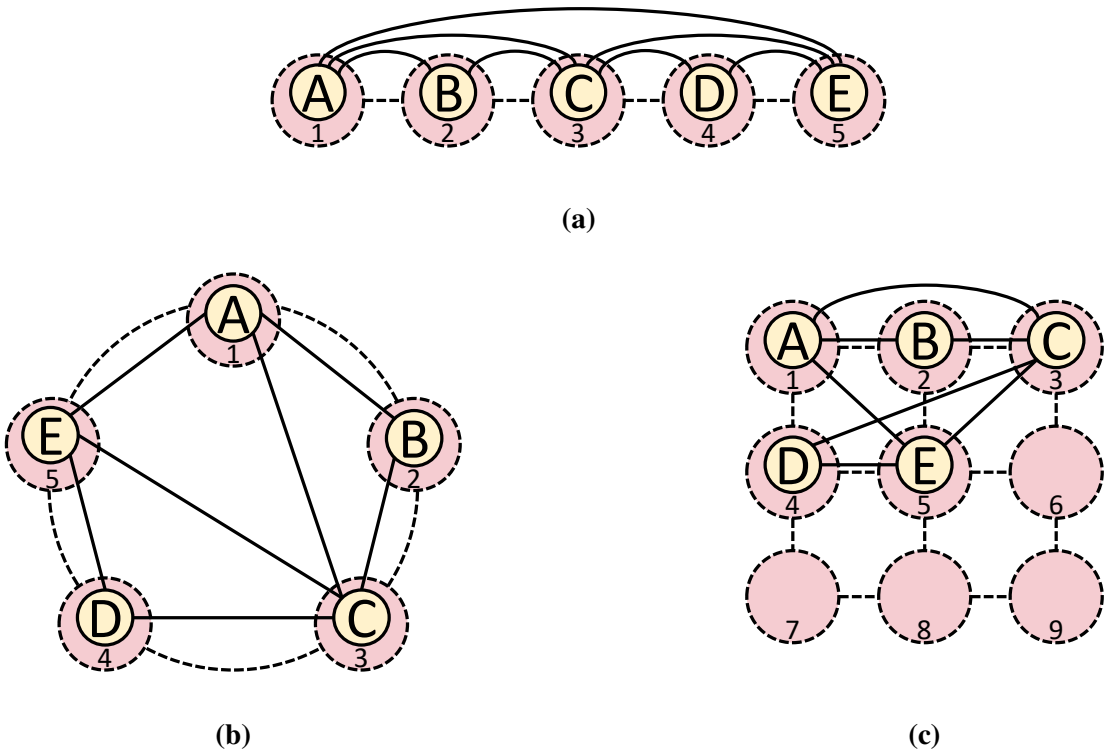


Figure 1.11 Example of an embedding in a path host graph (a), a cycle host graph (b), and a grid host graph (c).

(u,v)	$(\varphi(u),\varphi(v))$	$\psi_P(\varphi(u),\varphi(v))$	$\psi_C(\varphi(u),\varphi(v))$	$\psi_G(\varphi(u),\varphi(v))$
(A,B)	(1,2)	{(1,2)}	{(1,2)}	{(1,2)}
(A,C)	(1,3)	{(1,2),(2,3)}	{(1,2),(2,3)}	{(1,2),(2,3)}
(A,E)	(1,5)	{(1,2),(2,3),(3,4),(4,5)}	{(5,1)}	{{(1,2),(2,5)}, {(1,4),(4,5)}}
(B,C)	(2,3)	{(2,3)}	{(2,3)}	{(2,3)}
(C,D)	(3,4)	{(3,4)}	{(3,4)}	{{(3,6),(6,5)(4,5)}, {(3,2),(2,5),(5,4)}, {(3,2),(2,1),(1,4)}}
(C,E)	(3,5)	{(3,4),(4,5)}	{(3,4),(4,5)}	{{(3,6),(6,5)}, {(3,2),(2,5)}}
(D,E)	(4,5)	{(4,5)}	{(4,5)}	{(4,5)}

Table 1.1 Example of the definition of ψ in different host graphs: ψ_P for the path, ψ_C for the cycle and ψ_G for the grid.

Problems defined over a path host graph, are probably the most studied subfamily within the context of GLPs, followed by cycles and grids. However, other general structures are also preferred as host graphs, such as: trees, hypercubes, or torus, among others. Next, the most relevant host graphs are formally defined.

A path graph, denoted as P_n, is defined in [21, 53] as a graph with n vertices, that can be listed in order $v_1, v_2, \ldots, v_n$ such that the edges are (v_i, v_{i+1}) where $i = 1,2,\ldots,n-1$. The path graph is a tree with two nodes with degree 1, and the other $n-2$ nodes (if exist) of degree 2. A path graph is usually drawn so that all its vertices and edges lie on a single straight line [95]. In the context of GLP, a path host graph, H, for an input graph G, satisfies $|V_G| = |V_H| = n$. Figure 1.10(b) illustrates a path graph with $n = 5$.

A cycle graph, denoted as C_n, is defined as a path graph where the edge set contains the edge (v_n, v_1). Therefore, the number of vertices of C_n is equal to the number of edges, and each vertex has degree 2; that is, every vertex has exactly two edges incident with it [53]. Similarly to the path host graph, a cycle host graph has the same number of vertices as the input graph. Figure 1.10(c) depicts a cycle graph with $n = 5$. In this dissertation, the subfamily of GLPs whose embedding occurs in a cycle will be named as Circular Graph Layout Problem (CGLP).

A grid graph is defined as the Cartesian product $P_n \times P_m$ of path graphs of n and m

vertices, respectively [2]. Grid graphs are a common type of lattice graph, whose drawing in an Euclidean space $\mathbb{R}^2$ forms a regular tiling. These graphs can be represented as a 2-tuple (i,j) that locates a point in the plane. Considering the GLPs context, given an input graph G, the grid host graph satisfies $|V_H| = \lceil\sqrt{n}\rceil \cdot \lceil\sqrt{m}\rceil$ and therefore, $1 \leq i \leq \lceil\sqrt{n}\rceil$ and $1 \leq j \leq \lceil\sqrt{m}\rceil$. An example of grid host graph for an input graph with $n = m = 5$ is depicted in Figure 1.10(d). Even though the grids may not have an equal number of rows and columns, the problem tackled in this Doctoral Thesis restricts the host graph to have an equal number of vertices in both dimensions as the one presented in Figure 1.10(d).

The second way to classify GLPs is according to the optimized objective function. The most relevant objective functions involve the "bandwidth" calculation. The bandwidth is defined as the length (or cardinality) of a path assigned to an edge of the input graph. Based on the bandwidth, the following objective functions are highlighted: maximum bandwidth, minimum bandwidth, or the sum of the bandwidth (also known as minimum linear arrangement), among others. In the literature, other objective functions can be found on the basis of more complex calculations, such as the "cutwidth" or "edge bisection". The maximum "cutwidth" is computed as the maximum number of edges that traverse the space between every of two consecutive vertices of the arrangement. The edge bisection is calculated as the number of edges that link disjoint sets of vertices of the same size, obtained as a partition of the host graph. An exhaustive study on GLPs objective functions can be found in [65].

A particular GLP depends mainly on the objective function to be optimized and the host graph on which the embedding is to be performed. The combination of these two elements has provided a wide range of problems denoted as the GLPs family.

1.2.2 Literature review

In the literature, it can be found a wide variety of articles studying GLPs from different perspectives. The earliest works related to the GLPs date back to the end of the 20th century. At the moment when this dissertation was written, about 45,000 related publications were collected. To illustrate the number of publications, Figure 1.12 shows a graph with the evolution of the number of publications over the years and the cumulative percentage of

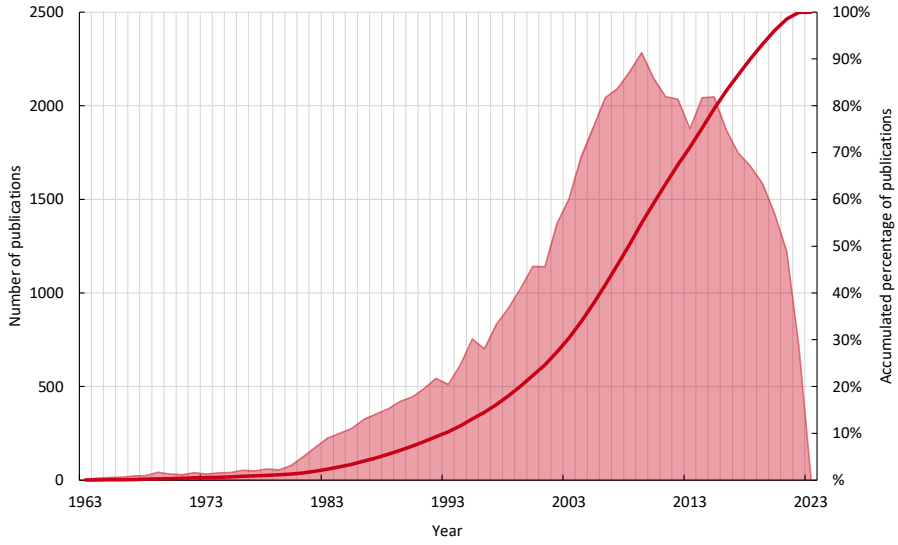


Figure 1.12 Evolution of the number of publications[3] related to the GLPs.

publications.

Most of the papers address GLPs from a theoretical point of view: studying their computational complexity, proposing equations to determine the value of the optimal embedding, determining lower and upper bounds, or finding relationships between problems, etc. This type of work generally focuses not only on an objective function and a particular host graph but also on a particular input graph, which usually has a well-known topology. On the other hand, researchers also study these problems from a more general perspective proposing exact and approximate algorithms for any type of input graph. Table 1.2 shows a summary of contributions organized by contribution typology and the objective function being optimized for the two most studied host graphs, the path, and the cycle. Specifically, the citations colored in green are theoretical publications, the ones in yellow are exact approaches, and the ones in red correspond to heuristic approaches.

The combination of an objective function with a host graph determines a particular GLP. The maximization of the bandwidth on the path host graph is known as Antibandwidth

[3]Data is collected from Web of Science and the query: `https://www.webofscience.com/wos/woscc/summary/ebb5950b-66dd-4aac-a45f-97854680b05b-62b12cce/relevance/1`.

Minimization Problem (ABP), while for the cycle host graph it is denoted as Cyclic Antibandwidth Problem (CAB). Similarly, the problem that minimizes the bandwidth for the path host graph is known as Bandwidth Minimization Problem (BMP), and for the cycle host graph, the Cyclic Bandwidth Minimization Problem (CBMP). The minimization of the sum of the bandwidth is denoted as the Minimum Linear Arrangement Problem (MinLA) for the path and the Cyclic Bandwidth Sum Problem (CBS) for the cycle. Finally, the minimization of the cutwidth is denoted as Cutwidth Minimization Problem (CMP), and for the cycle as Cyclic Cutwidth Minimization Problem (CCMP).

	Host graph	
Objective Function	**Path**	**Cycle**
Max. Bandwidth	[3] [146] [172] [237] [249] [11] [58] [156] [204]	[54] [146] [172] [199] [237] [12] [31] [157]
Min. Bandwidth	[105] [119] [127] [146] [200] [264] [105] [165] [148] [176] [135]	[52] [119] [127] [141] [151] [152] [264] [213] [201] [202] [210]
Min. Sum. Bandwidth	[103] [83] [65] [82] [4] [26] [48] [96] [168] [206] [13] [136] [194] [205]	[259] [37] [126] [221] [98] [209] [208] [33]
Min. Cutwidth	[36] [106] [128] [212] [241] [154] [158] [167] [185] [44] [59] [162] [186] [192] [49]	[1] [8] [27] [43] [68] [125] [128] [203] [223] [222] [224] [34] [28] [110] [124]

Table 1.2 Classification and review of the state-of-the-art of some GLPs organized by publication type. Specifically, the citations colored in green are theoretical publications, the ones in yellow are exact approaches, and the ones in red correspond to heuristic approaches.

This Doctoral Thesis focuses on GLPs where the host graph is a cycle. However, an additional problem has been studied where the host graph is a grid. GLPs with grid host

graphs have not been studied as deeply as those with paths or cycles host graphs. For that reason, they have not been included in Table 1.2. However, it is worthwhile to collect the most relevant works related to this family of problems. Specifically, there are several theoretical papers [42, 149, 150], one paper proposing exact algorithms based on Constraint Satisfaction Problem (CSP) [211] and two heuristic approaches [32, 211].

Among the theoretical papers collected in this section, it is worth highlighting those that aim to demonstrate the complexity of the problems. Originally, L.J. Stockmeyer, in a private communication to Michael R. Garey and D.S. Johnson, made a reduction of the CMP to the Simple Max-Cut problem [93, 133] in order to demonstrate that the CMP is NP-complete [82, 83]. However, this contribution was never published, although this result was reported in 1977 by F. Gavril [84]. Further studies of the computational complexity of the CMP led to the demonstration that it remains NP-complete for some specific input graph [64, 159, 177]. Once there was a demonstration of a GLP as belonging to the NP-complete, other researchers demonstrated the membership of similar problems, where the embedding is performed into a path graph, to that same complexity class [187].

As it can be observed, the complexity of problems whose embedding is performed on path host graphs has been extensively studied. However, this type of work is barely found when moving to other GLPs, where the host graph is not a path. Generally, authors address this issue just by relating the complexity of a particular problem to another problem in which complexity class is known. However, it is possible to find studies for specific graphs.

Before concluding this section, a crucial aspect of the experimentation process, especially when analyzing and comparing the algorithms, must be pointed out. To evaluate the performance of the proposed algorithms, the researchers test them with different types of instances, which, in the context of the GLPs, correspond to the input graph to be embedded. The instances that can be found in the literature for these problems can be classified into three groups. The first set consists of graphs with a specific topological structure, such as the Cartesian product of several graphs, hypercube graphs, grids, or cones, among others. The second type includes graphs coming from different scientific disciplines or inspired by real-world applications. In particular, the Harwell-Boeing collection [61] is of great interest in this group, as it has been widely used for many other problems modeled with graphs. Finally, the last category encompasses all graphs that have been generated synthetically or

artificially, generally at random, or following a certain probability distribution [157, 164]. All the sets of instances used in the comparisons to test the proposed algorithms are collected in each of the articles that comprise this dissertation and summarized in Section 4.1.

1.2.3 Historical perspectives and applications

Algorithms applied to GLPs have a wide range of applications in various fields, including computer science, engineering, and social sciences. This section provides several additional applications for GLPs.

As stated previously in this chapter, the first known application and origin of the GLPs, is the design of VLSI circuits. VLSI circuit can be modeled using a graph, where the edges represent the wires, and the vertices represent the modules [191]. In this sense, GLPs are used to determine the placement of transistors on a chip to minimize the length of the wires that connect them. This can help to reduce not only the amount of time it takes for signals to travel between transistors, which can improve the performance of the chip but also the production cost.

Once graph embedding problems became established in the scientific community as a family of problems with both theoretical and applied interest, researchers proposed their application to a diverse set of areas. Some of these applications, which are presented below, were previously collected in [65].

Visualizations and graph drawing

GLPs are commonly used to design the visualization of networks. Since networks or graphs can model data from a wide variety of disciplines, GLPs emerge as a methodology that could address their visualization. The following is a list of some possible disciplines in which the visualization of networks by means of GLPs may be of greatest interest.

- Biological networks, such as networks of proteins or genes. These visualizations can help researchers understand the relationships between different biological entities and how they interact with each other [35, 137, 263].

- Communication networks, such as telephone or Internet connection networks. These visualizations can help network engineers to analyze the structure and properties of the network and identify potential problems or areas for optimization. These visualizations can also help to identify patterns and relationships within the data [240].

- Computer-aided design to visualize complex systems and to help designers and engineers to interpret the structure and properties of the systems they are working with [181].

- Visualizations of transportation networks, like roads or air routes, can be useful for planners and engineers. They can show how the network is organized and what its features are. They can also help find possible problems or areas that need improvement [50, 262].

- Social networks, such as networks of friends on social media platforms. These visualizations can help to identify patterns and relationships within the network, such as the formation of communities or the identification of key influencers. In particular, one potential application of a circular graph layout is the representation of a group of friends who are connected to each other through mutual acquaintances. The circular layout can help to show the relationships between the different members of the group and how they are connected to each other [19, 73, 182, 242].

Generally, to deal with these visualizations, graph-drawing systems are commonly used [78, 129]. An automated graph-drawing system is a computer program that takes a graph as input and produces a visual representation of it, typically in the form of a diagram. The goal of an automated graph-drawing system is to create a clear and aesthetically pleasing visualization of the graph that makes it easy for a human viewer to understand the relationships between the vertices and the edges [169, 197].

Automatic graph-drawing systems use various algorithms to determine the position of the vertices and the route of the edges in the diagram. Among them, one of the most relevant is the Sugiyama framework [236]. Some of the most common objectives of these algorithms are to minimize the number of edge crossings, avoid edge overlaps, and balance the distribution of the vertices in the diagram [89]. Today, many graph-drawing systems

have been proposed, such as Graphviz [79], yEd [250], the Open Graph Drawing Framework (OGDF) [40] or Tulip [9], among others.

Numerical analysis

Traditionally, GLPs have been used in the context of numerical analysis where researchers often work with large matrices that have a high number of zero entries, known as sparse matrices. Existing algorithms are inefficient when applied to very sparse matrices. To improve the efficiency of these algorithms, it is typically desirable to reorder the rows and columns of the matrix in such a way that the non-zero entries of the matrix are as close as possible to the diagonal of the matrix. It is in this need for reorganization of rows and columns where GLPs emerge. In particular, GLPs are used to model the data as a graph and then to find the best arrangement of the vertices and edges of the graph. By minimizing the distance between non-zero entries and the diagonal, it can improve the performance of algorithms that rely on the sparsity of the matrix. Some references about these applications can be found in [14, 86, 217, 228].

Data analysis

GLPs could be used as a framework and support system in the area of data analysis to identify patterns and relationships within a dataset. This application could be related to other fields like data mining, market research, or network analysis, among others.

For example, suppose a large dataset of customer transactions from an online retail store. The objective is to identify patterns and trends in the data that could improve the store's marketing and sales strategies. One way to do this is to model the data with a graph where each product is represented by a vertex and each transaction is represented by an edge, connecting those products that a customer has purchased in the same transaction. Therefore, those products that have been purchased together on numerous occasions will be highly connected. Then, a GLP can be defined just by using the network of products as the input graph of the problem and the host graph could be any simple structure such as a path, grid, or cycle. Then, the objective will be to minimize the bandwidth, i.e., the distance between each product on the selected host graph. Finally, the layout generated

could be used to identify groups of products whose customers are in the habit of buying them together, which can help us target our marketing efforts more effectively [104, 144, 145].

Facility planning

Facility planning can be found in many contexts, such as healthcare, education, urban planning, manufacturing process, logistics, or retail, among others. In healthcare, for example, facility planning involves designing hospitals and clinics that are functional, efficient, and safe for patients and medical staff. In logistics, it involves designing warehouses, distribution centers, and other facilities that support the movement and storage of goods. In retail, it involves designing stores that are attractive and inviting to customers, while also being functional and efficient for employees. However, the most relevant applications of GLPs in facility planning are focused on the manufacturing process.

The manufacturing process is usually related to a family of problems referred to as the physical organization of a production system. The best layout is often the one that results in the greatest overall efficiency of transactions or flows between facilities. Costs can also be an important factor in choosing an alternative distribution plan for your application. Some models are based on an analysis of the relative location of facilities. In this sense, Facility Layout Problems (FLPs) can be understood as a particular family of problems within GLPs, where the facilities and their relationships can be modeled as an input graph, and the desired layout is the host graph [108, 170].

FLPs have been studied as much as the GLPs. Indeed, there is a clear relation between layouts and host graphs. For example, regular layout configurations are preferred over irregular ones. In this sense, the most relevant ones are linear single-row layouts, linear double-row layouts, semicircular layouts, and closed-loop layouts. Among the above layouts, the loop layout was found to be more attractive due to their relatively low initial costs and high flexibility in material handling [38, 147, 219]

Overall, algorithms applied to GLPs play a key role in many fields, and they are an essential tool for understanding and analyzing complex systems and data.

1.3 Hypothesis and objectives

In all research projects, the objectives must be determined prior to their implementation. Generally, these objectives are based on the need to address or solve a given problem and they are aimed to obtain an expected result, usually known as the starting hypothesis.

After a deep review of the state of the art of GLPs, we identified a family of problems whose graph embedding is performed on host graphs with a regular cycle-like structure. These problems, which are presented in depth in Chapter 2, are of great interest from both academic and applied points of view. In addition, we consider the possibility of improving existing approaches by finding better solutions in a reduced computational time.

Since GLP are classified as NP-hard problems, that is, there is no exact method able to solve them in a reasonable amount of time, we believe that if we apply new heuristic methods to solve these problems, they could be solved more efficiently compared to previous proposals in the state of the art.

Based on these ideas, the hypothesis (Section 1.3.1) and the objectives (Section 1.3.2) are elaborated.

1.3.1 Hypothesis

The hypothesis presented in the following is considered the cornerstone of this research project since it serves as a guide. The hypothesis proposed for the development of this Doctoral Thesis can be summarized in the following terms:

> *Heuristic algorithms, used in conjunction with metaheuristic techniques, are methods capable of finding high quality, potentially near-optimal solutions to optimization problems modeled by graphs consisting of embedding a candidate graph in a host graph with circular structure.*

1.3.2 Objectives

Derived from the hypothesis previously introduced, the main objective of this Doctoral Thesis is stated as follows:

Design and implement heuristic algorithms, used in combination with metaheuristic techniques, to address optimization problems modeled by graphs consisting of embedding graphs in cycles.

To achieve the main objective, it is necessary to cover the following specific objectives for each of the problems tackled:

- **Review and analyze the current state of the art of each GLP**. This objective starts by studying the GLP family from a general perspective and then focuses on problems where the embedding is performed on a particular host graph. Later, an exhaustive study of the problem under consideration is carried out. As a result, it is expected to collect and document strategies, algorithms, and procedures used to tackle related problems. Finally, the sets of instances on which these algorithms have been tested have to be collected.

- **Obtain properties and structural characteristics of the problem**. Since heuristic procedures are generally problem-dependent, learning the characteristics of the problem can help to propose very advanced and efficient techniques.

- **Model the problem so that it can be approached computationally**. This objective involves the selection of the most appropriate programming language and paradigm.

- **Design and develop a heuristic algorithm to solve the problem**. For this purpose, heuristic and metaheuristic techniques will be used. In particular, we intend to use some algorithms or strategies discussed in Section 1.1.2. This objective includes identifying the most appropriate heuristic and metaheuristic algorithms for each problem. In addition, algorithms typically have certain parameters that must be adjusted.

- **Compare experimentally the proposed algorithm with the state-of-the-art algorithms**. The state-of-the-art algorithms identified will be executed to perform an exhaustive and fair comparison, over a previously used set of instances. The state-of-the-art algorithms will ideally be the ones originally implemented by the authors or, in the worst case, re-implemented.

- **Prepare a document that includes the work carried out and the conclusions of the results obtained**. In the final phase of the research, the Doctoral Thesis report will be written, describing the problems identified and addressed. For these problems, the state of the art will be collected, and both the contributions made and the results obtained will be documented.

- **Disseminate the results by publishing them in research forums such as journals, conferences, or workshops**. The results of the research work will be subject to a review process by independent institutions that will culminate with their publication in journals of national and international prestige in the optimization research area or in conferences.

1.3.3 Research methodology

The development of this Doctoral Thesis is based on the scientific method as a methodology to generate new knowledge. Specifically, this method is based on observation, measurement, experimentation, formulation, and modification of the hypothesis. Moreover, this method can be applied to any scientific area whose purpose is to provide new knowledge. In this case, the scientific method is used as a guide to propose heuristic and metaheuristic algorithms to address the combinatorial optimization problems introduced in previous sections. Figure 1.13 shows the research process proposed in this Doctoral Thesis.

The process begins with the problem statement (phase 1). This is followed by a study of the state of the art, that is, learning about the most recent advances related to the problem at hand (phase 2). In the same phase, the tasks of identifying and reproducing previous algorithms and obtaining the instances (also known as inputs or test cases) are highlighted, since they are fundamental for the correct development of the research.

The following five phases of the research process, detailed in Figure 1.13, are repeated iteratively (phases 3, 4, 5, 6, and 7). Starting with the formulation of a hypothesis (phase 3), a study of the problem is carried out to enable the extraction of characteristics and their modeling (phase 4). Then, an algorithm is proposed and implemented to address the problem concerned. Generally, the algorithms implemented will be heuristic algorithms, metaheuristic algorithms, or advanced strategies that increase the efficiency or robustness of

the proposed algorithm (phase 5). After the implementation of an algorithm, it is necessary to analyze its performance through a set of experiments (phase 6). These experiments are classified into preliminary experiments if the objective is to establish the best configuration of the parameters of the proposed algorithm; or competitive experiments, aimed to compare the proposed algorithm with state-of-the-art algorithms. Finally, the starting hypothesis is validated or refuted (phase 7). The most relevant results of the research conducted are disseminated through publications in journals, conferences, or workshops (phase 8).

Finally, it should be emphasized that each of the phases of the research process described in Figure 1.13 is related to one or more of the objectives, set out in the previous section. Therefore, the correct compliance to this scheme favors the achievement of the objectives set.

1.4 Structure of the document

To conclude the introductory chapter of this Doctoral Thesis, the structure of the document is described in summary form, including a brief description of each part and chapter that comprise it.

- **Part I** summarizes the research developed, including the introduction and formalization of the problems, the existing heuristics and metaheuristics generally used to tackle optimization problems, and the results of each problem, including the conclusions, obtained. Specifically, this part is divided into the following chapters.
  - **Chapter 1** introduces the concept of optimization and optimization problems and collects the most relevant methods to solve them. In addition, this chapter presents the GLPs, the most relevant related work, and the motivation to study them. This chapter concludes with the statement of the hypothesis and objectives of the Doctoral Thesis.
  - **Chapter 2** presents the GLPs addressed in this Doctoral Thesis. For each of the studied problems, the objective function is formalized and explained with an example. In addition, other variants are described.

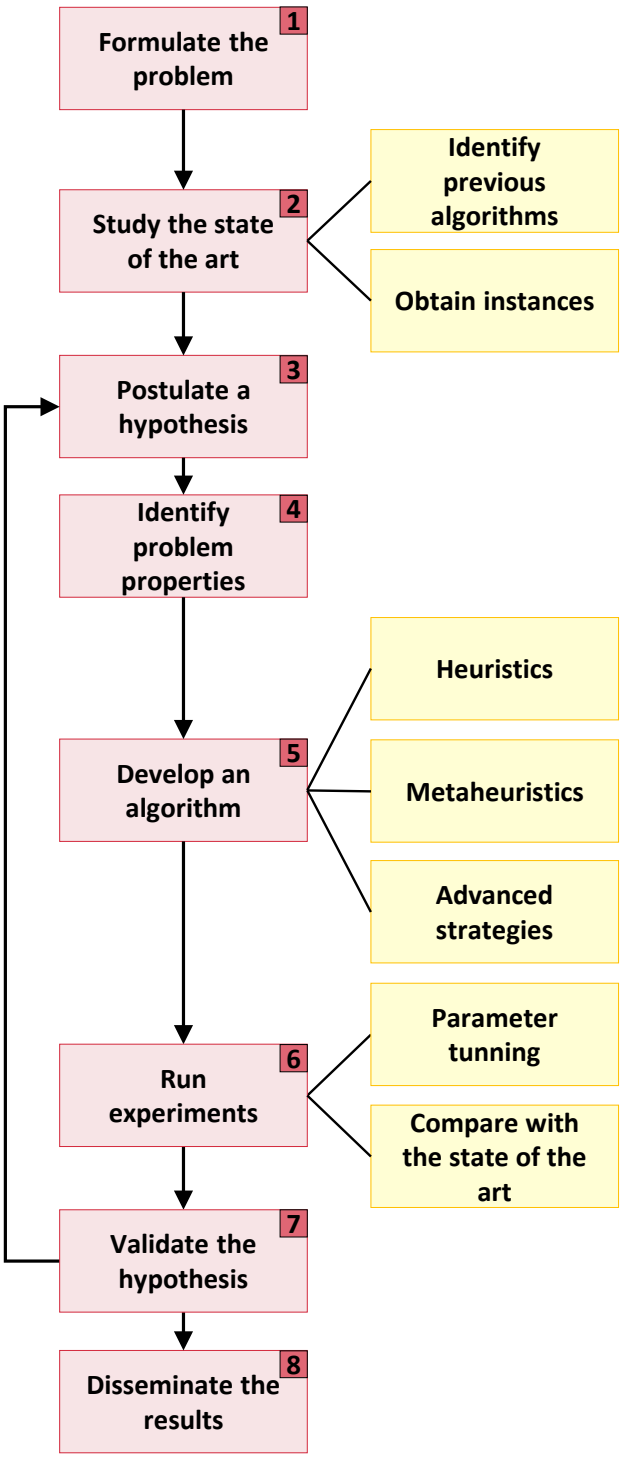


Figure 1.13 Representation of the adaptation of the scientific method to the context of this Doctoral Thesis.

  - **Chapter 3** describes the algorithmic proposal to tackle the considered problems. In particular, the main components of the methodology are classified into constructive procedures, improving procedures, metaheuristics, and advanced strategies.
  - **Chapter 4** is concerned with the results obtained by the algorithms proposed for each problem considered. In addition, each chapter section highlights the main contributions made.
  - **Chapter 5** ends with the conclusions derived from the research, as well as the main contributions, presenting possible lines of future work.

- **Part II** contains the publications, both journal and conference, associated with this Doctoral Thesis, including additional information about the journal in which it was published and a summary of the publication. This part is divided into the following chapters:

  - **Chapter 5** presents a summary of the publications obtained as the result of this Doctoral Thesis.
  - **Chapters 6–10** include published articles along with information on the journal in which they are published.
  - **Chapter 11** lists additional publications obtained during the Doctoral Thesis period, but not directly related to the main objective of the investigation.

- **Part III** corresponds to the appendix of this Doctoral Thesis. Specifically, it contains a summary in Spanish that includes background, objectives, methods, and results, and conclusions. In addition, it includes the bibliography and the glossary of keywords.

Chapter 2

Studied Graph Layout Problems

In this chapter, the problems addressed in this Doctoral Thesis are introduced. For each of the problems, the objective function, the goal of the problems, and an illustrative example of the evaluation of a solution are described. In particular, the studied problems are the Cyclic Cutwidth Minimization Problem (Section 2.1), the Cyclic Antibandwidth Problem (Section 2.2), the Cyclic Bandwidth Sum Problem (Section 2.3), and finally, the Two-Dimensional Bandwidth Minimization Problem (Section 2.4).

2.1 Cyclic Cutwidth Minimization Problem

The first problem studied in this Doctoral Thesis is the Cyclic Cutwidth Minimization Problem (CCMP). The CCMP consists of embedding general input graphs into a cycle host graph. To formally define the CCMP is necessary to consider again the assignment functions φ and ψ (see Equations 1.3 and 1.4 respectively, introduced in Section 1.2.1). Then, the objective function of this problem is based on the concept of a cut of an edge in the host graph (i.e., edges in E_H). The cut of an edge $(w, z) \in E_H$, also known as congestion [212], is defined as the number of paths, P_H, assigned by ψ that traverse (w, z). Formally, given φ and ψ, the cut of an edge $(w, z) \in E_H$ is defined as:

$$\text{cut}(\varphi, \psi, (w, z)) = |\{(u, v) \in E_G : (w, z) \in \psi(u, v)\}|. \tag{2.1}$$

Note that, for this problem, the function ψ does not need to assign the shortest path between vertices u and v but is a matter to be decided in the optimization process. However, for the sake of simplicity, we use in this research the one that assigns the shortest path (see Equation 1.5).

Then, the objective function, denoted as *ccw* or cyclic cutwidth, is calculated as the maximum cut for all edges $(w,z) \in E_H$. In mathematical terms:

$$ccw(G,\varphi,\psi) = \max_{(w,z) \in E_H} cut(\varphi,\psi,(w,z)). \tag{2.2}$$

Finally, this min-max optimization problem consists of finding the assignment $(\varphi^\star,\psi^\star)$, among all possible assignments $(\varphi,\psi) \in \Phi$, that minimizes the cyclic cutwidth:

$$(\varphi^\star,\psi^\star) \leftarrow \operatorname*{arg\,min}_{(\varphi,\psi) \in \Phi} ccw(G,\varphi,\psi). \tag{2.3}$$

Figure 2.1 depicts the evaluation of the solution previously presented in Figure 1.11(b) of the input graph illustrated in Figure 1.10(a). The evaluation of a solution for the CCMP, implies the calculation of the cut associated with each edge of the host graph. For example, to compute the cut of the edge $(1,2)$ first it is necessary to obtain those paths associated with the edges of the input graph that contain the edge $(1,2)$. In this example, there are two paths that meet this condition: $\psi_C(\mathrm{A},\mathrm{B}) = \{(1,2)\}$ and $\psi_C(\mathrm{A},\mathrm{C}) = \{(1,2),(2,3)\}$ (see Table 1.1). Figure 2.1(a) highlights the edge (A,B) and its associated path in yellow. Similarly, the edge (A,C) has been highlighted in green. Therefore, $\mathrm{cut}((1,2),\varphi,\psi_C) = |\{(\mathrm{A},\mathrm{B}),(\mathrm{A},\mathrm{C})\}| = 2$. Figure 2.1(b) illustrates the value of the cut for each edge of the host graph. Finally, the value of the objective function is $ccw(G,\varphi,\psi_C) = \max\{2,2,2,2,1\} = 2$.

The Cyclic Cutwidth Minimization Problem is one of those min-max optimization problems where the objective function consists of minimizing a maximum value. This kind of problem usually presents flat landscapes or fitness landscapes [59, 192, 207]. This concept was previously introduced in Section 1.1.2 and means that many solutions are qualified with the same value of the objective function, although they are structurally different. For example, the solution $ccw(G,\varphi,\psi_C) = \max\{2,2,2,2,1\} = 2$, will be equivalent to any other solution whose objective function value is 2 such as $ccw(G,\varphi',\psi'_C) = \max\{1,1,2,1,1\} = 2$. However, as the reader may have noted, the solution (φ',ψ'_C) seems

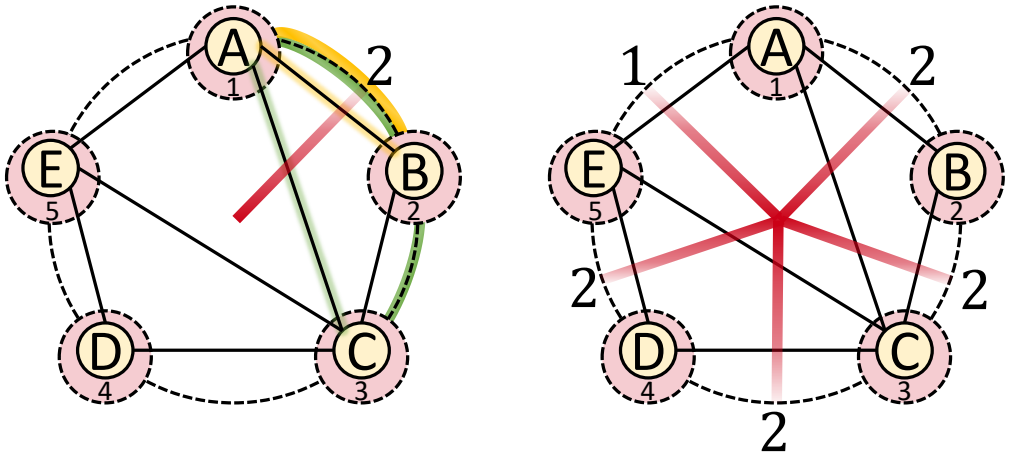


Figure 2.1 (a) Evaluation of the cut of the edge $(1,2) \in E_H$. (b) Evaluation of all cuts in a solution for the CCMP resulting in an objective function value of 2.

to be better than (φ, ψ_C) since it closer to reduce the value of the objective function.

In this sense, determining search directions becomes a very difficult task when decisions are based only on the change of the objective function value produced by a move. Finding a meaningful way of differentiating solutions with the same objective function value is important because the structure of one solution may be more promising than the structure of another one in terms of a later improvement in the search.

2.2 Cyclic Antibandwidth Problem

The second problem tackled within this Doctoral Thesis is the Cyclic Antibandwidth Problem (CAB), which aim is to embed the input graph in a cycle host graph maximizing the minimum distance of all adjacent vertices, i.e., the bandwidth.

Again, to formally define the CAB it is necessary to consider the functions φ and ψ. Then, the objective function of this problem is based on the concept of the bandwidth of an edge in the input graph. The bandwidth of an edge $(u,v) \in E_G$ is defined as the cardinality of the path $p \in P_H$, assigned by ψ to the edge (u,v). Formally, given φ and ψ, the bandwidth of edge $(u,v) \in E_G$ is defined as:

$$bw(\varphi, \psi, (u, v)) = |\psi(u, v)|. \tag{2.4}$$

Then, the objective function, denoted as *cab* or cyclic antibandwidth, is calculated as the maximum bandwidth for all edges $(u, v) \in E_G$. In mathematical terms:

$$cab(G, \varphi, \psi) = \min_{(u,v) \in E_G} bw(\varphi, \psi, (u, v)). \tag{2.5}$$

Finally, this max-min optimization problem consists of finding the assignment $(\varphi^\star, \psi^\star)$, among all possible assignments $(\varphi, \psi) \in \Phi$, that minimizes the cyclic antibandwidth:

$$(\varphi^\star, \psi^\star) \leftarrow \arg\max_{(\varphi,\psi) \in \Phi} cab(G, \varphi, \psi). \tag{2.6}$$

Based on the solution previously presented in Figure 1.11(b) of the input graph illustrated in Figure 1.10(a), we illustrate now in Figure 2.2 the evaluation of the objective function of this problem with an example. In particular, in order to compute the bandwidth of an edge, the cardinality of it associated path is calculated. For example, the bandwidth of the edge (A,C) is computed as $|\psi_C(\mathrm{A},\mathrm{C})| = |\{(1,2),(2,3)\}| = 2$. Both, the edge and its assigned path are highlighted in green in Figure 2.2(a). Similarly, the bandwidth of the edge $|\psi_C(\mathrm{D},\mathrm{E})| = |\{(4,5)\}| = 1$, highlighted with yellow in the same figure. Figure 2.2(b) illustrates the value of the bandwidth for each edge of the host graph. Finally, the value of the objective function is the minimum bandwidth across all edges, which is 1 in this example. In mathematical terms: $cab(G, \varphi, \psi_C) = \min\{1,2,1,1,1,2,1\} = 1$.

2.3 Cyclic Bandwidth Sum Problem

The third problem studied within the GLP family is the Cyclic Bandwidth Sum Problem (CBS), which consists of minimizing the sum of the bandwidth, that is, the distance, of the edges of an input graph computed in the cycle host graph.

In order to compute the objective function of a solution to the CBS, the calculation of the bandwidth for each edge of the input graph is needed. In this case, the bandwidth is computed as defined for the CAB in Equation 2.4. Then, the evaluation of the objective

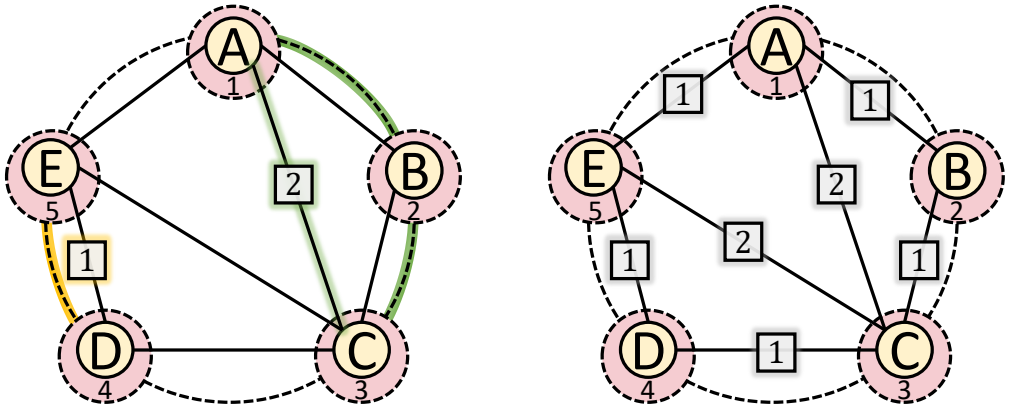


Figure 2.2 (a) Evaluation of the bandwidth of edges $(\mathrm{A},\mathrm{C}),(\mathrm{D},\mathrm{E}) \in E_G$. (b) Evaluation of all edges of the input graph in the cycle host graph.

function of the CBS for a particular embedding (φ, ψ) of the input graph G, denoted as *cbs*, is calculated as the sum of the bandwidth of each edge of the input graph. More formally:

$$cbs(G, \varphi, \psi) = \sum_{(u,v) \in E_G} bw((u,v), \varphi, \psi) \tag{2.7}$$

Finally, the objective of CBS is to find an embedding $(\varphi^\star, \psi^\star)$ among all possible embeddings, $(\varphi, \psi) \in \Phi$, that minimizes the Equation (2.7). In mathematical terms:

$$(\varphi^\star, \psi^\star) \leftarrow \underset{(\varphi,\psi) \in \Phi}{\arg\min}\ cbs(G, \varphi, \psi) \tag{2.8}$$

Based on the solution presented in Figure 2.2 for the CAB that shows the evaluation of the bandwidth for each edge of the input graph, we now illustrate the calculation of the objective function for the CBS. Specifically, to compute it, the bandwidth value associated with each edge is summed. More formally: $cbs(G, \varphi, \psi_C) = 1+2+1+1+1+2+1 = 9$.

2.4 Two-Dimensional Bandwidth Minimization Problem

The main objective of this doctoral thesis focuses on the proposal of heuristic and metaheuristic algorithms for CGLP. However, after successfully addressing three problems belonging to the latter family, it was decided to apply the knowledge learned to other GLPs defined over a different host graph emerging a new additional hypothesis during the last year of the Doctoral Thesis:

> *The heuristic techniques proposed for CGLP are also useful to find quality solutions for other GLPs in which the embedding is performed in another type of host graph.*

Among the possible GLPs existing in the literature, it is decided to work on the Two-Dimensional Bandwidth Minimization Problem (2DBMP). The Two-Dimensional Bandwidth Minimization Problem (2DBMP) consists of minimizing the bandwidth of the input graph when embedding it into a grid host graph.

In the related literature, the most relevant bandwidth function for grids is denoted L_1-norm [149, 211], which is also known as Taxicab norm distance or Manhattan distance [46, 149]. To compute the L_1-norm distance or the bandwidth of an edge of the input graph $(u, v) \in E_G$, the cardinality of the path assigned to (u, v) is calculated. To do so, ψ is used alike for the CAB and the CBS (see Equation 2.4).

Then, the objective function, denoted as *2dbmp*, is calculated as the maximum bandwidth for all edges $(u, v) \in E_G$. In mathematical terms:

$$2dbmp(G, \varphi, \psi) = \max_{(u,v) \in E_G} bw((u, v), \varphi, \psi) \tag{2.9}$$

Finally, the 2DBMP for a graph G consists of finding an embedding $(\varphi^\star, \psi^\star_G)$ among all possible embeddings of the problem that minimizes Equation 2.9. More formally:

$$(\varphi^\star, \psi^\star) \leftarrow \underset{(\varphi^\star, \psi^\star) \in \Phi}{\arg\min} \; 2dbmp(G, \varphi, \psi) \tag{2.10}$$

Figure 2.3 depicts the evaluation of the solution previously presented in Figure 1.11(c) of the input graph illustrated in Figure 1.10(a). As it can be observed, all vertices of V_G

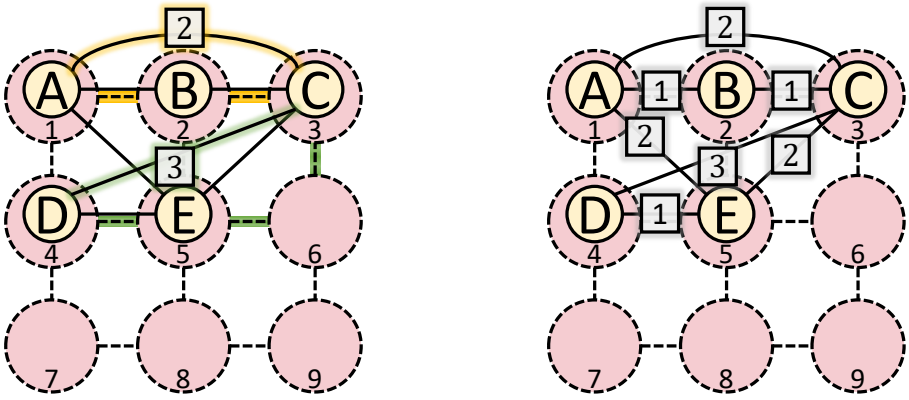


Figure 2.3 (a) Evaluation of the bandwidth of edges $(\mathrm{A},\mathrm{C}),(\mathrm{C},\mathrm{D}) \in E_G$. (b) Evaluation of all edges of the input graph in the grid host graph.

have been assigned to a vertex of V_H through the definition of φ. However, unlike the CGLP previously presented, there are vertices of the host graph that have not been assigned to any vertex of the input graph.

In order to evaluate the objective function of the example described in Figure 2.3, it is required to calculate the distance between each pair of adjacent vertices in V_G (i.e., for each edge of E_G) by using Equation 2.4. For instance, considering edge (C,D), three possible paths can be assigned to it (see Table 1.1). Since the choice does not influence the bandwidth calculation, one of them is selected for this example. In particular, the path $\psi_G(\mathrm{C},\mathrm{D}) = \{(3,6),(6,5),(5,4)\}$ has been selected and is highlighted in Figure 2.3(a). Therefore, $bw(\varphi,\psi_G,(\mathrm{C},\mathrm{D})) = 3$. Similarly, the bandwidth of the edge (A,C), highlighted in yellow in the same figure is $bw(\varphi,\psi_G,(\mathrm{A},\mathrm{C})) = |\psi_G(\mathrm{A},\mathrm{C})| = |\{(1,2),(1,3)\}| = 2$. This calculation is performed over the rest of the edges of G. The obtained distances are depicted in Figure 2.3(b). Finally, the value of the objective function is the maximum across all distances, which is 3 in this example. In mathematical terms, $2dbmp(G,\varphi,\psi_G) = \max\{1,2,2,1,3,2,1\} = 3$.

Chapter 3

Algorithmic proposal

This Doctoral Thesis proposes various heuristic and metaheuristic procedures to solve the presented GLPs. The use of heuristics allows finding good solutions efficiently when exact methods are not able to find feasible solutions or require too much computational time. Heuristic algorithms are based on knowledge and experience and are often used to quickly find good solutions to complex problems. Constructive procedures are responsible for generating solutions from scratch, while improvement procedures start with an initial solution and iteratively improve it by making small changes. Metaheuristics are higher-level heuristic strategies that guide the use of other heuristics to solve a problem.

This chapter describes the heuristic and metaheuristic procedures proposed in this Doctoral Thesis. Section 3.1 summarizes the constructive procedures proposed, while Section 3.2 describes the improving methods based on defining neighborhoods. Section 3.3 presents the metaheuristic approaches that combine the proposed heuristics. Finally, Section 3.4 discusses some advanced strategies that complement the improving methods. The proposed procedures aim to efficiently and effectively solve the presented GLPs.

3.1 Constructive procedures

Constructive procedures are heuristic methods used for the generation of solutions. Generally, constructive procedures start from an empty initial solution and end with a complete solution, i.e., a feasible solution (see Section 1.1.2).

In this section, all the issues to be taken into account when proposing a constructive for a GLP are presented. In addition, the constructive procedures proposed for each of the problems studied are collected.

First, the construction of a solution for a GLPs consists of defining the mathematical functions φ and ψ. In the four problems studied in this Doctoral Thesis (see Chapter 2), the function ψ always depends on the function φ since the path assigned to each of the edges of the input graph will always be the one with the smallest cardinality (see Equation 1.4). Therefore, for the sake of clarity, in the construction of a solution we focus just on defining the φ function.

It is worth remembering that the φ function assigns a vertex of the host graph to each vertex of the input graph, and this assignation has been denoted as an embedding or a projection. Any definition of the φ function that satisfies that each vertex of the input graph is assigned to a vertex of the host graph, will result in a feasible solution to the problem.

In this dissertation, beyond proposing specific constructive algorithms for each of the problems studied, we also propose a general scheme or framework for the definition of any constructive algorithm for the graph layout family of problems. Specifically, the definition of a constructive algorithm for a GLP should consider the followings steps:

1. Generate a set of candidate vertices made of unassigned vertices of the input graph. This set will be formally denoted as CL_G.

2. Generate a set of candidate vertices made of vertices of the host graph. This set will be formally denoted as CL_H.

3. Select a vertex u of CL_G following specific criteria.

4. Select a vertex v of CL_H following specific criteria.

5. Assign the vertex of the host graph to the vertex of the input graph, $\varphi(u) = v$.

6. Update the sets CL_G and CL_H.

These steps are repeated until a feasible solution is reached, that is until the set CL_G is empty. Based on this scheme, the researcher's task will be to determine the way in which

each of the steps is carried out. For example, the creation of the candidate vertex lists (points 1 and 2) can be dynamic (a list of candidate vertices that is updated at each iteration) or static (a list initialized with all the vertices of the graph). However, the most determinant aspect of a construct that follows this scheme focuses on the selection of a vertex from both CL_G and CL_H. In this research, we have studied different strategies for the selection of the vertices of both sets. In general, these strategies are not complex and focus on the properties of the graph to be embedded.

The simplest constructive that can be proposed for any GLP following this scheme is a random constructive. In this case, all candidate lists are statically generated and at each iteration, one vertex is randomly selected from each set. Algorithm 1, shows a classical scheme of the implemented randomized constructive. Specifically, a constructive procedure for a GLP receives as input parameter the input and the host graph and returns the constructed solution. To do so, first, the procedure generates an empty solution (step 2), and the sets of candidate vertices CL_G (step 3) and CL_H (step 4) are initialized. Then, while CL_G is not empty, a vertex for both, CL_G and CL_G is selected at random (steps 6 and 7 respectively). Then, the assignation is performed (step 8), and sets are updated (steps 9 and 10).

Algorithm 1: Example of a random constructive procedure for a GLP

```
1  Procedure RandomConstructive(G(V_G, E_G), H(V_H, E_H)):
2      φ ← ∅ ;
3      CL_G ← V_G;
4      CL_H ← V_H;
5      while CL_G ≠ ∅ do
6          u ← random(V_G);
7          v ← random(V_H);
8          φ ← φ ∪ {φ(u) = v};
9          V_G = V_G \ {u};
10         V_H = V_H \ {v};
11     end
12 return φ
```

Despite the simplicity and the predictable poor quality of the solutions generated by the random constructive procedure, the definition of such a procedure is considered crucial for the development of this type of research for several reasons. The generated solutions are

used as a preliminary comparison framework or as an upper/lower bound in the absence of benchmark results obtained by a state-of-the-art algorithm. It is also used to validate that the implementation is correct and to detect possible errors in the early stages of algorithm coding, especially in the procedure which evaluates the objective function of the solutions. In addition, it is used as a starting point for other more advanced methods to analyze its performance, such as the improvement capability of the improvement procedure. An easy extension of this method would be adding a multistart strategy to this method which would result in an improvement in the quality of the best solution found after a few iterations.

There are other more complex and intelligent construction procedures known as greedy constructive procedures. These procedures start from an empty solution, and at each iteration, they add the best possible element to the solution according to a criterion. Generally, this criterion is a mathematical function that the candidate elements have to optimize.

Following the 6-point framework outlined above, Algorithm 2 compiles the pseudocode of a general greedy constructive procedure for a GLP. The main difference with the random construction (Algorithm 1) is in the selection of the vertices of the sets CL_G and CL_H (steps 6 and 7, respectively). Specifically, as an example, two greedy criteria g_1 and g_2 are posed. In each iteration, the vertices that maximize (analogously minimize) these functions, will be the ones chosen to perform the assignment $\varphi(u) = v$ (step 8). Note that the selection criteria do necessarily have to be mathematical functions to be optimized, as they may be based on other more simple or complex criteria.

A greedy criterion is a rule that selects the best candidate vertex according to the value of a function at each step of a constructive procedure. Depending on the criterion used, the evaluation of a candidate vertex could be very time-consuming, and the resultant solution might not even be good in terms of quality. Therefore, researchers generally propose criteria that can be calculated quickly and that bear some relation to the function to be optimized.

In this Doctoral Thesis, different criteria have been proposed for the selection of vertices of both sets. Next, criteria proposed for the selection of a vertex of the input graph are presented (see Section 3.1.1, Section 3.1.2 and Section 3.1.3)

Algorithm 2: Example of a greedy constructive procedure for a GLP

```
1  Procedure GreedyConstructive(G(V_G, E_G), H(V_H, E_H)):
2      φ ← ∅
3      CL_G ← V_G
4      CL_H ← V_H
5      while CL_G ≠ ∅ do
6          u ← arg max_{u' ∈ V_G} g_1(φ, u')
7          v ← arg max_{v' ∈ V_H} g_2(φ, u', v')
8          φ ← φ ∪ {φ(u) = v}
9          V_G = V_G \ {u}
10         V_H = V_H \ {v}
11     end
12 return φ
```

3.1.1 Criteria for selecting a vertex of the input graph

The most used greedy criterion for the selection of a vertex from the input graph, in the context of this dissertation, is based on the idea of quantifying the urgency with which a vertex should be assigned based on the state of a partial solution. This idea was first proposed by A.J. McAllister in 1999 in the context of the MinLA [168] and it has been adapted to consider a cycle host graph in this research.

In particular, the greedy selection function to select the next vertex from the candidate graph is defined as follows. Let γ be a mathematical function such that, given a partial solution φ, associates a vertex of the input graph $u \in CL_G \subseteq V_G$ to a natural number. More formally, $\gamma : CL_G \rightarrow \mathbb{N}$. In order to compute the number associated with a vertex, two sets must be defined. Let $A(u)$ be the set of vertices adjacent to u that have already been assigned to a vertex of the hos graph, and let $U(u)$ be the set of vertices adjacent to u that still remains to be assigned, such that $deg(u) = |A(u)| + |U(u)|$. Both sets are mathematically formalized as follows:

$$A(u) = \{v \in V_G : (u, v) \in E_G \wedge v \notin CL_G\}, \tag{3.1}$$

$$U(u) = \{v \in V_G : (u, v) \in E_G \wedge v \in CL_G\}, \tag{3.2}$$

Then, we define the value of γ function for a vertex $u \in CL_G$ and a partial solution φ as:

$$\gamma(u) = |A(u)| - |U(u)|. \tag{3.3}$$

The rationale of the γ function relay on the "attractiveness" of a vertex to be selected next to perform the next assignment. In this sense, the proposed greedy function considers a vertex as "attractive" if all its adjacent vertices have already been assigned. Conversely, an unassigned candidate vertex is "unattractive" when none of its adjacent vertices have been assigned. Therefore, the vertex that maximizes Equation 3.3 is selected to be assigned next.

This greedy criterion was satisfactorily used to tackle the CCMP (see Chapter 7). However, in the following research in which the CBS and 2DBMP were studied, a weakness of the strategy was observed. To explain this deficiency, the following situation is presented as an example. Consider a vertex a where $|A(a)| = 1$ and $|U(a)| = 1$, the value of γ for u is $\gamma(a) = |A(a)| - |U(a)| = 1 - 1 = 0$. Similarly, consider a vertex b where $|A(b)| = 3$ and $|U(b)| = 4$, then, $\gamma(b) = |A(b)| - |U(b)| = 3 - 4 = -1$. Since $\gamma(a) > \gamma(b)$, vertex a will be chosen to be added next. Apparently, it seems reasonable to pick vertex a because it has more adjacent vertices in the solution than vertices yet to be assigned. However, vertex b has more vertices in the solution than a, and it also seems reasonable that it should be assigned more urgently.

In practice, there is no absolute or clear answer for determining which vertex should be considered more important or urgent. In fact, experimentation has led to the observation that it depends on the problem or even on the input graph considered. For that reason, in this Doctoral Thesis an adaptation of Equation 3.3 is proposed by adding two parameters, w_1 and w_2 to be adjusted by the researcher. The adaptation of the proposed greedy criterion, denoted as $\gamma_w(u)$, is detailed in the following equation:

$$\gamma_w(u) = w_1 \cdot |A(u)| - w_2 \cdot |U(u)|, \tag{3.4}$$

where $0 \leq w_1, w_2 \leq 1$ and $w_1 + w_2 = 1$. The rationale behind these two parameters is to balance the relevance of having many adjacent vertices assigned ($w_1 > w_2$) or a reduced number of adjacent vertices unassigned ($w_1 < w_2$). Notice that if $w_1 = w_2$, then the strategy

is equivalent to the original proposal introduced in Equation 3.3. Again, all unassigned vertices from the input graph are evaluated, and the vertex with the largest γ_w-value is chosen to be assigned next.

Finally, it should be noted that both greedy criteria proposed are adaptive since the value associated with each candidate vertex has to be updated at each iteration of the construction phase. This concept is closely connected with such relevant construction procedures as the one proposed in the GRASP methodology, which will be further explained in Section 3.1.4.

3.1.2 Criteria for selecting a set of vertices of the input graph

In this research, the greedy criterion proposed in the previous section has been used to generate the initial solutions of three problems, the CCMP, the CBS and the 2DBMP. As the reader may have noticed, these three problems share the property that, directly or indirectly, placing the adjacent vertices of the input graph as close as possible in the embedding leads to better-quality solutions. However, in the CAB the objective is just the opposite: adjacent vertices should be placed as far as possible from each other. It is for this reason that to address the CAB it is necessary to propose a constructive strategy based on a different criterion.

In particular, in [12, 64], a criterion was presented to detect several groups of vertices that are suitable to be jointly assigned to adjacent vertices of the host graph since this assignment will not impact negatively the objective function. Specifically, those groups of vertices are the ones that simply satisfy that they are not adjacent to each other. To detect those groups, a spanning tree of the input graph is generated using a Breadth First Search (BFS) algorithm [229].

The BFS algorithm starts at a given vertex (root) and explores all the adjacent vertices at the present depth before moving on to the vertices at the next depth level. This process is repeated until all vertices have been visited. The BFS, while exploring the vertices of the graph, organizes them into levels according to the depth at which they have been found. For example, the root vertex has a depth of 0, while those adjacent to it have a depth of 1, and so on, resulting in a spanning tree.

The spanning tree organizes the vertices in layers or levels that have the same unit

distance with respect to a root vertex. Then, for each level of the spanning tree, vertices are then assigned to consecutive vertices of the host graph. Once the vertices of the first level selected have been assigned, the next non-consecutive level of the spanning tree is selected, and so on.

The rationale for this strategy is to ensure that adjacent vertices of the input graph are not adjacent to each other in the embedding. However, vertices at the same level of the spanning tree can be adjacent among them, in that case, only those that are not adjacent to some already assigned vertex will be assigned.

In order to adequately apply this strategy, it is necessary to take some decisions. First, a vertex must be selected as the root node. Secondly, a strategy must be established to explore the levels and finally, another strategy must determine the order in which the vertices of the same level are assigned. In this research, we have not delved into these issues and the decisions have been taken at random (see Section 3.1.4). However, these tasks open new chances for future research.

3.1.3 Criteria for selecting a vertex of the host graph

Once the vertex of the input graph has been chosen, an available vertex from the host graph must be selected to embed it. In this research, we propose two different approaches: one based on graphical patterns and the other one based on a greedy criterion.

The first approach determines the order in which host vertices are selected on the basis of a graphical pattern. In the case of the cycle host graph, we studied two patterns denoted as “sequential pattern” (Figure 3.1(a)) and “alternating pattern” (Figure 3.1(b)). In the case of the grid host graph, three patterns are proposed: the “sequential pattern”(Figure 3.1(c)), the “diagonal pattern” (Figure 3.1(d)) and the “zigzag pattern” (Figure 3.1(e)). In each of the figures, the sequence is indicated with blue arrows and numbers inside each of the host vertices, being the number 1 the vertex selected in the first iteration, and numbers 5 and 9 the vertices selected in the last iteration for the cycle, and grid host graphs, respectively.

These strategies have been proposed and studied for the GLPs studied, however, their performance has been inefficient for all of them except for the CCMP. For the rest of the problems, the second approach, based on a greedy function, is used.

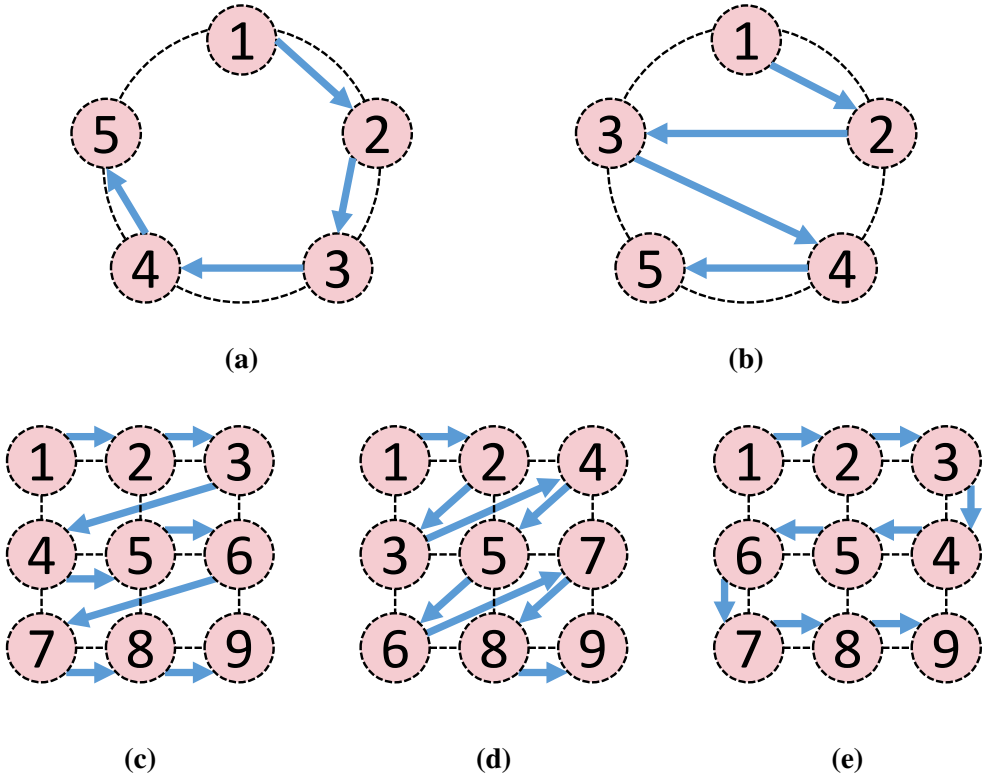


Figure 3.1 Example of the order followed to assign vertices of the input graph to vertices of the host graph using the sequential (a) and alternating (b) patterns in a cycle host graph, and the sequential (c), diagonal (d) and zigzag (e) patterns in a grid host graph.

The second proposed approach selects a vertex of the host graph based on greedy function denoted as λ. This function is based on the concept of "contribution to the objective function" of an assignation $\varphi(u) = v$. Therefore, λ is defined according to the particular objective function addressed (λ_1 for the CAB, λ_2 for the CBS, and λ_3 for the 2DBMP). For example, given a partial solution φ, one vertex of the input graph u and one vertex of the host graph v, the λ function for the CAB will be defined as follows:

$$\lambda_1(u, v) = \min_{w \in V_G \setminus CL_G} \{ \min_{p \in P_H} \{p(v, \varphi(w))\}\}. \tag{3.5}$$

In a simple way, this function calculates the minimum distance that a vertex u would be from all its adjacent vertices that are already part of the solution ($w \in V_G \setminus CL_G$) if it were assigned to the vertex of the host graph v. To calculate the minimum distance, in addition, it is necessary to calculate the shortest path between v and $\varphi(w)$. Therefore, the best possible candidate host vertex $v^\star$ for an input vertex u is:

$$v^\star = \arg\max_{v \in CL_H} \lambda_1(u, v). \tag{3.6}$$

Similarly, this greedy function is adapted for the CBS in Equations 3.7 and 3.8, and for the 2DBMP in Equations 3.9 and 3.10.

$$\lambda_2(u, v) = \sum_{w \in V_G \setminus CL_G} \{ \min_{p \in P_H} \{p(v, \varphi(w))\}\}. \tag{3.7}$$

$$v^\star = \arg\min_{v \in CL_H} \lambda_2(u, v). \tag{3.8}$$

$$\lambda_3(u, v) = \max_{w \in V_G \setminus CL_G} \{ \min_{p \in P_H} \{p(v, \varphi(w))\}\}. \tag{3.9}$$

$$v^\star = \arg\min_{v \in CL_H} \lambda_3(u, v). \tag{3.10}$$

Given the computation time required to calculate the proposed greedy criterion, especially when the input graph is large, a reduction of the set CL_H to a subset of these vertices, denoted as CL'_H is proposed. This set is adaptive, and it is updated at each iteration of the

constructive procedure by adding all host vertices adjacent to any other adjacent host vertex that has been previously assigned. Formally, the update of the set can be mathematically defined as follows:

$$CL'_H = \{w \in CL_H : (w, z) \in E_H \wedge z \notin CL_H\}. \tag{3.11}$$

This strategy has been successfully implemented for the CAB, CBS and the 2DBMP.

3.1.4 Randomization of the procedures

In general, any constructive procedure aims to generate solutions of the best possible quality, i.e., to generate the optimal solution to the problem or to get as close as possible to it so that other intensification methods can reach it quickly. However, applying the construction procedure repeatedly may sometimes be desirable to yield diverse starting solutions for other intensification procedures, such as the local search. In this sense, diversification could provide lower quality but more diverse solutions, allowing the exploration of more regions of the solution space. Strategies that encourage diversity are especially interesting when combined with procedures in which it is necessary to generate more than one solution, such as multi-start procedures or population-based algorithms.

In this Doctoral Thesis we have used the GRASP constructive extensively. GRASP is a multistart metaheuristic for producing good-quality solutions of combinatorial optimization problems. The probabilistic component that randomly chooses one of the best candidates from the Restricted Candidate List [71, 72], usually denoted as RCL, is the key characteristic of this procedure.

Inspired by these ideas, we have adapted the proposed greedy criterion to promote the diversity of the solutions generated. Specifically, the greedy criterion γ_w, used for the selection of an input graph vertex, is randomized by selecting the weights w_1 and w_2 each iteration (see Equation 3.4).

In addition, other randomized strategies are proposed to further promote the diversity of the solutions obtained, which are listed below:

- The selection of the first vertex of the input graph, necessary to initialize the candidate list CL_G, can be performed randomly. Similarly, the selection of the first vertex

of the host graph to initialize the candidate list CL_H, can be performed randomly.

- The selection of the root vertex for the generation of a spanning tree based on the BFS algorithm is performed randomly. In addition, the order in which the levels are traversed and the order in which each vertex of a level is added to the solution is also chosen randomly.

- Ties produced when evaluating any of the proposed criteria are broken at random.

3.2 Improving methods

One of the most relevant improving or intensification procedures is the local search procedure. This procedure has been used in all the investigations developed within this Doctoral Thesis.

The approach of a local search implies the definition of neighborhood structures and, therefore, of movement operators. In this research, two moves widely used in the context of combinatorial optimization when the solution is modeled with a graph are proposed: insertion and exchange (also known as swap).

The insert move consists of removing one vertex of the input graph from its current assignation and assigning it (i.e., inserting) into another vertex of the host graph. This move implies a "shift" of some vertices to "make room" to the vertex that is going to be inserted. An example of an insert move is illustrated in Figure 3.2. In particular, given the embedding depicted in Figure 3.2(a), we illustrate the insertion of vertex A of the input graph into vertex 4 of the host graph. Both vertices have been highlighted. Then, in Figure 3.2(b) we show the resultant solution after the operation. As it can be observed, the operation implies the shift of vertices D and E to make room for vertex A. The direction of the shift (clockwise or counterclockwise) should be decided experimentally.

Formally, the insert move is denoted as $\mathit{Insert}(\varphi, u, v)$, where φ is the current embedding in which vertex $u \in V_G$ is going to be assigned to vertex $v \in V_H$. Then, the associated neighborhood, denoted as $N_I(\varphi)$ is defined as the set of solutions that can be reached by applying the insert operator for each $u \in V_G$ and $v \in V_H$. Mathematically:

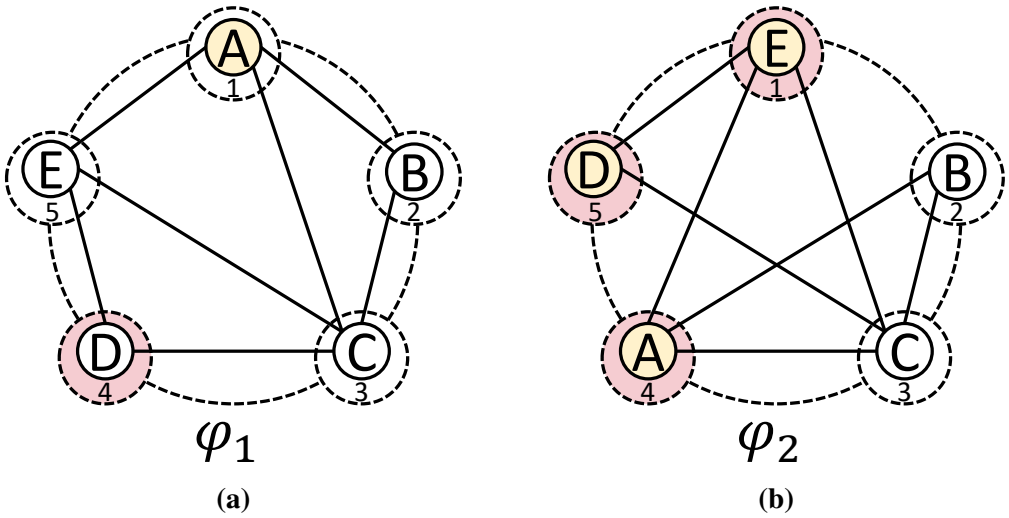


Figure 3.2 (a) Example of an embedding φ_1. (b) Resultant embedding φ_2 obtained after the operation *Insert*$(\varphi_1, \mathrm{A}, 4)$.

$$N_I(\varphi) = \{\textit{Insert}(\varphi, u, v) \forall u \in V_G, v \in vV_H, \varphi(u) \neq v\}. \tag{3.12}$$

The second neighborhood proposed in this research is defined by the swap move. The swap move consists of exchanging the assignation of vertices u and w (i.e., $\varphi(u) = z$ and $\varphi(w) = v$). An example of a swap in the solution φ of the previous example is depicted in Figure 3.3. In particular, in Figure 3.3(a), vertices A and E of the input graph are assigned to vertices 1 and 4 respectively. Then, Figure 3.3(b) illustrates the resultant solution after exchanging their assigned vertices of the host graph.

Formally, the swap move is denoted as *Swap*(φ, u, w), where φ is the current embedding where vertices $u, w \in V_G$ are going to exchange their assignments. Then, the associated neighborhood, denoted as $N_S(\varphi)$ is defined as the set of solutions that can be reached by applying the insert operator for each $u \in V_G$ and $v \in V_H$. More formally:

$$N_S(\varphi) = \{\textit{Swap}(\varphi, u, w) \forall u, w \in V_G, u \neq w\}. \tag{3.13}$$

Given a neighborhood structure, a local search algorithm attempts to find a solution that improves the quality of the best solution found so far. If it is able to find a better one, it replaces the best solution found with the new one. This process is repeated until reaching

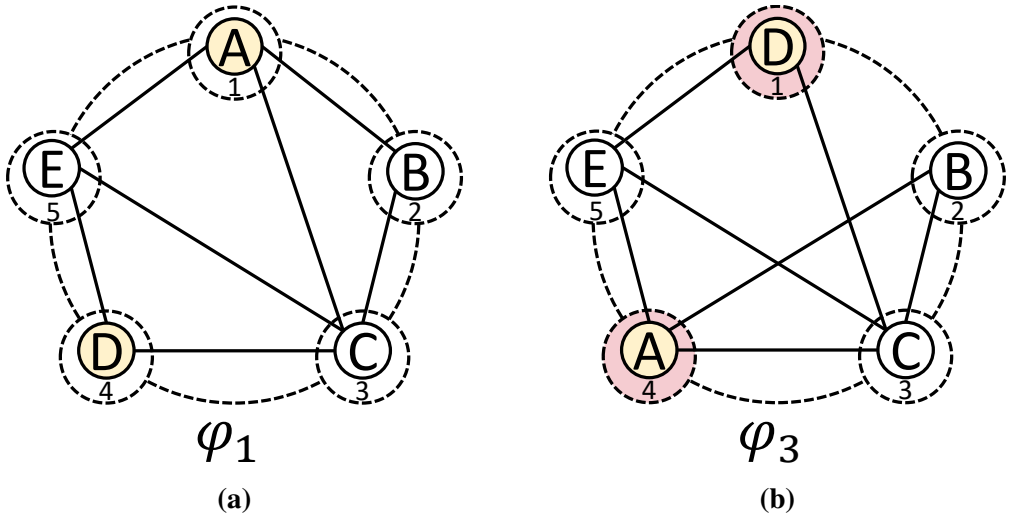


Figure 3.3 (a) Example of an embedding φ_1. (b) Resultant embedding φ_3 obtained after the operation $Swap(\varphi_1, \mathrm{A}, \mathrm{D})$.

a locally optimal solution. Algorithm 3 presents the pseudocode of a generic local search procedure for a neighborhood N and follows a "best improvement" strategy. Note that, the of function, computes the objective function of a solution φ.

To implement a "first improvement strategy", the neighborhood exploration (step 6) should be performed iteratively, checking if the solution found is better than the current one. In that case, the neighborhood exploration is stopped and the best solution found is updated.

3.3 Metaheuristics

In this Doctoral Thesis, different metaheuristic methods are used to solve the GLPs addressed. The metaheuristic procedures applied in this research combine several of the above heuristic procedures to efficiently explore the search space in order to find optimal solutions. Since metaheuristics are generic strategies that are not problem-dependent, some metaheuristics among those presented below have been proposed to address multiple problems in the context of this Doctoral Thesis (multistart procedures, TS, VNS and its variants, and IG).

Algorithm 3: An example of a local search procedure for a minimization problem following a "best improvement" strategy

```
1  Procedure LocalSearch(φ):
2      φ* ← 1
3      improves ← 1
4      while improves = 1 do
5          improves ← 0
6          φ' ← arg min_{φ'' ∈ N(φ*)} of(φ'')
7          if of(φ') < of(φ*) then
8              φ* ← φ'
9              improves ← 1
10         end
11     end
12 return φ*
```

3.3.1 Multistart procedures

In this research, a multistart optimization algorithm has been proposed to tackle all the CGLP addressed, the CCMP, the CAB, and the CBS. Multistart strategies are used in this context to escape from local optima where heuristic procedures get stuck. They are based on the idea of generating new solutions iteratively. Optionally, many methods combine the initial construction with a local search procedure as an intensification strategy. The pseudocode of a generic scheme of a multistart procedure for a minimization problem is presented in Algorithm 4. The multistart algorithm takes as input the maximum number of iterations (i_{max}). Then, it initializes an empty solution (step 2). It next enters a loop where it performs i_{max} iterations of an algorithm (step 3). Note that, the function *algorithm* should be replaced by the particular algorithm proposed for each problem (step 4). When the loop finishes, it returns the best solution found among all the restarts.

The Algorithm 4 is a simplified and simple version of a multistart algorithm. In practice, researchers use combinations of more complex termination criteria such as the number of unimproved iterations, the minimum number of iterations, or the maximum time.

Algorithm 4: Example of a multistart procedure

```
1 Procedure Multistart(i_max):
2     φ ← ∅
3     for i ← 1 to i_max do
4         φ' ← algorithm(φ)
5         if of(φ') < of(φ) then
6             φ ← φ'
7         end
8     end
9 return φ
```

3.3.2 Tabu Search

Tabu Search (TS) is a metaheuristic introduced by F. Glover in 1986 as a way to improve traditional local search algorithms [88]. The main idea of TS is to combine local search with memory-based strategies to explore the search space more efficiently and escape local optima.

TS works by maintaining a tabu list, which is a list of solutions that are considered "tabu" or forbidden for a certain number of iterations. The algorithm starts with an initial solution and iteratively generates a set of neighboring solutions, but only considers those that are not on the tabu list. If a better solution is found among the neighbors, the algorithm updates the current solution and adds the previous solution to the tabu list. The tabu list is updated at each iteration by removing solutions that are no longer tabu and adding new solutions that have become tabu [90, 91].

The tabu list acts as a memory of solutions that have been visited and helps the algorithm avoid revisiting solutions that have already been explored. This can help the algorithm escape from local optima and explore the search space further. Many ideas and extensions are discussed in [87, 90, 91].

In particular, we add a simple tabu search short-term memory to the local search proposed for the CCMP. Unlike TS original designs that use memory based on attributes, it is not necessary to include an aspiration criterion, since no tabu move can reach a solution that the search has not already visited [140]. Additionally, the short-term memory

components are enough to produce high-quality solutions, however, any optimization algorithm should find the right balance between intensification and diversification. To that aim, the tabu search framework introduces the general idea of long-term memory in order to diversify the search. However, other simpler approaches, such as the one proposed in this research to tackle the CCMP, combines TS with a multistart strategy to diversify the search.

3.3.3 Variable Neighborhood Search

Variable Neighborhood Search (VNS) was proposed by N. Mladenović and P. Hansen as a general method to solve hard combinatorial optimization problems [99, 100, 102]. The basic principle of this methodology is to perform systematic changes in the neighborhood structure to escape from local optima traps.

The use of different neighborhood structures was a novel idea in 1997 when VNS first appeared, unlike classical search procedures such as SA or TS that used a single neighborhood structure. VNS is based on the following three ideas [101]:

- A local optimum with respect to one neighborhood may not be optimal with respect to another neighborhood.
- A global optimum is a local optimum with respect to all possible neighborhood structures.
- Local optima with respect to one or more neighborhoods may be relatively close. Although there is no formal demonstration of this concern, it has been observed that, in some problems or neighborhoods, a local optimum is closely related to a global optimum.

Based on these ideas, it is possible to find many variants in the literature. Some of the most relevant ones are: Reduced Variable Neighborhood Search (RVNS), Basic Variable Neighborhood Search (BVNS), Variable Neighborhood Descent (VND), General Variable Neighborhood Search (GVNS), Parallel Variable Neighborhood Search (PVNS), or Variable Formulation Search (VFS), among others [59, 99, 101, 192].

In this research, we proposed the use of BVNS variant to tackle the CBS. Additionally, we propose the use of GVNS, and VND for the CAB.

The pseudocode of the BVNS variant is presented in Algorithm 5. Particularly, BVNS receives two input parameters, a feasible solution (φ) and the last neighborhood to be explored (k_{max}). The BVNS is conformed by three main steps: a shake procedure which helps to escape from local minima traps (step 4); a local search procedure based on the exploitation of a neighborhood structure (step 5); and the neighborhood change procedure, that determines which neighborhood is explored next (step 6) depending on the improvements found in the current neighborhood. These three steps are repeated until reaching the maximum number of neighborhoods explored, k_{max}. The neighborhood change procedure follows a standard design that increases the value of k by one unit when there is not an improvement in the current iteration, and it resets the value of k to one when an improvement is found.

Algorithm 5: Basic Variable Neighborhood Search Procedure

```
1 Procedure BVNS (φ, k_max):
2     k ← 1
3     while k ≤ k_max do
4         φ' ← Shake(φ, k)
5         φ'' ← LocalSearch(φ')
6         φ ← NeighborhoodChange(φ, φ'', k)
7     end
8 return φ
```

The main difference between BVNS and GVNS is in step 5 of Algorithm 5. As it can be seen, in BVNS the improving procedure is a local search. In contrast, in the GVNS variant, this step is replaced by a VND which is characterized by the deterministic exploration of several neighborhood structures during the search. In that case, the neighborhoods are explored sequentially and in descending order through local search procedures. This method returns a solution that is locally optimal with respect to all neighborhoods explored [60]. Algorithm 6 illustrates the pseudocode of a generic VND procedure for a minimization problem. In particular, the procedure receives two input parameters, the input solution (φ) and the set of neighborhoods (N) where $N = \{N_1, \ldots, N_{k_{max}}\}$. The procedure starts by

initializing k to the value one and then, it enters a loop in which it iterates through each neighborhood in the set. Each neighborhood is explored with the goal of finding a better solution than the current one found so far (step 4). If the solution found is better than the current solution (step 5), the algorithm updates the current solution (step 6). If the improved solution is not better than the current solution, the algorithm moves on to the next neighborhood (step 9). The loop continues until all neighborhoods have been explored. Then, the best solution found is returned.

Algorithm 6: Variable Neighborhood Descent Procedure

```
1  Procedure VND (φ, N):
2      k ← 1
3      while k ≤ k_max do
4          φ' ← LocalSearch(N_k, φ)
5          if of(φ') < of(φ) then
6              φ ← φ'
7              k ← 1
8          else
9              k ← k + 1
10         end
11     end
12 return φ
```

3.3.4 Iterated Greedy

Iterated Greedy (IG) is a metaheuristic based on the repeated application of two main phases: a partial destruction of a solution, and a reconstruction to reach a new feasible solution. These two phases are usually repeated until termination criteria is met [216, 235].

A solution is partially destroyed in the destruction phase to create a new feasible solution. The goal of this phase is to diversify the search by moving into unexplored regions of the solution space. The destruction phase typically involves small changes to the current solution to create a new solution that is feasible, but not necessarily better. This can be done in various ways, depending on the problem being solved. In addition, researchers propose the use of parameters to control the percentage of the solution to be destroyed. Therefore,

small modifications intensify the exploration in a narrow region of the solution space. On the contrary, large variations diversify the search in a broader region.

The reconstruction phase receives a partially destroyed solution and restores it to create a new feasible solution that is better than the current one. The reconstruction is done by a greedy procedure that, in some cases, will be similar to the one used to generate the initial solution.

IG is widely hybridized or combined with search algorithms such as local search procedures. In fact, for constructive heuristics, a natural extension is to improve the solutions generated by applying a local search method being, in the simplest case, an iterative improvement algorithm. This extension results in the procedure proposed to address the 2DBMP.

A standard version of IG combined with a local search procedure is presented in Algorithm 7. The procedure receives the input graph G and the maximum number of iterations i_{max}, as parameters. The algorithm starts by generating an initial solution with the greedy constructive procedure (step 2). Then, after obtaining an improved solution through the local search procedure (step 3), the procedure enters a loop (step 4). In each iteration, some elements are removed from the current solution using the destruction method (step 5). Next, the solution is greedily reconstructed (step 6) and improved again by the local search procedure (step 7). In each iteration, IG determines whether the perturbed and improved solution (φ''') is better than the incumbent one (φ) (step 8). If so, φ is updated accordingly. These three last steps (destruction, reconstruction, and local search) are repeated until reaching a maximum number of iterations. Once the termination condition is met, IG returns the best solution found.

3.4 Advanced strategies

The procedures presented in this chapter can be further improved with the use of the three advanced strategies. Although these strategies were designed within the context of GLPs, the ideas behind them might be applied to other heuristic searches. The first strategy explores computationally efficient ways of calculating the value of the objective function after a move, which is particularly important in large neighborhoods (Section 3.4.1). The second

Algorithm 7: Iterated Greedy procedure

```
1  Procedure IG(G, i_max):
2      φ = GreedyConstructive(G)
3      φ ← LocalSearch(φ)
4      for i ← 1 to i_max do
5          φ' ← Destruction(φ)
6          φ'' ← Reconstruction(φ')
7          φ''' ← LocalSearch(φ'')
8          if of(φ''') < of(φ) then
9              φ ← φ'''
10         end
11     end
12 return φ
```

strategy deals with flat landscapes, where many solutions have the same objective function value (Section 3.4.2). Finally, the third strategy also addresses efficiency, but from the point of view of reducing the number of moves to be evaluated within a neighborhood structure (Section 3.4.3).

3.4.1 Efficient evaluation of a solution after a move

The most time-consuming part of an optimization algorithm is usually the improvement strategy since it explores the neighborhood of a solution. Sometimes, a local search may need to examine almost all the neighboring solutions, especially when finding better solutions becomes harder or when using a "best improvement" strategy. Neighborhood exploration involves calculating the value of the objective function for each of the solutions that are part of it. For this reason, the time spent calculating the quality of a solution is critical to the performance of a heuristic algorithm, such as a local search. Therefore, researchers have proposed an intelligent or efficient evaluation of the objective function. In this Doctoral Thesis, two levels of optimization in the calculation of the objective function are proposed.

The first and simplest level focuses on the detection of the elements of the solution that have changed after a movement has been performed. Two moves were proposed in Section

3.2: insertions and exchanges. In the case of the insertion, denoted as $\mathit{Insert}(\varphi, u, v)$, at least two vertices may be affected by the move, the vertex inserted u and the vertex currently assigned to the position where u is going to be inserted $\varphi(w) = v$. In addition, all displaced (or shifted) vertices will also be impacted by the movement. Finally, vertices adjacent to any of the directly affected vertices would also have to be considered.

Considering the insertion move depicted in Figure 3.2, vertices A, D, and E are directly influenced by the $\mathit{Insert}(\varphi_1, \mathrm{A}, 4)$, while vertices B and C, are indirectly affected by the motion of their adjacent vertices.

Similarly, when considering a $\mathit{Swap}(\varphi, u, w)$ move, vertices u and w and their adjacent vertices are affected. Considering again the example depicted in Figure 3.3, vertices A and D are directly affected by the move $\mathit{Swap}(\varphi, \mathrm{A}, \mathrm{D})$, while the rest of the vertices (B, C, E) are affected for being adjacent the swapped vertices.

From a computational point of view, the complexity of the computation of the optimized objective function is also in the same order as the original objective function calculation order, since all the vertices might be affected by the move, so in the worse case they would be equivalent. However, despite the fact that this proposal does not reduce the theoretical complexity, from a practical point of view, and as we will illustrate in the experimental section, the number of edges affected by a move is considered, on average, smaller than the total number of edges of the graph, especially when the input graph is large.

The second level of optimization related to the calculation of the objective function is devoted to reducing the complexity involved in its calculation. However, this optimization is only possible for some specific objective functions and move operators. On the other hand, extra data structures are needed whose updating and storing do not deteriorate the performance of the algorithm. Given the particularity and complexity of this procedure, the reader is referred to the articles in which it is published. Specifically, this has been implemented for the CAB in [31] and for the CBS in [33].

3.4.2 Tiebreak criterion for solutions with the same objective function value

An essential element for the successful application of local search procedures is the use of the objective function value to guide the search. However, sometimes, the search space presents plateaus (valleys) in the fitness landscape, also known as flat landscapes [16, 210, 207] where many solutions have the same objective function value associated, despite the structure of the solution might differ significantly. This situation is likely to occur in optimization problems formulated as max-min (or min-max) problems, where the objective function consists of maximizing a minimum value (or minimizing a maximum value). Generally, these type of problems has numerous solutions associated with the same value of the objective function. When this occurs, it is difficult to determine which of the compared solutions is more promising to continue the search. In this case, the objective function value alone does not provide enough information to find effective search directions. *A priori*, two solutions with the same objective function value are equivalent for a local search procedure. Therefore, it is necessary to identify certain properties of the solution that determine which solution is more promising to continue the search. Some of the proposals that can be found in the literature to mitigate the impact of this problem consist of using additional objective functions as a tie-breaker tool [186, 192, 207]. These alternative objective functionsalternative objective function are only computed when the value of the original objective function for the compared solutions is the same.

In this research, we face a max-min optimization problem, the CAB (see Section 2.2), and two min-max optimization problems, the CCMP (see Section 2.1) and the 2DBMP (see Section 2.4). To overcome this difficulty, we propose a tiebreak criterion based on the frequency of a particular cut (in the case of the CCMP) or bandwidth (for the CAB and 2DBMP). More formally, given a solution (ψ, φ), we define f_c as the number of edges of the host graph with an associated cut equal to c as follows:

$$f_c = \{(w,z) \in E_H : \text{cut}(\varphi, \psi, (w,z)) = c\}. \tag{3.14}$$

Then, let c_{max} be the maximum cut among all edges of the host graph of n vertices, and therefore, $c_{max} = ccw(G, \varphi, \psi)$. Then, given an embedding (φ, ψ), the tie-breaking

function t_c for the CCMP is defined next:

$$t_c(\varphi, \psi) = \sum_{c=0}^{c_{max}} n^c \cdot f_c. \tag{3.15}$$

Analogously, in the case of bandwidth-based problems, we define f_b as the number of edges of the input graph with an associated bandwidth equal to b. In mathematical terms:

$$f_b = \{(u, v) \in E_G : bw(\varphi, \psi, (u, v)) = b\}. \tag{3.16}$$

Then, let b_{max} be the maximum bandwidth among all edges of the input graph and n the number of vertices of the input graph. Then, given an embedding (φ, ψ), the tie-breaking function t_b for the CAB and t'_b for the CBS are formalized in Equations 3.17 and 3.18 respectively:

$$t_b(\varphi, \psi) = \sum_{b=1}^{b_{max}} n^{b_{max}-c} \cdot f_b. \tag{3.17}$$

$$t'_b(\varphi, \psi) = \sum_{b=1}^{b_{max}} n^c \cdot f_b. \tag{3.18}$$

The proposed tie-break criteria take into consideration not only the objective function of an embedding but also additional semantic information related to promising solutions. Specifically, when the objective function value of two solutions is equal, both solutions are evaluated using the tie-breaking functions (i.e., t_c t_b and t_b). The solution with the lower value is chosen as the most promising one. The rationale behind this decision is to penalize those solutions with many edges with a cut or bandwidth close to the value of the objective function. It is worth mentioning that if the t-value for two solutions is the same, they are considered equivalent (in terms of the tie-breaking criterion).

3.4.3 Neighborhood reduction strategy

Considering that for the problems of interest, the neighborhoods are generally of immense size, it seems necessary to propose more advanced techniques for the exploration of these neighborhoods. This problem led to the well-known neighborhood reduction techniques.

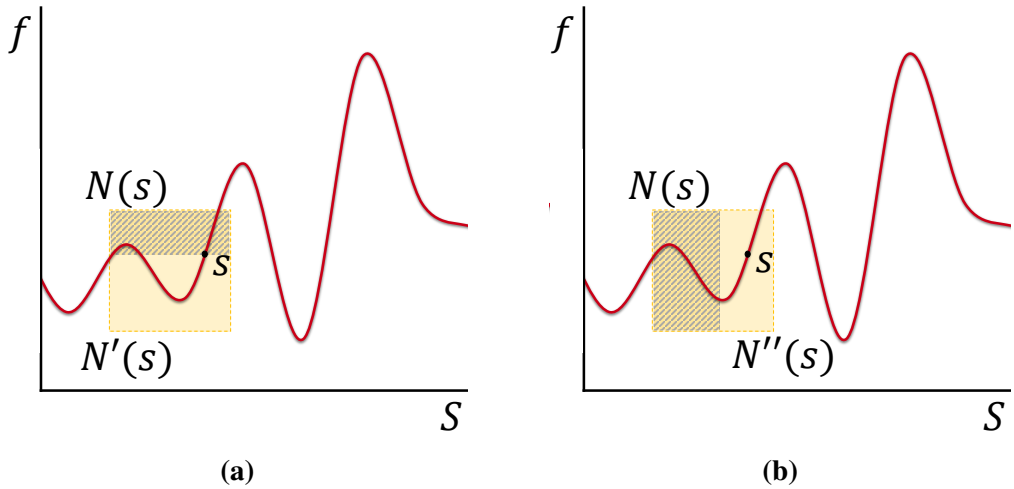


Figure 3.4 Representation of the landscapes of a minimization optimization problem. (a) Reduction of the neighborhood $N(s)$ to the subset $N'(s)$. (b) Reduction of the neighborhood $N(s)$ to the subset $N''(s)$

Ideally, a reduction technique focuses on promising solutions, avoiding waste of time in the evaluation of solutions that produce a deterioration in the objective function (see Figure 3.4(a)). However, in practice, given the complexity of the neighborhoods, some techniques are limited to reducing their size by setting a percentage or a fixed number of solutions to explore (see Figure 3.4(b)).

In this Doctoral Thesis, neighborhood reduction strategies have been proposed for some of the problems studied. In the CCMP, we were able to determine those potential moves that, in the worst case, will lead to a solution with the same quality as the current solution found [34].

Neighborhood exploration for the CAB was constrained by two neighborhood reduction strategies. Specifically, the exchange and insertion moves proposed in Section 3.2 depend on two vertices, one from the input graph and one from the host graph in the insertion move, and two vertices from the input graph in the swap case. In this research, two reduction strategies are proposed to reduce the number of vertices of both graphs. Specifically, the first strategy reduces the number of vertices of the input graph that can be moved, while the second strategy reduces the number of vertices of the host graph where the input vertices can be inserted, in the case of insertions. Similarly, it reduces the number of vertices of the

input graph that can be swapped, for the exchange operation.

Finally, in the 2DBMP we avoid the exhaustive exploration of the proposed neighborhood based on swaps by identifying all possible exchanges that would worsen the quality of the solution. Therefore, only moves to solutions of equal or better quality are considered.

3.5 Final proposals

In this chapter, the main heuristic and metaheuristic techniques applied in the context of this Doctoral Thesis have been described. Next, Table 3.1 summarizes the strategies proposed for each of the tackled problems: the CCMP, the CAB, the CBS, and the 2DBMP. Specifically, the algorithms and strategies are organized into greedy constructive, local search, metaheuristics, and advanced strategies.

As demonstrated, although heuristic strategies are generally problem-dependent, in this research we have been able to find strategies with more general applicability. However, it should be noted that the objective function tie-breaking criterion is not implemented for CBS because, unlike the other three problems where a maximum or minimum must be minimized or maximized, in this problem, a sum must be minimized. This makes the quality of the solutions much more diverse and thus hampers the emergence of large sets of solutions with the same objective function value.

Moreover, the four elements that organize this table can be understood as the framework for the development of an algorithmic proposal for any optimization problem.

Finally, it is worth mentioning that in this chapter only the algorithms and strategies that have been used in the final proposals have been included. However, to achieve this final configuration, it has been necessary to experiment and test a wide variety of strategies and algorithms.

3.6 Software development

The aim of this Doctoral Thesis is to optimize a specific problem or domain using heuristic and metaheuristic techniques. This involves combining computer science, mathematics and statistics, and operational research, which are all broad disciplines. Figure 3.5 shows a

		CCMP	CAB	CBS	2DBMP
Greedy Constructive	γ	✓			
	γ_w			✓	✓
	BFS based		✓		
	Patterns	✓			
	λ		✓	✓	✓
	Randomized	✓	✓	✓	✓
	Adaptive	✓	✓	✓	✓
Local Search	N_I	✓	✓	✓	
	N_S		✓	✓	✓
Metaheuristic	Multistart	✓	✓	✓	
	TS	✓			
	VNS		✓	✓	
	IG				✓
Advanced strategies	Efficient OF	✓	✓	✓	✓
	Tiebreak	✓	✓		✓
	Neigh. Reduction	✓	✓		✓

Table 3.1 Summary of the strategies and algorithms proposed for the problems addressed.

Venn Diagram [245] that illustrates this combination. To apply mathematics and statistics to real-world problems, we need to formalize and model them mathematically. We also need to extract or elicit the requirements of the stakeholders and how they fit into the business context, which can be translated in the optimization area as the understanding of the constraints, objectives, and input data of the problem to be addressed. In addition, software developers use computer science methodologies to extract this domain knowledge and propose a software solution, as we did in this thesis for an optimization problem. The software development also requires deployment and maintenance of the program [246].

The relationship between building software for a company and solving an optimization problem follows similar processes. It is for this reason that in this Doctoral Thesis, special importance has been given to software development. Therefore, in this section, the most relevant aspects related to software engineering are collected.

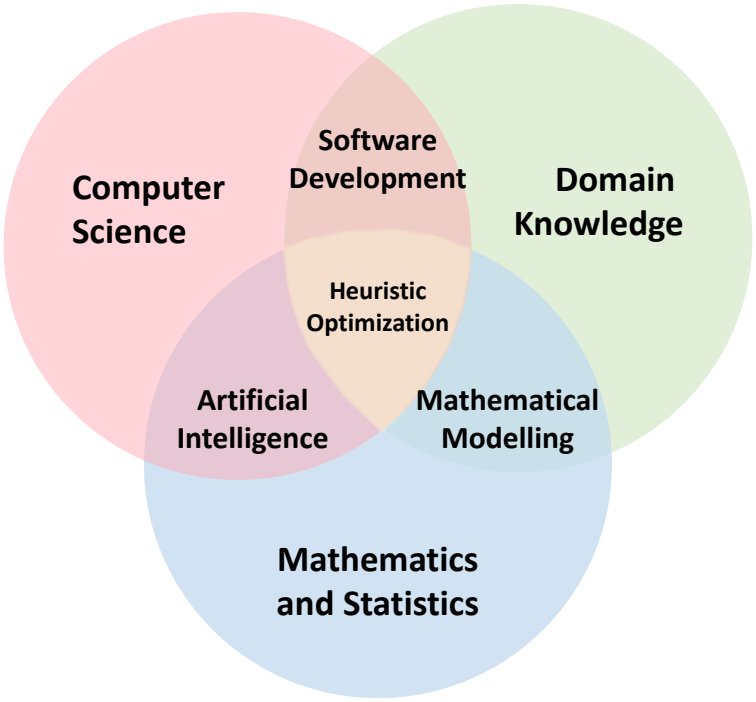


Figure 3.5 Disciplines which influence the heuristic optimization field in a Venn Diagram.

3.6.1 Implementation issues

The quality of a proposed algorithm is mainly measured using two metrics as a basis: the quality of the solutions obtained and the execution time. The implementation of an algorithm is as important as the proposed algorithm itself, i.e., if the algorithm is good, but the implementation is not, it will be difficult to obtain quality solutions in a reasonable amount of time.

Another essential issue to consider, when implementing a heuristic optimization algorithm is the programming language. Some of the most relevant general-purpose programming languages are Java [7], Python [243], C [132], C++ [234], or C# [112] among others. Although all programming languages are valid for coding an algorithm, certain aspects must be taken into account, such as complexity for learning, speed of execution, use of external libraries, or the possibility of running it on different operating systems easily [70].

In this Doctoral Thesis, the Java programming language has been chosen over the rest of the aforementioned languages. The flexible nature of Java enables programmers to create code that can be executed on any system or device, regardless of its architecture or platform. Since it is one of the most widely used programming languages in the world, it has continuous maintenance and a large community. Some Java advanced features are: the language is compiled and interpreted, platform-independent, portable, robust, multithreaded, and object-oriented [77, 196].

It is worth focusing on Java object-oriented feature. Since Java is strongly oriented to the object paradigm, almost everything in Java is an object and the main functionalities of the program live within objects and classes. This makes it considerably easier to achieve one of the objectives proposed in this Doctoral Thesis: "Model the problem so that it can be approached computationally".

The organization of the code into classes guides the developer to structure the code with the goal of solving a problem. For example, since this dissertation solves optimization problems that are closely related one to each other, the code developed will be similar. By organizing the code properly, a common development framework can be created for all of them.

Figure 3.6 shows the main classes of the application software developed to approach a particular problem, the CCMP. Note that, the diagram presented is based on the UML class diagram [22]. In particular, this diagram is organized into six packages.

The package "util" contains those classes that usually handle certain functionalities of the algorithms and that are generally common to any optimization problem such as instance reading, time control, random number generation, or a report generator.

The package"models" contains two of the fundamental classes in the implementation of any optimization problem: the instance and the solution. Both classes are abstract classes, as shown by the blue circle with the letter "C" next to the class name. Abstract classes are classes with general applicability to any optimization problem. Therefore, when implementing a particular problem, the particular classes of that problem are coded to inherit the generic behavior of the abstract classes.

The package "algorithms", and its subpackages, "constructives", and "improving", contain the procedures that operate on the solutions: construct them, modify them, etc. In other

words, each of these classes represents an algorithm.

Finally, the package "experiment" represents the program to be executed. In the "Experiment" class, a concrete experiment is executed, and it is used to compare different algorithms. Therefore, this class requires the reading of instances, the use of algorithms to generate solutions, and methods to generate reports of the desired metrics.

Among all the classes represented in the diagram shown in Figure 3.6, the Solution class stands out above the rest, since its correct implementation is critical in the final performance of the algorithm. Objects of this class represent a solution to the considered problem and are going to be handled by the proposed algorithms.

In the case of the GLPs, from a formal point of view, the solution is represented by means of the tuple (φ, ψ). When representing these concepts by a Java class, we have chosen to use two synchronized data structures. Specifically, two arrays of integer values denoted R and R^{-1} are used. An array is a one-dimensional matrix composed of an index that identifies each memory position within the array, and the corresponding content of those positions in memory. Therefore, an array can be understood as a set of key-value tuples.

In array R, the key represents the vertices of the host graph, while the value represents the vertices of the input graph. Therefore, a correspondence key-value represents the assignment of an input vertex to a host vertex. In this sense, there are n key-value pairs, being n the number of vertices of the host graph. Figure 3.7 illustrates the representation of this array with a table. In this case, the header of the array corresponds to the keys of the array (i.e., vertices of the host graph) while the second row of this graphical representation contains the values associated with each key (i.e., vertices of the input graph). This is the straightforward implementation of the φ function.

Oppositely, in $R^{-}1$, the key represents the vertices of the input graph, while the value represents the vertices of the input graph. Figure 3.7, illustrates the array R^{-1} of the solution depicted in Figure 3.7. Therefore, determining the assigned host vertex to a vertex of the input graph, or vice versa, has a computational complexity of $O(1)$.

It should be noted that an array structure is very appropriate for this task since it allows direct access to its components. In addition, the choice of this structure has taken into account that it is generally quite efficient when it comes to queries and modifications.

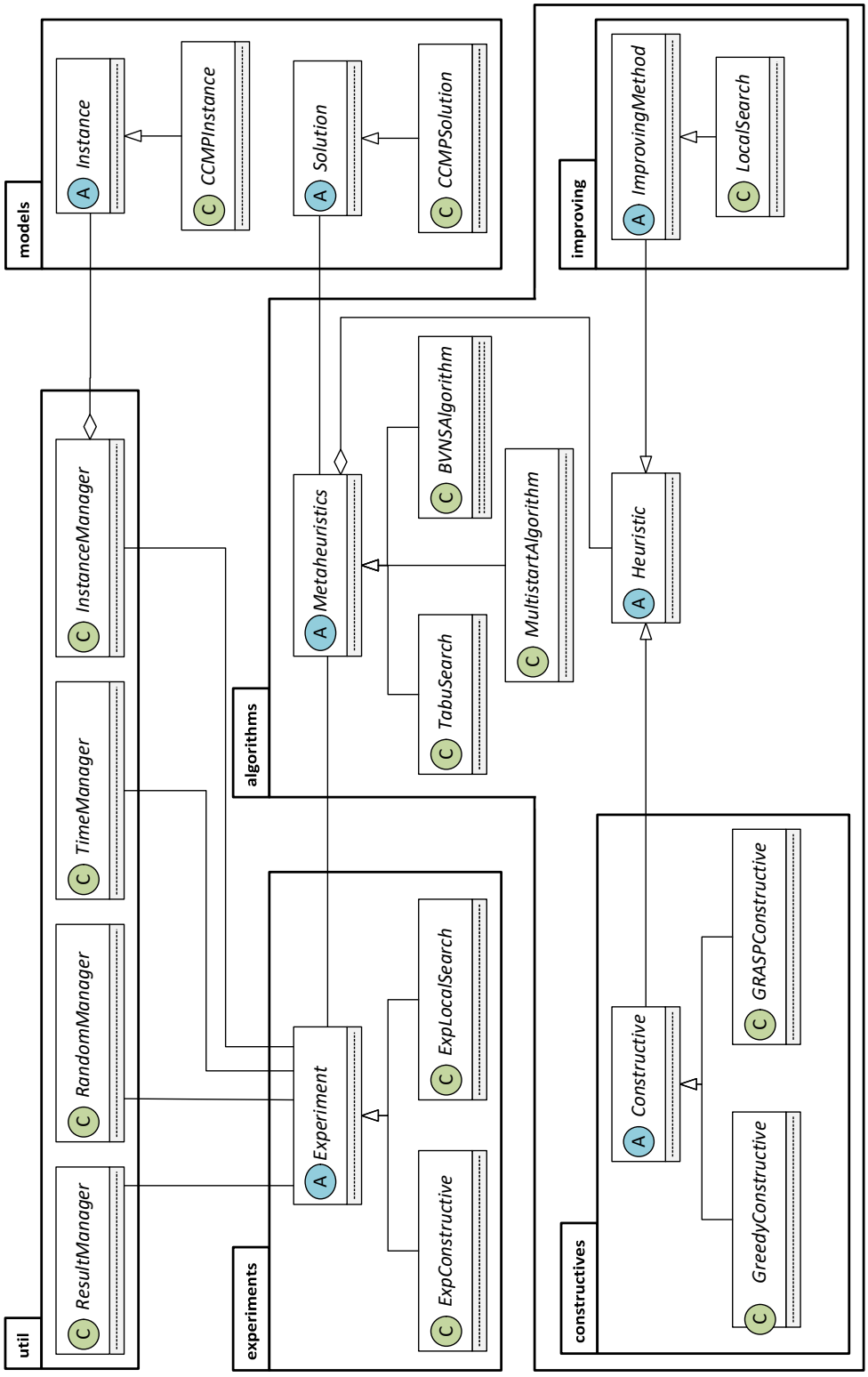


Figure 3.6 Class diagram of the software developed for the CCMP.

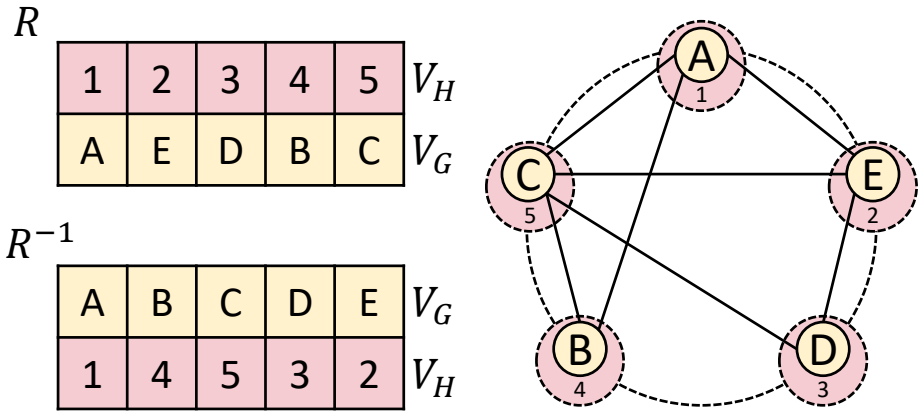


Figure 3.7 Example of the generic representation of a solution using two arrays.

Finally, some other auxiliary data structures are typically used to facilitate the calculation of the objective function of a solution. For example, in the case of CBS, a structure is used to store, for each edge of the input graph, the value of the associated bandwidth. In this way, in the case of calculating the value of the objective function of the solution after a move, only the value of the edges affected should be updated.

3.6.2 Solution visualization

Geometric representations and drawings have been the subject of study since about 3000 BC. This dissertation begins by quoting the famous citation of Archimedes "Do not disturb my circles!" (in Latin, "*Noli turbare circulos meos!*") indicating the early connection between graph drawing and geometry [62].

Classic geometry is a branch of mathematics that deals with the properties and relationships of points, lines, angles, surfaces, and solids. Undoubtedly, Euclid revolutionized classic geometry after presenting his best-known work, the book "Element" [69, 111].

Today, there are more advanced technologies to deal with geometry. Here, automated graph drawing emerges as a system to position vertices (or nodes) and edges (or arcs) to produce graphs with desired properties (See Section 1.2.3) [89, 239].

In this research, we consider crucial to develop software to represent and generate drawings of the solutions obtained. By observing the graphical representation of a solution, it is

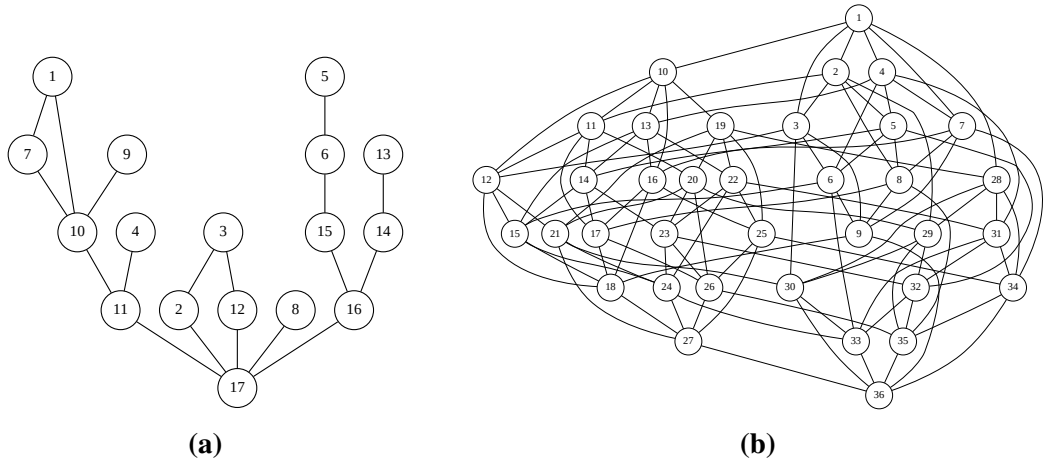


Figure 3.8 (a) Example of a graph of the Small Instances set. (b) Example of a regular *Toroidal Mesh* topology graph.

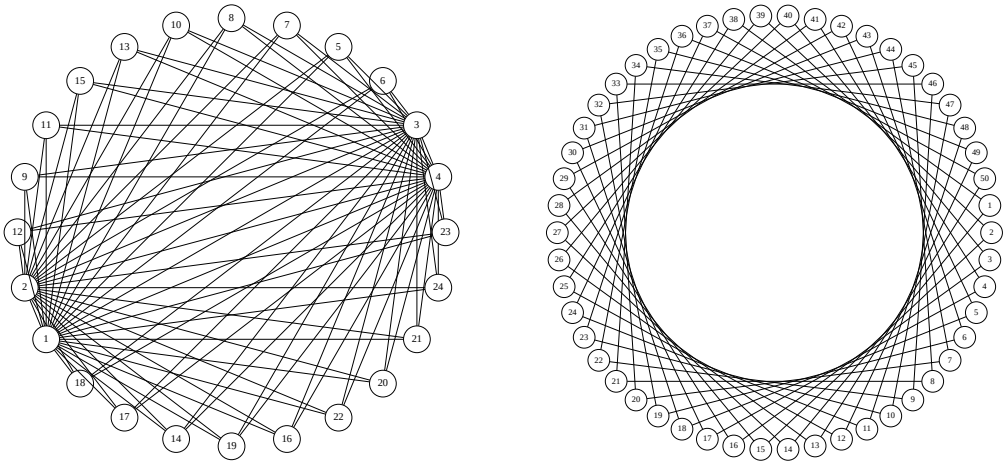
possible to visually conjecture how close it is to an optimal solution, what difficulties an algorithm may present, or to analyze the evolution of the quality of the solutions during the search.

Therefore, solutions obtained by the different procedures are transformed into an image, in vector format. Also, just as a data scientist does a preliminary analysis of the data, when working on GLPs it is important to know the structure and topology of the input graphs.

Figures 3.8 and 3.9 have been made through the library developed in the context of this Doctoral Thesis. In particular, this library has been coded in Java, in combination with DOT and Graphviz. DOT is a plain text descriptive language that provides a simple way to describe graphs [79]. Using the open-source software Graphviz [67, 80], graphs described in the DOT language are converted into images, vector images, or PDFs.

The development of such software not only allows us to visualize solutions in order to understand the performance of the algorithms (see Appendix A), but it can also be used to compare the generated solutions with other graph-drawing software. In this sense, the conventional wisdom that a picture is worth a thousand words applies here to evaluate our proposals and compare them it with other graph-drawing software, beyond statistical study based on specific objective functions.

Although a wide variety of free libraries for graph representation can be found on the Internet, it is important to note that none of them allow the representation of a graph as the



(a) **(b)**

Figure 3.9 Graphical representation of two solutions for the CCMP. In particular, figure (a) shows the Complete Split graph $K_4 \nabla \overline{K_{20}}$, made of 24 vertices and 86 edges; and (b) depicts the Circulant graph C_{50}^{14}, made of 50 vertices and 50 edges.

embedding of an input graph in a given host graph. This library can be reused to represent solutions to any other problem of GLP.

3.6.3 Resources used

The developed research has been carried out at Universidad Rey Juan Carlos (URJC), within the Group for Research in Algorithms For Optimization (GRAFO). Both the university and the research group have supported this research by allowing the use of their resources. The means necessary for the achievement of this Doctoral Thesis are described below.

GRAFO consists of professors and students belonging to different universities and research areas. This group, in addition to providing funding to support the Ph.D. student, has shared their knowledge, contacts of international researchers, and the necessary scientific equipment. The characteristics of the hardware used are specified next:

- Dell XPS 13 9300. Intel Core i7-1065G7 4-core CPU @ 1.30GHz. RAM 16.0 GB
- High-performance computing cluster. Experiments were carried out on Ubuntu 20.04 Virtual Machine. AMD EPYC 7282 16-core CPU @ 2.80GHz. RAM 16.0 GB

On the other hand, the URJC has provided the Ph.D. candidate with a workplace as well as multiple other facilities ranging from software licenses to bibliographic material. The most relevant resources in the context of this Doctoral Thesis are listed below:

- GitHub Campus Program is a premium access package to GitHub for education-focused institutions. GitHub is a web-based platform for version control and collaboration that uses the Git control system. It allows developers to store and track changes to their code and collaborate with other developers on projects.
- JetBrains Professional License. Jetbrains is a software development company that is that provides integrated development environments (IDEs) and other developer tools. In particular, IntelliJ IDEA and PyCharm (IDEs for Java and Python development respectively) have been used.
- Microsoft Office is a collection of productivity software developed and published by Microsoft. It includes applications such as Word, Excel, PowerPoint, Outlook, OneNote, and OneDrive, among others.
- Adobe Acrobat Reader is a software to view, print, and annotate Portable Document Format (PDF) files.
- SPSS Statgraphics is a statistical analysis software package for statistical analysis and visualization [74].
- Library of the URJC, with access to inter-library loan, the Madroño Consortium, specialized databases such as Web Of Science, or access to journals of Elsevier, Springer, IEEE, ACM, and other digital libraries.
- Microsoft Visio and StarUML are diagramming programs generally used to visualize business data. Both softwares support the Unified Modeling Language (UML) framework. In the context of this Doctoral Thesis, they have been used to model the problems, the flow of the algorithms, and the structure of the input data.

Chapter 4

Joint discussion of results

This chapter summarizes and synthesizes the results obtained during the experimentation carried out in this Doctoral Thesis, as well as the drawing of some conclusions and implications based on the results obtained when considering the research as a whole. It also provides a comprehensive and critical evaluation of the research, demonstrating the contribution of the proposed algorithms and strategies for each problem studied.

Specifically, this chapter is organized into three sections. The section 4.1 presents the guidelines followed for the experimental evaluation of the proposed algorithms in a rigorous way. Then, Section 4.2, describes the tuning process of an algorithm as well as some directions to illustrate the merit of the proposals. Finally, Section 4.3, describes the framework followed to compare the proposed algorithms with the best algorithms found in the state of the art.

4.1 Analysis of the performance of the algorithms

Analyzing the performance of an algorithm is essential to evaluate its effectiveness and efficiency in solving a problem. However, it is usually not sufficient to rely solely on a theoretical approach to evaluate them. Especially in the case of heuristic and metaheuristic algorithms that are designed to find good solutions, but not necessarily optimal ones, to optimization problems. In this case, their performance can be affected by a number of factors that will not be captured by a purely theoretical analysis.

To properly assess a metaheuristic, it is often necessary to perform an empirical analysis, where the algorithm is run on a series of test cases and its performance is measured using appropriate performance metrics. In this Doctoral Thesis, a very specific experimentation scheme is followed. These guidelines are summarized in the following steps:

Step 1: **Experimental Design**. Before analyzing the performance of an algorithm, the assumptions and objectives must be clearly defined. This also involves the selection of test cases or instances. These test cases should be representative of the types of inputs that the algorithm is likely to encounter in practice. Also, this step involves obtaining other codes of state-of-the-art algorithms.

Step 2: **Measurement**. To properly analyze the performance of an algorithm, some performance metrics to measure have to be defined. This might include metrics like time complexity, space complexity, or the quality of the solutions produced by the algorithm.

Step 3: **Implementation and execution**. The proposed algorithms are executed on the selected test cases. In some non-deterministic environments, it would be interesting to run the algorithm several times to take into account the randomness or variability of the results.

Step 4: **Collect, report, and analyze the results**. The raw results are manipulated to obtain the metrics specified in the previous step. Generally, the analysis of the results entails plotting graphs, and charts or calculating statistical measures.

Step 5: **Draw conclusions and validate the hypothesis**. From the analysis of the results, conclusions are drawn, and the initial hypothesis is validated. This step involves the extraction of knowledge from the data, identifying areas where the algorithm performs particularly well or poorly, and suggesting ways to improve its performance.

Based on the steps described above, we now focus on two of them, which are decisive for the correct execution of the rest of the steps and therefore of the research in general. In particular, in Section 4.1.1 we list and analyze the instances used in the studied GLPs. Then, Section 4.1.2 collects the most relevant metrics for comparing algorithms.

4.1.1 Instances

Instances used in this Doctoral Thesis are introduced in Section 1.2.2. Specifically, these instances are graphs that are used as the input graph of the addressed GLP. Table 4.1 lists the instances utilized, organized by problem and set. From the table, it can be observed that most of the graphs used have a well-known topology except the Small Instances, Random Graphs, and Harwell-Boeing sets. Moreover, for all the problems studied, sets of instances with a known optimum can be found (they are marked with a "*" in the table). Instances with a known optimum are useful to test the effectiveness of heuristic algorithms. However, when designing an algorithm for general graphs, the behavior of the method on a specific family, with a particular topology, might not reach the optimal solutions.

From the observation of Table 4.1 it can be concluded that there is a minority of instances that represent real scenarios. In particular, the Harwell-Boeing set includes instances that model problems in diverse areas of engineering [61].

Finally, it is worth mentioning that all instances listed in Table 4.1 can be easily obtained from literature articles, which kindly provide links for downloading. Moreover, graphs with known topology have the advantage that they can be generated using commercial software.

The selection of input instances to evaluate an algorithm should be performed carefully. When determining the set of instances, researchers should take into account their diversity, difficulty, and structure. In addition, a large number of instances may facilitate a fairer comparison and analysis of the proposed methods. In fact, some statistical tests require a sufficiently large sample in order to be performed correctly. However, it also implies a higher effort in the validation process.

Generally speaking, the considered instances are usually divided into two subsets. The first one is used to adjust the algorithm parameters and the second one to evaluate its performance. The calibration of the parameters of the metaheuristics is an important and tricky task that will be further explained in Section 4.2. From the point of view of the selection of the instances, a poor selection of instances can cause the algorithm to overfit. Overfitting is one common problem in artificial intelligence or machine learning algorithms that arises when an algorithm performs well on the training data but poorly on new or unseen data [109, 257]. Additionally, this set division can lead to a more accurate measure of the

	CCMP	CAB	CBS	2DBMP
3D Meshes		18		
Bipartite graph				*6
Cartesian product of graphs			6	*15
Caterpillar		36		
Complete binary tree		21		
Complete graph				*2
Complete Split	*21			
Cone	*10			
Cycle Pow			*4	6
Cycles		*22	*2	*3
Double Stars		18		
Grids		*21		
Hamming		*22		
Harwell-Boeing	38	21	90	45
Hypercubes		5		
Join of Hypercubes	*9			
Paths		*22	*2	*3
Petersen graph				*1
r-level t-ary trees				*4
Random Graphs		28		
Small Instances	84			
Toroidal Mesh	*17	*33		
Wheel			*2	5
Total	179	247	106	90

Table 4.1 Sets of instances used to test and compare the proposed algorithms for the studied GLPs. "*" indicates instances with known optima.

performance of an algorithm or approach, as it allows the testing of the algorithm on new data rather than relying on the same data used to train or develop it.

In order to address the problems that a poor selection of instances may entail, during the course of the Doctoral Thesis a parallel research was carried out, which ended with the proposal of an automatic method for the selection of a preliminary set of instances. The reference to the associated publication with this research can be found in Chapter 11.

4.1.2 Metrics

Before executing an algorithm, the performance measures and indicators may be selected. Then, a statistical analysis will be applied to the obtained results. Measurement metrics are the way researchers have found to evaluate the performance and effectiveness of the proposed algorithms in solving a particular problem.

Metrics can generally be divided into the following groups: quality metrics, efficiency metrics, and robustness [15, 39, 230].

Quality metrics measure the quality or effectiveness of the solutions produced by an algorithm. An example of a quality metric, in the context of the GLP, is the objective function value of the solution. Based on the objective function value, other metrics can be derived, such as the difference to the global optimum solution or the distance to the lower/upper bounds.

Efficiency metrics usually measure the time that an algorithm needs to solve. Comparing the execution time of algorithms is the most commonly used metric for evaluating the efficiency of an algorithm in solving a problem. However, it is important to note that the execution time of an algorithm can be affected by a variety of factors, such as the computer or the specific environment in which the algorithm is run. For example, the execution time of an algorithm may be slower on a computer with slower processing speeds or less principal memory, or in an environment with higher levels of background activity due to other processes in execution. The programming language is also a matter to consider. Although researchers strive to make fair comparisons, the analysis of the efficiency of algorithms is an issue that needs to be studied deeper.

Other more precise metrics focus on the study of efficiency from the perspective of complexity, such as the time complexity, or the space complexity. The time complexity measures the amount of time it takes for an algorithm to solve a problem as the size of the input data increases, while the space complexity, measures the amount of memory required to solve a problem. However, it is not always possible to calculate the complexity of an algorithm [23].

In this research, in order to promote a fair and unbiased comparison, the state-of-the-art

algorithms have been run in the same experimental environment as the proposed algorithms. In addition, stopping criteria have been proposed to restrict the execution time to the times proposed by the researchers of the previous studies.

Finally, the third group of metrics focuses on algorithm robustness. There is no clear definition of this metric, although its objective is clear, to measure the ability of the algorithm to maintain its performance under a range of changing conditions or circumstances. For example, in the case of GLP these metrics should analyze the ability of the algorithm to find solutions to sets of input graphs with diverse topologies. Moreover, their resolution must be done with the same parameter adjustment, not being valid for a specific adjustment for each type of graph.

The above metrics are often combined with statistical analyses. The simplest analysis combines performance indicators by aggregation, averaging, or deviation operations that summarize the total experimentation into a single number. In this research, and in general in the context of single-objective optimization, we report the average value of the objective function, the deviation from the best solution found in the experiment (or known in the literature), and the number of best (or optimal) solutions found.

Other more complex analyses are based on statistical tests. Statistical tests can be a useful tool for analyzing and comparing the performance of different algorithms and can help to provide a scientifically valid basis for making conclusions about their effectiveness. In particular, in the context of this Doctoral Thesis, tests are used to determine whether there is a significant difference between the performance of two compared algorithms.

Given the particular properties of the problem, a specific statistical hypothesis testing tool is chosen. Statistical tests are commonly classified as parametric (there are assumptions about the distribution of the population from which the sample was taken) and nonnonparametric (the sample that does not follow any specific distribution) [6, 117]. Then, depending on the number of algorithms compared, different tests are used. For example, for one or two parametric samples, the most widely used statistical test is the Student's t-test [183], and for more than two samples the ANOVA test [118]. In this Doctoral Thesis, we propose and analyze algorithms comparing the quality, i.e., the objective function value, of the generated solutions. Since we assume that the sample formed by the values of the objective function does not follow any known distribution, nonparametric tests are needed.

In this case, when two algorithms are compared, the Wilcoxon test [251] test is used, while when three or more algorithms are compared, we use the Friedman test [265]. These tests can be easily performed with commercial software such SAS [220] or SPSS [74] and free software like R [198].

4.2 Preliminary testing

Most algorithms, and in particular heuristic and metaheuristic procedures, require the tuning of various parameters that influence the quality of the obtained results. The parameter values associated with the metaheuristics used must be the same for all instances and they are determinants in all the metrics mentioned above. Therefore, finding the best parameters for an algorithm can be considered an optimization problem itself, where the goal is to find the best performance of an optimization method provided through a series of test cases.

Parameter tuning strategies can be divided into two categories: offline and online. Offline parameter tuning involves setting the algorithm parameters prior to the execution, while online parameter tuning involves adjusting the parameters dynamically during execution. In particular, the research carried out in this Doctoral Thesis uses the first strategy, the offline parameter tuning [178, 238].

This parameter setting can be done manually or automatically. A manual parameter tuning is usually done by analyzing one parameter at a time, and its optimal value is determined empirically. In this case, no interaction between parameters is studied. To overcome this problem, "experimental design" is used [18]. Although manual tuning is time-consuming and does not guarantee to find the best algorithm configuration, it is very important from a scientific point of view as it provides great control and understanding of the tuning process and the proposed strategies.

In contrast, automatic tuning use software that often follows a common general scheme. Given a set of input parameters to configure the algorithm, it generates a list of possible configurations of the algorithm. Then, the algorithm to be tuned is evaluated over a preliminary set of instances and finally, the best configurations are returned. Automatic parameter setting usually finds the parameter setting faster, but overall, it is a less tedious procedure than manual setting. However, since it only provides a list of the best settings, it is often

less intuitive and difficult to understand the tuning process. In addition, a diverse and large set of test cases is needed for the correct operation of programs of this style to avoid possible biases when running over the full test set. Examples of automatic tuning software are *ParamILS* [121], *Sequential Model-based Algorithm Configuration* [120] or *irace* [153], among others [178].

In this Doctoral Thesis, both strategies are combined. In general, the same experimentation scheme is followed in the research carried out to address the problems studied. First, preliminary experiments are conducted to determine the best strategies for constructing a solution. Next, the best search strategies are determined, i.e., which neighborhood should be explored (insertions or swaps) and how it should be explored ("first improvement" or "best improvement" strategies). Then, the contribution of the proposed advanced strategies is analyzed. In general, these experiments, in addition to adjusting the parameterization of the algorithm, can be used to illustrate the behavior of these strategies and their contribution to the final performance of the algorithm. This experiment is followed by the configuration of the parameters related to the proposed metaheuristic as well as the algorithm termination criteria. Finally, before moving on to the competitive experiments, an analysis of the evolution and influence of the proposed strategies on the final proposed algorithm is performed. The following are examples of some experiments carried out in the research conducted in [31, 32, 33, 34].

As stated before, the first preliminary experiment focuses on the constructive configuration. Table 4.2 shows an example of the results reported when comparing different strategies for determining the best constructive procedure in the context of the CBS. In particular, Table 4.2 is extracted from [33] and it is devoted to analyzing the influence of the greedy selection λ (see Equation 3.7), in the performance of the constructive procedure based on the γ greedy criteria (see Equation 3.4)

As it can be observed, to compare the performance of the algorithms proposed, in each experiment, we usually report the averaged value of the objective function (Avg. *cbs*), the deviation from the best solution found in the experiment (Dev. (%)), the number of best solutions found in the experiment (#Best) and the running time in seconds (CPU T. (s)).

Although tables are usually the simplest and most straightforward way to analyze the

	$\gamma_w + Seq.Pattern$	$\gamma_w + \lambda$
Avg. *cbs*	18116.61	16838.72
Dev. (%)	5.93	0.36
# Best (18)	4	14
CPU T. (s)	1.35	1.46

Table 4.2 Influence of the greedy selection λ of the host vertex, in the performance of the constructive procedure proposed for the Cyclic Bandwidth Sum Problem.

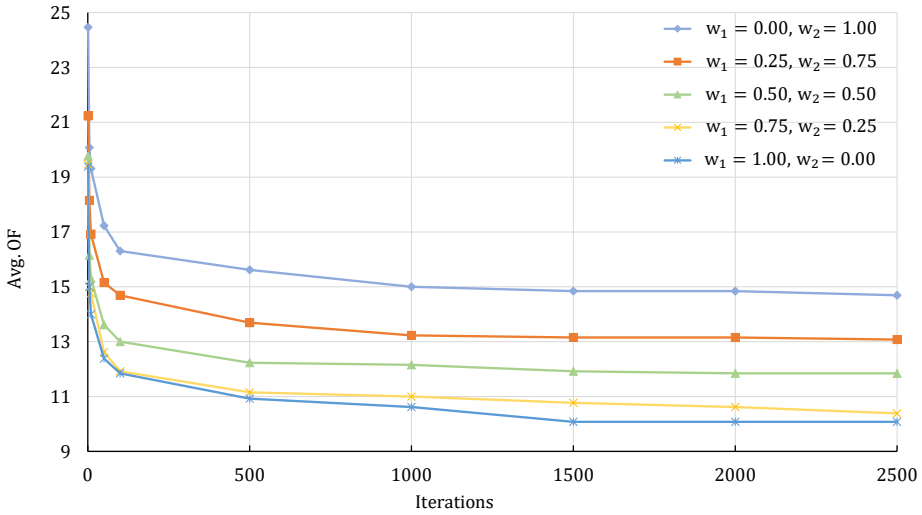


Figure 4.1 Evolution of the average objective function value when increasing the number of constructions for different values of w_1 and w_2 in γ_w for the constructive procedure proposed for the 2DBMP in [32].

performance of algorithms, graphs can also be useful for visualizing the performance of algorithms, especially when comparing the performance of multiple algorithms over a range of different input sizes or other variables.

For example, the in context of the 2DBMP we illustrated in Figure 4.1 the influence of the parameters w_1 and w_2 in γ_w that balance the influence of the adjacent assigned/unassigned vertices respectively (see Equation 3.4). Particularly, in Figure 4.1 we depict the average performance of the constructive procedure for five different configurations of these two parameters when the number of constructions increases from 1 to 2500.

The next experiment is usually devoted to analyzing the influence of incorporating the

	LS	LS+T	LS+T+E	LS+T+E+R
Avg. *2dbmp*	32.54	7.77	7.77	7.85
Dev. (%)	350.04	4.62	4.62	7.05
#Best	0	8	8	9
CPU T. (s)	72.51	7873.28	22.76	2.18

Table 4.3 Contribution of advanced strategies to the local search proposed for the Two-Dimensional Bandwidth Minimization Problem in [32].

proposed advanced strategies into the local search procedure. For example, Table 4.3, reported in the context of the 2DBMP in [32] study the contribution of each strategy proposed (i.e., the tiebreak criterion (T), the efficient move calculation (E), and the neighborhood reduction strategy (R) with the original local search procedure in isolation (LS). In a nutshell, it can be observed how the tiebreak criterion (LS+T) drastically improves the quality of the solutions obtained with respect to the original local search (LS) although the time increases considerably. Then, LS+T+E is able to reduce the time required to reach the same solutions by 99.71%. Finally, the reduction strategy (LS+T+E+R) reduces the time required for LS+T+E by an order of magnitude, although the average quality of the solutions obtained deteriorates slightly.

The next experiment is devoted to configuring the termination criteria or some parameters related to the particular metaheuristic algorithm proposed. In general, this experimentation is performed automatically with an automatic tuning software. In the developed research we used *irace* (Iterated Race for Automatic Algorithm Configuration) [153]. *irace* produces an output in the form of a table that summarizes the results of the configuration search process. The output typically includes the values of the performance metric for each combination of algorithm and hyperparameter settings that were evaluated, as well as any other relevant information, such as the running time or the number of iterations required to reach each combination. See for example [31] where we adjusted the largest neighborhood (k_{max}) and the maximum time for each iteration (t_{max}) of the proposed GVNS for addressing the CAB. The output obtained as a result of executing *irace* is depicted in Figure 4.2.

Finally, the final preliminary experiment explores the increase in solution quality when going from the constructive procedure to the local search with advanced strategies and the

```
# Iteration: 4
# nbIterations: 4
# experimentsUsedSoFar: 993
# timeUsed: 0
# remainingBudget: 7
# currentBudget: 7
# number of elites: 3
# nbConfigurations: 3
# Best configurations:
ID   k_max    t_max
1    0.01     30
2    0.01     25
3    0.01     40
```

Figure 4.2 Example output of *irace* when tuning the GVNS procedure proposed for the Cyclic Antibandwidth Problem in [31].

	Constructive	LS+T+E+R	MS-TS
Avg. *ccmp*	60.61	43.61	43.06
Dev. (%)	41.03	3.15	0.11
#Best	0	12	17
CPU T. (s)	0.01	0.51	2.14

Table 4.4 Performance differences between the procedure components and the full procedure proposed for the Cyclic Cutwidth Minimization Problem in [34].

final tuned procedure that is compared with state-of-the-art algorithms. As an example, Table 4.4 reports the results obtained for the algorithms proposed for the CCMP in [34]. As can be appreciated, local search significantly improves the quality of the solution obtained by the constructive procedure. Combining both procedures in the TS framework yields a further improvement in quality, although the difference between local search and TS is significantly smaller than the difference in solution quality between constructive and local search.

Before concluding this section, it is worth mentioning another type of preliminary experimentation, as important as what it has been discussed so far. In particular, another objective of the preliminary experimentation is to verify that everything works correctly

and as expected. For example, if the optima values are known, it can be expected that the proposed algorithm will never report solutions whose value of the associated objective function is higher/smaller than the global optimum. Another example could be to check that the efficient computation of the objective function reports the expected value.

The preliminary experimentation process usually ends when the most efficient algorithm and parameters configurations have been determined. In addition to verifying and validating the correct functioning of the implemented code. Then, it is then compared with the methods proposed in the state of the art.

4.3 Competitive testing

Competitive testing is commonly used as a way to assess the effectiveness of different approaches or strategies. In the context of this Doctoral Thesis, competitive tests compare the performance of the best state-of-the-art algorithms in finding the best solutions for an input data set. However, making a fair comparison is not as simple as comparing some of the metrics discussed in Section 4.1.2. In general, it is important to keep in mind that there is no "best" algorithm, but rather the best algorithm for a specific task and set of requirements. Different algorithms have different strengths and weaknesses, and the algorithm that is best suited for a given might depend on several factors, including the characteristics of the data set, the performance metrics that are most significant, and the resources available.

For example, exact algorithms are usually slower than heuristic procedures. However, they are among the few algorithms capable of determining whether the solution reached corresponds to the optimal solution of a problem. On the other hand, in the area of GLPs one can find algorithms that focus on solving some specific input graphs, while others are of a more general nature. Also, extremely fast algorithms can be found, capable of finding good quality solutions in reduced computation times. These algorithms should be carefully compared with other algorithms aiming to find better-quality solutions without considering short execution times.

Another aspect to consider in competitive experimentation is the number of algorithms to be compared. Comparing a few algorithms may not provide a complete survey of the

	CCMP	CAB	CBS	2DBMP
Upper/lower bounds	✓	✓	✓	✓
Exact algorithms	0	0	0	3
Approximate algorithms	1	2	3	1

Table 4.5 Algorithmic proposals existing in the state of the art for the problems studied in this research.

options available in solving a problem, while comparing many algorithms may be time-consuming, making it difficult to draw meaningful conclusions, or consider algorithms that are of minor importance to the comparison.

In the case of competitive comparison of algorithms for solving optimization problems, and, more specifically in GLPs, the fairest comparison is one in which exact algorithms, which can ensure that the found solution is optimal, and heuristic algorithms, focused on the problem at hand, are compared. It may also be of interest to include in the comparison upper/lower bounds obtained both through theoretical studies and exact/approximate algorithms.

Table 4.5 shows, for the four problems studied in this Doctoral Thesis, the number of algorithmic proposals present in the state of the art. These proposals are organized by the existence, or not, of lower/upper bounds, exact algorithms, or approximate algorithms (usually heuristic and metaheuristic procedures). Note that the algorithms proposed in this research are not reported in the table.

When performing the competitive tests, we always compare our proposal with the exact algorithm, if it exists, and with the best approximate procedures. Sometimes, two or more algorithms perform well in different sets of instances. For example, for the CAB and the 2DBMP, our proposal is compared with two state-of-the-art algorithms.

To conclude this section, Table 4.6 presents a summary of the competitive tests carried out in each studied problem, comparing the best algorithm of the state of the art at that moment, and our best algorithmic proposal. Note that for the CAB two algorithms in the state of the art are compared, and the objective was to maximize an objective function value. Although the following chapter contains the most relevant conclusions of this research, it is worth mentioning some of them based on the results compiled in Table 4.6.

		State of the art		Our proposal
CCMP	Avg. *ccw*		57.94	57.24
	Dev. (%)		3.54	0.69
	#Best (179)		141	157
	CPU T. (s)		1434.01	177.42
CAB	Avg. *cab*	149.39	164.70	164.09
	Dev. (%)	21.11	7.57	5.45
	#Best (267)	99	94	167
	CPU T. (s)	150.00	150.00	150.00
CBS	Avg. *cbs*		60209.01	59408.65
	Dev. (%)		6.14	0.10
	#Best (106)		50	100
	CPU T. (s)		6019.53	1485.50
2DBMP	Avg. *2dbmp*		4.82	3.57
	Dev. (%)		32.90	0.00
	#Best (86)		40	86
	CPU T. (s)		231.77	53.39

Table 4.6 Summary of the comparison of the best-proposed algorithms with the best state-of-the-art algorithms for the CCMP, CAB, CBS, and 2DBMP, respectively.

Overall, the results suggest that each proposed algorithm is generally superior to the state-of-the-art algorithm in terms of solution quality. The proposed algorithm consistently achieves higher average solution quality, lower deviation from the best values found, a larger number of best solutions found, and a shorter running time. These results lead to affirm that the proposed algorithms are promising approaches for solving optimization problems and, in particular, to GLPs.

It is worth noting that the specific conclusions that can be drawn from the results are available in the articles attached to this dissertation. In particular, Chapter 7 contains the results for the CCMP, Chapter 8 the results for the CAB, Chapter 9 the results for the CBS and finally, Chapter 10 the results for the 2DBMP.

Chapter 5

Conclusions and future work

The last chapter of this Doctoral Thesis summarizes, in Section 5.1, the main conclusions and contributions of the research carried out. The conclusions also include an analysis of the degree of achievement of the proposed objectives, as well as the validation of the hypothesis set at the beginning of this research. Finally, Section 5.2 presents research lines that remain open and that would allow researchers in the field to continue with the investigation initiated in this Doctoral Thesis.

5.1 General conclusions

In this Doctoral Thesis, four optimization problems belonging to the GLPs family of problems have been addressed. These problems consist of finding an embedding of an input graph in a host graph with a well-known topology or structure, in a way that a given objective function is optimized. In particular, three of the studied problems perform the embedding of general graphs on a cycle host graph, while the other one does it on a grid host graph. Moreover, the objective functions optimized are based on two mathematical functions, the cutwidth, and the bandwidth.

The GLP family of problems is a broad and diverse class of optimization problems that have many practical applications, including network design, resource allocation, and graph drawing. Solving these problems efficiently and accurately is an important challenge since most of them belong to the NP-hard or NP-complete category. Since exact algorithms are

inefficient for solving these types of problems, it is necessary to propose techniques that, although they are not able to certify if the solutions found are optimal, obtain high-quality solutions in reduced computation times. Among the possible techniques to address these problems, heuristic and metaheuristic algorithms have been proven to be very effective. Consequently, heuristic and metaheuristic algorithms are proposed in this dissertation to tackle the studied GLPs.

These ideas are the basis hypothesis stated in this Doctoral Thesis. This hypothesis has been successfully validated for the three problems studied: the Cyclic Cutwidth Minimization Problem, the Cyclic Antibandwidth Problem, and the Cyclic Bandwidth Sum Problem. Then, the hypothesis is subsequently extended by modifying the host graph in which the embedding is performed (a grid) and it has been successfully validated for the Two-Dimensional Bandwidth Minimization Problem.

In order to corroborate that the initial hypothesis is true, different objectives have been set and have consequently guided the research. The following is an analysis of the degree of achievement of the proposed objectives.

The first objective was to review and analyze the current state of the art. This objective has been achieved, since the research project includes an exhaustive review of the literature for each of the problems studied, and a more general review of related problems. In addition, this review includes an analysis of existing approaches to solving the problems.

The second objective was to obtain the properties and structural characteristics of the problem. This objective has also been achieved, as the proposed strategies in this research exploit some characteristics of the problems. The objective function tie-breaking criterion or neighborhood reduction techniques are a clear example of such strategies.

The third objective was to design and develop a heuristic algorithm to solve each particular problem. Based on the information provided, the objective has been achieved, as the research includes the development of a new algorithmic approach for each problem tackled. Although the proposed algorithms are described in detail in the accompanying papers, the first part of this dissertation provides insight into the work developed.

The fourth objective was to experimentally compare the proposed algorithm with state-of-the-art algorithms. The research project includes a thorough evaluation of the proposed

algorithm using a variety of experimental methods: preliminary experiments made of automatic and manual tuning, competitive testing, or statistical analysis, among others. The results of these experiments are compared to those obtained using other existing algorithms, providing a comprehensive assessment of the performance of the proposed algorithm.

The fifth objective was to elaborate a document that includes the work carried out and the conclusions of the results obtained. The elaboration of this dissertation is a clear proof of the achievement of this objective. However, other reports have been prepared for purely dissemination purposes that are more closely related to the objective mentioned next.

Finally, the sixth objective was to disseminate the results by publishing them in research forums. This objective has also been achieved, as this research includes several publications in scientific forums such as journals and conferences. Participation in conferences has helped to share the results of the study with the research community in general and has also allowed us to benefit from the comments and suggestions of other researchers. Similarly, the reviewers proposed by the journals to evaluate our articles have enriched and improved our research.

Overall, it seems that the objectives stated at the beginning of this Doctoral Thesis have been achieved, and the most relevant results have been of interest to the scientific community, allowing their dissemination in different forums.

Regardless of the fulfillment of the objectives of the Doctoral Thesis, the approach of the algorithms has allowed obtaining knowledge and experience that can be applied to other problems or in future research, not necessarily bound to GLPs. In addition, although the articles published for each problem have their own conclusions, it is possible to draw general conclusions from the research conducted. The most relevant conclusions learned from this research are summarized in eleven points:

1. This work presents a general formalization of GLPs, which involves defining the problem in precise and mathematical terms. This formalization provides a framework for understanding the key features and challenges of these problems and helps to identify commonalities and differences between problems in the GLPs family.

2. Search horizons in max/min or min/max problems require alternative objective functions. This highlights the importance of carefully considering the objective function

when designing algorithms for these problems and suggests that different approaches may be needed to optimize different objective functions.

3. There are common strategies that can be applied to multiple related problems. This suggests that it may be possible to develop general-purpose algorithms or approaches that are effective across a range of different graph embedding problems. It is from this conclusion that a line of research emerges and will be described in Section 5.2.

4. The strategy for constructing an initial solution is key in all the algorithms proposed in this research. The proposed scheme is considered as a guide to successfully tackle a GLP. In particular, the step-by-step construction based on selecting a vertex of the input graph and assigning it to a vertex in the host graph has a major impact on the efficiency and effectiveness of the proposed algorithms. Tailoring these strategies to the specific characteristics and requirements of the problem has been shown to be successful in generating good initial solutions.

5. The combination of greedy constructive techniques with a multi-start approach can be a powerful approach for solving optimization problems, particularly when combined with a local search. This approach allows for the rapid construction of good initial solutions, which can be improved using local search procedures to find high-quality solutions.

6. Classic neighborhoods, such as swaps and inserts, perform well for GLPs. Not only for combinatorial optimization problems but also for GLPs, the swap and insertion operators allow finding good quality solutions when exploring neighborhoods by local search algorithms. Moreover, their combination with the efficient computation of the objective function or neighborhood reduction techniques makes them key elements to tackle a problem of these characteristics.

7. In general, it is not necessary to explore a neighborhood exhaustively in order to find an improved solution in a GLP. Instead, it is usually sufficient to explore a subset of potential solutions to find an improvement. This observation is based on the fact that GLPs may have a complex search space. Instead, it is likely to find a good solution by exploring a subset of the search space and applying local search techniques to

improve the solution. In general, those potential solutions can be determined by the degree of the vertices that are going to be moved or the contribution of a vertex to the objective function if it is moved.

8. Given the formalization of a GLP, one solution might have multiple equivalent solutions. That is, one solution can be considered equal to another when the relative position of each vertex with respect to its adjacents is the same. In that case, it can be understood visually as the geometric motion of translation and rotation of an embedding. This issue results in neighborhoods composed of numerous equivalent solutions. Moreover, this opens a line of future work in the detection of these equivalent solutions, but also in the exploitation of these properties to propose more efficient methods.

9. Advanced search strategies, such as those proposed in this research, are an important aspect of optimization problem-solving and can be considered a combination of programming knowledge and scientific expertise. Typically, these procedures are created and used by scientists and researchers who may not be proficient in programming but who have thorough knowledge in understanding of the problem and the algorithms and strategies needed to solve it. Therefore, they should not be undervalued by the computer science community. Indeed, they should be reflected in the disclosures made to increase community awareness.

10. There is a lack of exact algorithms for the resolution of GLPs. Given the complexity of the problems, it seems reasonable that exact algorithms have hardly been proposed for these types of problems. Possibly, the combination of heuristic and exact algorithms, such as matheuristics, can result in effective methods to solve some problems belonging to the GLP family.

11. Finally, the representation of the solutions by its drawing is fundamental to analyzing the correct functioning of the algorithm and proposing more effective strategies. Furthermore, by observing the visualizations of the solutions, it is possible to elaborate conjectures or ideas of how close a solution may be to optimality.

Overall, the learned lessons listed here offer valuable insights into the nature of graph embedding problems and suggest directions for future research and the development of more efficient algorithms for solving related problems. In fact, at the time of writing this dissertation, other researchers have based their algorithmic proposal on the research reflected here, being able to improve the results obtained by our proposed methods [110]. This can be considered as proof that our work has contributed significantly to the advancement of knowledge in the GLP field, and it has had a real impact on the scientific community.

The specific conclusions obtained for each particular problem and possible lines of research derived from these conclusions are presented in each of the papers compiled in Part II of this dissertation.

5.2 Future lines of research

During the development of a Doctoral Thesis, it is common to identify open lines of work, which can be an alternative starting point for future research. As a result of the research process initiated with this Doctoral Thesis, the following future work is proposed:

1. **Exact and matheuristic algorithms**. Future research may focus on developing exact methods to solve some GLP. Although, given the difficulty of the problems, they will allow checking if the quality of the solutions obtained for the smallest instances corresponds to the optimal one. This can entail developing branch-and-bound algorithms that effectively search across the space of potential solutions to identify the best one, as well as developing mathematical models that can be used to assess and optimize solutions. In addition, the proposition of such techniques may lead researchers to combine them with heuristic algorithms that eventually result in matheuristic algorithms.

2. **Review on graph embeddings**. It may also be useful to extend the review of graph embedding problems to other host structures. This could involve reviewing the literature to identify major research contributions and trends and synthesizing this information into a coherent view of the current state of the field. Such a study could be

a useful resource for researchers who wish to understand the current research landscape and identify potential areas for future research. In addition, the last paper of this nature was published in 2002 [65] and, since then, a wide variety of papers and applications have been published.

3. **GLP variants**. Another important area of research is the study of other graph embedding problems. This may involve exploring different types of input graphs and objective functions, as well as examining the feasibility of embedding graphs in different types of host graphs. By broadening the scope of these problems, it may be possible to gain a deeper understanding of the underlying challenges and find more general solutions. This line of research may result in future work raised in the next item.

4. **Black-box algorithm**. Possible research to be carried out may be aimed at developing a black-box algorithm. This could involve combining and applying the knowledge gained from previous research efforts in order to create a more comprehensive and effective solution, for more general input and host graphs.

5. **Multiobjective optimization**. A potential new research direction in the field of graph embedding could focus on the development of algorithms and approaches for solving multiobjective optimization problems. In this context, a multiobjective optimization problem is one with multiple conflicting objectives that need to be optimized simultaneously. This type of problem has not been sufficiently studied or raised in the field of graph embedding.

6. **Neighborhoods**. Possible future research could include the development of more advanced neighborhoods for use in local search algorithms. Generally, neighborhoods used in this Doctoral Thesis are based on insertion or exchange moves or a combination of these. However, it is reasonable to consider other more complex and sophisticated operators. For example, a move that removes some vertices from the input graph, then solves the reduced problem, and finally inserts back the removed vertices, could be considered. Other neighborhoods could implement operators involving multiple vertices (more than two vertices), for example.

7. **Equivalent solutions**. Another possible research line could focus on studying how equivalent solutions may affect the performance and effectiveness of the algorithms. As stated, embeddings into a cycle and grid host graphs lead to many equivalent solutions which, further than having the same objective function, are indistinguishable in terms of their representation. This could involve research into the use of techniques such as symmetry breaking or the use of additional constraints that could significantly reduce the search space.

8. **Graph drawing software**. Finally, the development of graph drawing software has a dual benefit. On the one hand, it could be a useful tool for researchers who study graph embedding problems. Since, as mentioned above, such a tool could make it easier for users to analyze graphs in order to propose more effective and efficient algorithms. On the other hand, the development of this tool would allow the general public to visualize generic graphs represented by cyclic or grid structures.

In conclusion, the above ideas highlight the significant opportunity for progress in the area of GLPs by conducting novel research. Researchers pursuing the suggested topics of investigation can extend and improve the existing algorithms, as well as conduct a theoretical analysis of interest in the area. When combined, both perspectives offer the potential to reveal new insights in this intrepid field.

Part II

Publications

Chapter 6

Overview

In this Doctoral Thesis, several contributions to the body of knowledge related to the GLPs are presented, drawing on a variety of techniques and tools from computer science and mathematics that have been applied to the studied problems.

Specifically, this second part of the dissertation is divided into six chapters. This is the first of the chapters and aims to contextualize chronologically each of the research studies performed and the associated publications, both in journals and scientific forums. Then, a total of four chapters are devoted to the problems addressed in this Doctoral Thesis. Specifically, one chapter per optimization problem (Chapter 7 for the CCMP, Chapter 8 for the CAB, Chapter 9 for the CBS and Chapter 10 for the 2DBMP). For each of these problems, the related publications and their most relevant contributions are presented. Finally, the last chapter (Chapter 11) presents other publications related to the research presented in this dissertation.

Figure 6.1 presents a chronological representation of the main events and publications along a timeline, with the different milestones reached in this Doctoral Thesis. In this figure, each milestone is classified by color, where publications are identified with green, participation in conferences or workshops with yellow, and notable events with red.

As can be observed in Figure 6.1, research on GLPs began at the end of 2018 in the context of a grant developed at the Universidad Politécnica de Madrid, where the author was studying for a B.Sc. in Software Engineering. This grant was requested by the author of this Doctoral Thesis, together with one of the supervisors, Eduardo G. Pardo. During the

first few months, an exhaustive study of all GLPs was performed, finally focusing on the CGLPs. Specifically, for each of the problems studied in the literature, the most relevant contributions, previous algorithms, sets of instances used, the most relevant authors, etc. were identified. Based on this previous state-of-the-art research, the focus was placed on the CCMP. The preliminary results obtained were subsequently presented at the GRAFO Workshop. In addition, an article summarizing the research on the CCMP was written and accepted, and it was published in February 2021 (see Chapter 7) [34].

The B.Sc. studies were complemented by M.Sc. studies in Artificial Intelligence. During this period, research began on the second problem presented in this dissertation, the CAB. In addition, the Ph.D. candidate participated in two workshops and started doctoral studies at the Universidad Rey Juan Carlos, becoming part of the GRAFO research group. Furthermore, the Ph.D. was partially financed by one of the most recognized predoctoral scholarships in Spain, the *Ayudas para la formación de profesorado universitario (FPU)*, applied for in collaboration with the other supervisor, Abraham Duarte. With the start of doctoral studies in October 2020, research on the third problem addressed, CBSs, began. The first results obtained for the CAB were presented at an international conference, severely affected by the Covid-19 pandemic, in March 2021. During the same year, the study on the 2DBMP began, and a presentation was made at a workshop and a national conference. Within the framework of this conference, an article in a Scimago Journal & Country Rank (SJR) journal was published. In particular, [28] presented some results for the CCMP not included in [34].

At the beginning of 2022, two articles were published in Journal Citation Reports (JCR) journals (CAB [31] and CBS [33]). In addition, presentations were made at various scientific forums: a national conference, a workshop, and an informative contest. In the latter contest, the doctoral candidate was awarded as the winner in the Three Minute Doctoral Thesis (3MT) competition among the best participants of public universities in Madrid in the area of engineering and architecture. In July 2022, a research stay at the University of Colorado Boulder (Denver, USA) supervised by Professor Manuel Laguna was made. During this stay, the Ph.D. candidate strengthened ties and collaborations with renowned researchers in the area of heuristics and metaheuristics, such as Fred Glover and Rafael Martí. During the stay, the fourth article of this Doctoral Thesis was published [32] (see Chapter

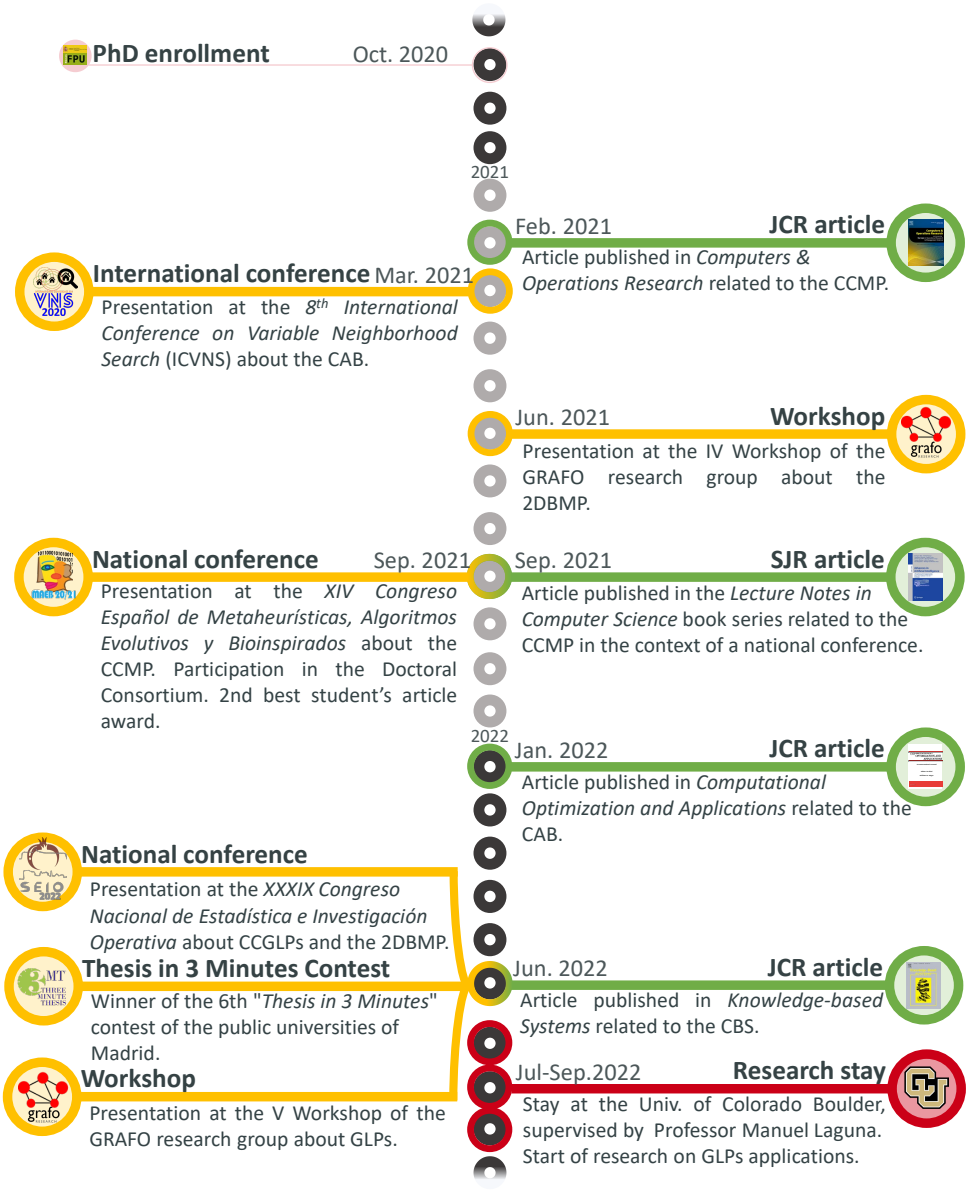
10). The year ended with a presentation at an international conference and a workshop. In addition, an article not directly related to the topic of this Doctoral Thesis was published [161]. In this publication, we studied the methodological and reproducibility perspectives when addressing optimization problems through heuristic or metaheuristic approaches (see Chapter 11). Afterward, the doctoral candidate focused on writing and defending the Doctoral Thesis.

In the following chapters, publications in JCR journals are included, together with a figure with relevant information about the publication: title, authors, corresponding author, DOI, volume, article number, date of publication, number of records obtained to date and name of the journal. In addition, information related to the journal is included: research areas, the rank of the categories, and the Journal Impact Factor. All information related to the journal has been extracted from the Web of Science[1] platform.

[1]Web of Science is a citation database platform provided by Clarivate Analytics, widely used by researchers to find relevant literature, track research trends, and analyze citation metrics. Web of Science can be accessed at `https://www.webofscience.com`.

continued on the next page

Figure 6.1 Timeline of the relevant events associated with this Doctoral Thesis.



continued on the next page

Figure 6.1 Timeline of the relevant events associated with this Doctoral Thesis.

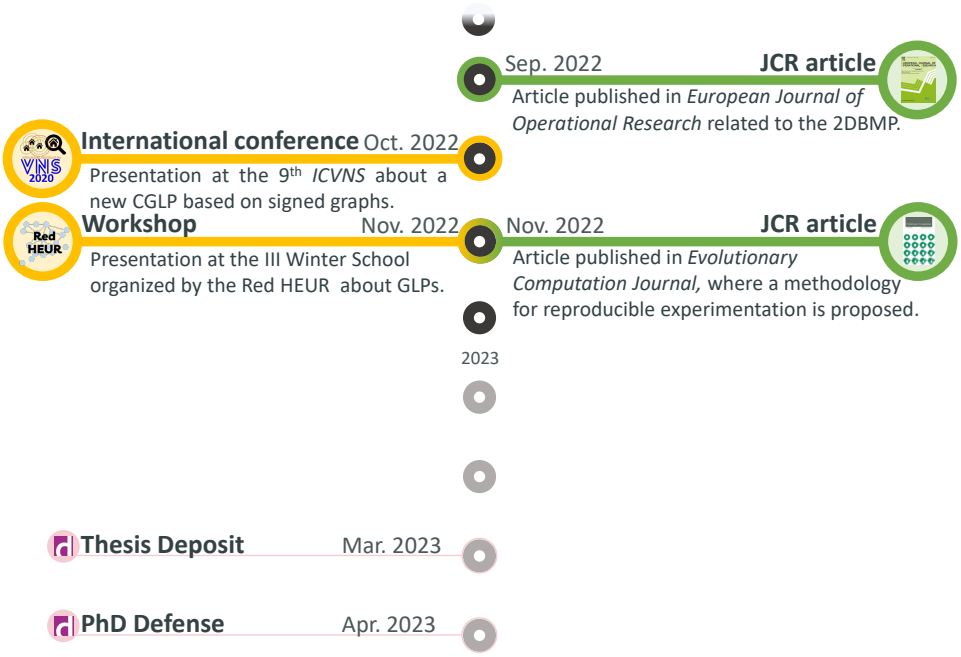


Figure 6.1 Timeline of the relevant events associated with this Doctoral Thesis.

Chapter 7

Cyclic Cutwidth Minimization Problem

The Cyclic Cutwidth Minimization Problem is the first of the GLPs studied in this Doctoral Thesis and it was previously introduced in Section 2.1. As a result of the research conducted, two articles have been published:

1. S. Cavero, E. G. Pardo, M. Laguna, and A. Duarte. Multistart search for the cyclic cutwidth minimization problem. *Computers & Operations Research*, 126:105116, 2021 [34].

2. S. Cavero, E. G. Pardo, and A. Duarte. Influence of alternative objective functions in the optimization of the cyclic cutwidth minimization problem. In *Advances in Artificial Intelligence: 19th Conference of the Spanish Association for Artificial Intelligence, CAEPIA 2020/2021, Málaga, Spain*, pages 139–149. Springer, Cham, 2021 [28].

Moreover, a presentation has been made at a national conference:

3. S. Cavero, E. G. Pardo, and A. Duarte. Influence of the alternative objective functions in the optimization of the cyclic cutwidth minimization problem. *XIX Conferencia de la Asociación Española para la Inteligencia Artificial* (CAEPIA 2021), in Málaga, Spain, 2021 [29].

Among the previous publications, the article, titled: *"Multistart search for the Cyclic Cutwidth Minimization Problem"* [34], was published in a JCR journal. Figure 7.1 compiles

some information about the journal. Note that, in addition to the Ph.D. candidate and his supervisors, Professor Manuel Laguna from the University of Colorado Boulder (Denver, USA) also participated in this research. The problem tackled in this research, the CCMP has previously been studied for specific classes of graphs with regular structure. However, work on general candidate networks is scarce. We can only point out a population-based metaheuristic of the GA family. In this investigation, we take a different approach and develop a single-solution neighborhood search based on TS. The main contributions of this proposal are summarized next:

- **Greedy constructive procedure** based on the adjacency of the vertices. The vertices are selected according to a greedy criterion originally inspired by the ideas presented in [168]. Then, each selected vertex is assigned sequentially to an available host vertex.

- **Local search procedure** based on the insert neighborhood which is explored following a best improvement strategy. This local search procedure is further improved with three advanced strategies: an alternative objective function, an efficient calculation of the objective function, and a neighborhood reduction strategy to explore just those potential better solutions.

- The previous heuristic procedures are combined with two well-known metaheuristics: **Tabu Search** (TS) [87, 90] and a **Multistart procedure** [163, 166]. On the one hand, tabu short-term memory is added to the aforementioned local search to store attributes of previously visited solutions. On the other hand, the multistart strategy is used to escape from local optima by starting the search again from a different point in the solution space.

Finally, our proposal was compared to the previous state-of-the-art algorithm. The TS solutions are found in about one order of magnitude less time than MA. The performance of TS is better on random graphs than on structured graphs. For all instances, the solutions obtained are on average closer to the best solutions than the GA solutions. Furthermore, statistical tests indicate the merits of our proposal.

Multistart search for the Cyclic Cutwidth Minimization Problem

Sergio Cavero, Eduardo G. Pardo, Manuel Laguna and Abraham Duarte

Computers & Operations Research. Volume 126, 105116, 2021.

https://doi.org/10.1016/j.cor.2020.105116

Journal Information

Research Areas:

- Computer Science (Interdisciplinary Applications)
- Engineering (Industrial)
- Operations Research & Management Science.

Category Rank:

- Operations Research & Management Science: 20/87 (Q1)
- Computer Science (Interdisciplinary Applications): 35/113 (Q2)
- Engineering (Industrial): 19/50 (Q2)

Journal Impact Factor: 5.159

Data obtained from Journal Citation Reports 2021

Figure 7.1 Journal information related to the publication [34].

In the second publication related to the CCMP titled *"Influence of alternative objective functions in the optimization of the cyclic cutwidth minimization problem"*, we extended the previous article by studying different alternative objective functions and their combinations were analyzed in the search for better quality solutions beyond flat landscapes. The article is published in the journal *Advances in Artificial Intelligence* as part of the collection *Lecture Notes in Artificial Intelligence* (SJR / Q2). In particular, we analyze the influence in the search of using alternative objective functions within local search procedures to deal with the flat landscape problem presented in Section 1.1.2. In addition, we propose different alternative objective functions for the CCMP and compare its performance.

To conclude this chapter, we include a copy of the most relevant paper published for the CCMP in the context of this Doctoral Thesis.

Computers and Operations Research 126 (2021) 105116

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/caor

Multistart search for the Cyclic Cutwidth Minimization Problem

Sergio Cavero [a], Eduardo G. Pardo [a,*], Manuel Laguna [b], Abraham Duarte [a]

[a] Universidad Rey Juan Carlos, Spain
[b] University of Colorado Boulder, USA

ARTICLE INFO

Article history:
Received 21 March 2020
Revised 30 September 2020
Accepted 5 October 2020
Available online 15 October 2020

Keywords:
Cyclic cutwidth
Graph layout problem
Circular embedding
Tabu Search
Multistart search

ABSTRACT

The Cyclic Cutwidth Minimization Problem (CCMP) is a Graph Layout Problem that consists of finding an embedding of the vertices of a candidate graph in a host graph, in order to minimize the maximum cut of a host edge. In this case, the host graph is restricted to be a cycle. In this paper, we identify a new lower bound for the problem, and also a property which allows search procedures to drastically reduce the neighborhood size during the search. Additionally, we propose the use of an alternative objective function for min–max optimization problems, never used before in the context of the CCMP. These strategies have been combined within a multistart search procedure for this problem. The obtained method is compared with the state of the art for the CCMP using sets of problem instances previously published. Statistical tests indicate the merit of our proposal.

1. Introduction

Some families of optimization problems can be represented as graph layout problems where the objective consists of defining a mapping of a candidate graph into a regular structure, called the host graph. In these problems, there are two mapping functions. The first one assigns each vertex in the candidate graph (candidate vertex) to a vertex in the host graph (host vertex). The second function assigns to each edge in the candidate graph (candidate edge) a path in the host graph (host path). The most common approaches found in the literature for this family of problems are those where the host graph is a line. These problems are commonly referred to as linear layout problems (Pardo et al., 2016). However, there exist mappings over more complex regular structures such as trees, grids, or cycles (Díaz et al., 2002). Regardless of the structure of the candidate and host graphs, the optimization problem may be defined over several objective function choices.

We tackle a minimization problem consisting of embedding a general candidate graph in a cycle host graph. Let $\mathcal{C} = (\mathcal{V}_\mathcal{C}, \mathcal{E}_\mathcal{C})$ be a connected, unweighted, and undirected candidate graph where $\mathcal{V}_\mathcal{C}$ and $\mathcal{E}_\mathcal{C}$ represent the sets of vertices and edges, respectively. Analogously, let $\mathcal{H} = (\mathcal{V}_\mathcal{H}, \mathcal{E}_\mathcal{H})$ be a host cycle graph with the following properties:

- $n = |\mathcal{V}_\mathcal{C}| = |\mathcal{V}_\mathcal{H}| = |\mathcal{E}_\mathcal{H}|$.
- The degree of each vertex $v \in \mathcal{V}_\mathcal{H}$ is 2.
- $\mathcal{H}$ is a Eulerian and Hamiltonian graph.
- The disposition of the vertices $\mathcal{V}_\mathcal{H}$ in the Euclidean space is such that all adjacent vertices are placed at the same distance.

Considering these definitions, the host graph in our context is represented as a cycle. A bijective function φ assigns each vertex in the candidate graph to a single vertex in the host graph. This function is defined as $\varphi : \mathcal{V}_\mathcal{C} \rightarrow \mathcal{V}_\mathcal{H}$, where $\forall\, v \in \mathcal{V}_\mathcal{C}\ \exists\ w \in \mathcal{V}_\mathcal{H}$ such that $\varphi(v) = w$. An injective function ψ assigns candidate edges to host paths. A path is a sequence of edges that connects two vertices without repeating any edges or vertices. Let $\mathcal{P}_\mathcal{H}$ be the set of all possible host paths in $\mathcal{H}$. Then, since $\mathcal{H}$ is a cycle, for every candidate edge $(u, v) \in \mathcal{E}_\mathcal{C}$ there are two possible paths in $\mathcal{P}_\mathcal{H}$, starting in $\varphi(u)$ and ending in $\varphi(v)$. The ψ function is defined as the mapping of candidate edges to host paths, i.e., $\psi : \mathcal{E}_\mathcal{C} \rightarrow \mathcal{P}_\mathcal{H}$.

Fig. 1 shows an example of a candidate graph ($\mathcal{C}$), a host graph ($\mathcal{H}$), a possible embedding of the candidate graph in the host graph, and the two possible host paths between a pair of adjacent vertices (A and D). In particular, Fig. 1(a) shows a candidate graph with $\mathcal{V}_\mathcal{C} = \{A, B, C, D, E, F\}$ and $\mathcal{E}_\mathcal{C} = \{(A, B), (A, D), (A, E), (A, F), (B, C), (B, D), (C, D), (D, E)\}$. Therefore, the host graph must have the same number of vertices $\mathcal{V}_\mathcal{H} = \{1, 2, 3, 4, 5, 6\}$ and a set of corresponding edges $\mathcal{E}_\mathcal{H} = \{(1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 1)\}$, as presented in Fig. 1(b). One possible mapping of the candidate graph into the host graph is showed in Fig. 1(c). Each vertex in $\mathcal{V}_\mathcal{C}$ is assigned to a vertex in $\mathcal{V}_\mathcal{H}$. For instance, vertex A in $\mathcal{V}_\mathcal{C}$ is assigned to vertex 1 in $\mathcal{V}_\mathcal{H}$. The assignment is denoted by $\varphi(A) = 1$. Similarly, vertex B is assigned to vertex 4 ($\varphi(B) = 4$),

* Corresponding author. C/Tulipán s/n, 28933, Móstoles, Spain.
E-mail addresses: sergio.cavero@urjc.es (S. Cavero), eduardo.pardo@urjc.es (E.G. Pardo), laguna@colorado.edu (M. Laguna), abraham.duarte@urjc.es (A. Duarte).

https://doi.org/10.1016/j.cor.2020.105116
0305-0548/

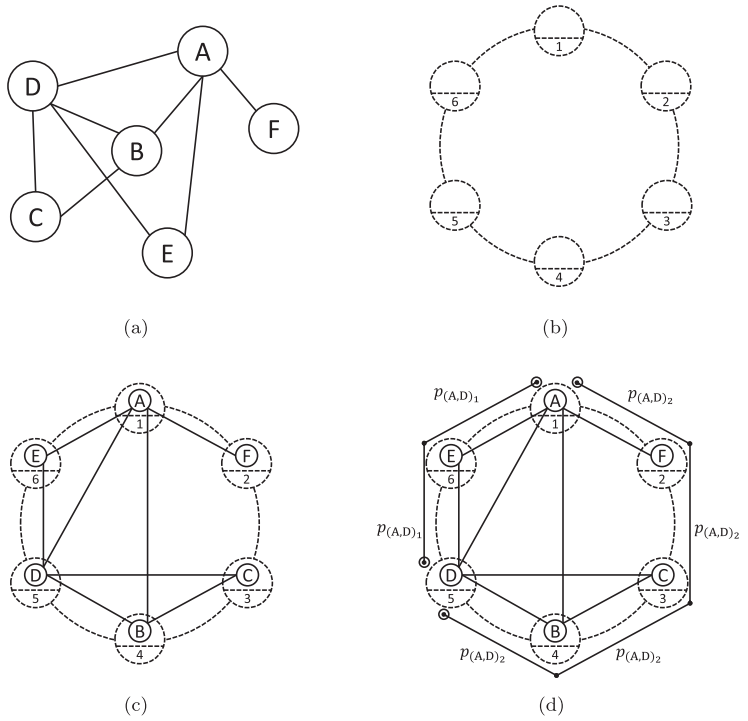


Fig. 1. (a) A candidate graph, $\mathcal{C}$. (b) A host graph, $\mathcal{H}$. (c) A possible embedding of $\mathcal{C}$ in $\mathcal{H}$. (d) Two possible paths in $\mathcal{H}$ for $(A, D) \in V_{\mathcal{C}}$ named as $p_{(A,D)_1}$ and $p_{(A,D)_2}$ respectively.

and so forth for all other candidate vertices. Then, candidate edges are assigned to host paths. For instance, edge (A, D) must be assigned to a host path with $\varphi(A) = 1$ and $\varphi(D) = 5$ as terminal vertices. The possible paths are $p_{(A,D)_1} = \{\varphi(A), \varphi(E), \varphi(D)\} = \{1, 6, 5\}$ and $p_{(A,D)_2} = \{\varphi(A), \varphi(F), \varphi(C), \varphi(B), \varphi(D)\} = \{1, 2, 3, 4, 5\}$. These paths are shown in Fig. 1(d). Note that for any candidate edge, there are only two possible host paths (clockwise and counterclockwise) when the host graph is a cycle. We further define ψ as a function that selects the shortest path. The length of the path is determined by the number of host edges traversed. In our example, the length of $p_{(A,D)_1}$ is 2 and the length of $p_{(A,D)_2}$ is 4. Therefore, $\psi(A, D) = p_{(A,D)_1} = \{1, 6, 5\}$. The function must be applied to all candidate edges $\mathcal{E}_{\mathcal{C}}$. When the length of both host paths is the same, ψ selects the clockwise path.

1.1. Problem description

We study the Cyclic Cutwidth Minimization Problem (CCMP) for general candidate graphs and cycle host graphs. Our objective function for the CCMP is based on the concept of a cut of an edge in the host graph (i.e., edges in $\mathcal{E}_{\mathcal{H}}$). Given the assignment functions φ and ψ, the cut of an edge $e \in \mathcal{E}_{\mathcal{H}}$ is defined as the number of host paths assigned by ψ that traverse e. This calculation has been referred to in the literature as congestion (Rolim et al., 1995).

Formally, we define the cut of host edge $(w, z) \in \mathcal{E}_{\mathcal{H}}$ associated with φ and ψ as:

$$\underset{(w,z)\in\mathcal{E}_{\mathcal{H}}}{\text{cut}}((w, z)\varphi, \psi) = |\{(u, v) \in \mathcal{E}_{\mathcal{C}} : (w, z) \in \psi(u, v)\}|, \tag{1}$$

where the $\psi(u, v)$ path is defined as:

$$\psi(u, v) = \begin{cases} \{w, z\} & \text{if } \varphi(u) = w \wedge \varphi(v) = z \\ \{w, z, \ldots, \varphi(v)\} & \text{if } \varphi(u) = w \\ \{\varphi(u), \ldots, w, z\} & \text{if } \varphi(v) = z \\ \{\varphi(u), \ldots, w, z, \ldots, \varphi(v)\} & \text{otherwise.} \end{cases} \tag{2}$$

The objective function (ccw) is denoted as the cyclic cutwidth and is calculated as follows:

$$ccw(\mathcal{C}, \varphi, \psi) = \max_{(w,z)\in\mathcal{E}_{\mathcal{H}}} \text{cut}((w, z)\varphi, \psi). \tag{3}$$

Since paths can be derived from the vertex assignments (see Eq. 2), a solution is fully characterized by φ. Therefore, for the purpose of the optimization problem, we can simplify the notation for ccw, by making it depend only on $\mathcal{C}$ and φ. Our min–max optimization problem consists of finding, among all possible candidate vertex assignments $\varphi \in \Phi$, the assignment $\varphi^\star$ that minimizes the cycle cutwidth:

$$\varphi^\star \leftarrow \arg\min_{\varphi\in\Phi} ccw(\mathcal{C}, \varphi). \tag{4}$$

Fig. 2 depicts the evaluation of the solution represented in Fig. 1, i.e., the evaluation of the assignments φ and ψ. The host graph is shown in dashed black lines and the candidate graph in gray solid lines. The host paths assigned by ψ to each candidate edge in $\mathcal{E}_{\mathcal{C}}$ are shown outside the cycle. Then, the cut associated with each edge of the host graph is indicated as the number of paths that traverse the edge. For example, edge $(1, 2)$ is traversed by paths $p_{(A,F)}$ and $p_{(A,B)}$. Therefore, $\text{cut}((1, 2), \varphi, \psi) = |\{(A, F), (A, B)\}| = 2$. Similarly, edge $(2, 3)$ is traversed by only path

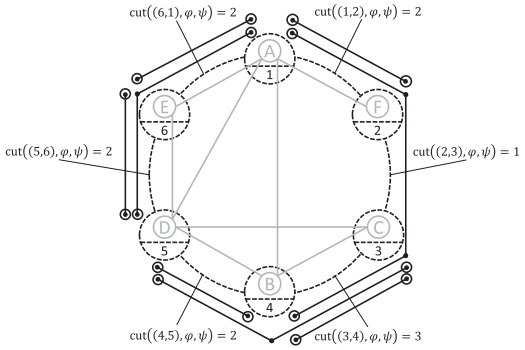


Fig. 2. Evaluation of a possible embedding of $\mathcal{C}$ in $\mathcal{H}$..

$p_{(A,B)}$. Therefore, $\text{cut}((2,3),\varphi,\psi) = |\{(A,B)\}| = 1$, and so on. The value of the objective function for the vertex assignment φ on graph $\mathcal{C}$ is $ccw(\mathcal{C},\varphi) = \max\{2,1,3,2,2,2\} = 3$.

1.2. Literature review

The CCMP is closely related to the Cutwidth Minimization Problem (CMP). Both problems share the same objective function, however the host graph for the CMP is a line, while the host graph for the CCMP is a cycle. The practical applications of these problems are also common, and can be found in various areas that include circuit design (Cohoon and Sahni, 1987), engineering (Makedon and Sudborough, 1989), and graph drawing (Makedon and Sudborough, 2001).

The CMP was originally proposed in the 70s as a theoretical model in the context of circuit design (Cohoon and Sahni, 1987). The problem belongs to the $\mathcal{NP}$-hard class (Gavril, 1977). Due to the problem complexity, several heuristic procedures have been proposed in Cohoon and Sahni (1987), Resende (2009), Pantrigo et al. (2012), Pardo et al. (2013), Duarte et al. (2016) and Martins Santos and Moreira de Carvalho (2019). The literature also includes exact procedures for the CMP on some special classes of graphs (Harper, 1966; Rolim et al., 1995; Thilikos et al., 2005). In addition, two Branch & Bound procedures have been proposed for general graphs (Palubeckis, 2012; Martí et al., 2013) and mathematical formulations have been proposed in Luttamaguzi et al. (2005) and López-Locés et al. (2014).

The bounds proposed in Johnson (2003) establish a relationship of the objective function values for the CMP and the CCMP. In particular, for a candidate graph $\mathcal{C}$, if we let $lcw(\mathcal{C})$ and $ccw(\mathcal{C})$ be the optimal objective function values corresponding to the CMP and CCMP, then:

$$\frac{lcw(\mathcal{C})}{2} \leqslant ccw(\mathcal{C}) \leqslant lcw(\mathcal{C}). \tag{5}$$

The CMP and the CCMP are equivalent when $\mathcal{C}$ is a tree (Chavez and Trapp, 1998).

There are several exact algorithms for the CCMP for particular types of candidate graphs. The optimal value of the CCMP for complete graphs is known by construction (Rios, 1996). For mesh graphs, it is possible to determine the optimum for grids with dimensions larger than 3×3 (Schröder et al., 1999; Clarke, 2002; Schröder et al., 2004). The optimal solution is known for three-dimensional meshes, as long as one dimension is 2 and the other two are greater than or equal to 2 (Sciortino et al., 2002). Similarly, the optimal solution is known for cylindrical meshes for which one of the dimensions is greater than or equal to 2 and the other dimension is greater than or equal to 3 (Schröder et al., 2004). Exhaustive search procedures can be used to find optimal solutions for the CCMP on Q_3 hypercubes (Abbott, 1966). This result has been extended to Q_4 (James, 1996), Q_5 (Aschenbrenner, 2001) and Q_6 (Castillo, 2003) hypercubes. The CCMP has also been studied on general hypercubes (Erbele et al., 2003). Finally, exact algorithms exist for complete bipartite graphs (Johnson, 2003), complete tripartite graphs (Allmond, 2006), and n-partite graphs (Allmond, 2006).

No exact algorithms exist for the CCMP on general graphs. Recently, the practical interest of the CCMP has motivated researchers in the optimization community to apply heuristic techniques to this problem. For instance, in Jain et al. (2016) the authors describe a Memetic Algorithm (Moscato et al., 1989) for the CCMP. They propose six constructive heuristics to generate an initial population of solutions for their solution method. The method includes a local search procedure that attempts to move vertices from positions where the maximum cut occurs. The Memetic Algorithm was evaluated with six types of graph instances: complete splits, join of hypercubes, cones, toroidal meshes, and two random types. The procedure matched all known-optimal solutions and produced the best-known solutions for all other instances, establishing itself as the state of the art for the CCMP.

1.3. New lower bound for the CCMP

Despite the fact that not many theoretical results have been derived for the CCMP for general graphs, several authors have proposed lower bounds for the linear version of the problem (Palubeckis, 2012; Martí et al., 2013). In particular, we focus our attention on the ideas introduced in Martí et al. (2013). In that paper the authors proposed four new lower bounds for the linear cutwidth minimization problem. Based on these ideas, in this section we propose a new lower bound for the CCMP.

Let $d(v)$ be the degree of a candidate vertex $v \in \mathcal{V}_c$, the CCMP of the candidate graph $\mathcal{C}$, denoted as $ccw(\mathcal{C})$, can be bounded by the ceiling of the half of $d(v)$. Let us illustrate this with the example in Fig. 3. Let i be the host vertex assigned to v ($\varphi(v) = i$) and let $d(v) = 4$, as we show in the figure. Additionally, let j and k be the host vertices such that $\{(j,i),(i,k)\} \in \mathcal{E}_{\mathcal{H}}$. Therefore, j and k are adjacent vertices to i in the host graph. For each u, such that $(u,v) \in \mathcal{E}_{\mathcal{C}}$, assigned to any other host vertex distinct from i, there exists a path in the host graph, determined by the ψ function, assigned to the edge (u,v) that contains either the edge (j,i) or the edge (i,k). In Fig. 3, we illustrate the assignation of each edge of v to a different path in the host graph. In this case, the adjacent vertices to v are denoted as u_1, u_2, u_3, and u_4 and its associated paths are denoted as $p_{(v,u_1)}$, $p_{(v,u_2)}$, $p_{(v,u_3)}$, and $p_{(v,u_4)}$ respectively. Therefore, each (u,v) contributes with one unit to the cut of either the host edge (j,i) or the host edge (i,k). In this example, $p_{(v,u_1)}$ and

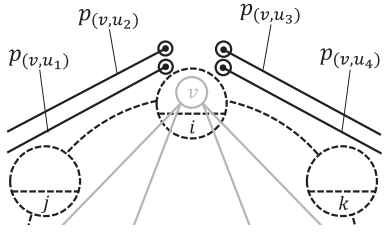


Fig. 3. Illustration of the lower bound derived from vertex v.

$p_{(v,u_2)}$ contribute to the cut of the host edge (j,i), while $p_{(v,u_3)}$ and $p_{(v,u_4)}$ contribute to the cut of the host edge (i,k). Since we are facing a min–max optimization problem, in the best scenario half of the vertices adjacent to v would contribute to the edge (j,i) while the other half to the edge (i,k) with independence of the assignment of rest of the vertices of the graph.

Among all the vertices in $\mathcal{V}_C$, the one with the largest $d(v)$ will provide the tightest bound. Mathematically, this lower bound, *LB*, can be expressed as follows:

$$ccw(\mathcal{C}) \geqslant LB = \max_{v \in \mathcal{V}_C} \left\lceil \frac{d(v)}{2} \right\rceil. \quad (6)$$

This new lower bound has been calculated for each instance used in our computational tests and the results per instance are shown in the Appendix A, where it is possible to compare the LB with the results obtained by the Multistart TS procedure proposed in this paper.

1.4. Our contributions

In addition to the new lower bound proposed for the CCMP (introduced in Section 1.3), the main contribution of this work is the development of a metaheuristic procedure that includes sound fundamental as well as advanced search strategies for the CCMP. We are able to show, through extensive computational experimentation, that our proposal is competitive with the current state of the art for solving the CCMP. The method consists of a multistart search, where the starting points are generated in greedy fashion and the improvement phase is based on neighborhoods and a tabu memory structure (Glover, 1997). We perform several preliminary tests to find the best configuration of our procedure, i.e., to determine the best set of search parameter values. Through this process, we are able to determine the contribution of the various elements embedded in the proposed procedure.

After a set of tuning experiments to identify the best combination of parameter values, we employ the resulting procedure configuration for competitive testing. The test set consists of problem instances from the literature. Those instances are grouped into two main categories, graphs with known structure and random graphs. The results of these tests indicate that, in many cases, our proposed solution method is able to find solutions of better quality than the state-of-the-art-procedures, and in less computational time. Furthermore, we show that these differences are statistically significant.

The description of our work is organized as follows. Section 2 describes our algorithmic proposals. Section 3 presents several advanced strategies. Section 4 introduces the test set, describes the computational experiments, and discusses the results. Conclusions and final thoughts are in Section 5.

2. Algorithmic proposal

Our solution method has two main components, a procedure to construct an initial solution and an improvement method based on tabu search (TS) Glover, 1997. We first provide details of a procedure to construct initial solutions (Section 2.1). Then, we discuss how a local search operates on these initial solutions (see Section 2.2). This section continues with a description of the TS mechanisms that help the local search escape local optimal points (see Section 2.3). Finally, the combination of the previous components is described in Section 2.4 where we present our multistart procedure.

2.1. Constructive procedure

Constructing a solution for the CCMP consists of performing two tasks: 1) assigning the vertices of the candidate graph to the vertices of the host graph (i.e., defining the domain and range of the φ function); and 2) assigning the edges of the candidate graph to a path in the host graph (i.e., defining the domain and range of the ψ function). We have tried different strategies to construct initial solutions for the CCMP. These strategies were tested over a subset of instances and the best of them was selected for the final configuration of our algorithm. We next describe in detail the best strategy identified among the tested ones and, at the end of the section, we enumerate and briefly describe the rest of the strategies tested.

The constructive algorithm used in the final configuration of our procedure starts with all candidate vertices unassigned. We number the host vertices starting from the top vertex (i.e., the vertex that in a graphical representation would be at 12 o'clock) and continuing clockwise. At each step, we select a candidate vertex and assign it to the next available host vertex. Since the host graph is a cycle, without loss of generality, at each step of the procedure, we move sequentially in the clockwise direction. That is, the first candidate vertex is assigned to host vertex 1, the second candidate vertex is assigned to host vertex 2, and so forth. The first assignment is random. That is, a candidate vertex is randomly selected and it is assigned to host vertex 1. After the first assignment, all unassigned vertices in the candidate graph are evaluated with a greedy function to determine the most attractive vertex to assign next. The greedy function to select the next unassigned vertex from the candidate graph is inspired by previously published ideas (Mcallister, 1999). The construction ends after all candidate vertices have been assigned to a host vertex.

The greedy selection function to select the next vertex from the candidate graph is defined as follows. Let $\mathcal{A}$ be the set of candidate vertices that have already been assigned and let $\mathcal{U}$ be the set of unassigned candidate vertices. Let $d(v)$ be the degree of candidate vertex v. Also, let $d_{\mathcal{A}}(v)$ be the number of assigned candidate vertices that are adjacent to v and $d_{\mathcal{U}}(v)$ be the number of unassigned vertices adjacent to v. That is,

$$d_{\mathcal{A}}(v) = |\{u \in \mathcal{A} : (v,u) \in \mathcal{E}_C\}|,$$

$$d_{\mathcal{U}}(v) = |\{u \in \mathcal{U} : (v,u) \in \mathcal{E}_C\}|,$$

such that $d(v) = d_{\mathcal{A}}(v) + d_{\mathcal{U}}(v)$. Then, we define the greedy value g for an unassigned vertex v as:

$$g(v) = d_{\mathcal{A}}(v) - d_{\mathcal{U}}(v).$$

The g function measures the urgency of the candidate vertex under consideration to the assigned candidate vertices versus to its proximity to the unassigned candidate vertices. We would like to select the unassigned candidate vertex that, relative to other unassigned candidate vertices, is closest to the candidate vertices that have already been assigned. Therefore, the unassigned candidate vertex with the largest g value is chosen to be assigned next. The greedy function is such that it makes an unassigned candidate vertex very attractive if all vertices adjacent to it have already been assigned. Conversely, an unassigned candidate vertex is unattractive when none of its adjacent vertices have been assigned.

Algorithm 1 summarizes the greedy constructive procedure. The candidate graph $\mathcal{C} = (\mathcal{V}_C, \mathcal{E}_C)$ is the input to this procedure. Initially, all candidate vertices are unassigned (step 2). Steps 3–4 make the assignment of a randomly selected candidate vertex to host vertex 1. The assigned vertex is removed from the set of unassigned candidate vertices (step 5). A for-loop is then executed (steps 6–10) to assign all remaining candidate vertices in $\mathcal{U}$. Step

7 evaluates all unassigned candidate vertices to identify the one with the largest greedy value (with ties broken arbitrarily). The chosen candidate vertex $next_C$ is assigned to the next available host vertex $next_H$ (step 8). The assigned candidate vertex, $next_C$, is removed from $\mathcal{U}$ (step 9). Once all candidate vertices are assigned, the procedure returns φ, which contains the mapping of candidate vertices to host vertices (step 11).

Algorithm 1. Constructive procedure

1: **Procedure** `GreedyConstructive`($\mathcal{C}$)
2: $\mathcal{U} \leftarrow \mathcal{V}_C$
3: $next_C \leftarrow$ `rand`($\mathcal{U}$)
4: $\varphi(next_C) \leftarrow 1$
5: $\mathcal{U} \leftarrow \mathcal{U} \setminus \{next_C\}$
6: **for all** $next_H > 1$ **do**
7: $\quad next_C \leftarrow \arg\max_{v \in \mathcal{U}} g(v)$
8: $\quad \varphi(next_C) \leftarrow next_H$
9: $\quad \mathcal{U} \leftarrow \mathcal{U} \setminus \{next_C\}$
10: **end for**
11: **return** φ

In order to implement an algorithm for the CCMP, it is also necessary to define an efficient data structure to represent a solution φ. In this case, we propose the use of an array data structure. An array is a one-dimension matrix composed of an index that identifies each memory position within the array, and the corresponding content of those positions in memory. Therefore, an array can be seen as a set of tuples key-value. The key represents the vertices of the host graph, while the value represents the vertices of the candidate graph. Therefore, a correspondence key-value represents the assignment of a candidate vertex to a host vertex. In this sense, there are n key-value pairs, being n the number of vertices of either the host or the candidate graphs. In Fig. 4, we depict several representations of this array (one per subfigure). In this case, the header of the array corresponds to the keys of the array while the second row of this graphical representation contains the values associated with each key.

Fig. 4 shows an example of the steps followed by the constructive procedure for the graph introduced in Fig. 1(a). For this graph, the construction is completed in six iterations (i.e., one for each vertex). As shown in Fig. 4(a), the procedure starts with a random selection of candidate, say for instance vertex C, which is assigned to host vertex 1. In each step, we indicate which vertex is $next_H$ and $next_C$. The first one follows the numerical (clockwise) order. The second one is selected by computing the value of the g function. Fig. 4(a) shows the g value for the vertices in $\mathcal{U}$ at this step (i.e., A, B, D, E, and F). For instance, $g(A) = d_{\mathcal{A}}(A) - d_{\mathcal{U}}(A) = 0 - 4 = -4$. Similarly, $g(B) = d_{\mathcal{A}}(B) - d_{\mathcal{U}}(B) = 1 - 2 = -1$, and so forth. Once all the unassigned vertices have been evaluated, the greedy selection chooses the vertex with the largest g value (with ties broken arbitrarily). The number of vertices evaluated decreases by one at each step. In addition to the graphical representation of the current partial solution, Fig. 4 includes tables associated with the vertex assignments, i.e, $\varphi(C) = 1$, $\varphi(B) = 2$, $\varphi(D) = 3$, $\varphi(E) = 4$, $\varphi(A) = 5$, and $\varphi(F) = 6$.

The ψ function can be derived from the φ mapping (see Fig. 4 (f)). The domain of ψ is given by the edges of the candidate graph, while the range is given by the paths in the host graph. Of the possible paths, we choose the shortest that connects the end candidate vertices. The topology of the host graph restricts the range of ψ to two possible paths per candidate edge, one clockwise and another one counterclockwise (see Fig. 1(d)). If there is a tie in the length of the two possible paths, the procedure selects the clockwise path. Since ψ can be derived from the φ mapping, we consider that φ is a full characterization of the solution to the problem.

From an implementation point of view, we need to define an additional data structure to evaluate a solution. Particularly, this data structure is an array which stores integer values that represent the different cuts of each edge in the host graph. Therefore, the size of this array is equal to the number of edges in the host graph. In this case, we use the same key-value representation previously mentioned, the key identifies the edge in the host graph, while the value represents the number of cuts of this edge. To calculate the number of cuts of each host edge, we scan all the edges of the candidate graph and obtain the associated paths in the host graph determined by the ψ function. Each time that a host edge is included in a path, it implies to increment in one unit the value of the number of cuts of that host edge. This procedure is in the order of $O(n^2)$. Once all the cuts for a particular solution have been stored in the array, we need to fully traverse it in order to find the largest cut value, which is the value of the objective function for the CCMP.

In addition to the aforementioned constructive algorithm, we also tested other constructive variants. Even though those strategies were not so successful for this problem, it might be also interesting to enumerate them. First, we tried a randomization of the greedy selection represented by g, following a GRASP constructive approach (Feo and Resende, 1994; Resende et al., 2010). Secondly, we used a second greedy function based on the quality of the objective function, instead of the function g, to select the next candidate vertex. Again, we also implemented a randomized variant of this constructive, based on the new greedy function. Unfortunately, the performance of those methods was far from the best results obtained by the one described in this section. Therefore, for the sake of brevity we do not include the experiments related to these additional strategies in the preliminary experiments section.

2.2. Local search

From a starting solution, a local search is an intensification strategy designed to find the local optimum in a predefined neighborhood. Our Local Search (LS) is defined in an insertion neighborhood. An insertion is a classical move in both graph layout and permutation problems. It consists of removing a candidate vertex from its current position in the host graph and inserting it in a different position. For instance, Fig. 5 depicts the move of vertex A from position 5 (i.e., $\varphi(A) = 5$) to position 2. We denote this operation as $\varphi' = Insert(\varphi, A, 2)$, where φ' is the solution after the move. Fig. 5 shows the solution before the insertion (φ) and the solution after the insertion (φ'). The figure highlights the vertices from the candidate and host graphs that are affected by the move. As customary in insertion moves, the displaced elements must be shifted. In our context, the vertices can be shifted in the clockwise or counterclockwise direction. Since the host graph is a cycle, shifting in one direction or the other results in the same solution. In our example, when candidate vertex A is moved to position 2, displaced vertices could have shifted in the clockwise direction (i.e., B moves to 3, D to 4, and E to 5). Instead, our moves are such that we always shift the displaced candidate vertices in the counterclockwise direction, as shown in Fig. 5.

Considering the aforementioned insertion move, the neighborhood associated with candidate vertex v of solution φ is defined as the solutions that can be reached by the insertions (embedding) of v in all positions (vertices) in the host graph that are different from its current position:

$$N_\varphi(v) = \{Insert(\varphi, v, u) : \forall\, u \in V_{\mathcal{H}}, u \neq \varphi(v)\}$$

For a candidate graph with n vertices, the size of the complete neighborhood (i.e., considering all candidate vertices) is $n \cdot (n-1)$.

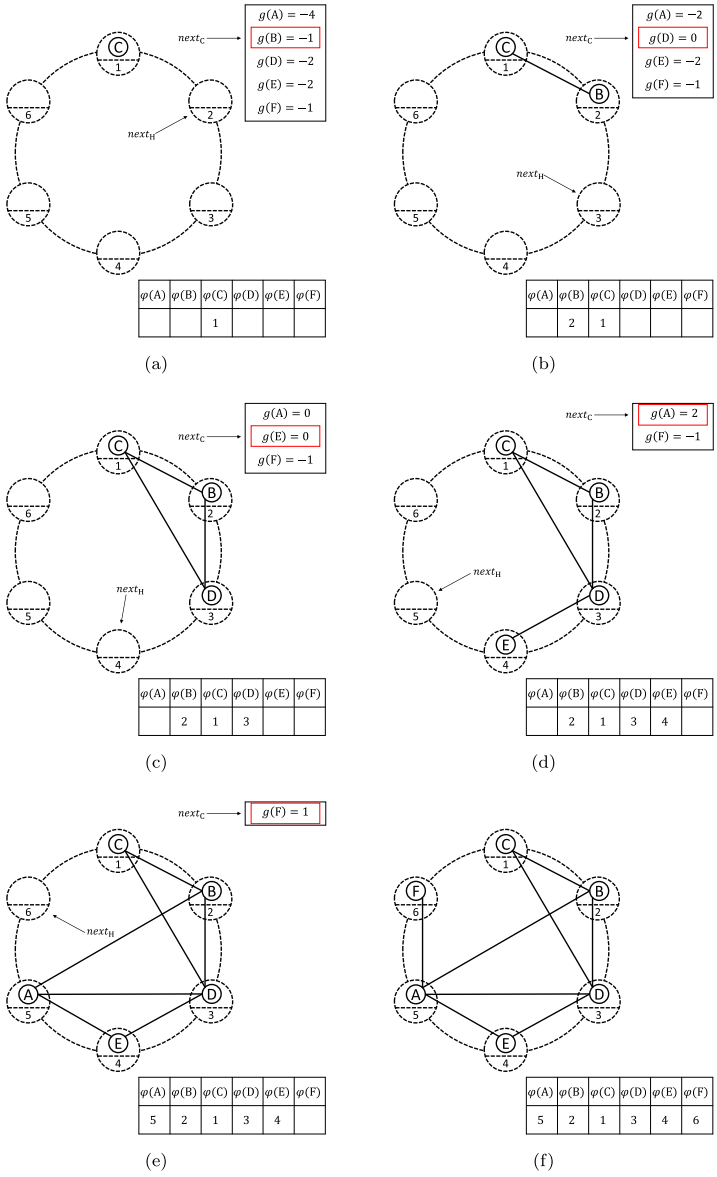


Fig. 4. Example of the steps followed to construct a solution.

Algorithm 2 summarizes the steps of the local search with insert moves. The input to this procedure is the candidate graph $(\mathcal{C} = (\mathcal{V}_c, \mathcal{E}_c))$ and the initial solution (φ). The procedure combines first and best improving strategies. The first improvement strategy is related to performing the insertion of the first vertex which improves the current solution, while the best improvement strategy is referred to only considering the best insertion of each vertex. Particularly, all candidate vertices in $\mathcal{V}_c$ are scanned at random using a shuffle function (see step 4). Therefore, for each vertex v being considered, the procedure finds the best possible insertion in $N_\varphi(v)$ (step 6), as in the best improvement strategy, with an algorithmic complexity of $O(n^2)$. Then, if this insertion results in an improvement (step 7), since we are following a first improvement strategy, the current solution is updated (step 8), the improvement flag is switched to True (step 9), and the scanning of the candidate vertices starts again from this new solution (step 10). The do-while loop (steps 2–13) is repeated until no insertion of a candidate vertex is able to improve the current solution.

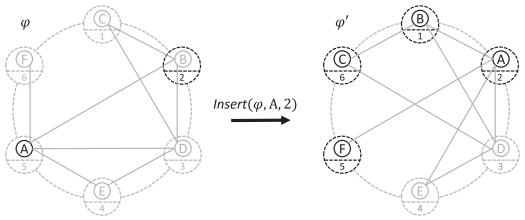


Fig. 5. Example of an embedding φ and the resultant embedding φ' obtained after the operation $Insert(\varphi, A, 2)$.

Algorithm 2. Local search

```
1: Procedure LocalSearchBestImprovement (C, φ)
2: do
3:    improve ← False
4:    V ← shuffle(V_C)
5:    for v ∈ V do
6:       φ′ ← arg min_{φ″∈N_φ(v)} ccw(C, φ″)
7:       if ccw(φ′) < ccw(φ) then
8:          φ ← φ′
9:          improve ← True
10:         break
11:      end if
12:   end for
13: while improve
14: return φ
```

Notice that in addition to the local search defined in an insertion neighborhood, we also tested the performance of another local search, in the interchange neighborhood. The interchange is classical move in both graph layout and permutation problems. In particular, the interchange move consists of selecting two candidate vertices from the solution and exchanging their assigned vertices in the host graph. Unfortunately, the performance of this strategy was far from the best results obtained by the local search based on the insertion move. Therefore, for the sake of brevity we do not include the experiments related to this additional strategy in the preliminary experiments section.

2.3. Tabu search

Tabu Search (TS) is a metaheuristic originally introduced in 1977 (Glover, 1977) and later formalized in Glover (1986) as a general method for solving hard optimization problems. Many ideas and extensions are discussed in Glover (1989), Glover (1990), Glover (1997) and Glover and Laguna (1997). A recent review of the strategies associated with tabu search are compiled in Laguna (2017). TS is a, so-called, single-solution neighborhood search metaheuristic methodology. TS introduces the concept of memory with the goal of making the best possible decisions based on the previous information collected throughout the search, instead resorting to randomization.

We add a simple tabu search short-term memory to the local search summarized in Algorithm 2. The TS memory consists of recording a number of recently visited solutions. A move (i.e., an insertion) is classified tabu if it transforms the current solution into a tabu solution (i.e., a solution that is currently in the short-term memory). Unlike TS designs that use memory based on attributes, in our design it is not necessary to include an aspiration criterion, since no tabu move can reach a solution that the search has not already visited. The size of the TS memory (i.e., the number of tabu solutions) is the search parameter known as *TabuTenure*.

Instead of stopping at the first local optimum (i.e., the first time that a move cannot be found to improve the current solution) as in Algorithm 2, the search is allowed to continue by selecting the non-improving move that deteriorates the objective function the least. This move is the "best" non-improving move. Before making a move, the current solution is added to the TS memory and the "oldest" tabu solution is deleted. The oldest tabu solution is the one added *TabuTenure* iterations ago. The search continues after a number of iterations without improvement are executed. The number of non-improving iterations used to stop is a search parameter (*Non-Improving*), computed with respect to the number of vertices of the input graph (n). Preliminary testing revealed $TabuTenure = 0.2n$ as an effective value for this search parameter. These experiments also showed that the best results can be expected when *Non-Improving* is set to $0.1n$, with a minimum value of 6 and a maximum value of 15. The short-term TS memory components are sufficient to produce very high-quality solutions, however, any optimization algorithm should find the right balance between intensification and diversification (Laguna, 2017). To that aim, tabu search framework introduces the general idea of long-term memory in order to diversify the search. However, other simpler approaches (Glover and Laguna, 1997), as the one proposed in this paper, propose the combination of tabu search with a multistart strategy to diversify the search as we present in Section 2.4.

2.4. Multistart procedure

In this paper we propose a multistart optimization algorithm to tackle the CCMP. Multistart strategies are used in this context to escape from local optima where heuristic procedures get stuck. It based on the idea of applying an intensification strategy to different starting points in the space search. The pseudocode of our proposal is presented in Algorithm 3. In this case, at each iteration, the procedure starts from a different solution constructed with the greedy constructive algorithm introduced in Section 2.1. Then, the intensification strategy used is the TS presented in Section 2.3. When the TS stops, a new solution is generated with Algorithm 1. This process continues until a maximum time limit (t_{max}) or a maximum number of iterations (r_{max}) is reached, but guaranteeing that a minimum number of iterations (r_{min}) is performed for each candidate graph (C). As we will see in the preliminary experiments section, the minimum and maximum number of iterations have been set to 10 and 30 respectively.

Algorithm 3. General procedure

```
1: Procedure MultistartTabuSearch (C, t_max, r_min, r_max)
2: restarts ← 0
3: best ← ∅
4: do
5:    φ ← GreedyConstructive(C)
6:    φ′ ← TabuSearch(C, φ)
7:    if best == ∅ or ccw(φ′) < ccw(best)
8:       best ← φ′
9:    end if
10:   restarts ← restarts + 1
11: while restarts < r_min or (time < t_max and restarts < r_max)
12: return φ
```

3. Advanced strategies

The procedures presented in Section 2 can be further improved with the use of the three advanced strategies proposed in this section. Although these strategies were designed within the CCMP

context, the ideas behind them might be applied to other heuristic searches. The first strategy deals with landscapes where many solutions have the same objective function value. In this case, the objective function value alone does not provide enough information to find effective search directions. A secondary evaluation is able to differentiate solutions for which objective function values are the same. The second strategy explores computationally efficient ways of calculating move values. This is particularly important in large neighborhoods. The third one also tackles efficiency but from the point of view of reducing the number of moves to evaluate.

3.1. Secondary solution evaluation

In heuristic search, a flat landscape condition occurs when large fractions of the solution space have the same objective function value (Pinana et al., 2004; Resende et al., 2010). This means that structurally different solutions may be associated with the same value of the objective function. Determining search directions becomes a very difficult task when decisions are based only on the change of the objective function value produced by a move. Finding a meaningful way of differentiating solutions with the same objective function value is important because the structure of one solution may be more promising than the structure of another in terms of improving the incumbent solution later in the search.

Flat landscapes are typically associated with min–max or max–min optimization problems (Pardo et al., 2013; Duarte et al., 2016), that is, those problems where the objective is to minimize a maximum value or to maximize a minimum value. To overcome this difficulty, researchers have proposed the use of one or more secondary solution evaluations. These evaluations are only activated when solutions have the same objective function value and they are designed to indicate preferences regarding the structure of the solutions being compared (Pantrigo et al., 2012; Pardo et al., 2013).

The secondary evaluation that we employ is based on ideas presented in Pantrigo et al. (2012) and Pardo et al. (2013). As our problem formulation indicates, the CCMP is an optimization problem that has the goal of finding a solution that minimizes the maximum cut produced by the assignment of candidate vertices to a host graph. It is possible for multiple solution configurations to have the same maximum cut. When comparing two solutions with the same maximum cut (i.e., the same objective function value), we are interested in knowing which one of the two has a better "improvement potential" if a move (or series of moves) could reduce the current maximum cut.

We use the improvement potential concept to differentiate solutions with the same objective function value. In particular, if two solutions have the same objective function value, we compare the number of times that the largest cut occurs in each them. The solution with the smallest number of largest cut is deemed better (i.e., the solution has the larger improvement potential of the two solutions under consideration). If this calculation is not able to differentiate between the solutions, then we compare the number of times that the second largest cut occurs in any of them and so on. We do this until the result of the calculation is able to distinguish between the two solutions.

3.2. Efficient move calculation

The exploration of a solution neighborhood usually is the most computational intensive element in search procedures. Neighborhood search methods require the evaluation of moves to determine what to do next. Developing efficient ways of calculating move values is critical in heuristic search. We propose an efficient calculation of the value of the *Insert* move that we defined in Section 2.2. The main idea consists of isolating the effect of an *Insert* move. Specifically, we only need to consider the edges of the candidate vertices that are reassigned with the corresponding move.

We illustrate the move evaluation with the example depicted in Fig. 6. Fig. 6(a) shows the solution before the move of vertex C to position 2, characterized by $Insert(\varphi, \mathrm{C}, 2)$. Notice that, for the sake of clarity, in the figure we have simplified the notation of the cut of each host edge, introduced in Eq. (1), with a label c_i $(1 \leqslant i \leqslant 5)$. Fig. 6(b) shows the solution after $Insert(\varphi, \mathrm{C}, 2)$. The move evaluation consists of first deleting the paths associated with the edges adjacent to vertices B and C, which are the only vertices that change positions (host vertices) after the move. We also delete the contribution of these paths to the objective function calculation. Then, new paths are assigned to these edges and the contribution of these paths is added to the objective function. The table in Fig. 6(c) shows these calculations. Step 0 shows the cuts in the current solution. Step 1 shows the contributions that are removed and step 2 shows the contributions that are added. The cut values associated with the new solution are shown in step 3. These values are obtained by adding the corresponding values in the previous steps.

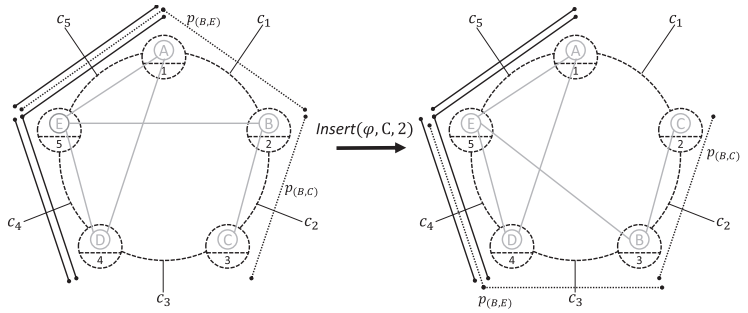
From a computational point of view, and taking into consideration the array data structure used to evaluate a solution (introduced in Section 2.1), calculating the value of the objective function from the scratch is in the order of $O(n^2)$ being n the number of vertices of the graph. On the other hand, if the array data structure used to evaluate a solution is already calculated, performing a move only implies to reconsider the contribution of the edges of the candidate vertices being reassigned (instead of all of them). The complexity of this operation is also in the order of $O(n^2)$, since all the vertices might be affected by the move. However, despite the fact that this proposal does not reduce the theoretical complexity, from a practical point of view, and as we will illustrate in the experimental section, the number of edges affected by a move is considerably smaller than the total number of edges of the graph.

3.3. Search regions of interest

Exhaustive neighborhood searches become increasingly impractical when the size of the neighborhood grows either polynomially or exponentially with the size of the problem. In our case, the entire neighborhood of a solution is $n \cdot (n-1)$, being n the number of vertices. To complement the quick move calculation described in the previous subsection, we add a strategy to focus the search on regions of interest. These regions are reached by a set of promising moves and therefore the strategy consists of identifying such moves. This partial exploration of the entire neighborhood is similar to the strategy of moving to the first improving solution, which has been documented to work well in multi-start settings (Hansen and Mladenovic, 2006). Our neighborhood search combines the two strategies and in addition we embed the notion of a partial exploration that focuses on Regions Of Interest (ROI).

The ROI of a candidate vertex v, denoted by $\mathrm{ROI}(v)$, is the subset of host vertices such that if v was assigned to any of them, the current objective function value might change. In other words, $\mathrm{ROI}(v)$ is the set of host vertices that could cause a reassignment of paths associated with the edges of v. Clearly, the idea here is to avoid changing the assignments of candidate vertices to positions (host vertices) that will cause no changes in the path assignments (and therefore no changes in the objective function value).

From an implementation point of view, the vertices that are included in the set $\mathrm{ROI}(v)$ are derived from the assignments of the adjacent vertices to v. For each u such that $(u, v) \in \mathcal{E}_c$ (i.e., u



step	cut	c_1	c_2	c_3	c_4	c_5
0	Fig. 5(a)	1	1	0	2	3
1	Remove	-1	-1	0	0	-1
2	Add	0	+1	+1	+1	0
3	Fig. 5(b)	0	1	1	3	2

(c)

Fig. 6. (a) Solution before the move *Insert*(φ, C, 2). (b) Solution after the move *Insert*(φ, C, 2). (c) Efficient revaluation of the objective function.

is adjacent to v in the candidate graph), we add to ROI(v) the following vertices of the host graph: $\varphi(u)$, $\varphi(u)-1$, $\varphi(u)+n/2$, $\varphi(u)+(n/2)+1$, and $\varphi(u)+(n/2)-1$. Notice, that without loss of generality, in order to properly identify the host vertices added to ROI(v) we are supposing that host vertices are denoted by numbers from $1 \ldots n$ and that $\varphi(u)-1 < \varphi(u) < \varphi(u)+1$. Additionally, the set ROI(v) also includes the host vertices derived from the following situation: when two adjacent vertices in the candidate graph (let denote them as v_1 and v_2) are assigned to two vertices in the host graph such that, the length of the two possible paths between them is the same, then $\varphi(v_1)$, $\varphi(v_1)-1$, $\varphi(v_2)$, and $\varphi(v_2)-1$ are also included in the set ROI(v).

The exploration of the regions of interests is then done in two steps, we first identify ROI(v) for a v chosen at random, and then we evaluate all the moves associated with inserting v in all the host vertices in ROI(v). The best move is selected if it improves the current solution. We repeat this process for all v until no improvement is performed.

It is worth mentioning that this strategy is independent of the considered objective function. Therefore, it can be used as a single strategy in isolation or in combination with the secondary objective function (See Section 3.1).

4. Experimental results

Before describing our computational experiments and discussion or results, we introduce the instances of our test set (Section 4.1). Preliminary experiments are described in Section 4.2. These experiments are devoted to adjust the parameters of our solution procedure and also to analyze the contribution of the proposed search strategies. We compare the best configuration with the state of the art in Section 4.3.

It is important to notice that all our algorithms have been implemented in Java 8, and all the experiments have been run in an Intel Core i7-4702MQ CPU 2.20 Ghz with 16 GB RAM.

4.1. Instances

We use the following set of instances (graphs) from the literature in our computational tests Jain et al., 2016:

- **Small**: the number of vertices and edges in these random graphs varies between 16 and 24, and between 18 and 49, respectively. There are 84 instances in this set.
- **Harwell-Boeing**: these graphs arise from a wide variety of problems in scientific and engineering disciplines. The selected problems are a diverse subset of the original Harwell-Boeing set (Duff et al., 1992). Particularly, we have selected the 38 graphs used in Jain et al. (2016). These graphs have sizes between 39 and 685 vertices and from 46 to 3720 edges.
- **Regular**: these subset includes four different types of graphs (Complete Split Graph, Toroidal Mesh, Join of Hypercubes, and Cone Graph) with a predefined and well-known structure, but with an unknown optima. This set includes a total of 57 graphs with a number of vertices ranging from 12 to 1000 and a number of edges ranging from 46 to 6225.

Our experiments do not include the set of regular graphs with known optimal solutions included in Jain et al. (2016). These graphs are such that do not provide insightful results in comparative testing, since modern heuristics can easily find the optimal value.

4.2. Preliminary experiments

The preliminary experiments have the goal of identifying the best values for the search parameters, as well as assess the merit of the proposed search strategies. They have been performed over a reduced set of instances consisting of 10% of all problem instances (i.e., 18 graphs). We will refer to this subset of instances as the preliminary set.

The procedure to construct solutions described in Section 2.1 is not totally deterministic. The first candidate vertex to be assigned

is chosen at random. Also, when the greedy value is the same for all unassigned candidate vertices, ties are broken arbitrarily. Therefore, it is interesting to observe the performance of the procedure when constructing more than one solution with the aim of determining if the procedure is able to produce diverse solutions. Table 1 reports the average value for the considered instances of the objective function (Avg.), the deviation of the best solution with respect to the overall best within the experiment (Dev. (%)), the number of best solutions found in the experiment (#Best) and the time in seconds (CPU Time (s)) when applying the procedure to construct 1, 5, 10, 20, 30, 40, and 50 solutions. Note that we are not comparing against the best known solutions but against the best solutions found within this experiment. The results in Table 1 show that the best solutions for the 18 instances in the preliminary set are found when the procedure constructs 50 solutions. The improvement is significant from constructing one solution to constructing 50 solutions. However, the difference in solution quality is negligible after executing the procedure 20 times.

As we introduced in Section 2.4, our whole procedure performs a variable number of iterations for each graph, depending on the available time. In this sense, each iteration starts from a new solution constructed with our greedy procedure. Considering the results presented in Table 1, the number of different iterations used per instance in our final experiments has been set in the range [10,30] which means that the procedure continues until a maximum time limit is reached, as long as at least 10 but no more than 30 restarts are performed.

We then compare the performance of our greedy construction (Greedy) and a totally random construction (Random). In Table 2 we report the results found when executing the algorithms with a time limit of one second for each graph. Since this experiment do not include parameter adjustments, it has been performed over both: the preliminary set and the whole set of instances. Particularly, in this table, in addition to the Avg., Dev. (%) and #Best we report the average number of solutions constructed (Avg. #Cons.) in the aforementioned time limit. As expected, the average number of constructions performed by the random constructive is larger than the ones obtained by the greedy constructive. This difference is due to the necessity of the greedy procedure of evaluating each unassigned candidate vertex before selecting it. However, the solution quality indicators favor the greedy constructions over a totally random approach. This fact is observed either in the preliminary and the whole set of instances.

The second preliminary experiment is devoted to testing the influence of the advanced strategies introduced in Section 3. We start by running the local search of Section 2.2 and reporting the results in Table 3 (LS). The table compares these results with the outcomes from running LS with the efficient move evaluation of Section 3.2 (LS+E) and the results from focusing on regions of interest of Section 3.3 (LS+E+ROI). Since the local search follows a strictly descent pattern, the secondary evaluation of Section 3.1 does not play a role (it is included in the three compared strategies of this experiment). Recall that the secondary evaluation is used to distinguish between moves that result in no change of the objective function value.

Table 2
Comparison between random and greedy constructions.

	Preliminary set (18)		Whole set (179)	
	Random	Greedy	Random	Greedy
Avg.	87.22	53.06	98.81	75.92
Dev. (%)	113.00	9.46	84.61	7.15
#Best	4	14	39	146
Avg. #Cons.	151098.83	55025.72	193418.67	61339.90

Table 3 reveals that all variants are able to reach the same solution quality. Moving from LS to LS+E, we observe the decrease of one order of magnitude in computational time to explore the same average number of solutions (Avg. #Sol.). The neighborhood reduction strategy associated with ROI is able to further reduce the computational burden (by two additional orders of magnitude). We point out that all variants were run starting from the same initial solution. Again, these findings have been observed either in the preliminary and the whole set of instances.

To adjust the two parameters associated with the tabu search elements in our procedure, we performed a full factorial design with values that we made dependent on the graph size. In particular, for the tabu tenure we tested 0.05, 0.1, 0.2, 0.3, and 0.4 of n, where n is the number of vertices in the candidate graph. We tested the same percentages for the maximum number of iterations without improvement, and limited the value to be within a minimum (6 iterations) and a maximum (15 iterations). For the sake of brevity, we do not include the corresponding table of results. We only identify the best setting for tabu tenure ($0.2n$) and number of iterations without improvement ($0.1n$).

Our final preliminary experiment explores the increase in solution quality when going from simple greedy constructions (Greedy) to the local search with advanced strategies (LS+E+ROI) and to the tuned procedure that includes TS elements (TS). Table 4 reports the results. As expected, adding local search results in a noticeable improvement in solution quality. The embedding of tabu search elements results in additional quality improvement, although the difference between LS+E+ROI and TS is significantly smaller than the solution quality difference between Greedy and LS+E+ROI.

4.3. Competitive testing

In our competitive testing, we compare our tuned procedure with the one proposed in Jain et al. (2016), the Memetic Algorithm (MA) that we described in the literature review. Notice that we have directly used the original code implemented by the authors in Jain et al. (2016), who kindly provided us with the source code of their algorithms for our comparisons. In order to execute the MA procedure we have used the parameters indicated by the authors. Particularly, the maximum number of generations of each run of the MA is set to 400 generations. Additionally, the reported quality values are the best ones obtained after 30 runs of the algorithm and the time reported corresponds only to the run where the best solution was found. Table 5 shows the results of this test. The results are grouped by set and graph type (i.e., random or struc-

Table 1
Performance of the constructive procedure based on the number of iterations.

Iterations	1	5	10	20	30	40	50
Avg.	60.61	57.28	55.00	54.17	54.11	54.06	53.94
Dev. (%)	32.82	12.83	6.20	0.33	0.28	0.17	0.00
#Best	3	9	11	15	16	17	18
CPU Time (s)	0.004	0.010	0.015	0.024	0.032	0.041	0.048

Table 3
Contribution of advanced strategies to the local search.

	Preliminary set (18)			Whole set (179)		
	LS	LS+E	LS+E+ROI	LS	LS+E	LS+E+ROI
Avg.	43.61	43.61	43.61	60.15	60.15	60.15
#Best	18	18	18	179	179	179
CPU Time (s)	591.98	35.93	0.53	1249.13	68.36	3.31
Avg. #Sol.	4312427.50	4312427.50	60463.78	4569844.91	4569844.91	86776.99

tured). TS refers to our proposed method and MA refers to the memetic algorithm in Jain et al. (2016). We use the same metric as before, where average deviation from best (Dev.) is calculated considering the collective-best solutions found with either TS or MA. The "Total" section at the bottom of the table provides averages across all graph types.

The general observations from examining this table are that:

- The TS solutions are found in about one order of magnitude less time than MA.
- The performance of TS is better on random graphs than on structured graphs.
- For all problem types, TS solutions are on average closer to the best solutions (maximum deviation of 2.40%) than the MA solutions (maximum deviation of 12.09%).

Table 4
Performance differences between the procedure components and the full procedure.

	Greedy	LS+E+ROI	TS
Avg.	60.61	43.61	43.06
Dev. (%)	41.03	3.15	0.11
#Best	0	12	17
CPU Time (s)	0.01	0.51	2.14

Table 5
Comparison with the state of the art.

			TS	MA
Random graphs	Small Instances (84)	Avg.	3.90	3.98
		Dev. (%)	0.00	1.80
		#Best	84	78
		CPU Time (s)	0.32	16.67
	Harwell-Boeing (38)	Avg.	67.26	70.71
		Dev. (%)	2.40	12.09
		#Best	31	10
		CPU Time (s)	187.59	4081.87
Regular-structured graphs	Complete Split (21)	Avg.	217.76	217.34
		Dev. (%)	0.18	0.00
		#Best	13	21
		CPU Time (s)	994.16	2522.81
	Join of Hypercubes (9)	Avg.	80.89	80.34
		Dev. (%)	0.27	0.00
		#Best	7	9
		CPU Time (s)	56.08	369.91
	Toroidal Mesh (17)	Avg.	30.24	30.24
		Dev. (%)	1.47	1.24
		#Best	13	16
		CPU Time (s)	142.31	1871.53
	Cone (10)	Avg.	154.80	155.00
		Dev. (%)	0.10	0.26
		#Best	9	7
		CPU Time (s)	80.32	1405.45
Total (179)		Avg.	57.24	57.94
		Dev. (%)	0.69	3.54
		#Best	157	141
		CPU Time (s)	177.42	1434.01

We performed two statistical tests with the goal of identifying significant performance differences, the Wilcoxon's signed rank test and the *t*-test for paired samples.

Specifically, we use Wilcoxon's signed rank test (Wilcoxon, 1992) to identify differences between the objective function values of the best solutions found by TS and MA. We apply the one-tail version of this paired test to random graphs and structured graphs separately. Our null hypothesis is that there is no difference in the median of the objective function values, while the alternative hypothesis is that the median of one set of values is smaller than the other. The first test with all 122 random graphs results in a *p*-value of 0.0008. Therefore, we can confidently reject the null hypothesis in favor of the alternative hypothesis that the median of the TS objective function values is less than the median of the MA values. This fact confirms the better performance of the TS over these instances. The Wilcoxon test with all 57 structured graphs results in a *p*-value of 0.0069. This also indicates a strong rejection of the null hypothesis in favor of concluding that the median of the MA objective function values for structured graphs is less than the median of the TS values.

We complement this experiment by conducting a *t*-test for paired samples. The resulting *p*-values of 0.032 and 0.106 for random and structured graphs, respectively, means that, in the case of random graphs, we could still reject the null hypothesis at a reasonable level of significance, say 5%. However, we would have to accept a Type I error of over 10% if we would like to reject the null hypothesis for the structured graphs in favor of concluding that the average MA objective function values for structured graphs is less than the TS average.

We believe that the solid performance of MA on structured graphs is due to the six different ways in which solutions are constructed to initialize the search. This fact is also supported by the authors of the MA procedure, since they tested in Jain et al. (2016) the six constructive heuristics in isolation for the different kinds of instances. The differences found in the performance of each constructive procedure for each kind of graph, suggested them the inclusion of solutions provided by all the constructives, in the initial population of their MA. At least one of these constructives can be customized to exploit a particular regular structure and give the search procedure the advantage of starting the search at high-quality initial points. Customization of a solution-construction procedure is not possible for graphs without a regular structure, such as random graphs. Including various forms of constructing solutions within a single procedure is indeed a reasonable idea as long as the application can afford the price to be paid on the increased computational effort.

Given the previous assumptions, as a final experiment, we evaluate the influence of the constructive procedures in the overall performance of the compared methods. In Table 6, we report the average quality (Avg.), the number of best solutions found (#Best), and the average improvement, Imp. (%), achieved by the TS or the MA with respect to its corresponding constructive procedure. Notice that, the overall method (TS or MA) also includes the con-

Table 6
Study of the influence of the constructive procedures in the overall performance of the compared methods.

		Our approach			P. Jain et al. [32]		
		Greedy	TS	Imp.(%)	6 cons.	MA	Imp.(%)
Small Instances (84)	Avg.	5.90	3.90	51.62	4.57	3.98	15.81
	#Best	5	79		37	47	
Harwell-Boeing (38)	Avg.	87.24	67.26	47.48	95.95	70.71	40.72
	#Best	1	37		3	35	
Complete Split (21)	Avg.	360.38	217.76	72.55	217.95	217.33	0.18
	#Best	0	21		14	7	
Join of Hypercube (9)	Avg.	98.78	80.89	31.54	84.11	80.33	6.83
	#Best	0	9		0	9	
Toroidal Mesh (17)	Avg.	36.53	30.24	27.06	31.12	30.24	3.15
	#Best	7	10		11	6	
Cone (10)	Avg.	184.70	154.80	22.25	157.00	155.00	2.29
	#Best	0	10		2	8	
Total (179)	Avg.	82.31	57.24	48.21	64.04	57.94	16.85
	#Best	13	166		67	112	

structive phase and therefore, if we just report the total number of best solutions found by TS or MA it would always match the total number of instances. However, since we are interested in determining which part of the method is responsible of the best solution found, in Table 6, we have separated the number of best solutions found by the constructive procedure, from the total number of best solutions found by the metaheuristic. The column Greedy in Table 6 reports the initial solution provided to the TS in the best iteration. On the other hand, the MA runs 6 constructive procedures (6 cons. column in Table 6) for 30 iterations with an initial population size of 60 or 90 individuals (depending on the size of the graph) at each iteration. We have performed all the possible initial constructions for the 30 iterations with the code provided by the authors in Jain et al. (2016) and we have reported the best solution found among all of them. As it can be seen in Table 6, considering all the instances together (row Total) the TS was able to improve 166 initial solutions out of 179, while 13 solutions generated by the Greedy constructive were not improved. Furthermore, the TS improved the Avg. value of the objective function in a 48.21% over the Greedy procedure. On the other hand, MA was able to improve 112 initial solutions out of 179, and the average improvement of the MA with respect to the constructive procedures resulted in a 16.85%. If we take a closer look to the sets of instances separately, we observe a different influence of the initial solution in the search, depending on the type of instance. Particularly, in the sets of instances where the MA obtains a more competitive performance (the graphs with a regular structure) there are a smaller improving difference with respect to its constructive procedure. This indicates that the solution quality for those sets of instances is mainly due to the constructive phase. In this sense, the differences in the Complete Split Graphs are the more remarkable, since the TS was able to improve a 72.55% with respect to the initial solution, while the MA was only able to improve a 0.18%, obtaining almost the same final quality. We believe that this behavior is due to the fact that in Jain et al. (2016) the authors proposed six tailored-made constructive procedures able to perform very well for the particular sets of instances used. We can conclude that, TS is able to largely improve the initial solutions provided by a general constructive procedure. On the other hand, the use of a set of tailored-made constructive procedures when handling different sets of instances, might be useful to provide high-quality starting points for improvement methods.

Appendix A includes the individual results of our competitive tests. This could help researchers to perform future comparisons.

5. Conclusions

We studied the Cyclic Cutwidth Minimization Problem consisting of embedding a candidate graph into a cycle (host) graph in order to minimize the maximum cut. This problem has been previously studied for specific classes of graphs with a regular structure. However, work on general candidate graphs is sparse. We can point to only one recent heuristic approach for general graphs. The approach in the literature is a population-based metaheuristic from the family of memetic algorithms. We took a different approach and developed a single-solution, neighborhood search. In the process of creating an effective and efficient solution method, we adapted three strategies that have general applicability:

1. Efficient move value calculation.
2. Secondary move evaluation to distinguish moves that the primary evaluation (based on the objective function) is not able to distinguish.
3. Neighborhood search space reduction via regions of interest.

Our work establishes some new benchmarks for the Cyclic Cutwidth Minimization Problem and provides validation for strategies that promise to accelerate the execution of heuristic searches without sacrificing solution quality. Finally, we have identified a general new lower bound for CCMP, valid for any kind of graph and inspired in the linear version of the problem.

CRediT authorship contribution statement

Sergio Cavero: Conceptualization, Investigation, Data curation, Methodology, Software. **Eduardo G. Pardo:** Conceptualization, Investigation, Validation, Writing - original draft. **Manuel Laguna:** Supervision, Formal analysis, Writing - review & editing. **Abraham Duarte:** Supervision, Writing - review & editing, Funding acquisition.

Acknowledgment

This work was partially supported by the Ministerio de Ciencia, Innovación y Universidades under Grant Ref. PGC2018-095322-B-C22 and Grant Ref. FPU19/04098 by Comunidad de Madrid and European Regional Development Fund with Grant Ref. P2018/TCS-4566. We would also like to thank P. Jain, K. Srivastava, and G. Saran, authors of the previous most competitive method in the state of art (Jain et al., 2016) for sharing their code with us.

Appendix A. Individual results

These are the results of our competitive testing. These values were used to calculate the summary presented in Table 5.

Small Instances

	LB	TS			MA		
Instance	***ccw***	***ccw***	**CPUt (s)**	**Dev. (%)**	***ccw***	**CPUt (s)**	**Dev. (%)**
p17_16_24	4	5	0.27	0.00	5	16	0.00
p18_16_21	3	4	0.20	0.00	4	13	0.00
p19_16_19	3	3	0.26	0.00	3	11	0.00
p20_16_18	4	4	0.20	0.00	4	12	0.00
p21_17_20	3	3	0.12	0.00	3	12	0.00
p22_17_19	3	3	0.24	0.00	3	15	0.00
p23_17_23	3	4	0.39	0.00	4	16	0.00
p24_17_29	4	5	0.39	0.00	6	15	20.00
p25_17_20	3	3	0.20	0.00	3	13	0.00
p26_17_19	3	3	0.16	0.00	3	12	0.00
p27_17_19	3	3	0.30	0.00	3	12	0.00
p28_17_18	3	3	0.18	0.00	3	12	0.00
p29_17_18	3	3	0.16	0.00	3	13	0.00
p30_17_19	2	3	0.20	0.00	3	13	0.00
p31_18_21	3	3	0.22	0.00	3	13	0.00
p32_18_20	3	3	0.32	0.00	3	15	0.00
p33_18_21	3	3	0.17	0.00	3	15	0.00
p34_18_21	3	3	0.32	0.00	3	14	0.00
p35_18_19	3	3	0.09	0.00	3	14	0.00
p36_18_20	3	3	0.24	0.00	3	13	0.00
p37_18_20	4	4	0.14	0.00	4	13	0.00
p38_18_19	2	3	0.15	0.00	3	13	0.00
p39_18_19	3	3	0.26	0.00	3	12	0.00
p40_18_32	4	6	0.68	0.00	6	17	0.00
p41_19_20	3	3	0.16	0.00	3	11	0.00
p42_19_24	2	4	0.18	0.00	4	16	0.00
p43_19_22	3	3	0.22	0.00	3	13	0.00
p44_19_25	3	4	0.35	0.00	4	15	0.00
p45_19_25	4	4	0.23	0.00	4	15	0.00
p46_19_20	3	3	0.16	0.00	3	12	0.00
p47_19_21	2	3	0.19	0.00	3	14	0.00
p48_19_21	3	3	0.14	0.00	3	13	0.00
p49_19_22	3	3	0.21	0.00	3	14	0.00
p50_19_25	3	3	0.29	0.00	3	15	0.00
p51_20_28	3	4	0.56	0.00	4	24	0.00
p52_20_27	4	4	0.34	0.00	4	15	0.00
p53_20_22	3	3	0.27	0.00	3	14	0.00
p54_20_28	4	5	0.32	0.00	5	17	0.00
p55_20_24	3	3	0.14	0.00	3	15	0.00
p56_20_23	2	3	0.30	0.00	3	14	0.00

(*continued on next page*)

(continuation)	LB	TS			MA		
Instance	ccw	ccw	**CPUt (s)**	**Dev. (%)**	ccw	**CPUt (s)**	**Dev. (%)**
p57_20_24	4	4	0.33	0.00	4	14	0.00
p58_20_21	3	3	0.07	0.00	3	14	0.00
p59_20_23	3	4	0.37	0.00	4	15	0.00
p60_20_22	3	3	0.63	0.00	3	19	0.00
p61_21_22	3	3	0.27	0.00	3	15	0.00
p62_21_30	3	5	0.34	0.00	5	17	0.00
p63_21_42	5	7	0.62	0.00	8	27	14.29
p64_21_22	3	3	0.17	0.00	3	14	0.00
p65_21_24	2	3	0.39	0.00	4	18	33.33
p66_21_28	4	5	0.42	0.00	5	19	0.00
p67_21_22	3	3	0.07	0.00	3	14	0.00
p68_21_27	4	4	0.26	0.00	4	18	0.00
p69_21_23	3	3	0.19	0.00	4	15	33.33
p70_21_25	3	4	0.26	0.00	4	16	0.00
p71_22_29	2	4	0.38	0.00	4	19	0.00
p72_22_49	4	9	1.26	0.00	9	28	0.00
p73_22_29	3	4	0.38	0.00	4	16	0.00
p74_22_30	4	5	0.51	0.00	5	18	0.00
p75_22_25	3	4	0.32	0.00	4	16	0.00
p76_22_30	4	4	0.84	0.00	5	19	25.00
p77_22_37	3	6	0.56	0.00	6	24	0.00
p78_22_31	3	5	0.37	0.00	5	22	0.00
p79_22_29	3	5	0.39	0.00	5	21	0.00
p80_22_30	3	5	0.38	0.00	5	20	0.00
p81_23_46	4	8	0.45	0.00	8	27	0.00
p82_23_24	4	4	0.25	0.00	4	20	0.00
p83_23_24	4	4	0.17	0.00	4	19	0.00
p84_23_26	3	4	0.27	0.00	4	21	0.00
p85_23_26	3	4	0.32	0.00	4	18	0.00
p86_23_24	3	3	0.22	0.00	3	17	0.00
p87_23_30	3	4	0.28	0.00	4	18	0.00
p88_23_26	3	4	0.16	0.00	4	16	0.00
p89_23_27	3	4	0.33	0.00	4	19	0.00
p90_23_35	3	5	0.41	0.00	5	21	0.00
p91_24_33	3	5	0.33	0.00	5	21	0.00
p92_24_26	2	4	0.40	0.00	4	18	0.00
p93_24_27	4	4	0.40	0.00	4	17	0.00
p94_24_31	3	4	0.49	0.00	4	23	0.00
p95_24_27	3	4	0.44	0.00	4	19	0.00
p96_24_27	3	3	0.38	0.00	3	17	0.00
p97_24_26	4	4	0.36	0.00	4	17	0.00
p98_24_29	3	4	0.66	0.00	4	27	0.00
p99_24_27	3	4	0.14	0.00	4	22	0.00
p100_24_34	4	5	0.41	0.00	5	24	0.00
Avg.	3.16	3.90	0.32	0.00	3.98	16.67	1.80

Harwell-Boeing

	LB	TS			MA		
Instance	*ccw*	*ccw*	**CPUt (s)**	**Dev. (%)**	*ccw*	**CPUt (s)**	**Dev. (%)**
494_bus	5	25	288.88	25.00	20	3427	0.00
662_bus	5	31	306.09	0.00	38	18132	22.58
685_bus	6	37	304.98	0.00	50	14894	35.14
arc130	62	122	302.89	0.00	143	2350	17.21
ash292	7	34	309.38	0.00	47	5102	38.24
ash85	5	14	13.37	0.00	16	232	14.29
bcspwr01	3	4	0.45	0.00	5	28	25.00
bcspwr02	3	5	1.46	0.00	5	44	0.00
bcspwr03	5	10	18.36	0.00	11	399	10.00
bcspwr04	8	36	306.27	2.86	35	5483	0.00
bcspwr05	5	25	82.35	0.00	26	3347	4.00
bcsstk01	6	25	12.03	0.00	27	89	8.00
bcsstk02	33	561	8.42	2.75	546	130	0.00
bcsstk04	22	302	410.32	0.00	311	1599	2.98
bcsstk05	12	113	165.78	0.00	114	2387	0.88
bcsstk06	14	254	2125.71	18.14	215	17319	0.00
bcsstk22	4	10	43.15	0.00	11	447	10.00
can__144	7	25	53.82	25.00	20	1177	0.00
can__161	8	45	84.09	0.00	48	517	6.67
can__292	17	67	309.42	0.00	105	10488	56.72
curtis54	8	11	6.20	0.00	12	65	9.09
dwt__209	8	47	147.80	0.00	55	6286	17.02
dwt__221	6	27	90.53	0.00	28	9233	3.70
dwt__234	5	12	8.40	0.00	14	969	16.67
dwt__245	6	32	140.78	0.00	38	3529	18.75
fs_183_1	52	113	303.52	0.00	136	6773	20.35
gent113	13	70	187.25	0.00	80	1721	14.29
gre__115	5	25	33.82	0.00	28	916	12.00
gre__185	4 6	46	79.24	9.52	42	2797	0.00
ibm32	6	15	3.40	0.00	15	55	0.00
impcol_b	9	44	15.69	0.00	47	247	6.82
impcol_c	7	33	34.09	0.00	41	1163	24.24
lns__131	6	24	18.29	0.00	25	738	4.17
lund_a	10	107	221.78	8.08	99	3413	0.00
lund_b	10	100	300.58	0.00	101	2399	1.00
saylr3	3	46	302.45	0.00	60	18000	30.43
west0132	11	48	84.28	0.00	62	104	29.17
will57	5	11	2.96	0.00	11	112	0.00
Avg.	10.84	67.26	187.59	2.40	70.71	4081.87	12.09

Complete Split

	LB	TS			MA		
Instance	*ccw*	*ccw*	**CPUt (s)**	**Dev. (%)**	*ccw*	**CPUt (s)**	**Dev. (%)**
$K_4 \nabla \overline{K_{10}}$	7	13	0.92	0.00	13	27	0.00
$K_4 \nabla \overline{K_{15}}$	9	17	0.48	0.00	17	49	0.00
$K_4 \nabla \overline{K_{20}}$	12	23	2.63	0.00	23	69	0.00
$K_4 \nabla \overline{K_{30}}$	17	33	5.27	0.00	33	123	0.00
$K_4 \nabla \overline{K_{50}}$	27	53	16.71	0.00	53	273	0.00
$K_4 \nabla \overline{K_{100}}$	52	103	98.15	0.00	103	101	0.00
$K_5 \nabla \overline{K_5}$	5	10	0.93	0.00	10	19	0.00
$K_5 \nabla \overline{K_{10}}$	7	16	1.20	0.00	16	34	0.00
$K_5 \nabla \overline{K_{20}}$	12	29	3.26	0.00	29	88	0.00
$K_5 \nabla \overline{K_{50}}$	27	68	15.71	0.00	68	303	0.00
$K_5 \nabla \overline{K_{100}}$	52	133	117.84	0.00	133	887	0.00
$K_6 \nabla \overline{K_{15}}$	10	27	1.81	0.00	27	81	0.00
$K_6 \nabla \overline{K_{50}}$	28	81	30.88	1.25	80	468	0.00
$K_6 \nabla \overline{K_{100}}$	53	156	145.45	0.65	155	1419	0.00
$K_{10} \nabla \overline{K_{15}}$	12	50	5.09	0.00	50	149	0.00
$K_{10} \nabla \overline{K_{50}}$	30	139	91.92	0.78	138	765	0.00
$K_{10} \nabla \overline{K_{100}}$	55	264	310.32	0.38	263	2655	0.00
$K_{20} \nabla \overline{K_{50}}$	35	302	310.33	0.33	301	2299	0.00
$K_{20} \nabla \overline{K_{100}}$	60	552	925.71	0.18	551	7384	0.00
$K_{50} \nabla \overline{K_{50}}$	50	940	2705.05	0.21	938	17726	0.00
$K_{50} \nabla \overline{K_{100}}$	75	1564	16087.62	0.06	1563	18059	0.00
Avg.	30.24	217.76	994.16	0.18	217.34	2522.81	0.00

Cones

	LB	TS			MA		
Instance	*ccw*	*ccw*	**CPUt (s)**	**Dev. (%)**	*ccw*	**CPUt (s)**	**Dev. (%)**
$C_{10,10}$	6	26	1.58	0.00	26	64	0.00
$C_{10,15}$	9	39	4.64	0.00	39	127	0.00
$C_{10,20}$	11	51	3.67	0.00	51	194	0.00
$C_{10,50}$	26	126	46.89	0.00	127	619	0.79
$C_{15,15}$	9	58	17.40	0.00	58	184	0.00
$C_{15,20}$	11	77	18.98	0.00	78	252	1.30
$C_{15,50}$	26	190	162.52	0.00	191	1676	0.53
$C_{20,20}$	11	102	9.65	0.99	101	432	0.00
$C_{20,50}$	26	252	226.61	0.00	252	2003	0.00
$C_{50,50}$	26	627	311.24	0.00	627	8504	0.00
Avg.	16.1	154.80	80.32	0.10	155.00	1405.45	0.26

Join of Hypercubes

	LB	TS			MA		
Instance	*ccw*	*ccw*	**CPUt (s)**	**Dev. (%)**	*ccw*	**CPUt (s)**	**Dev. (%)**
$Q_2 + Q_3$	5	12	1.02	0.00	12	26	0.00
$Q_2 + Q_4$	9	23	2.25	0.00	23	62	0.00
$Q_2 + Q_5$	17	46	10.77	0.00	46	200	0.00
$Q_3 + Q_3$	6	21	2.92	0.00	21	49	0.00
$Q_3 + Q_4$	10	40	7.11	0.00	40	118	0.00
$Q_3 + Q_5$	18	79	34.47	0.00	79	337	0.00
$Q_4 + Q_4$	10	76	11.67	0.00	76	257	0.00
$Q_4 + Q_5$	18	147	132.50	1.38	145	675	0.00
$Q_5 + Q_5$	19	284	302.05	1.07	281	1622	0.00
Avg.	12.45	80.89	56.08	0.27	80.34	369.91	0.00

Torodial Mesh

	LB	TS			MA		
Instance	*ccw*	*ccw*	**CPUt (s)**	**Dev. (%)**	*ccw*	**CPUt (s)**	**Dev. (%)**
$C_3 \times C_3 \times C_3$	3	14	0.52	0.00	14	57	0.00
$C_3 \times C_3 \times C_4$	3	17	1.22	0.00	17	85	0.00
$C_3 \times C_3 \times C_5$	3	17	1.73	0.00	17	112	0.00
$C_3 \times C_3 \times C_{10}$	3	17	20.20	0.00	17	294	0.00
$C_3 \times C_4 \times C_4$	3	20	2.43	0.00	20	107	0.00
$C_3 \times C_4 \times C_5$	3	20	7.30	0.00	20	150	0.00
$C_3 \times C_4 \times C_{10}$	3	22	56.54	10.00	20	530	0.00
$C_3 \times C_5 \times C_5$	3	23	7.38	0.00	23	299	0.00
$C_3 \times C_5 \times C_{10}$	3	23	89.24	0.00	23	1013	0.00
$C_3 \times C_10 \times C_{10}$	3	38	300.66	0.00	46	6214	21.05
$C_4 \times C_4 \times C_4$	3	26	5.82	0.00	26	217	0.00
$C_4 \times C_4 \times C_5$	3	26	18.04	0.00	26	421	0.00
$C_4 \times C_4 \times C_{10}$	3	26	115.00	0.00	26	1114	0.00
$C_4 \times C_5 \times C_5$	3	32	31.46	6.67	30	945	0.00
$C_4 \times C_5 \times C_{10}$	3	32	260.36	6.67	30	1640	0.00
$C_5 \times C_5 \times C_5$	3	37	50.13	0.00	37	670	0.00
$C_{10} \times C_{10} \times C_{10}$	3	124	1451.31	1.64	122	17948	0.00
Avg.	3	30.24	142.31	1.47	30.24	1871.53	1.24

References

Abbott, H.L., 1966. Hamiltonian circuits and paths on the n-cube. Canadian Mathematical Bulletin 9, 557–562.

Allmond, H., 2006. On the cyclic cutwidth of complete tripartite and n-partite graphs. REU Project.

Aschenbrenner, R., 2001. A proof for the cyclic cutwidth of q5, REU Project. Cal State Univ, San Bernardino.

Castillo, C., 2003. A proof for the cyclic cutwidth of q6, REU Project. Cal State Univ, San Bernardino.

Chavez, J.D., Trapp, R., 1998. The cyclic cutwidth of trees. Discrete Applied Mathematics 87, 25–32.

Clarke, D.W., 2002. The cyclic cutwidth of mesh cubes.

Cohoon, J.P., Sahni, S., 1987. Heuristics for backplane ordering. Journal of VLSI and computer systems 2, 37–60.

Díaz, J., Petit, J., Serna, M., 2002. A survey of graph layout problems. ACM Comput. Surv. 34, 313–356.

Duarte, A., Pantrigo, J.J., Pardo, E.G., Sánchez-Oro, J., 2016. Parallel variable neighbourhood search strategies for the cutwidth minimization problem. IMA Journal of Management Mathematics 27, 55. https://doi.org/10.1093/imaman/dpt026.

Duff, I.S., Grimes, R.G., Lewis, J.G., 1992. Users' guide for the harwell-boeing sparse matrix collection (release i).

Erbele, J., Chavez, J.D., Trapp, R., 2003. The cyclic cutwidth of qn, Manuscript. California State University, San Bernardino, USA.

Feo, T., Resende, M., 1994. A greedy randomized adaptive search procedure for maximum independent set. Operations Research 42, 860–878.

Gavril, F., 1977. Some np-complete problems on graphs. In: Proceedings of the Eleventh Conference on Information Sciences and Systems, pp. 91–95.

Glover, F., 1977. Heuristics for integer programming using surrogate constraints. Decision Sciences 8, 156–166.

Glover, F., 1986. Future paths for integer programming and links to artificial intelligence. Computers & Operations Research 13, 533–549.

Glover, F., 1989. Tabu search–part i. ORSA Journal on Computing 1, 190–206.

Glover, F., 1990. Tabu search–part ii. ORSA Journal on Computing 2, 4–32.

Glover, F., 1997. Tabu search and adaptive memory programming–advances, applications and challenges. In: Interfaces in Computer Science and Operations Research. Springer, pp. 1–75.

Glover, F., Laguna, M., 1997. Tabu Search. Kluwer Academic Publishers, USA.

Hansen, P., Mladenovic, N., 2006. First vs. best improvement: an empirical study. Discrete Applied Mathematics 154, 802–817.

Harper, L.H., 1966. Optimal numberings and isoperimetric problems on graphs. Journal of Combinatorial Theory 1, 385–393.

Jain, P., Srivastava, K., Saran, G., 2016. Minimizing cyclic cutwidth of graphs using a memetic algorithm. Journal of Heuristics 22, 815–848.

James, B., 1996. The cyclical cutwidth of the three-dimensional and fourdimensional cubes. Cal State Univ., San Bernardino McNair Scholar's Program Summer Research Journal.

Johnson, M., 2003. The linear and cyclic cutwidth of the complete bipartite graph, REU Project, Cal State Univ., San Bernardino.

Laguna, M., 2017. Tabu Search. Springer International Publishing, Cham, pp. 741–758.

López-Locés, M.C., Castillo-García, N., Huacuja, H.J.F., Bouvry, P., Pecero, J.E., Rangel, R.A.P., Barbosa, J.J.G., Valdez, F., 2014. A new integer linear programming model for the cutwidth minimization problem of a connected undirected graph. In:

Recent Advances on Hybrid Approaches for Designing Intelligent Systems. Springer, pp. 509–517.

Luttamaguzi, J., Pelsmajer, M., Shen, Z., Yang, B., 2005. Integer programming solutions for several optimization problems in graph theory, Technical Report, Center for Discrete Mathematics and Theoretical Computer Science, DIMACS.

Makedon, F., Sudborough, I.H., 1989. On minimizing width in linear layouts. Discrete Applied Mathematics 23, 243–265.

Martí, R., Pantrigo, J.J., Duarte, A., Pardo, E.G., 2013. Branch and bound for the cutwidth minimization problem. Computers & Operations Research 40, 137–149.

Martins Santos, V.G., Moreira de Carvalho, A.M., 2019. Tailored heuristics in adaptive large neighborhood search applied to the cutwidth minimization problem. European Journal of Operational Research. https://doi.org/10.1016/j.ejor.2019.07.013.

Mcallister, A.J., 1999. A new heuristic algorithm for the linear arrangement problem.

Moscato, P., 1989. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms, Caltech concurrent computation program, C3P Report 826 (1989).

Palubeckis, G., 2012. A branch-and-bound algorithm for the single-row equidistant facility layout problem. OR Spectrum 34, 1–21.

Pantrigo, J.J., Martí, R., Duarte, A., Pardo, E.G., 2012. Scatter search for the cutwidth minimization problem. Annals of Operations Research 199, 285–304.

Pardo, E.G., Mladenović, N., Pantrigo, J.J., Duarte, A., 2013. Variable formulation search for the cutwidth minimization problem. Applied Soft Computing 13, 2242–2252.

Pardo, E.G., Martí, R., Duarte, A., 2016. Linear Layout Problems, Handbook of Heuristics. Springer International Publishing. ISBN: 978-3-319-07153-4.

Pinana, E., Plana, I., Campos, V., Martı, R., 2004. Grasp and path relinking for the matrix bandwidth minimization. European Journal of Operational Research 153, 200–210.

Resende, M.G.C., 2009. A.D.V. In: Method and system for network migration scheduling. United States Patent Application Publication. US2009/0168665.

Resende, M.G.C., Martí, R., Gallego, M., Duarte, A., 2010. Grasp and path relinking for the max–min diversity problem. Computers & Operations Research 37, 498–508.

Rios, F., 1996. Complete graphs as a first step toward finding the cyclic cutwidth of the n-cube. Cal State Univ., San Bernardino McNair Scholar's Program Summer Research Journal.

Rolim, J., Sýkora, O., Vrt'o, I., 1995. Optimal cutwidths and bisection widths of 2-and 3-dimensional meshes. In: International Workshop on Graph-Theoretic Concepts in Computer Science, Springer, pp. 252–264.

Schröder, H., Sýykoa, O., Vrt'o, I., 1999. Cyclic cutwidth of the mesh. In: International Conference on Current Trends in Theory and Practice of Computer Science. Springer, pp. 449–458.

Schröder, H., Sýkora, O., Vrt'o, I., 2004. Cyclic cutwidths of the two-dimensional ordinary and cylindrical meshes. Discrete Applied Mathematics 143, 123–129.

Sciortino, V., Chavez, J.D., Trapp, R., 2002. The cyclic cutwidth of a $p2 \times p2 \times$ pn mesh, REU Project, Cal State Univ., San Bernardino.

Shahrokhi, F., Sýkora, O., Székely, L.A., Vrt'o, I., 2001. On bipartite drawings and the linear arrangement problem. SIAM Journal on Computing 30, 1773–1789.

Thilikos, D.M., Serna, M., Bodlaender, H.L., 2005. Cutwidth ii: Algorithms for partial w-trees of bounded degree. Journal of Algorithms 56, 25–49.

Wilcoxon, F., 1992. Individual comparisons by ranking methods, in: Breakthroughs in statistics, Springer, pp. 196–202.

Chapter 8

Cyclic Antibandwidth Problem

The Cyclic Antibandwidth Problem is the second of the GLPs studied in this Doctoral Thesis, and it was previously introduced in Section 2.2. As a result of the research conducted, an article has been published:

1. S. Cavero, E. G. Pardo, and A. Duarte. A general variable neighborhood search for the cyclic antibandwidth problem. *Computational Optimization and Applications*, 81(2):657–687, 2022 [31].

Moreover, a presentation has been at an international conference:

2. S. Cavero, E. G. Pardo, and A. Duarte. A vns approach for a variant of the antibandwidth problem. *8th International Conference on Variable Neighborhood Search* (ICVNS 2021), in Abu Dhabi, U.A.E., 2021 [30].

The article, titled "*A general variable neighborhood search for the cyclic antibandwidth problem*" [31] was published in a JCR journal. Figure 8.1 compiles some information about the journal. The CAB was previously studied for specific classes of graphs using exact methods. However, just two approaches can be found in the literature for general graphs: a MA and an ABC combined with TS.

To handle the problem, we propose a GVNS in a multistart algorithmic design. The main components of the algorithm proposed to address the CAB are summarized next:

- Two-phase **greedy constructive procedure**. The first phase determines the next vertex to be added to the solution based on the adjacency of the vertices and the BFS algorithm [229]. Then, a greedy function based on the objective function of the problem is used to locate the selected vertex in the host graph.

- **Two local search procedures** are proposed. The first one explores the insert neighborhood following a best improvement strategy. The second local search explores the swap neighborhood following a first improvement strategy. Additionally, the proposal takes advantage of two exploration strategies: a criterion for breaking the tie of solutions with the same objective function and an efficient evaluation of neighboring solutions. Furthermore, two neighborhood reduction strategies are proposed.

- The previous heuristic procedures are framed within the **GVNS** metaheuristic. This framework combines an efficient VND, based on the aforementioned local search procedures, with a novel destruction–reconstruction shaking procedure.

Our procedure has been tested in isolation to verify that the new proposed GVNS method, in combination with the additional advanced strategies, is capable of accelerating the execution of heuristic search procedures without sacrificing the quality of the solutions found. Then, the final configuration of the method has been successfully compared with the previous state-of-the-art procedures, becoming the new state-of-the-art method for the CAB. The merit of the empirical results obtained was corroborated using non-parametrical statistical tests.

To conclude this chapter, we include a copy of the most relevant paper published for the CAB in the context of this Doctoral Thesis.

A GVNS for the cyclic antibandwidth problem

Sergio Cavero, Eduardo G. Pardo and Abraham Duarte

Computational Optimization and Applications. Volume 81(2), 657–687, 2022.

`https://link.springer.com/article/10.1007/s10589-021-00334-y`

Journal Information

Research Areas:

- Operations Research & Management Science
- Engineering (Industrial)

Category Rank:

- Operations Research & Management Science 78/267: (Q2)
- Mathematics (Applied) 59/87: (Q3)

Journal Impact Factor: 2.005

Data obtained from Journal Citation Reports 2021

Figure 8.1 Information related to the publication [31].

Computational Optimization and Applications (2022) 81:657–687
https://doi.org/10.1007/s10589-021-00334-y

A general variable neighborhood search for the cyclic antibandwidth problem

Sergio Cavero[1] · **Eduardo G. Pardo**[1] · **Abraham Duarte**[1]

Received: 8 February 2021 / Accepted: 12 November 2021 / Published online: 17 January 2022

Abstract
Graph Layout Problems refer to a family of optimization problems where the aim is to assign the vertices of an input graph to the vertices of a structured host graph, optimizing a certain objective function. In this paper, we tackle one of these problems, named Cyclic Antibandwidth Problem, where the objective is to maximize the minimum distance of all adjacent vertices, computed in a cycle host graph. Specifically, we propose a General Variable Neighborhood Search which combines an efficient Variable Neighborhood Descent with a novel destruction–reconstruction shaking procedure. Additionally, our proposal takes advantage of two new exploration strategies for this problem: a criterion for breaking the tie of solutions with the same objective function and an efficient evaluation of neighboring solutions. Furthermore, two new neighborhood reduction strategies are proposed. We conduct a thorough computational experience by comparing the algorithm proposed with the current state-of-the-art methods over a set of previously reported instances. The associated results show the merit of the introduced algorithm, emerging as the best performance method in those instances where the optima are unknown. These results are further confirmed with nonparametric statistical tests.

Keywords Cyclic antibandwidth problems · Graph layout problem · Metaheuristics · Variable neighborhood search · Combinatorial optimization

✉ Abraham Duarte
abraham.duarte@urjc.es

Sergio Cavero
sergio.cavero@urjc.es

Eduardo G. Pardo
eduardo.pardo@urjc.es

[1] Universidad Rey Juan Carlos, Madrid, Spain

1 Introduction

In recent years, there has been a growing interest in studying the assignment of the vertices of a generic input graph to the vertices of a regular host graph (e.g., path, cycle, grid, or torus, among others), optimizing a particular objective function [11]. This idea has been previously denoted in the literature as the embedding of a graph into another graph [16, 29]. Also, it is possible to find references to this concept as the labeling of a graph (i.e., the vertices of the graph receive labels) [31], the layout of the graph (i.e., the vertices of a graph are shown in a line, cycle, grid, etc.) [11], or the mapping of a graph (i.e., a pair of functions assigning the vertices and edges of an input graph to the vertices and paths of a host graph, respectively) [17]. Within this family of problems, the scientific community has mainly paid attention to those optimization problems where the host graph is a path graph. See, for instance, the Cutwidth Minimization Problem [8, 28], the Minimum Linear Arrangement [23, 35], or the Vertex Separation Problem [7, 41]. These problems, where the mapping is performed over a path host graph, are usually known as Linear Layout Problems [27].

In this paper, we focus on the Cyclic Antibandwidth Problem, which belongs to a particular subfamily of graph layout problems where the aim is to embed the input graph in a cycle host graph. Within this subfamily, we can also find, among others, the Cyclic Cutwidth Minimization Problem [5, 17], the Cyclic Bandwidth Problem [34, 38], or Cyclic Bandwidth Sum Problem [36, 37].

To formally define the concept of embedding in a cycle, let $\mathcal{G} = (\mathcal{V}_\mathcal{G}, \mathcal{E}_\mathcal{G})$ be a connected, unweighted, and undirected candidate graph where the set of vertices is denoted as $\mathcal{V}_\mathcal{G}$ (with $|\mathcal{V}_\mathcal{G}| = n$) and its edge set as $\mathcal{E}_\mathcal{G}$ (with $|\mathcal{E}_\mathcal{G}| = m$). Analogously, let $\mathcal{C}_n = (\mathcal{V}_\mathcal{C}, \mathcal{E}_\mathcal{C})$ be a host cycle graph where $\mathcal{V}_\mathcal{C}$ and $\mathcal{E}_\mathcal{C}$ represent the sets of vertices and edges respectively (with $|\mathcal{V}_\mathcal{C}| = |\mathcal{E}_\mathcal{C}| = n$). Due to the fact that the host graph is a cycle, $\mathcal{C}_n$ is a 2-regular, Eulerian, Hamiltonian, and unit-distance graph.

Figure 1a shows an example of an input graph, $\mathcal{G}_1$, with six vertices $\mathcal{V}_\mathcal{G} = \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}, \mathrm{E}, \mathrm{F}\}$ and six edges $\mathcal{E}_\mathcal{G} = \{(\mathrm{A}, \mathrm{B}), (\mathrm{A}, \mathrm{C}), (\mathrm{B}, \mathrm{E}), (\mathrm{B}, \mathrm{F}), (\mathrm{D}, \mathrm{E}), (\mathrm{E}, \mathrm{F})\}$. In Fig. 1b, we show

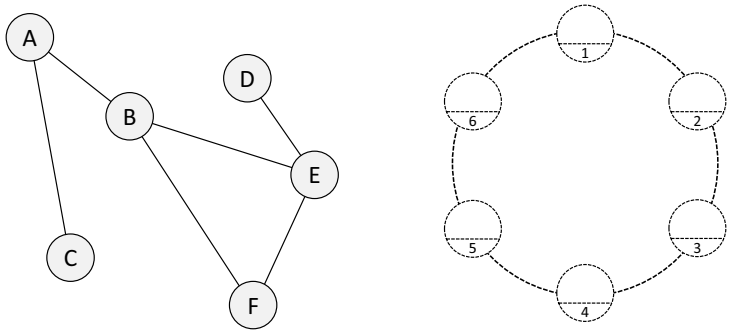


Fig. 1 **a** A candidate graph $\mathcal{G}_1$, **b** A host graph $\mathcal{C}_6$

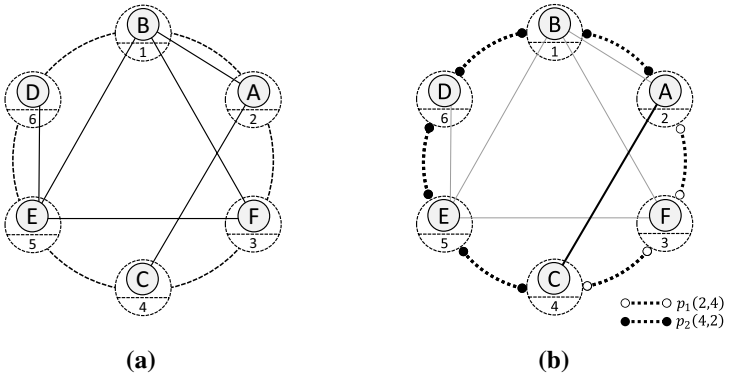


Fig. 2 **a** A possible embedding φ' of $\mathcal{G}_1$ in $\mathcal{C}_6$. **b** Two possible paths in $\mathcal{C}_6$ for $(2,4) \in \mathcal{E}_\mathcal{C}$ denoted as $p_1(2,4)$ and $p_2(4,2)$

a cycle graph $\mathcal{C}_6$ graph where the set of edges an vertices are $\mathcal{V}_\mathcal{C} = \{1,2,3,4,5,6\}$ and $\mathcal{E}_\mathcal{C} = \{(1,2),(2,3),(3,4),(4,5),(5,6),(6,1)\}$, respectively.

As it was aforementioned, an embedding consists in defining two mathematical functions. The first one, φ, is a bijective function that assigns each vertex of the candidate graph to a vertex of the host graph. In mathematical terms,

$$\varphi : \mathcal{V}_\mathcal{G} \rightarrow \mathcal{V}_\mathcal{C}, \text{ where } \forall v \in \mathcal{V}_\mathcal{G} \exists! u \in \mathcal{V}_\mathcal{C} | \varphi(v) = u. \tag{1}$$

The second one, ψ, is an injective function that maps each edge $(u,v) \in \mathcal{E}_\mathcal{G}$ to a set of paths in $\mathcal{C}_n$ whose endpoints are $\varphi(u)$ and $\varphi(v)$. It is worth mentioning that a path is a sequence of edges, without repeating neither vertices nor edges. Thus, considering that $\mathcal{C}_n$ is a cycle graph, given two vertices $u, v \in \mathcal{V}_\mathcal{G}$ there exists only two feasible paths p_1, p_2 between $\varphi(u), \varphi(v) \in \mathcal{V}_\mathcal{C}$. More formally,

$$\begin{gathered} \psi : \mathcal{E}_\mathcal{G} \rightarrow \mathcal{P}_\mathcal{C}, \text{where } \forall (u,v) \in \mathcal{E}_\mathcal{G} \exists \{p_1, p_2\} \in \mathcal{P}_\mathcal{C} \text{with} \\ \varphi(u), \varphi(v) \in \mathcal{V}_\mathcal{C} \wedge \psi((u,v)) = \begin{cases} p_1(\varphi(u), \varphi(v)) \\ p_2(\varphi(v), \varphi(u)) \end{cases}, \text{with } p_1 \neq p_2, \end{gathered} \tag{2}$$

where $\mathcal{P}_\mathcal{C}$ is the set of all feasible paths.

Attending to the definition of Eq. 2, the corresponding two paths, p_1 and p_2, can be derived through φ from the vertex assignment. For the sake of clarity, we simplify the notation by making the embedding only dependent on $\mathcal{G}$ and φ.

A possible embedding φ' of $\mathcal{G}_1$, into $\mathcal{C}_6$ is shown in Fig. 2a, where we can observe that each vertex of the input graph is mapped into a vertex of the host graph. For example, the vertex A of $\mathcal{G}_1$ is assigned to vertex 2 of $\mathcal{C}_6$, which is formally denoted as $\varphi(\text{A}) = 2$ (see Eq. 1). Similarly, B is assigned to 1 ($\varphi(\text{B}) = 1$), C is assigned to 4 ($\varphi(\text{C}) = 4$), and so on.

As it was aforementioned, ψ assigns two paths in $\mathcal{C}_6$ for each edge in $\mathcal{G}_1$. In Fig. 2b, we highlight the two possible paths between vertices $\varphi(\text{A}) = 2$ and $\varphi(\text{C}) = 4$

Table 1 Assignment of the vertices and edges of $\mathcal{G}_1$ into $\mathcal{C}_6$ vertices and paths respectively given the embedding of Fig. 2a

$(u, v) \in \mathcal{E}_\mathcal{G}$	$\varphi(u)$	$\varphi(v)$	$p_1(\varphi(u), \varphi(v))$	$p_2(\varphi(v), \varphi(u))$
(A, B)	2	1	{(2, 3), (3, 4), (4, 5), (5, 6), (6, 1)}	{(1, 2)}
(A, C)	2	4	{(2, 3), (3, 4)}	{(4, 5), (5, 6), (6, 1), (1, 2)}
(B, E)	1	5	{(1, 2), (2, 3)(3, 4), (4, 5)}	{(5, 6), (6, 1)}
(B, F)	1	3	{(1, 2), (2, 3)}	{(3, 4)(4, 5), (5, 6), (6, 2), }
(D, E)	6	5	{(6, 1), (1, 2), (2, 3)(3, 4), (4, 5)}	{(5, 6)}
(E, F)	5	3	{(5, 6), (6, 1), (1, 2), (2, 3)}	{(3, 4), (4, 5)}

in $\mathcal{C}_6$, denoted as $p_1(2, 4)$ and $p_2(4, 2)$. Specifically, $p_1(2, 4) = \{(2, 3), (3, 4)\}$, while $p_2(4, 2) = \{(4, 5), (5, 6), (6, 1), (1, 2)\}$.

Based on the input graph of Fig. 1a and the embedding of Fig. 2a, we report in Table 1, for each edge of the candidate graph (column $(u, v) \in \mathcal{E}_\mathcal{G}$), the assigned vertices in the host graph (columns $\varphi(v)$ and $\varphi(v)$), and the two paths assigned to the edge (columns $p_1(\varphi(u), \varphi(v))$ and $p_2(\varphi(v), \varphi(u))$. Based on a graphical representation and a clockwise move, we consider the path $p_1(\varphi(u), \varphi(v))$ as the one starting in $\varphi(u)$ and ending in $\varphi(v)$. Similarly, we consider the path $p_2(\varphi(v), \varphi(u))$ as the one starting in $\varphi(v)$ and ending in $\varphi(u)$.

1.1 Problem statement

In this paper, we tackle the Cyclic Antibandwidth (CAB) problem. Before formalizing this problem, we introduce a basic notation. Let $\mathcal{G}$ and $\mathcal{C}_n$ be the input and the host graph, respectively; and let φ be the function that defines an embedding, which is a solution to the CAB. We then define the Cyclic Antibandwidth of a graph as:

$$\text{CAB}(\mathcal{G}, \varphi) = \min_{(u,v)\in\mathcal{E}_\mathcal{G}} \{|p_1(\varphi(u), \varphi(v))|, |p_2(\varphi(v), \varphi(u))|\}, \tag{3}$$

where $|\cdot|$ determines the length of the path. To simplify the notation, we denote with $p((\varphi(u), (\varphi(v))$ as the shortest path between $p_1(\varphi(u), \varphi(v))$ and $p_2(\varphi(v), \varphi(u))$. Considering that the host graph is a cycle, it trivially holds for the CAB problem that $|p_2(\varphi(v), \varphi(u))| = n - |p_1(\varphi(u), \varphi(v))|$.

Finally, the aim of this optimization problem is to find an embedding $\varphi^\star$ (i.e., a solution) that maximizes Eq. 3:

$$\varphi^\star \leftarrow \text{CAB}(\mathcal{G}) = \arg\max_{\varphi\in\phi} \text{CAB}(\mathcal{G}, \varphi), \tag{4}$$

where ϕ represents the set of all possible feasible solutions of the problem.

Based on the input graph $\mathcal{G}_1$, depicted in Fig. 1a, and the embedding φ', depicted in Fig. 2a , we now illustrate how to evaluate the objective function of this problem with an example. In particular, the $\text{CAB}(\mathcal{G}_1, \varphi')$ can be calculated with the information reported in Table 1. First, we calculate the length of each of the paths assigned to the edges of the input graph. For instances, considering the edge (A, B),

the associated paths have lengths of 1 and 5 respectively. Similarly, the paths of the edge (B, F) have lengths of 2 and 4, and so on with the rest of the edges of $\mathcal{G}_1$. Then, for each edge, the path with the minimum length is selected. Finally, the value of the objective function is the minimum length across all paths, which is 1 in this example. In mathematical terms:

$$\begin{aligned} \mathrm{CAB}(\mathcal{G}_1, \varphi') = \min\{&p(\varphi(\mathrm{A}), \varphi(\mathrm{B})),\ p(\varphi(\mathrm{A}), \varphi(\mathrm{C})),\ p(\varphi(\mathrm{B}), \varphi(\mathrm{E})),\\ &p(\varphi(\mathrm{B}), \varphi(\mathrm{F})),\ p(\varphi(\mathrm{D}), \varphi(\mathrm{E})),\ p(\varphi(\mathrm{E}), \varphi(\mathrm{F}))\} =\\ = \min\{&1, 2, 2, 2, 1, 2\} = 1. \end{aligned} \tag{5}$$

1.2 Literature review

The Cyclic Antibandwith problem was proved to be $\mathcal{NP}$-Hard for general graphs in [32]. It was first introduced in [18] as a variant of the Bandwidth Minimization Problem (BMP). The BMP consists in finding a permutation of the rows and columns of a binary symmetric matrix that minimizes its bandwidth, where the bandwidth of the matrix is the largest distance from a nonzero element to the main diagonal [2]. This concept was later reinterpreted in terms of graph theory [22], where the binary matrix can be represented as a graph. In this context, the bandwidth of a graph is the maximum distance between any two adjacent vertices embedded in a path graph [15, 33].

Considering also a path host graph, it emerges the Antibandwidth Maximization Problem (AMP), which was proposed in [18] too. The AMP is also known as the Dual Bandwidth problem or the Separation Number problem [11, 24, 43]. It consists in maximizing the minimum distance between any two adjacent vertices embedded in the host graph. This problem is closely related to the CAB since both problems share the same objective function, being the only difference the host graph (a path for the AMP and a cycle for the CAB). The real applications of these two optimization problems can be found in multiprocessor scheduling [18], frequency assignment [12], graph drawing [30], VLSI design [4], batch-processing workloads [39], or network scheduling [3], among others.

The relation between the CAB and AMP was defined in terms of upper/lower bounds in [24, 42] as follows:

$$\frac{1}{2}\mathrm{AMP}(\mathcal{G}) \leq \mathrm{CAB}(\mathcal{G}) \leq \mathrm{AMP}(\mathcal{G}). \tag{6}$$

The CAB has been exactly solved for some regular-structure graphs such as paths [42], cycles [42], two dimensional meshes [32], toroidal meshes [32], and Hamming graphs [6]. Also, Raspaud et al. [32] presented asymptotic results for hypercube graphs. However, since the problem is $\mathcal{NP}$-Hard, there is not a method able to solve the problem for general graphs

From a heuristic perspective, this problem was first approached by Bansal and Srivastava in [3]. The authors proposed a Memetic Algorithm (MA) where the initial population was generated by using a greedy constructive procedure based on the

well-known Breadth First Search (BFS) algorithm [40]. The MA was tested over different sets of instances that could be grouped into: those where the optimum is known and those where the optimum is unknown. Based on the results obtained in [3], the authors also conjectured about the optimal solution of some graphs with regular structure, such as: three-dimensional meshes, hypercubes, and double stars. Later, Lozano et al. [20] tackled the problem with a hybrid metaheuristic that combined Artificial Bee Colony (ABC) with Tabu Search. They also used the BFS introduced in [3] as the constructive procedure to generate the initial population of the ABC method. Computational experienced showed that the algorithm proposed in [20] was favorably compared with the MA approach in [3] over the same set of instances, becoming the new state of the art of the CAB.

1.3 Our contributions

In this paper, we propose a Multistart General Variable Neighborhood Search (MS-GVNS) for the Cyclic Antibandwidth problem. The GVNS combines a Variable Neighborhood Descent (VND) with a shake procedure. Particularly, we propose a two-neighborhood VND, combined with a novel destruction–reconstruction shaking procedure. Also, we introduce a new constructive procedure based on a BFS strategy, to provide initial solutions to the GVNS. Additionally, we consider a criterion to guide the algorithm through the search space where solutions have the same objective function, and an efficient mechanism to evaluate neighboring solutions. To accelerate the scanning strategies of the neighborhoods, we also propose two methods focused on evaluating a very reduced number of solutions from the whole neighborhood.

The rest of the paper is organized as follows: in Sect. 2 we describe our algorithmic proposal. In Sect. 3, we present the advanced strategies for the local search. Then, in Sect. 4 we introduce the set of instances used in this paper, the preliminary experiments, and the competitive test performed in order to compare our algorithms with the two previous approaches. Finally, in Sect. 5 we expose our conclusions.

2 Algorithmic proposal

Dealing with hard optimization problems, especially those which fall into the $\mathcal{NP}$ -Hard class of problems (as it is the case of the CAB), usually requires the use of approximate procedures. These methods need to be able to find high-quality solutions in a reasonable amount of time. Among them, we can find the heuristic procedures, which are able to: produce feasible solutions, reach local optima starting from a particular solution, and even move from one local optimum to another one (through the use of a special kind of heuristics named metaheuristics). However, they cannot determine if any of the local optima reached is also the global optimum of the problem tackled.

Heuristic search procedures usually require some type of diversification to overcome local optimality [21]. One effective way to achieve diversification is to restart

the algorithm from new promising solutions once a region has been explored. Multi-start procedures were originally conceived as a way to go beyond classical local search methods by simply applying it to multiple random initial solutions. These procedures usually follow a generic scheme where the generation of new solutions and the corresponding improvement are alternated for a given number of iterations. In particular, our approach for the CAB problem starts by constructing solutions with a greedy procedure and then the improvement is performed with General Variable Neighborhood Search (GVNS). Algorithm 1 shows the pseudocode of the proposed multi-start procedure. It receives as input parameters, in step 1, a graph ($\mathcal{G}$), the maximum allowed time for the whole method (t_{max}), and the two parameters necessary for initializing each iteration of the `GVNS` (t'_{max} and k_{max}). Notice, that t'_{max} symbolizes the maximum allowed time for the `GVNS` at each iteration of the MS-GVNS and, therefore $t'_{max} < t_{max}$. At each iteration, a solution, φ, is constructed with the greedy method `GreedyConstructive` (step 4) described in Sect. 2.1, then it is improved with the method `GVNS` (step 5), described in Sect. 2.2. After that, it is decided whether the new solution obtained is better than the best solution found so far (step 6). Notice that, the method `CAB` returns the value of the objective function of an input solution for a given graph. If the new solution found is better than the previous one, it becomes the new best solution (step 7). Then, a new iteration starts (step 3). This process is repeated until a maximum time limit (t_{max}) is reached (with *time* indicating the current elapsed time since the procedure started running). Finally, the best solution found is returned (step 10).

Algorithm 1: MultiStart GVNS (MS-GVNS)

```
1  MS-GVNS (G, t_max, t'_max, k_max)
2  φ* ← ∅
3  while time < t_max do
4  |   φ ← GreedyConstructive(G)
5  |   φ' ← GVNS(G, φ, t'_max, k_max)
6  |   if CAB(G, φ') > CAB(G, φ*) then
7  |   |   φ* ← φ'
8  |   end
9  end
10 return φ*
```

2.1 Constructive procedure

We propose a constructive method inspired on previous ideas reported in [3, 11]. Our procedure can be divided in two main stages: (1) generate an initial partial solution by assigning several groups of vertices that can be assigned together from the input graph to the host graph; (2) assign the rest of the vertices one by one to the previous structure, finding the most suitable assignment for each of them. Next, we describe the first stage.

The input graph contains several groups of vertices suitable to be assigned together to consecutive vertices of the host graph, since this assignation does not affect in a negative way to the objective function. Particularly, those groups of

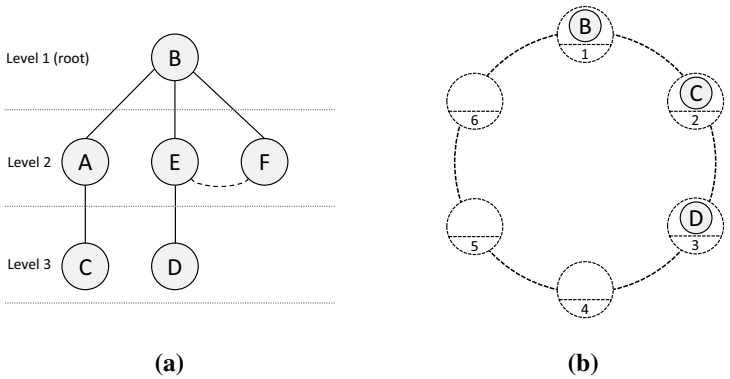


Fig. 3 Example of the construction of a solution (first stage)

vertices are the ones which satisfy the fact that they are not adjacent among them. To detect those groups, we first generate a level structure (spanning tree) of the input graph by using a Breadth First Search (BFS) algorithm [40]. The aim of this strategy is to separate the vertices into subsets (named levels) that have the same unit distance with respect to a randomly selected root vertex. Then, we choose a level at random from the spanning tree, and, from that level, we select the group of non-adjacent vertices. These vertices are then assigned to consecutive vertices of the host graph. Notice that the order used to assign the vertices within this group to the host graph is not relevant. Also, it is important to remark that some vertices from the selected level might remain unassigned, since they are adjacent to any of the selected vertices in the group.

Once the vertices of the first level selected have been assigned, we repeat this process on the nonconsecutive levels of the spanning tree one by one. The reason behind this strategy is to guarantee that the selected vertices are nonadjacent. Notice that this process is only applied to half of the levels (i.e., supposing that the levels are labeled starting from one to l, this is equivalent to scan all odd levels or all even levels).

Let us illustrate this first stage with an example. In Fig. 3a we depict a feasible spanning tree of the input graph shown in Fig. 1a. As we can observe, in level 1, there is only one vertex (B), which is the root of the spanning tree. In the second level, there are 3 vertices (A, E, and F), where the edge between vertices E and F is represented with a dotted line. Finally, vertices C and D are located in the third level. As it was aforementioned, we select an initial level at random (say, for instance, level 1). Then, all nonadjacent vertices in the selected level are assigned to the host graph ($\varphi(\mathrm{B}) = 1$). We repeat this process with the non-consecutive/odd levels. In this example level 3 is selected next, and their non-adjacent vertices are assigned next (i.e., $\varphi(\mathrm{C}) = 2$, and $\varphi(\mathrm{D}) = 3$). Notice that in the case that there exists an edge between two vertices in the same level, we only assign one of them, which would be selected at random.

The selection of a random root vertex for generating the spanning tree, and the selection of a random level to start the construction of the solution, favors the diversification when performing multiple constructions, as it is the case of a multistart procedure.

Next, we describe the second stage of our constructive procedure which assigns one by one the remaining unassigned vertices of the input graph. Particularly, nonassigned vertices are scanned at random and assigned to its best possible vertex of the host graph, which is determined with the help of a greedy function.

The purpose of this function is to determine the suitability of a vertex of the host graph as a candidate for an input vertex. Particularly, this function is based on the proximity to any adjacent vertex previously assigned to the solution (i.e., it quantifies the distance in the cycle of the input vertex to its nearest adjacent).

To define this greedy function, let $\mathcal{A}$ be the set of vertices of the input graph that have been already assigned to any of the vertices of the host graph, and $\mathcal{U}$ the set of unassigned vertices, such that $\mathcal{A} \cup \mathcal{U} = \mathcal{V}_{\mathcal{G}}$. Given a vertex of the input graph $u \in \mathcal{U} \subset \mathcal{V}_{\mathcal{G}}$ and a vertex of the host graph, $v \in \mathcal{V}_{\mathcal{C}}$, we define a function $g(u, v)$ that ponders the quality of a possible assignment $\varphi(u) = v$ as follows:

$$g(u, v) = \min_{w \in \mathcal{A}} \{|p(v, \varphi(w))|\}, \text{with } (u, w) \in \mathcal{E}_{\mathcal{G}}, \tag{7}$$

where $p(v, \varphi(w))$ is the path in the host graph from the candidate host vertex v to any other host vertex w with an adjacent to u assigned. Therefore, the best possible assignation $v^\star$ for a vertex u is:

$$v^\star = \arg\max_{v \in \mathcal{V}_{\mathcal{C}}} g(u, v). \tag{8}$$

Let us illustrate this second stage with an example. Taking into consideration the example depicted in Fig. 3b, $\mathcal{A} = \{\mathrm{B}, \mathrm{C}, \mathrm{D}\}$ and $\mathcal{U} = \{\mathrm{A}, \mathrm{E}, \mathrm{F}\}$. Then, we randomly select a vertex from $\mathcal{U}$ (say for instance vertex E). Since there are six host vertices, there are six possible assignments of E to the solution. For the sake of simplicity, in Fig. 4 we illustrate only two of them. In Fig. 4a we show the assignation $\varphi(\mathrm{E}) = 6$ while in Fig. 4b we shown the assignation $\varphi(\mathrm{E}) = 5$. In those figures we highlight the shortest path from the host vertex considered to any host vertex containing an adjacent to E. The greedy function values for the previous assignations are $g(\mathrm{E}, 6) = 1$ and $g(\mathrm{E}, 5) = 2$. Similarly, the value of the function g for the rest of the assignations not illustrated here are: $g(\mathrm{E}, 1) = 1$; $g(\mathrm{E}, 2) = 1$; $g(\mathrm{E}, 3) = 1$; $g(\mathrm{E}, 4) = 1$.

Notice that in the case that the candidate vertex is assigned to a host vertex which already contains other input vertex, the solution becomes unfeasible. In those cases we follow a straightforward criterion by performing the necessary chain of clockwise shifts of already assigned vertices, as we illustrate in Fig. 5. Particularly, in Fig. 5a, D is assigned to 3 which previously contained F. Then, in Fig. 5b we observe that F has been shifted and assigned to 4 and, similarly, A has been shifted and assigned to 5, and finally, E has been shifted and assigned to 6.

In Algorithm 2 we summarize the pseudocode of the greedy constructive proposed in this paper. Particularly, the steps 3 to 12 correspond to the first stage of the algorithm, while steps 13 to 17 correspond to the second stage. The algorithm

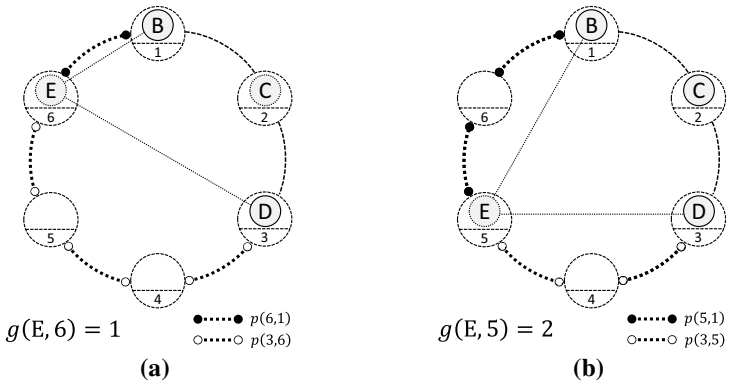


Fig. 4 **a** Assignment of the input vertex E to the host vertex 6 ($\varphi(\mathrm{E}) = 6$). **b** Assignment of the input vertex E to the host vertex 5 $\varphi(\mathrm{E}) = 5$

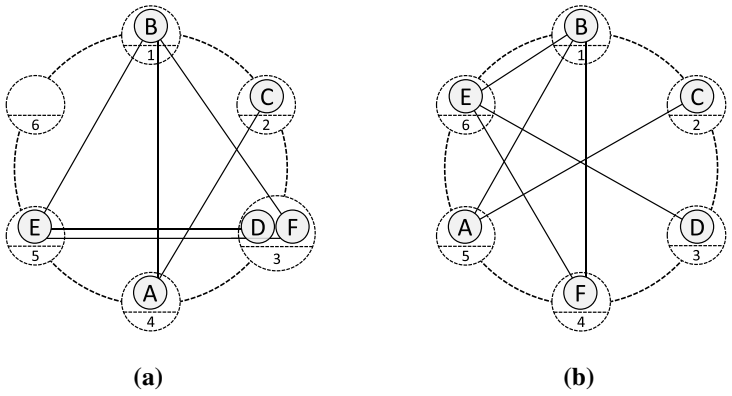


Fig. 5 **a** Infeasible assignment of the input vertex F to the host vertex 3 ($\varphi(\mathrm{F}) = 3$). **b** Reassignation of the vertices F, A and E after the assignation $\varphi(\mathrm{F}) = 3$

receives a candidate graph ($\mathcal{G}(\mathcal{V}_\mathcal{G}, \mathcal{E}_\mathcal{G})$) as an input, where $\mathcal{V}_\mathcal{G}$ and $\mathcal{E}_\mathcal{G}$ represent the sets of vertices and edges, respectively. First, a solution φ and the set of unassigned vertices ($\mathcal{U}$) are initialized (step 3). Then, we select a vertex at random from the graph (step 3) that is used as a root for constructing the spanning tree (step 4). Later, we select a random level from the tree (step 5) and we extract the list of non-consecutive levels starting from that one (step 6). In steps 7 to 12 we scan each of the levels selected one by one. For each level we first extract all the vertices at the level (step 8) and then we select the group of non adjacent vertices (step 9). The selected vertices are then removed from the set of unassigned vertices (step 10) and assigned at random to the next available candidate vertices (step 11). Once the first stage is finished, we start with the second stage, where the rest of unassigned vertices

(remaining in $\mathcal{U}$) are assigned one by one. Particularly, for each unassigned vertex u (step 14) we look for the best possible host vertex in the host graph (using the greedy function g previously introduced) and perform the assignation of u to that host vertex (step 16). Notice that this method performs a reassignation of the host vertices if the best position selected is already taken. Finally, when the solution is completed (i.e., all candidate vertices have been assigned to a different host vertex) the method returns the constructed solution (step 18).

Algorithm 2: Greedy constructive

```
1  ConstructSolution (G(V_G, E_G))
2  φ ← ∅, U ← V_G
3  root ← RandomVertex(V_G)
4  tree ← GenerateSpanningTree(root, V_G)
5  start ← RandomIndex(tree)
6  levels ← GetNonConsecutiveLevels(tree, start)
7  for i ← 0 to |levels| do
8  |   vertices ← GetVertices(levels, i)
9  |   nonAdjacent ← GetNonAdjacentVertices(vertices)
10 |   U ← U \ nonAdjacent
11 |   AssignNextCandidateVertex(φ, nonAdjacent)
12 end
13 while |U| ≠ ∅ do
14 |   u ← GetVertex(U)
15 |   U ← U \ u
16 |   AssignBestCandidateVertex(φ, u)
17 end
18 return φ
```

2.2 General variable neighborhood search

Variable Neighborhood Search (VNS) is a metaheuristic proposed by Hansen and Mladenovic in [26] which is based on the concept of changes in the neighborhood structure during the search, to prevent heuristic procedures from getting stuck in local optima [9]. It is possible to find many different variants of VNS in the literature, which propose different ways of exploring the neighborhood structures (deterministic or stochastic). Moreover, some variants of VNS have already been used to successfully tackle other problems within the Graph Layout family of problems [8, 28]

The classical best-known variant, Basic Variable Neighborhood Search (BVNS), includes three main procedures: shake, improve, and neighborhood change. The shake procedure performs stochastic moves in the current neighborhood being explored, with the aim of escaping from the basin of attraction; the improve method is a deterministic procedure based on the use of a local search, able to find the local optimum for a particular neighborhood structure. Finally, the neighborhood search procedure determines if there is an improvement in the solution found in the current iteration of the method, with respect to the best overall solution found.

In this paper, we propose the use of an extension of the previous procedure, named General Variable Neighborhood Search (GVNS) [14, 25]. This VNS variant

differs from BVNS in the improvement procedure step. Particularly, instead of using a simple local search, which explores a specific neighborhood, it uses a Variable Neighborhood Descent (VND) [13] to explore a group of neighborhoods. The GVNS is used as the improvement strategy for our multistart approach for the CAB. In Algorithm 3, we introduce the pseudocode of the GVNS. This procedure receives the input graph $\mathcal{G}$, an initial solution (φ), the maximum allowed time (t'_{max}), and the largest neighborhood to be explored (k_{max}). Later, in Sects. 2.2.1 and 2.2.2 we describe in detail the shake and VND procedures, respectively.

Algorithm 3: General VNS

```
1  GVNS (G, φ, t'_max, k_max)
2  k ← 1
3  do
4  |  do
5  |  |  φ' ← Shake(φ, k)
6  |  |  φ'' ← VND(G, φ')
7  |  |  k ← NeighborhoodChange(φ', φ'', k)
8  |  while k ≤ k_max
9  while time < t'_max
10 return φ
```

2.2.1 Shake procedure

The shake procedure is used to escape from local optima solutions by performing a perturbation able to cause a change in the current neighborhood. This technique allows to diversify the search and explore new regions of the solution space. To tackle the CAB, we proposed a random destruction–reconstruction shake procedure. Particularly, this strategy first removes a set of input vertices (selected at random) from its currently assigned host vertex. Then, it performs also at random, new assignations of the input vertices removed within the available host vertices.

The number of vertices to be removed depends on a search parameter (k) which is received as an input to the method. The value of k ranges from 1 to k_{max} (received as an input to the GVNS) and, in each iteration, it is determined by the neighborhood change procedure within the GVNS.

In addition to the aforementioned shake procedure, we also tested a variant of the method where the reconstruction phase was based on a greedy function. Particularly, we used the greedy criterion previously introduced in the constructive procedure, to assign the vertices in the reconstruction phase. However, the performance of this shake variant was slightly worse to the one previously introduced so it was discarded.

2.2.2 Variable neighborhood descent

The VND is used to explore a group of neighborhoods deterministically, in such a way that the obtained result is a local optimum with respect to all neighborhoods explored [9, 13].

In this paper, we propose two neighborhoods for the CAB and a different local search procedure to explore each of them. In Algorithm 4 we introduce the pseudocode of the VND method proposed. As we can observe, the method receives the input graph $\mathcal{G}$ and a feasible solution φ. Then, it explores the first neighborhood ($\mathcal{N}_1$) in the steps 6 to 15, following a best improvement strategy. This exploration continues while it is able to produce an improvement in the current solution. Then, it switches to the second neighborhood ($\mathcal{N}_2$) and explores it following a first improvement strategy, in the steps 16 to 31. If the method finds a better solution in this second neighborhood, the VND switches again to the first neighborhood (steps 33 to 38). Otherwise, when both neighborhoods have been explored without improvement, the procedure ends and returns the best overall solution found $\varphi^\star$.

Algorithm 4: VND procedure

```
1  VND (G, φ)
2  φ* ← φ
3  neighborhood ← 1
4  do
5      switch neighborhood do
6          case 1 :
7              do
8                  improved ← False
9                  φ' ← arg max_{φ''∈N1(φ)} CAB(G, φ'')
10                 if CAB(G, φ') > CAB(G, φ) then
11                     φ ← φ'
12                     improved ← True
13                 end
14             while improved
15         end
16         case 2 :
17             do
18                 improved ← False
19                 i ← 1
20                 do
21                     φ' ← φ_i ∈ N2(φ)
22                     if CAB(G, φ') > CAB(G, φ) then
23                         φ ← φ'
24                         improved ← True
25                         break
26                     else
27                         i ← i + 1
28                     end
29                 while i ≠ |N2(φ)|
30             while improved
31         end
32     end
33     if CAB(G, φ) > CAB(G, φ*) then
34         φ* ← φ
35         neighborhood ← 1
36     else
37         neighborhood ← neighborhood + 1
38     end
39 while neighborhood ≤ 2
40 return φ*
```

The first neighborhood ($\mathcal{N}_1$ in Algorithm 4) is based on the set of solutions that can be reached by using the classical Insert move operator. Particularly, this operator selects an input vertex and removes its current assignation to the host graph. Then, it tries to assign the removed vertex to any host vertex in the host graph, and it performs the best possible overall insertion, considering the objective function. Notice that when an Insert operation is performed, every vertex of the input graph is already assigned to another vertex of the host graph. Therefore, it is necessary that the method partially reassigns some of the input vertices to other host vertices, to make room for the new assignation. In this case, all affected input vertices are reassigned in a chain from its current host vertex to another host vertex adjacent to the

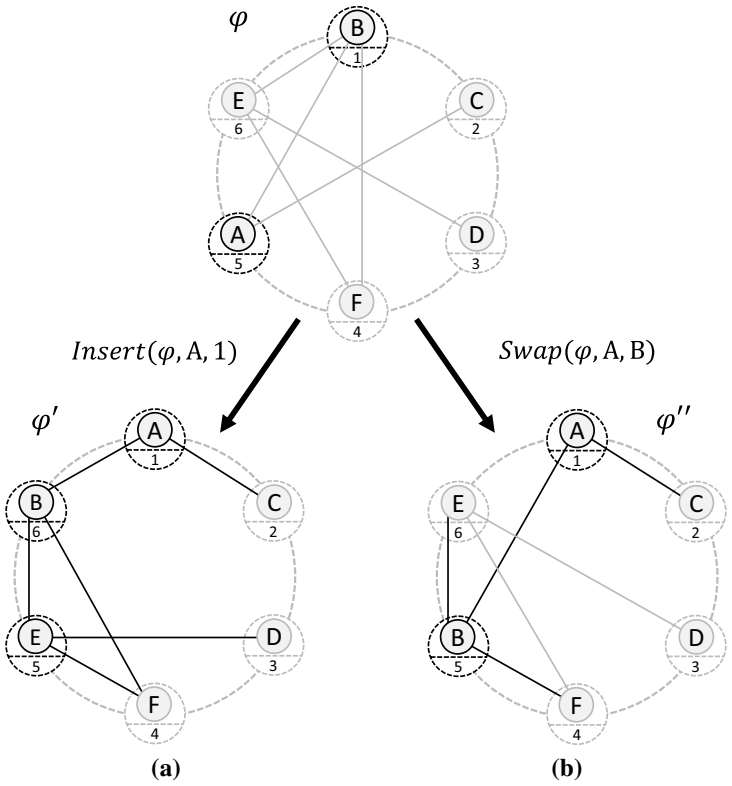


Fig. 6 Example of a solution of the swap neighborhood, φ', and insertion neighborhood, φ'' of a solution φ

current one, following a clockwise or counterclockwise criterion, which depends on the number of reassignments needed (i.e., the fewer, the better, with ties broken at random).

In Fig. 6 we illustrate with an example the operation $Insert(\varphi, \mathrm{A}, 1)$. The Insert operation starts from the solution φ (depicted at the top of the figure). As we can observe, in this case, the operation removes the input vertex A (assigned to the host vertex 5 in φ) and assigns it to the host vertex 1. Therefore, vertices B and E are reassigned to host vertices 6 and 5 obtaining the solution φ', depicted in Fig. 6a.

Finally, we formally define the neighborhood $\mathcal{N}_1$ for a solution φ as $\mathcal{N}_1(\varphi) = \{Insert(\varphi, u, w),\ \forall\, u \in \mathcal{V}_G, w \in \mathcal{V}_C : \varphi(u) \neq w\}$. The size of this neighborhood is $n(n-1)$ being n the number of vertices of the candidate graph.

The second neighborhood ($\mathcal{N}_2$ in Algorithm 4) is based on another classical operator, usually known as Swap. It consist in selecting two input vertices and interchanging the assigned host vertex to each of them.

In Fig. 6 we illustrate with an example the operation $Swap(\varphi, \mathrm{A}, \mathrm{B})$. The Swap operation starts from the solution φ (depicted at the top of the figure). As we can

observe, in this case, the operation removes the input vertex A (assigned to the host vertex 5 in φ) and assigns it to the host vertex 1. Similarly, it removes the input vertex B (assigned to the host vertex 1 in φ) and assigns it to the host vertex 5. The obtained solution (φ'') after the Swap is depicted in Fig. 6b.

Formally, we define the swap neighborhood $\mathcal{N}_2$ for a solution φ as $\mathcal{N}_2(\varphi) = \{Swap(\varphi, u, v)\ \forall\ u, v \in \mathcal{V}_G : u \neq v\}$. The size of this neighborhood is $n(n-1)/2$ being n the number of vertices of the candidate graph.

3 Advanced search strategies for exploring the neighborhoods

A neighborhood structure, defined over a particular solution, determines the set of solutions which can be reached by performing a predefined move. The size of this set and the way in which the exploration of the solutions contained in it is performed, are key parameters in determining the efficiency of a search procedure.

In Sect. 2.2.2 we defined two neighborhood structures ($\mathcal{N}_1$ and $\mathcal{N}_2$) for the CAB. Now, in Sect. 3.1 we propose, two advanced search strategies to efficiently explore those neighborhoods. Then, in Sect. 3.2 we propose two additional strategies to reduce the size of sets $\mathcal{N}_1$ and $\mathcal{N}_2$.

3.1 Neighborhood exploration strategies

The solution space of some optimization problems contains a large number of neighboring solutions with the same value of the associated objective function. This fact makes the task of exploring the neighborhood very hard, since a local search procedure usually reaches locally optimal solutions very quickly, being unable to determine which solution is better among those with the same objective function value. To avoid this drawback, in Sect. 3.1.1, we propose an advanced strategy to distinguish the more promising solutions in the neighborhood, further than the original value of the objective function. Additionally, when a local search method needs to evaluate the quality of neighboring solutions, it can take advantage of the information used to evaluate the the current solution. In this sense, neighboring solutions usually share many common characteristics, which makes possible to efficiently evaluate the quality of a neighboring solution after a move. In Sect. 3.1.2, we propose a strategy to handle this issue.

3.1.1 Advanced exploration strategy 1: tiebreak criterion for the objective function

Flat landscape is a term used in the literature to refer to solution spaces, where many solutions have the same value of the objective function associated, despite the fact that its structure might be different [28, 31]. The Cyclic Antibandwidth problem, as other problems belonging to the min/max or max/min categories,[1]

[1] Minimization of a maximum value or maximization of a minimum value.

presents a flat landscape [38, 41]. Flat landscapes are hard scenarios for search procedures, since the value of the objective function is frequently used to determine the search direction.

To avoid this drawback, we propose an advanced exploration strategy (AE_1) consisting in defining a tiebreak criterion that makes it possible to distinguish among solutions with the same value of the original objective function. Therefore, the rationale behind this strategy is to identify potentially promising solutions for search procedures. Particularly, when a tie is found in the original objective function, we use the tiebreak criterion to decide which solution is more suitable to continue the search.

This tiebreak criterion is inspired by the ideas introduced in [20]. Particularly, the authors of that paper proposed a very simple alternative objective function. It consists in quantifying the number of edges which have an assigned path with length equal to the value of the objective function, for that particular solution. Then, when two solutions are tied in terms of value of the original objective function, the solution with the smaller value of this alternative objective function is chosen.

To propose our tiebreak criterion, we define $\mathrm{CAB}(\varphi, (u, v))$ as the bandwidth of an edge $(u, v) \in \mathcal{E}_{\mathcal{G}}$ in a solution φ such that:

$$\mathrm{CAB}(\varphi, (u, v)) = |p(\varphi(u), \varphi(v))|. \tag{9}$$

where the path p, as was mentioned in Sect. 1.1, is the shortest path between p_1 and p_2. Then, we classify the edges in $\mathcal{E}_{\mathcal{G}}$ based on its bandwidth in a solution, in such a way that all the edges with the same bandwidth are grouped together. More formally, given a solution φ, let i ($1 \leq i \leq \lceil |\mathcal{V}_{\mathcal{G}}|/2 \rceil$) be the bandwidth of the edges in a particular set, denoted as S^i_{φ}. In mathematical terms:

$$S^i_{\varphi} = \{(u, v) \in \mathcal{E}_{\mathcal{G}} \ : \ \mathrm{CAB}(\varphi, (u, v)) = i\}. \tag{10}$$

With the previous definitions at hand, and given two solutions φ and φ', our tiebreak criterion consists in scanning all previous sets one by one. Particularly, we traverse the sets in ascending order of i, starting with i equal to the value of the original objective function. For a particular i, we compare the cardinality of S^i_{φ} and $S^i_{\varphi'}$, in such a way that, if the cardinality of both sets is equal, we increment i. Otherwise, the solution with the smallest cardinality is selected as the most promising one, avoiding the exploration of further sets.

Let us illustrate the evaluation of this tiebreak criterion with an example. Considering the solution φ' of Fig. 6a, the set $S^1_{\varphi'} = \{(\mathrm{A}, \mathrm{B}), (\mathrm{A}, \mathrm{C}), (\mathrm{B}, \mathrm{E}), (\mathrm{E}, \mathrm{F})\}$ is formed by the edges with bandwidth equal to 1. The set $S^2_{\varphi'} = \{(\mathrm{B}, \mathrm{F}), (\mathrm{D}, \mathrm{E})\}$ is formed by the edges with bandwidth equal to 2. Similarly, the edges in the solution φ'' in Fig. 6b are classified in: $S^1_{\varphi''} = \{(\mathrm{A}, \mathrm{C}), (\mathrm{B}, \mathrm{E}), (\mathrm{B}, \mathrm{F})\}$, $S^2_{\varphi''} = \{(\mathrm{A}, \mathrm{B}), (\mathrm{E}, \mathrm{F})\}$, and $S^3_{\varphi''} = \{(\mathrm{D}, \mathrm{E})\}$. Therefore, when comparing φ' and φ'' we observe that both solutions have the same objective function value, which is equal to 1. However, the tiebreak criterion indicates that φ'' is better than φ', since $|S^1_{\varphi''}| = 3$ is smaller than $|S^1_{\varphi'}| = 4$.

3.1.2 Advanced exploration strategy 2: efficient move calculation

A very time-consuming activity in the exploration of a neighborhood is related to the evaluation of the neighboring solutions to determine the most suitable move. To partially overcome this situation, we propose an advanced exploration strategy (AE_2) which efficiently calculates the quality of neighboring solutions by considering the current one.

Particularly, given a solution φ and a neighboring solution $\varphi' \in \mathcal{N}_x(\varphi)$, being x either 1 or 2 (see Sect. 2.2.2), we can calculate the objective function value and, if necessary, the tiebreak criterion of φ', by only observing the changes in the paths associated to certain edges of the input graph after the move.

Notice that a straightforward implementation of the evaluation of the objective function of φ', would need to analyze the whole set of input edges one by one. Then, for each edge, the procedure determines if the path associated with each edge derived from φ' is the shortest path among the scanned ones, to calculate the objective function value. Additionally, the evaluation of the tiebreak criterion would imply to classify the incumbent edge in the corresponding $S^i_{\varphi'}$. However, depending on the similarity between φ and φ', it is possible that most of the paths remain unaltered after the move and, therefore, its length does not change. Thus, these edges do not need to be analyzed, since they do not produce a change either in the objective function or in the tiebreak criterion.

This strategy consists in identifying the paths that require to be updated after the move, which obviously depends on the particular move. In this paper, we consider two possible moves: insertions and swaps. In the insertion case, as we introduced in Sect. 2.2.2, it might be necessary to reassign several vertices after a move, further than the inserted vertex. Then, it is necessary to analyze all paths associated with any edge with an endpoint in a reassigned vertex. On the other hand, in the swap case, the only paths affected after a move are those associated with the edges with an endpoint in one of the vertices involved in the move.

To illustrate the performance of this strategy with an example, we consider the solution φ depicted and the vertex A in Fig. 6. The straightforward implementation of the insert move implies to update the bandwidth of the 6 edges of the graph (i.e, $(A, B), (A, C), (B, E), (B, F), (D, E), (E, F)$) for a particular insertion. Then, exploring all the possible insertions for A (i.e., there are 5 host vertices) requires to perform $6 \times 5 = 30$ updates. On the other hand, we have quantified the number of updates needed when using AE_2. Particularly, in Table 2 we report the updates needed for each insert move of A, reducing the total number of updates from 30 to 25 (Table 3).

Similarly, in the case of the swap move, the straightforward implementation again requires 30 updates as in the case of the insertion move. However, the updates using AE_2 together with the swap operator are reduced from 30 to 18. Particularly, in Table 2, we report the updates needed for each possible swap move of A.

As we can observe, this strategy has a larger impact in the case of the exploration of the neighborhood defined by the swap operator than the neighborhood defined by the insert operator. This fact is later confirmed in the preliminary

Table 2 Example of the edges that need to be updated when using AE_2 together with the insert operator

	Evaluated edges
Insert(φ, A, 1)	(A, B), (A, C), (B, E), (B, F), (D, E), (E, F)
Insert(φ, A, 2)	(A, B), (A, C), (B, F), (D, E), (E, F)
Insert(φ, A, 3)	(A, B), (A, C), (B, F), (D, E), (E, F)
Insert(φ, A, 4)	(A, B), (A, C), (B, F), (E, F)
Insert(φ, A, 6)	(A, B), (A, C), (B, E), (D, E), (E, F)

Table 3 Example of the edges that need to be updated when using AE_2 together with the swap operator

	Evaluated edges
Swap(φ, A, B)	(A, B), (A, C), (B, E), (B, F)
Swap(φ, A, C)	(A, B), (A, C)
Swap(φ, A, D)	(A, B), (A, C), (D, E)
Swap(φ, A, E)	(A, B), (A, C), (B, E), (E, D), (E, F)
Swap(φ, A, F)	(A, B), (A, C), (B, F), (E, F)

experiments (see Sect. 4.2). Additionally, it is worth mentioning that the influence of AE_2 increases with large and dense graphs.

3.2 Neighborhood reduction strategies

Many optimization problems are computationally intractable due the large size of its solution space. To explore promising parts of that space, researchers define neighborhood structures which contain sets of solutions associated. As stated in [1], the larger the neighborhood, the longer it takes to perform a search which traverses all associated solutions.

In this sense, to tackle the CAB, we have introduced two neighborhood structures $\mathcal{N}_1$ (with size $n(n-1)$) and $\mathcal{N}_2$ (with size $n(n-1)/2$) being n the number of vertices of the input graph. In this section, we propose two strategies to reduce those neighborhoods, by exploring only the most promising solutions, avoiding to traverse the whole neighborhoods.

3.2.1 Neighborhood reduction 1: candidate vertices

The first neighborhood reduction strategy (R_1) narrows the number of either insert or swap operations performed. Specifically, given a solution φ, the whole exploration of $\mathcal{N}_1(\varphi)$ implies that all the vertices $v \in \mathcal{V}_\mathcal{G}$ are candidate to be inserted in any host vertex $w \in \mathcal{V}_\mathcal{C}$. The first neighborhood reduction strategy applied to the insert operator ($\mathcal{N}_1^{R_1}$) narrows the number of candidate vertices in $\mathcal{V}_\mathcal{G}$ to be inserted. In particular, we only consider those vertices which are endpoints of the edges in the set $S_\varphi^{i_{min}}$, with $i_{min} = \text{CAB}(\mathcal{G}, \varphi)$.

Similarly, the exhaustive exploration of $\mathcal{N}_2(\varphi)$ requires that all the vertices $v \in \mathcal{V}_\mathcal{G}$ are candidate to swap its associated host vertex with the host vertex associated to any other $w \in \mathcal{V}_\mathcal{G}$. Again, this strategy applied to the swap operator ($\mathcal{N}_2^{\mathrm{R}_1}$) narrows the number of candidate vertices in $\mathcal{V}_\mathcal{G}$ to be considered. Specifically, we only deal with those vertices which are endpoints of the edges in the set $S_\varphi^{i_{min}}$, with $i_{min} = \mathrm{CAB}(\mathcal{G}, \varphi)$.

In mathematical terms, we define $\mathcal{N}_1^{\mathrm{R}_1}$ and $\mathcal{N}_2^{\mathrm{R}_1}$ as follows:

$$\mathcal{N}_1^{\mathrm{R}_1} = \{Insert(\varphi, v, w), \text{with } v \in \mathcal{V}_\mathcal{G} \wedge (v, u) \in S_\varphi^{i_{min}},\ \forall u \in \mathcal{V}_\mathcal{G}, w \in \mathcal{V}_\mathcal{C}\} \quad (11)$$

$$\mathcal{N}_2^{\mathrm{R}_1} = \{Swap(\varphi, v, w), \text{with} v \in \mathcal{V}_\mathcal{G} \wedge (v, u) \in S_\varphi^{i_{min}},\ \forall u \in \mathcal{V}_\mathcal{G}, w \in \mathcal{V}_\mathcal{G}\} \quad (12)$$

Let us illustrate the evaluation of this strategy with an example. Considering the solution φ in Fig. 6, the edges classified in: $S_\varphi^{i_{min}} = \{(\mathrm{B}, \mathrm{E})\}$, i.e., edges with bandwidth equal to 1. Therefore, in order to construct $\mathcal{N}_1^{\mathrm{R}_1}$ (analogously $\mathcal{N}_2^{\mathrm{R}_1}$) we consider only the insertion (analogously the swap) of candidate vertices B and E over all host vertices, which reduces considerably the number of operations in comparison with using the whole set of vertices of the graph.

3.2.2 Neighborhood reduction 2: candidate assignments

Given a solution φ, the whole exploration of $\mathcal{N}_1(\varphi)$ or $\mathcal{N}_2(\varphi)$ implies that all the host vertices $v \in \mathcal{V}_\mathcal{C}$ are candidate to be considered in any move. The second neighborhood reduction strategy (R_2) narrows the number of candidate host vertices considered for the assignment of the vertices moved either in the insert or swap operators. Therefore, we denote as $\mathcal{N}_1{}^{\mathrm{R}_2}$ (analogously $\mathcal{N}_2^{\mathrm{R}_2}$) to the neighborhood structure, obtained after applying the insert (analogously swap) operator together with R_2. This strategy looks for the most suitable host vertices to assign a particular candidate one. To determine those vertices, we use the greedy criterion (g) and the definition of the best possible assignation for a candidate vertex, introduced in Eqs. 7 and 8, described in Sect. 2.1.

Particularly, the reduced set of host candidate vertices for the insert operator is denoted as $\mathcal{V}_{ins}^{R_2}$, while the reduced set of input vertices which are currently assigned to the best host candidate vertices for the swap operator is denoted as $\mathcal{V}_{swap}^{R_2}$. Notice, that the Swap operator is defined in such a way that it receives two candidate vertices belonging to $\mathcal{V}_\mathcal{G}$, while the Insert operator is defined in such a way that it receives a candidate vertex belonging to $\mathcal{V}_\mathcal{G}$ and a host vertex belonging to $\mathcal{V}_\mathcal{C}$.

Therefore, we formally define $\mathcal{N}_1^{\mathrm{R}_2}$ and $\mathcal{N}_2^{\mathrm{R}_2}$ as follows:

$$\mathcal{N}_1^{\mathrm{R}_2} = \{Insert(\varphi, v, w), \text{with } v \in \mathcal{V}_\mathcal{G}, w \in \mathcal{V}_{ins}^{R_2}\} \quad (13)$$

$$\mathcal{N}_2^{\mathrm{R}_2} = \{Swap(\varphi, v, w), \text{with } v \in \mathcal{V}_\mathcal{G}, w \in \mathcal{V}_{swap}^{R_2}\} \quad (14)$$

Let us illustrate the evaluation of this strategy with an example. Considering the solution φ' in Fig. 6a and the vertex A, the set of host vertices chosen to construct $\mathcal{N}_1^{\mathrm{R}_2}$ using the insert operator, is conformed by the host vertex $4 \in \mathcal{V}_\mathcal{C}$. This host vertex is obtained after maximizing the value of g when A is assigned to each host

vertex. Particularly, this evaluation results in: $g(A, 1) = 1$, $g(A, 2) = 1$, $g(A, 3) = 1$, $g(A, 4) = 2$, $g(A, 5) = 1$ and $g(A, 6) = 1$; being the assignation of A to the host vertex 4, the one which maximizes the value of g. Therefore, the set of input vertices to construct $\mathcal{N}_2^{R_2}$ using the swap operator, is conformed by the input vertex F, since $\varphi(F) = 4$.

4 Computational results

In this section, we report the computational experiments carried out to determine the effectiveness of the proposed algorithms. First, in Sect. 4.1, we present the set of instances used to evaluate and test our proposal. Then, in Sect. 4.2, we describe a set of preliminary experiments performed to illustrate the performance of our more significant strategies in isolation, and also to tune the parameters of our algorithms. Finally, in Sect. 4.3, we compare our best variant of the multistart GVNS procedure with previous state-of-the-art methods.

The proposed algorithms have been coded in Java 13, and all experiments have been run on an Intel Core i7-4702MQ CPU 2.20Ghz with 16 GB RAM.

4.1 Instances

The instances used to test our multistart GVNS were previously proposed in other papers of the state of the art which tackle the CAB [3, 20]. Particularly, we have divided all available instances into two groups:

- Instances with known optimum. This set consists of 132 regular-structured graphs, and it includes: paths (24 instances), cycles (24 instances), grids (23 instances), toroidal grids (37 instances), and Hamming graphs (24 instances).
- Instances with unknown optimum. This set consists of 163 instances divided into regular-structured graphs and random graphs. Among the regular graphs we can find: three-dimensional meshes (20 instances), double stars (20 instances), hypercubes (7 instances), caterpillars (40 instances), complete binary trees (24 instances). On the other hand, the random graphs are extracted from two different origins: harwell-boeing instances [10] (24 instances) and random connected graphs [20] (28 instances).

We have made available all previous instances at https://grafo.etsii.urjc.es/optsicom/cab/.

Additionally, we selected a reduced set of 28 instances (in the following the "preliminary data set") chosen proportionally from each of the previously described sets, which are used for our preliminary experiments. The small size of this data set is devoted to avoiding overfitting of the methods. Particularly, this preliminary data set consists of 2 paths, 2 cycles, 2 grids, 4 toroidal grids, 2 hamming, 2 three dimensional meshes, 2 double stars 2 hypercubes, 4 caterpillars, 3 complete binary tree and three Harwell-Boeing.

Table 4 Comparison of the proposed constructive procedure (Greedy) with the previous state-of-the-art constructive (GLAH) [3, 20] over the preliminary data set

	Greedy				GLAH [3, 20]			
Iterations	1	10	100	1000	1	10	100	1000
Avg.	122.25	138.46	143.93	145.07	124.32	126.43	128.50	129.14
CPUt(s)	0.001	0.005	0.043	0.393	0.007	0.067	0.579	7.038
Dev.(%) (exp.)	23.16	8.72	1.30	0.00	26.02	21.17	17.72	16.61
#Best (exp.)	4	8	19	28	3	5	6	7
Dev.(%) (all)	39.04	26.30	21.94	21.15	36.90	33.60	31.72	31.07
#Best (all)	0	0	3	3	0	2	2	2

4.2 Preliminary experiments

This section is devoted to illustrate the performance of the different strategies proposed through the paper in isolation. First, we test the suitability of our constructive procedure for our multistart design. Also, we compare it with the best previous constructive procedure in the state of the art. Second, we study the influence of the advanced strategies proposed to explore the neighborhood structures, on the performance of the search process. In our third experiment, we illustrate the save of time achieved when using the neighborhood reduction strategies proposed. In our final preliminary experiment, we show how to increase the quality of the results by combining several neighborhoods within a VND algorithm. At the same time, in this experiment we observe the evolution in the quality obtained when increasing the abstraction level of the methods used (constructive, local search, VND, GVNS). Finally, we present the values of the parameters used for the configuration of GVNS, obtained after a fine tuning process.

In our first preliminary experiment, we compare our greedy constructive procedure (Greedy), with the constructive proposed in previous papers in the state of the art (GLAH in [3, 20]). Since we are proposing a multistart algorithm, in this experiment we test the evolution of the constructive procedure in both time and quality, when performing 1, 10, 100, and 1000 constructions. Particularly, we selected the best solution found by the compared methods for each instance, after the predefined number of iterations and, in Table 4, we reported the average quality of the objective function (Avg.), the average CPU execution time measured in seconds (CPUt(s)), the average deviation with respect to the best solution found in the experiment (Dev.(%) (exp.)), the average deviation with respect to the best solution found in the state of the art (Dev.(%) (all)), the number of best solutions obtained in the experiment (#Best (exp.)) and the number of best solutions obtained over all (#Best (all)).

As we observe in Table 4, both constructive methods proposed are able to evolve when increasing the number of solutions constructed from 1 to 1000. However, the improvement in the averaged quality grows in a smaller proportion when more constructions are performed. As expected, the running time increases proportionally with respect to the number of constructions.

Table 5 Influence of the advanced strategies (S_1 and S_2) in the two local search procedures proposed (LS_1 and LS_2)

	$\mathcal{N}_1$ (Insert)			$\mathcal{N}_2$ (Swap)		
	LS_1	LS_1+AE_1	LS_1+AE_1+AE_2	LS_2	LS_2+AE_1	LS_2+AE_1+AE_2
Avg.	1.59	30.29	30.29	1.71	34.71	34.71
CPUt (s)	0.74	256.67	105.07	0.51	15.61	0.60
Dev.(%) (exp.)	91.27	0.00	0.00	91.05	0.00	0.00
#Best (exp.)	0	17	17	0	17	17
Dev.(%) (all)	92.94	23.68	23.68	92.31	16.84	16.84
#Best (all)	0	3	3	0	4	4

When comparing the proposed procedure (Greedy) with the state-of-the-art one (GLAH), we observe that our constructive method is able to systematically obtain better solutions in shorter times, when performing the same number of constructions. This fact is especially pronounced in the case of 1000 constructions. Therefore, our method will be selected for the final configuration of our MS-GVNS.

The aim of the second preliminary experiment is to study the influence of the advanced strategies proposed in Sects. 3.1.1 and 3.1.2 in the exploration of the neighborhood structures introduced in Sect. 2.2.2. Particularly, the first neighborhood considered in this paper, denoted as $\mathcal{N}_1$, is explored with a local search which traverses the neighborhood defined by the Insert operator, following a best improvement strategy. This local search is defined in steps 6 to 15 in Algorithm 4 and we denote it as LS_1 for the preliminary experiments. On the other hand, the second neighborhood considered in this paper, denoted as $\mathcal{N}_2$, is explored with a local search which traverses the neighborhood defined by the Swap operator, following a first improvement strategy. This local search is defined in steps 16 to 31 in Algorithm 4 and we denote it as LS_2 for the preliminary experiments.

In Table 5, we report the results obtained with both local search procedures in isolation (LS_1 and LS_2). Also, we report the results obtained with the combination of LS_1 and LS_2 with the first advanced strategy, introduced in Sect. 3.1.1, and denoted in Table 5 as AE_1. Let us remember that AE_1 consists in using a criterion to distinguish between solution with same objective function. Therefore, the combination of each local search with AE_1 is denoted in Table 5 as LS_1+AE_1, and LS_2 + AE_1, respectively. Finally, we report the results obtained when combining each local search procedure with AE_1 and also with the second advanced strategy, introduced in Sect. 3.1.2, which is denoted in Table 5 as AE_2. Let us remember that this strategy consists of the efficient calculation of the objective function for neighboring solutions. Therefore, the combination of each local search with AE_1 and AE_2 is denoted in Table 5 as LS_1+AE_1+AE_2 and LS_2+AE_1+AE_2, respectively.

The three variants tested for each local search start the search from the same initial solution. Only one iteration of each variant has been performed with a maximum allowed time for each instance of 3600 s. Notice, that some algorithms were not able to finish the search for all instances in the preliminary data set, in

the time limit established, without using the efficient evaluation proposed (AE_2). Therefore, we have removed those instances from this experiment to fairly illustrate the behavior of the proposed techniques. The quality indicators presented in Table 5 are the same introduced in the first preliminary experiment. Finally, let us remark that, since we are trying to test the influence of AE_1 and AE_2 on each local search independently, the deviation (Dev.(%) (exp.)) and the number of best solutions found (#Best (exp.)) are computed with respect to each local search separately.

Observing the results in Table 5 we can conclude that the inclusion of AE_1 in combination with both local search procedures, results in a large improvement in the performance of the methods in terms of quality. This technique avoids the methods from getting easily stuck in flat landscapes when considering a single objective function. However, since the exploration is larger, the CPU time also increases significantly. Additionally, we observe that the influence of AE_1 seems to be slightly smaller in $\mathcal{N}_1$ than in $\mathcal{N}_2$.

On the other hand, the inclusion of AE_2 in combination with both local search procedures and AE_1, results in an improvement in the performance of the methods in terms of CPU time. Particularly, the methods are able to reach the same solutions in terms of quality, but the time needed to evaluate a solution after a move is substantially smaller when using AE_2. Again, we observe that the influence of AE_2 seems to be smaller in $\mathcal{N}_1$ than in $\mathcal{N}_2$.

We complement this experiment by studying the performance of the two classical strategies used to explore a neighborhood within a local search: first improvement and best improvement. Particularly, we study these strategies within a local search combined with AE_1 and AE_2, over the same instances reported in Table 5. As expected, we observed that the first improving strategy was faster in both $\mathcal{N}_1$ and $\mathcal{N}_2$. Moreover, in the case of $\mathcal{N}_2$, the first improvement strategy presented a deviation of 0.87% with respect to the best solutions found in this experiment, and 13 #Best (exp.) solutions, while the best improvement strategy had a deviation of 1.76% and found 11 #Best (exp.) solutions. Therefore, we chose the first improvement strategy to explore the swap neighborhood ($\mathcal{N}_2$). On the other hand, in the case of $\mathcal{N}_1$, the best improvement strategy had a deviation of 2.26% and reached 15 #Best (exp.) solutions, while the first improvement strategy had a deviation of 3.56% and reached 13 #Best (exp.). Therefore, in this case, we chose the best improvement strategy as a balance between quality and CPU time.

The next preliminary experiment is devoted to determine the influence in the CPU time of the neighborhood reduction strategies, used in combination with our local search procedures. Particularly, we ran the local search for one iteration for each variant and observed the differences in time and quality. Let us remember that, we proposed two neighborhood reduction techniques R_1 and R_2 (see Sect. 3.2 for further details). The results of this experiment are compiled in Table 6, where we report the results obtained for the same subset of instances used in Table 5. Notice, that since we are trying to test the influence of R_1 and R_2 in the original neighborhoods proposed ($\mathcal{N}_1$ and $\mathcal{N}_2$) independently, the deviation and the number of best solutions reported in Table 6 are computed only with respect to the columns belonging to $\mathcal{N}_1$ or $\mathcal{N}_2$ separately.

Table 6 Influence of the neighborhood reduction strategies (R_1 and R_2) combined with the two local search procedures proposed (LS_1 and LS_2)

	$\mathcal{N}_1$ (Insert)				$\mathcal{N}_2$ (Swap)			
	$\mathcal{N}_1$	$\mathcal{N}_1^{R_1}$	$\mathcal{N}_1^{R_2}$	$\mathcal{N}_1^{R_1R_2}$	$\mathcal{N}_2$	$\mathcal{N}_2^{R_1}$	$\mathcal{N}_2^{R_2}$	$\mathcal{N}_2^{R_1R_2}$
Avg.	30.29	26.47	30.65	20.82	34.71	30.82	30.29	15.12
CPUt(s)	105.07	2.06	0.27	0.03	0.60	0.13	0.04	0.004
Dev.(%) (exp.)	2.58	10.97	4.47	27.85	0.44	9.14	21.93	51.85
#Best (exp.)	14	6	12	2	15	5	2	0
Dev.(%) (all)	23.68	30.60	25.29	43.10	16.84	24.34	36.06	60.62
#Best (all)	3	2	3	2	4	2	0	0

Particularly, in Table 6, we present the results obtained in several scenarios: exploring the whole neighborhoods (column $\mathcal{N}_1$ or $\mathcal{N}_2$); applying R_1 or R_2 in isolation (column $\mathcal{N}_1^{R_1}$ or $\mathcal{N}_2^{R_1}$ and $\mathcal{N}_1^{R_2}$ or $\mathcal{N}_2^{R_2}$); or applying R_1 and R_2 at the same time (column $\mathcal{N}_1^{R_1R_2}$ or $\mathcal{N}_2^{R_1R_2}$). Notice, that in terms of the solutions within those neighborhoods $\mathcal{N}_1^{R_1R_2} = \mathcal{N}_1^{R_1} \cap \mathcal{N}_1^{R_2}$. Similarly, $\mathcal{N}_2^{R_1R_2} = \mathcal{N}_2^{R_1} \cap \mathcal{N}_2^{R_2}$.

Analyzing the results presented in Table 6, as expected, we observe that the smaller the neighborhood, the shorter the time needed to explore it. This fact is especially accused if we compare the exploration of the whole neighborhoods with respect to the exploration of any of the reduced neighborhoods presented. Also, it is important to remark that either R_1 or R_2 are able to notably reduce the CPU time in both: $\mathcal{N}_1$ or $\mathcal{N}_2$, being R_2 able to further reduce the time. However, the combination of R_1 and R_2 results as the fastest overall strategy.

We observe that the impact of the strategies is larger in the case of $\mathcal{N}_1$ where using both strategies together saves 99.97% of the CPU time with respect to explore the whole neighborhood. On the other hand, if we observe the impact of the strategies in terms of quality, as expected, the smaller the neighborhood explored, the poorer the quality obtained. However, in this context, it is important to find a good balance between quality and running time.

The last preliminary experiment is devoted to illustrate the contribution of each of the strategies proposed for the final configuration of the algorithm. Therefore, we configured several variants of our method by increasing the complexity of the strategies introduced, to test that including additional strategies, with the same computing time, contributes to the final design of the algorithm. Then, the compared procedures were executed for a time limit of 100 s and the best solution of each method was reported. In Table 7 we report the results obtained with the constructive procedure (Greedy), the influence of the two local search procedures in isolation (LS_1+AE_1 +AE_2, and LS_2+AE_1+AE_2), the VND method (VND) and the GVNS (GVNS) configured with $k_{max} = 0.01n$, being n the number of vertices of the graph. In Table 7, we observe how the average of the objective function and the number of best solutions increase when considering a more advanced strategy. Consequently, the deviation of GVNS to the best solution found in the experiment is the smaller one. Also, we observe how combining several neighborhoods within a VND algorithm results

Table 7 Evolution of the quality of the solutions obtained with the different strategies proposed

	Greedy	LS_1+AE_1+AE_2	LS_2+AE_1+AE_2	VND	GVNS
Avg.	146.79	155.14	156.32	157.14	157.79
CPUt (s)	100.00	100.02	102.94	107.09	100.02
Dev.(%) (exp.)	16.12	2.62	1.56	0.72	0.26
#Best (exp.)	6	15	14	16	22
Dev.(%) (all)	19.38	6.49	5.54	4.81	4.43
#Best (all)	5	12	12	13	14

in a larger improvement than exploring the neighborhoods in isolation, during the same time horizon.

Finally, it is important to remark that we performed a fine tuning experiment to adjust the parameters of our final method. Particularly, we use the calibration package *irace* (Iterated Race for Automatic Algorithm Configuration) [19]. Using *irace*, we adjusted the largest neighborhood (k_{max}) and the maximum time for each iteration (t'_{max}) of the GVNS. The k_{max} value is chosen dynamically for each instance depending on the number of vertices (n) of the input graph. To determine the most suitable value, we tested different percentages of n in the range $[0.005, 0.05]$ in steps of 0.005. Similarly, we study the behavior of the GVNS with different time limits in the range $[10, 100]$ seconds in steps of 5 s. The *irace* determined that the best performance of the method was obtained with any of the following three configurations: 1. $k_{max} = 0.01n$ and $t'_{max} = 30$; 2. $k_{max} = 0.01n$ and $t'_{max} = 25$; and 3. $k_{max} = 0.01n$; and $t'_{max} = 40$. Among the proposed configurations proposed by *irace*, we selected $k_{max} = 0.01n$ and $t'_{max} = 25$ for our final design, since it is the fastest among the compared ones. Notice, that the final implementation of our GVNS includes, for each local search procedure within the VND, both advanced strategies (AE_1 and AE_2). Additionally, in the case of the local search based on the insert operator we explore the neighborhood obtained after applying the two neighborhood reduction strategies proposed. On the other hand, in the case of the local search based on the swap operator we only consider R_2 as a balance between time and quality.

4.3 Final experiments

In this section, we compare our MS-GVNS with the previous methods in the state of the art for the CAB: the Memetic Algorithm (MACAB) [3] and the Hybrid Artificial Bee Colony (HABC) [20]. Both procedures have been described in the literature review section. Notice that in the case of the HABC, which is the current state of the art for the problem, we have run the original source code provided by the authors. In the case of the MACAB, we report the results provided by the authors of the previous comparative experimentation [20], since we do not have the original source code. However, in that experiment, the MACAB was run (together with the HABC) on an Intel Core i7, with 3.2 Ghz processor and 12 GB of RAM. On the basis of the

Table 8 Comparison with the state of the art when running each method for 150 s, on the sets of instances with known optimum

		MS-GVNS	HABC	MACAB
Paths (22)	Avg.	111.68	111.86	111.91
	Dev. (%)	0.16	0.01	0.00
	#Opt.	17	21	22
Cycles (22)	Avg.	109.46	106.13	98.21
	Dev. (%)	3.60	4.81	18.75
	#Opt.	4	2	12
Grids (21)	Avg.	281.71	281.71	283.70
	Dev. (%)	0.47	0.72	0.27
	#Opt.	10	7	12
Toroidal grids (33)	Avg.	217.79	220.79	164.03
	Dev. (%)	20.40	17.20	39.37
	#Opt.	9	6	4
Hamming (22)	Avg.	54.91	60.27	21.77
	Dev. (%)	29.14	23.65	55.03
	#Opt.	0	0	0
Total (120)	Avg.	159.80	161.53	137.62
	Dev. (%)	11.72	10.08	24.40
	#Opt.	40	36	50

results obtained, we can consider that HABC is the current state of the art for the CAB (i.e., the best previous method in the literature).

To compare the procedures, we have followed the same criteria proposed in the state of the art. Particularly, we run all the algorithms over the whole set of instances (excluding the instances used in the preliminary experiments) with a maximum time limit of 150 s per instance (previously set in the state of the art). The results are grouped by sets of instances and then organized in two different tables. In Table 8 we report the results for the sets of instances with known optimum. Similarly, in Table 9 we report the results for the sets of instances where the optimum remains unknown. At the bottom of both tables we have added a row (labeled as "Total") which provides the average for all instances of the table. Since the provided results in these tables are reported on average, we refer the reader to https://grafo.etsii.urjc.es/optsicom/cab/ where it is possible to find the individual results per instance.

In Table 8, since we are using instances with known optimum, we report the result obtained by each of the compared methods with respect to the optimum value per instance. As we can observe, the results obtained by the methods are very similar in global terms, when comparing the deviation (Dev.(%)), the number of optimum values found (#Opt.) and the average of the objective function (Avg.). Particularly, the HABC is the method with the smallest deviation with respect to the optimum values (10.08%), closely followed by MS-GVNS (with a deviation of 11.72%), and then by MACAB (with a deviation of 24.40%). However, MACAB is the best overall method in terms of the number of optimum solutions found (50), followed by MS-GVNS (40) and by HABC (36). To determine if there are significant differences among the methods, we performed a Friedman

Table 9 Comparison with the state of the art when running each method for 150 s, on the sets of instances with unknown optimum

		MS-GVNS	HABC	MACAB
Three dimensional meshes (18)	Avg.	741.39	759.33	762.17
	Dev. (%)	1.83	0.38	0.07
	#Best	4	4	17
Double stars (18)	Avg.	10.61	10.17	10.61
	Dev. (%)	0.00	5.80	0.00
	#Best	18	10	18
Hypercubes (5)	Avg.	117.60	109.80	120.80
	Dev. (%)	1.11	4.04	0.00
	#Best	3	2	5
Caterpillar (36)	Avg.	142.92	141.67	141.44
	Dev. (%)	0.00	0.84	2.53
	#Best	36	15	4
Complete binary tree (21)	Avg.	175.14	164.19	130.71
	Dev. (%)	0.00	3.27	21.10
	#Best	21	10	0
Harwell–Boeing (21)	Avg.	69.81	69.71	43.81
	Dev. (%)	0.46	1.43	42.98
	#Best	17	13	3
Random connected (28)	Avg.	7.93	6.43	3.64
	Dev. (%)	0.00	19.73	45.39
	#Best	28	4	2
Total (147)	Avg.	167.59	167.29	159.00
	Dev. (%)	0.33	5.53	18.43
	#Best	127	58	49

test. The obtained p value of 0.28236 indicates that we can not affirm the existence of significant differences among the tested methods.

In Table 9 we present the results obtained for the sets of instances where the optimum remains unknown. In this case, since the optimal value per instance is not available, we report the deviation with respect to the best value obtained in the experiment, and consequently, we report the number of best values found (#Best) instead of the number of optimal values. The practical interest of using heuristic procedures over instances where the optimum is unknown is larger than in instances where the optimum is known.

In this case, the MS-GVNS is the best overall method in terms of average of the objective function, deviation from the best solution, and the number of best solutions found. Particularly, MS-GVNS obtains a deviation of (0.33%), followed by HABC (with a deviation of 5.53%), and MACAB (with a deviation of 18.43%). The number of best solutions found by MS-GVNS was 127, followed by HABC with 58 and by MACAB with 49 best solutions found. Again, to determine if there are significant differences among the methods, we performed a Friedman test. The obtained p value lower than 0.001 indicates the existence of

significant differences among the tested methods. Given the previous result, we have performed a post-test analysis which consists of ranking the compared methods using the average rank values computed with this test. According to this rank, the best method is the MS-GVNS (with a rank value of 2.48), followed by the HABC (with 1.95 rank value) and finally by the MACAB (with 1.57 rank value). Finally, to support the previous observations, we have performed a Wilcoxon test between the two methods which ranked in the first and second position in the previous ranking (i.e., the MS-GVNS and HABC). Again, the obtained p value lower than 0.001 corroborates the existence of significant differences between the methods, confirming the remarkable performance of MS-GVNS.

As a final experiment, given the existent relationship between the CAB and AMP, introduced in Eq. 6, we have computed the lower and upper bounds for those instances used in this paper, which had been previously solved in the context of the AMP in the literature. Particularly, we computed these bounds for the 24 instances belonging to the Harwell-Boeing data set. However, the average gap (the difference between the upper and lower bounds) for these instances was very large (40.85 on average). Additionally, the result obtained by our algorithmic proposal was never improved by the upper bound calculated for those instances. Furthermore, considering the gap between the solutions obtained with our procedure and the lower bounds introduced in Eq. 6 we observed an averaged gap of 29.72, which improves the gap previously obtained. Unfortunately, we were unable to prove any solution to be optimal (i.e., the difference between the upper and lower bounds is 0).

5 Conclusions

In this paper, we have studied the Cyclic Antibandwidth problem which consists in embedding an input graph into a cycle host graph, to maximize the bandwidth measured in the cycle. This problem has been previously studied for specific classes of graphs using exact methods. However, just two approaches can be found in the literature for general graphs: a Memetic Algorithm and an Artificial Bee Colony with Tabu Search.

To handle the problem, we propose a novel constructive procedure based on previous ideas. The solutions provided by this method were then improved with a General Variable Neighborhood Search in a multistart algorithmic design. Additionally, we propose the use of two strategies to reduce the neighborhood size: (1) determining the vertices likely to produce an improvement in the objective function; and (2) determining the most suitable assignation for a vertex without trying all possible assignations. Also, we introduce two advanced strategies to systematically traverse those neighborhoods in an efficient way: (1) using an objective function tiebreak criterion to deal with flat landscapes; and (2) performing an efficient evaluation of the objective function value for neighboring solutions.

Our procedure has been tested in isolation to verify that the new proposed GVNS method, in combination with the additional advanced strategies, is able to accelerate the execution of heuristic search procedures without sacrificing solution quality. Then, the final configuration of our method has been successfully compared with the

previous state-of-the-art procedures, becoming the MS-GVNS the new state-of-the-art method for the Cyclic Antibandwidth problem. The merit of the empirical results obtained was corroborated using non-parametrical statistical tests.

Acknowledgements This research has been partially supported by the Ministerio de Ciencia, Innovación y Universidades (Grant Ref. PGC2018-095322-B-C22 and Grant Ref. FPU19/04098) and by Comunidad de Madrid and European Regional Development Fund (Grant Ref. P2018/TCS-4566).

Data availability The data that support the findings of this study are available from the corresponding author upon request.

References

1. Ahuja, R.K., Ergun, O., Orlin, J.B., Punnen, A.P.: A survey of very large-scale neighborhood search techniques. Discret. Appl. Math. **123**(1), 75–102 (2002)
2. Alway, G., Martin, D.: An algorithm for reducing the bandwidth of a matrix of symmetrical configuration. Comput. J. **8**(3), 264–272 (1965)
3. Bansal, R., Srivastava, K.: A memetic algorithm for the cyclic antibandwidth maximization problem. Soft. Comput. **15**(2), 397–412 (2011)
4. Bhatt, S.N., Thomson Leighton, F.: A framework for solving VLSI graph layout problems. J. Comput. Syst. Sci. **28**(2), 300–343 (1984)
5. Cavero, S., Pardo, E.G., Laguna, M., Duarte, A.: Multistart search for the cyclic cutwidth minimization problem. Comput. Oper. Res. **126**, 105–116 (2021)
6. Dobrev, S., Královič, R., Pardubská, D., Török, L., Vrt'o, I.: Antibandwidth and cyclic antibandwidth of Hamming graphs. Discret. Appl. Math. **161**(10), 1402–1408 (2013)
7. Duarte, A., Escudero, L.F., Martí, R., Mladenovic, N., Pantrigo, J.J., Sánchez-Oro, J.: Variable neighborhood search for the vertex separation problem. Comput. Oper. Res. **39**(12), 3247–3255 (2012)
8. Duarte, A., Pantrigo, J.J., Pardo, E.G., Sánchez-Oro, J.: Parallel variable neighbourhood search strategies for the cutwidth minimization problem. IMA J. Manag. Math. **27**(1), 55–73 (2016)
9. Duarte, A., Sánchez-Oro, J., Mladenović, N., Todosijević, R.: Variable Neighborhood Descent. In: Martí, R., Pardalos, P.M., Resende, M.G.C. (eds.) Handbook of Heuristics, pp. 341–367. Springer, Cham (2018)
10. Duff, I.S., Grimes, R.G., Lewis, J.G.: Users Guide for the Harwell–Boeing Sparse Matrix Collection (Release I). RAL, Chilton (1992)
11. Díaz, J., Petit, J., Serna, M.: A survey of graph layout problems. ACM Comput. Surv. **34**(3), 313–356 (2002)
12. Hale, W.: Frequency assignment: theory and applications. Proc. IEEE **68**(12), 1497–1514 (1980)
13. Hansen, P., Mladenović, N.: Variable Neighborhood Search. In: Burke, E.K., Kendall, G. (eds.) Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, pp. 313–337. Springer, US, Boston, MA (2014)
14. Hansen, P., Mladenović, N., Todosijević, R., Hanafi, S.: Variable neighborhood search: basics and variants. EURO J. Comput. Optim. **5**(3), 423–454 (2017)
15. Harper, L.H.: Optimal numberings and isoperimetric problems on graphs. J. Comb. Theory **1**(3), 385–393 (1966)
16. Hromkovic, J., Muller, V., Sykora, O., Vrto, I.: On embeddings in cycles. Inf. Comput. **118**(2), 302–305 (1995)
17. Jain, P., Srivastava, K., Saran, G.: Minimizing cyclic cutwidth of graphs using a memetic algorithm. J. Heurist. **22**(6), 815–848 (2016)
18. Leung, J.Y.-T., Vornberger, O., Witthoff, J.D.: On some variants of the bandwidth minimization problem. SIAM J. Comput. **13**(3), 650–667 (1984)
19. López-Ibá nez, M., Dubois-Lacoste, J., P. Cáceres, L., Birattari, M., Stützle, T.: Iterated racing for automatic algorithm configuration. The irace package. Oper. Res. Perspect. **3**, 43–58 (2016)

20. Lozano, M., Duarte, A., Gortázar, F., Martí, R.: A hybrid metaheuristic for the cyclic antibandwidth problem. Knowl. Based Syst. **54**, 103–113 (2013)
21. Martí, R.: Multi-start methods. In: Handbook of Metaheuristics. International Series in Operations Research and Management Science, pp. 355–368. Springer, Boston (2003)
22. Martí, R., Laguna, M., Glover, F., Campos, V.: Reducing the bandwidth of a sparse matrix with tabu search. Eur. J. Oper. Res. **135**(2), 450–459 (2001)
23. Martí, R., Pantrigo, J.-J., Duarte, A., Campos, V., Glover, F.: Scatter search and path relinking : a tutorial on the linear arrangement problem. Int. J. Swarm Intell. Res. (IJSIR) **2**(2), 1–21 (2011)
24. Miller, Z., Pritikin, D.: On the separation number of a graph. Networks **19**(6), 651–666 (1989)
25. Mladenović, N., Dražić, M., Kovačevic-Vujčić, V., Čangalović, M.: General variable neighborhood search for the continuous optimization. Eur. J. Oper. Res. **191**(3), 753–770 (2008)
26. Mladenović, N., Hansen, P.: Variable neighborhood search. Comput. Oper. Res. **24**(11), 1097–1100 (1997)
27. Pardo, E.G., Martí, R., Duarte, A.: Linear Layout Problems. In: Martí, R., Panos, P., Resende, M.G. (eds.) Handbook of Heuristics, pp. 1–25. Springer, Cham (2016)
28. Pardo, E.G., Mladenović, N., Pantrigo, J.J., Duarte, A.: Variable formulation search for the cutwidth minimization problem. Appl. Soft Comput. **13**(5), 2242–2252 (2013)
29. Pardo, E.G., Soto, M., Thraves, C.: Embedding signed graphs in the line. J. Comb. Optim. **29**(2), 451–471 (2015)
30. Pastore, T., Martínez-Gavara, A., Napoletano, A., Festa, P., Martí, R.: Tabu search for min-max edge crossing in graphs. Comput. Oper. Res. **114**, 104830 (2020)
31. Piñana, E., Plana, I., Campos, V., Martí, R.: GRASP and path relinking for the matrix bandwidth minimization. Eur. J. Oper. Res. **153**(1), 200–210 (2004)
32. Raspaud, A., Schröder, H., Sýkora, O., Torok, L., Vrt'o, I.: Antibandwidth and cyclic antibandwidth of meshes and hypercubes. Discret. Math. **309**(11), 3541–3552 (2009)
33. Raspaud, A., Sýkora, O., Vrt'o, I.: Congestion and dilation, similarities and differences: a survey. In: Proceedings of the 7th International Colloquium on Structural Information and Communication Complexity, pp. 14 (2000)
34. Ren, J., Hao, J.-K., Rodriguez-Tello, E., Li, L., He, K.: A new iterated local search algorithm for the cyclic bandwidth problem. Knowl. Based Syst. **203**, 106–136 (2020)
35. Rodriguez-Tello, E., Hao, J.-K., Torres-Jimenez, J.: An effective two-stage simulated annealing algorithm for the minimum linear arrangement problem. Comput. Oper. Res. **35**(10), 3331–3346 (2008)
36. Rodriguez-Tello, E., Lardeux, F., Duarte, A., Narvaez-Teran, V.: Alternative evaluation functions for the cyclic bandwidth sum problem. Eur. J. Oper. Res. **273**(3), 904–919 (2019)
37. Rodriguez-Tello, E., Narvaez-Teran, V., Lardeux, F.: Dynamic multi-armed bandit algorithm for the cyclic bandwidth sum problem. IEEE Access **7**, 40258–40270 (2019)
38. Rodriguez-Tello, E., Romero-Monsivais, H., Ramirez-Torres, G., Lardeux, F.: Tabu search for the cyclic bandwidth problem. Comput. Oper. Res. **57**, 17–32 (2015)
39. Rost, M., Schmid, S.: Charting the complexity landscape of virtual network embeddings. In: 2018 IFIP Networking Conference (IFIP Networking) and Workshops, pp. 1–9 (2018)
40. Skiena, S.S.: Graph Traversal. In The Algorithm Design Manual, 2nd edn. Springer Publishing Company, Berlin (1997)
41. Sánchez-Oro, J., José Pantrigo, J., Duarte, A.: Combining intensification and diversification strategies in VNS. An application to the vertex separation problem. Comput. Oper. Res. **52**, 209–219 (2014)
42. Sýkora, O., Torok, L., Vrt'o, I.: The cyclic antibandwidth problem. Electr. Notes Discrete Math. **22**, 223–227 (2005)
43. Weili, Y., Xiaoxu, L., Ju, Z.: Dual bandwidth of some special trees. Journal–Zhengzhou Univ. Nat. Sci. Ed. **35**(3), 16–19 (2003)

Chapter 9

Cyclic Bandwidth Sum Problem

The Cyclic Bandwidth Sum Problem is the last CGLP studied in this Doctoral Thesis and it was previously introduced in Section 2.3. As a result of the research conducted, an article has been published:

1. S. Cavero, E. G. Pardo, A. Duarte, and E. Rodriguez-Tello. A variable neighborhood search approach for cyclic bandwidth sum problem. *Knowledge-Based Systems*, 246:108680, 2022 [33].

Moreover, a presentation has been made at a national conference. This presentation, although it could also be associated with the two previous problems (CCMP and CAB), collects, from a more general point of view, the lessons learned in the study of previous CGLP. The paper is specified next:

2. E. G. Pardo, S. Cavero, and A. Duarte. Un enfoque metaheurístico para problemas de ordenación circular. *XXXIX Congreso Nacional de Estadística e Investigación Operativa* (SEIO 2022), in Granada, Spain, 2021 [189].

The article, titled: titled "*A variable neighborhood search approach for cyclic bandwidth sum problem*" [33], is published in a JCR journal. Figure 9.1 compiles some information about the journal. Note that, in addition to the Ph.D. candidate and his supervisors, Professor Eduardo Rodriguez-Tello from the Cinvestav Tamaulipas (Mexico) also collaborated in this research. The CBS was previously studied from an exact perspective for

some types of graphs with a regular structure. Additionally, previous heuristic procedures have been proposed for general random graphs. In particular, the heuristic approaches are based on GVNS, MA, and a combination of the previous MA with the Multi-Armed Bandit framework. In this research, we introduce a new algorithm to address the CBS based on two key components: a new greedy constructive procedure and an efficient local search method. The main components of the algorithm proposed are summarized next:

- Two-phase **greedy constructive procedure**. The first phase determines the next vertex to be added to the solution based on the adjacency of the vertices and a weighted version of the greedy criteria proposed in previous works [34, 168]. Then, a greedy function based on the objective function of the problem is used to locate the selected vertex in the host graph.

- **Local search** that efficiently explores the insert neighborhood following a best improvement strategy. Specifically, the exploration consists of using a chain of swap moves whose result is equivalent to inserting a vertex into all vertices of the host graph. This idea, combined with an efficient calculation of the objective function after a move, allows the method to drastically reduce the complexity of the exploration.

- The solutions generated by the constructive method are further improved by a procedure based on the **BVNS** metaheuristic. This metaheuristic combines local search with perturbations of the solution using the shake procedure.

Finally, it is worth mentioning that our best algorithmic variant has been favorably compared to the best previous method in the state of the art. The results of these comparisons have been supported by statistical tests that corroborate the merit of our proposal and establish it as the new state-of-the-art method for the CBS.

To conclude this chapter, we include a copy of the most relevant paper published for the CBS in the context of this Doctoral Thesis.

A VNS approach for the cyclic bandwidth sum problem

Sergio Cavero, Eduardo G. Pardo, Abraham Duarte and Eduardo Rodriguez-Tello
Knowledge-Based Systems. Volume 246(2), 108680, 2022.
`https://link.springer.com/article/10.1007/s10589-021-00334-y`

Journal Information

Research Areas:

- Computer Science, Artificial Intelligence

Category Rank:

- Computer Science, Artificial Intelligence 24/144 (Q1)

Journal Impact Factor: 8.139

Data obtained from Journal Citation Reports 2021

Figure 9.1 Information related to the publication [33].

Knowledge-Based Systems 246 (2022) 108680

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

A variable neighborhood search approach for cyclic bandwidth sum problem

Sergio Cavero [a], Eduardo G. Pardo [a,*], Abraham Duarte [a], Eduardo Rodriguez-Tello [b]

[a] Universidad Rey Juan Carlos, Department of Computer Sciences, C/Tulipán s/n, 28903, Madrid, Spain
[b] Unidad Tamaulipas, Cinvestav, Km. 5.5 Carretera Victoria - Soto La Marina, Victoria, 87130, Tamps., Mexico

ARTICLE INFO

Article history:
Received 3 September 2021
Received in revised form 23 March 2022
Accepted 25 March 2022
Available online 6 April 2022

Keywords:
Cyclic bandwidth sum
Graph layout problem
Variable Neighborhood search
Greedy algorithm
Combinatorial optimization

ABSTRACT

In this paper, we tackle the Cyclic Bandwidth Sum Problem, consisting in minimizing the sum of the bandwidth of the edges of an input graph computed in a cycle-structured host graph. This problem has been widely studied in the literature due to its multiple real-world applications, such as circuit design, migration of telecommunication networks, or graph drawing, among others. Particularly, we tackle this problem by proposing a multistart procedure whose main components are a new greedy constructive algorithm and an intensification strategy based on the Variable Neighborhood Search metaheuristic. The constructive procedure introduces two different greedy criteria to determine each step of the construction phase, which can be used for other related problems. Additionally, we illustrate how to perform an efficient exploration of the neighborhood structure by using an alternative neighborhood. Our algorithmic proposal is evaluated over a set of 40 instances previously studied in the literature and over a new proposed set of 66 well-known instances introduced in this paper. The obtained results have been satisfactory compared with the ones obtained by the best previous algorithm in the state of the art. The statistical tests performed indicate that the differences between the methods are significant.

1. Introduction

In this research, we deal with the Cyclic Bandwidth Sum Problem (CBSP), a $\mathcal{NP}$-hard combinatorial optimization problem belonging to the graph layout family of problems [1]. From a theoretical perspective, a graph layout problem consists in embedding an input graph $G = (V_G, E_G)$ into a host graph $H = (V_H, E_H)$ such that a certain objective function is optimized. Particularly, an embedding is defined through the use of two functions: the first one, denoted as φ, maps the vertices of G into the vertices of H ($\varphi : V_G \rightarrow V_H$); the second one, denoted as ψ, associates the edges of G to a path in H ($\psi : E_G \rightarrow P_H$), being P_H all feasible paths of H. It is worth mentioning that a path, denoted as $p(u, v)$, is a unique-ordered sequence of edges or vertices with endings in u and v. Particularly, in this paper, a path is defined as a sequence of edges.

The most studied problems within the graph layout family are those where the host graph is a regular graph, such as path graphs [2,3], cycle graphs [4–6], tree graphs [7,8], or grid graphs [9], among others.

The Cyclic Bandwidth Sum problem was originally proposed in [10], where it was proved to be $\mathcal{NP}$-hard. Since then, researchers have focused their work on solving the problem exactly for some particular graphs with regular structure. The most relevant contributions study the CBSP for paths, cycles, wheels, power of cycles, and complete bipartite graphs [11,12]. These results are extended by establishing upper bounds based on the Cartesian product of some graphs (specifically, for paths, cycles and complete graphs [12]). However, as far as we know, there is not a method able to solve the problem for general graphs due to its computational complexity.

Recently, researchers have focused on the proposal of heuristic algorithms to address the CBSP for any type of input graph. Among the techniques used in the literature to tackle the CBSP and other related Graph Layout Problems we can find either trajectory-based metaheuristics (e.g., Greedy Randomized Adaptive Search Procedure [13], Variable Neighborhood Search [14], or Tabu Search [4], etc.) and computational intelligence algorithms (e.g., Particle Swarm Optimization [15], Ant Colony Optimization [16], or Artificial Bee Colony [17], etc.). Among the different methods, Variable Neighborhood Search methodology has been particularly successful for this kind of problems, mainly due to two key factors: the change of neighborhood during the search, and the combination of deterministic and stochastic explorations.

* Corresponding author.
E-mail addresses: sergio.cavero@urjc.es (S. Cavero), eduardo.pardo@urjc.es (E.G. Pardo), abraham.duarte@urjc.es (A. Duarte), ertello@cinvestav.mx (E. Rodriguez-Tello).

https://doi.org/10.1016/j.knosys.2022.108680
0950-7051/

The first heuristic approach found in the literature for the CBSP is based on the General Variable Neighborhood Search (GVNS) metaheuristic [18]. In this research, the authors proposed two local search methods and six different shake procedures. Additionally, the initial solutions provided to the algorithm were generated with a random constructive paired with a Reduced Variable Neighborhood Search. This first proposal was able to find solutions tighter than the previous upper bounds established in [12].

Later, in [19], authors proposed a constructive procedure, denoted as MACH, that finds better solutions than those obtained by the GVNS procedure proposed in [18]. This constructive was inspired by the Jaccard similarity index [20] and the Depth First Search algorithm [21]. The solutions obtained by the MACH proposal were later improved with a Memetic Algorithm (MA) which operators were adaptively adjusted by a Dynamic Multi-Armed Bandit (DMAB) framework [22], denoted DMAB+MA. Some of the ideas implemented in the final configuration of DMAB+MA had already been introduced in [5,23]. Particularly, in [5], the authors performed a depth comparative analysis of three new alternative objective functions for the CBSP. Furthermore, in [23], a total 24 MA versions were presented, and the results, altogether, were successfully compared with the previous state-of-the-art algorithm.

DMAB+MA can be considered as the current state-of-the-art method for solving the CBSP. Particularly, this algorithm is able to find the optimal solution for those instances with a known optimum and, additionally, it is able to find the best-known values for the rest of the instances tested in the literature.

Further than the aforementioned procedures devoted to the CBSP, it is also worth mentioning the relations of the CBSP to other similar problems, such as the Cyclic Cutwidth Minimization Problem (CCMP) [4,24] or the Minimum Linear Arrangement Problem (MinLA) [25,26]. Specifically, the CCMP is also a graph layout problem where the host graph is a cycle graph and looks for minimizing the maximum number of paths that contain a particular host edge, while the CBSP looks for minimizing the sum of the paths that traverse every host edge. Therefore, an upper bound for the CBSP can be derived from the CCMP, $\text{CBS}(G) \leq n \cdot \text{CCMP}(G)$, where n is the number of vertices of the input graph ($|V_G|$).

On the other hand, the MinLA and the CBSP look for the optimization of the same objective function, but differ in the type of graph used as a host graph (i.e., a path graph in the case of the MinLA, and a cycle graph in the case of the CBSP). Therefore, upper and lower bounds for the CBSP can be derived from the MinLA. In mathematical terms, given a graph G, $\frac{\text{MinLA}(G)}{2} \leq \text{CBS}(G) \leq \text{MinLA}(G)$.

The main contribution of this work is the introduction of new heuristic algorithms to find high-quality solutions to the CBSP. Our proposal includes a novel greedy constructive procedure that exploits the structural properties of the input graphs, as well as an efficient intensification strategy based on the Basic Variable Neighborhood Search (BVNS) methodology. The inspiration behind this approach is based on the previous results obtained with this combination of strategies. Specifically, both strategies can be used to find high-quality solutions for other related optimization problems belonging to the graph layout family. The contribution of each proposed component in our final algorithm is justified through a set of preliminary experiments. Furthermore, the resultant procedure is satisfactorily compared with the best previous method identified in the state of the art. In particular, our proposal is able to find better quality solutions (supported by statistical tests) in less computing time.

The rest of the paper is organized as follows. Next, in Section 2, we formalize the CBSP. In Section 3, we introduce the main algorithmic multistart scheme proposed in this paper to handle the CBSP. First, we provide a detailed description of the new constructive procedure proposed for the problem (Section 3.1). Then, we introduce the BVNS procedure used to improve the solutions (Section 3.2). Additionally, we also introduce an efficient strategy to perform an exhaustive exploration of the solution space. In Section 4, we perform a set of computational experiments devoted to tuning the parameters of our proposed algorithm, and we compare our best variant with the best previous algorithm in the literature. Finally, in Section 5, we present the conclusions drawn from this work.

2. Problem definition

The CBSP is a graph layout problem where the host graph, denoted as H, is restricted to be a cycle. Then H is a 2-regular, Eulerian, Hamiltonian, and unit distance graph, and it satisfies that $|V_H| = |E_H| = |V_G|$. Moreover, in this particular problem, ψ depends on φ, since we can express ψ in terms of φ:

$$\begin{aligned} &\psi((u,v)) = \arg\min\{|p(\varphi(u),\varphi(v))|, |p(\varphi(v),\varphi(u))|\} \\ &\forall\,(u,v) \in E_G \text{ and } \{p(\varphi(u),\varphi(v)),\ p(\varphi(v),\varphi(u))\} \subset P_H. \end{aligned} \tag{1}$$

Therefore, to simplify the notation, in the rest of the paper, we refer to a solution or embedding just by the φ function. Additionally, since H is a cycle, for every edge $(u,v) \in E_G$ there are two possible paths in P_H with endings in $\varphi(u)$ and $\varphi(v)$ (i.e., $p(\varphi(u),\varphi(v))$ and $p(\varphi(v),\varphi(u))$). Intuitively, in a graphical representation, we consider the path $p(\varphi(u),\varphi(v))$ as the one starting in $\varphi(u)$ and ending in $\varphi(v)$ following a clockwise order. Similarly, we consider $p(\varphi(v),\varphi(u))$ as the path starting in $\varphi(v)$ and ending in $\varphi(u)$ following a counterclockwise order. It is worth mentioning that the length of each of the two paths (determined by the number of edges that contains) might be different. However, they satisfy that $p(\varphi(u),\varphi(v)) \cup p(\varphi(v),\varphi(u)) = E_H$.

Once the concept of embedding has been introduced, we define the bandwidth (bw) of an edge $(u,v) \in E_G$ for an embedding φ as follows:

$$bw((u,v),\varphi) = \min\{|p(\varphi(u),\varphi(v))|, |p(\varphi(v),\varphi(u))|\}, \tag{2}$$

Then, the evaluation of the objective function of the CBSP for a particular embedding φ of the input graph G, is computed as a sum of the length of the paths in the host graph, associated with each edge of the input graph. More formally:

$$cbs(G,\varphi) = \sum_{(u,v)\in E_G} bw((u,v),\varphi) \tag{3}$$

Finally, the objective of CBSP is to find an embedding $\varphi^\star$ among all possible embeddings, Φ, that minimizes Eq. (3). In mathematical terms:

$$\varphi^\star \leftarrow \underset{\varphi\in\Phi}{\text{argmin}}\{cbs(G,\varphi)\} \tag{4}$$

In Fig. 1, we illustrate the concepts presented in this section with an example. Particularly, in Fig. 1(a) we show an input graph G with $V_G = \{\text{A, B, C, D, E, F}\}$, and $E_G = \{(\text{A, B}), (\text{B, C}), (\text{B, D}), (\text{C, F}), (\text{D, E}), (\text{E, F})\}$. Therefore, $|V_G| = 6$ and $|E_G| = 7$. In Fig. 1(b) we represent the host graph H for the graph G since $|V_G| = |V_H| = |E_H| = 6$. Then, in Fig. 1(c) we show an embedding, φ, of G into H. Particularly, the φ function is defined by the assignments: $\varphi(\text{A}) = 1$, $\varphi(\text{B}) = 2$, $\varphi(\text{C}) = 3$, $\varphi(\text{D}) = 6$, $\varphi(\text{E}) = 5$, and $\varphi(\text{F}) = 4$. Then, the edges of E_G are assigned to paths of H by the ψ function, that, as stated before, can be derived from φ. As an example, given the edge $(\text{D, F}) \in E_G$, there are two possible paths that can be assigned to this edge ($p(\varphi(\text{D}),\varphi(\text{F}))$ or $p(\varphi(\text{F}),\varphi(\text{D}))$), which are depicted in Fig. 1(d). Since $\varphi(\text{D}) = 6$ and $\varphi(\text{F}) = 4$, we are looking for the path in the host graph with the

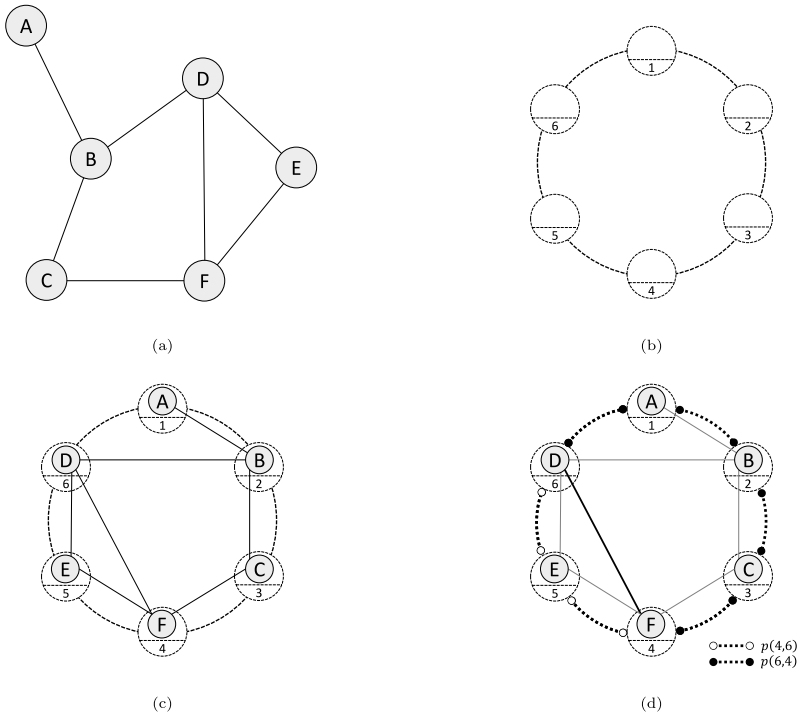


Fig. 1. (a) An input graph, G. (b) A host graph, H. (c) An example of an embedding. (d) The two possible paths that can be assigned to edge (D,F).

smallest cardinality. Particularly, $|p(6,4)| = |\{(6,5),(5,4)\}| = 2 < |p(4,6)| = |\{(6,1),(1,2),(2,3),(2,4)\}| = 4$. Therefore, $p(6,4)$ is assigned to edge (D, F), and so on for the rest of the edges in E_G.

Finally, to calculate the objective function of this particular embedding φ, we sum the length of the paths (i.e., the bw) assigned to each edge of the input graph:

$$\begin{aligned} cbs(G,\varphi) &= bw((\mathrm{A},\mathrm{B}),\varphi) + bw((\mathrm{B},\mathrm{C}),\varphi) + bw((\mathrm{B},\mathrm{D}),\varphi) \\ &\quad + bw((\mathrm{C},\mathrm{F}),\varphi) + bw((\mathrm{D},E),\varphi) + bw((\mathrm{D},\mathrm{F}),\varphi) \\ &\quad + bw((\mathrm{E},\mathrm{F}),\varphi) \\ &= |p(1,2)| + |p(2,3)| + |p(6,2)| + |p(3,4)| + |p(5,6)| \\ &\quad + |p(4,6)| + |p(4,5)| \\ &= 1+1+2+1+1+2+1 = 9. \end{aligned} \tag{5}$$

3. Algorithmic approach

In this section, we introduce our algorithmic proposal to tackle the Cyclic Bandwidth Sum Problem. Particularly, we present a multistart procedure which consists of two phases: the constructive phase, based on a greedy procedure (see Section 3.1), and the improvement phase, based on the Variable Neighborhood Search methodology (see Section 3.2). The rationale behind a multistart method is to diversify the search by restarting the procedure from a different initial point of the solution space. Moreover, the construction and improvement phases are repeatedly applied until a termination criterion is met.

The general scheme of our proposal, named MS-BVNS, is presented in Algorithm 1. The procedure receives three input parameters: a graph (G), the maximum number of consecutive iterations without amelioration (i_{max}), and the maximum number of neighborhoods to explore (k_{max}) in the improvement phase. The termination criteria of the algorithm (step 4) is determined by the parameter i_{max}. In each iteration, the construction and improvement phases are sequentially executed, and the solution found is compared with the best solution found in any previous iteration (step 7). When the procedure stops, the best solution found among all iterations is returned (step 14).

Algorithm 1 Multistart procedure

```
1:  Procedure MS-BVNS (G, i_max, k_max)
2:  i ← 0                                  ▷ # iterations without amelioration
3:  φ* ← ∅
4:  while i < i_max do
5:      φ ← GreedyConstructive(G)
6:      φ' ← BVNS(φ, k_max)
7:      if cbs(G, φ') < cbs(G, φ*) then
8:          φ* ← φ'
9:          i ← 0
10:     else
11:         i ← i + 1
12:     end if
13: end while
14: return φ*
```

3.1. Greedy constructive

We propose a new constructive procedure to generate feasible initial solutions for the CBSP. This procedure consists of three steps: (1) select an unassigned vertex u of the input graph; (2) select an unassigned vertex v of the host graph; and (3) assign u to v (i.e., $\varphi(u) = v$). These steps are repeated until all vertices of the input graph are assigned to a different vertex of the host graph.

The first iteration of the constructive procedure selects u and v at random. Then, in the following iterations, two greedy criteria, computed by the greedy functions g_1 and g_2, are used to select u and v, respectively.

Next, we formally define the two greedy functions g_1 and g_2. Let A_G be the subset of vertices of G that have already been assigned, and U_G be the subset of vertices of G that remains unassigned. Additionally, let $d_{A_G}(u) = |\{v \in A_G : (u, v) \in E_G\}|$ and $d_{U_G}(u) = |\{v \in U_G : (u, v) \in E_G\}|$, such that, the degree of $u \in V_G$, denoted by $d(u)$, satisfies $d(u) = d_{A_G}(u) + d_{U_G}(u)$.

Then, for any $u \in U_G$, we define the greedy function g_1 as follows:

$$g_1(u) = m \cdot d_{A_G}(u) - d_{U_G}(u), \tag{6}$$

where $m \in \mathbb{N}$ (including 0) is a search parameter. Particularly, if $m > 1$ it favors the selection of those unassigned input vertices with a larger number of adjacent vertices already assigned. On the other hand, if $m = 1$ the equation is similar to the idea introduced in [25]. Finally, if $m = 0$ the selection of the next vertex depends only on the number of unassigned adjacent vertices to u. The configuration of m is empirically studied in Section 4.2.

Notice that once all unassigned input vertices have been evaluated with g_1, we select the vertex which maximizes the value returned by g_1, with ties broken at random.

The rationale behind the g_1 function is to measure the immediacy of an unassigned vertex of the input graph to be embedded next. Generally speaking, for a particular candidate vertex, the larger number of adjacent vertices previously assigned, the greater the immediacy of the vertex to be assigned. The g_1 function is inspired by the ideas presented in a previous work [25] where the authors used a similar function for the same task applied to MinLA. The usefulness of these ideas was also extended in [4,27] where the function was again used in the context of the CCMP and its linear version, the Cutwidth Minimization Problem (CMP), respectively. As we presented in Section 1, the MinLA and CCMP are closely related problems to the CBSP. However, in this paper we contribute to the literature of graph layout problems with a novel approach, the introduction of the parameter m in the Eq. (6). The idea of letting m be a search parameter also increases its applicability to other problems, since greater values of m might be better for some problems than others.

Next, we formally define the greedy function g_2. Let A_H be the subset of vertices of H that have already been assigned, and U_H be the subset of vertices of H that remain unassigned. Among the vertices in U_H, we define the subset of candidate host vertices to be assigned next, denoted as C_H, as follows: $C_H = \{v \in U_H : (v, w) \in E_H \wedge w \in A_H\}$. Then, given an input vertex u to be assigned, we define the greedy function g_2, to evaluate any $v \in C_H$, as follows:

$$g_2(u, v) = \sum_{w \in A_H} \min\{|p(v, \varphi(w))|, |p(\varphi(w), v)|\}, \text{ with } (u, w) \in E_G, \text{ and } \{p(v, \varphi(w)), p(\varphi(w), v)\} \subset P_H. \tag{7}$$

Notice that once all unassigned host vertices in C_H have been evaluated with g_2, we select the host vertex which minimizes the value returned by g_2, with ties broken at random.

The logic behind the g_2 function is to determine the best host vertex assignation for a particular input vertex. At the first iteration, all host vertices have the same chance of being selected, therefore one host vertex is chosen at random. The rest of the iterations (except the last one, when there is only one candidate host vertex) there are two possible candidate host vertices. Then, g_2 measures the contribution of the input vertex to the objective function if the vertex is assigned to any of the available candidate host vertices. Algorithm 2 summarizes the greedy constructive procedure. The method receives a graph (G) as an input parameter. First, we generate and empty initial solution and we initialize the sets needed to calculate the functions g_1 and g_2 (step 2).

Then, the first vertices of both the input graph and the host graph are selected at random (steps 3–4) and the initial random assignment is performed (step 5). Next, the sets A_G, U_G, A_H, U_H and C_H are updated (steps 6–7).

As long as the set U is not empty (i.e., there are input vertices remaining to be assigned), the following assignments are made using the two greedy criteria previously described. Particularly, we select the vertex u^* that maximizes the greedy function g_1 (see (Eq. (6))). Next, given u^*, g_2 is calculated (see Eq. (7)), determining the vertex v^* of the host graph that will be assigned to u^*. Finally, the assignation $\varphi(u^*) = v^*$ is performed (step 11) and the corresponding sets are updated (steps 12–13). Finally, the constructed solution is returned (step 15).

Algorithm 2 Greedy constructive procedure

1: **Procedure** `GreedyConstructive` (G)
2: $\varphi \leftarrow \emptyset, A \leftarrow \emptyset, U \leftarrow V_G, A_H \leftarrow \emptyset, U_H \leftarrow V_H, C_H \leftarrow \emptyset$
3: $u \leftarrow \texttt{random}(U)$
4: $v \leftarrow \texttt{random}(V_H)$
5: $\varphi \leftarrow \varphi \cup \{\varphi(u) = v\}$
6: $A \leftarrow A \cup \{u\}, U \leftarrow U \setminus \{u\}, A_H \leftarrow A_H \cup \{v\}, U_H \leftarrow U_H \setminus \{v\}$
7: $C_H \leftarrow C_H \setminus \{v\} \cup \{w \in U_H : (v, w) \in E_H\}$
8: **while** $U \neq \emptyset$ **do**
9: $u^* \leftarrow \underset{u \in V_G}{\text{argmax}}\, g_1(u)$
10: $v^* \leftarrow \underset{v \in C_H}{\text{argmin}}\, g_2(u^*, v)$
11: $\varphi \leftarrow \varphi \cup \{\varphi(u^*) = v^*\}$
12: $A \leftarrow A \cup \{u^*\}, U \leftarrow U \setminus \{u^*\}, A_H \leftarrow A_H \cup \{v^*\}, U_H \leftarrow U_H \setminus \{v^*\}$
13: $C_H \leftarrow C_H \setminus \{v\} \cup \{w \in U_H : (v, w) \in E_H\}$
14: **end while**
15: **return** φ

In Fig. 2 we depict, with an example, the steps followed by the greedy procedure proposed to construct an initial solution. Specifically, we consider the input graph illustrated in Fig. 1(a). In this case, the constructive procedure needs six steps to generate a feasible solution for the problem. For each step, we show which elements constitute the sets: A_G, U_G, A_H and C_H.

In the first step, the input vertex B and the host vertex 1 are selected at random and $\varphi(\text{B}) = 1$. The partial solution obtained is depicted in Fig. 2(a). As it can be observed, sets A_G and C_H, contain vertices B and 1, respectively. Set U_G contains all non-assigned vertices of the input graph ($V_G \setminus \{\text{B}\}$), and set C_H the two possible vertices that meet the condition $\{v \in U_H : (v, 1) \in E_H \wedge 1 \in A_H\}$, 2 and 6. Then, g_1 function is calculated for each of the vertices of U_G. Since A presents the larger value among all the vertices, it is selected as the next vertex to be assigned of the input graph (let us consider for this example the search parameter $k = 5$). Then, the function g_2 is calculated for vertices 2 and 6. Since both have the same quality, the vertex 2 is chosen at random and the assignment $\varphi(\text{A}) = 2$ is performed. We illustrate the

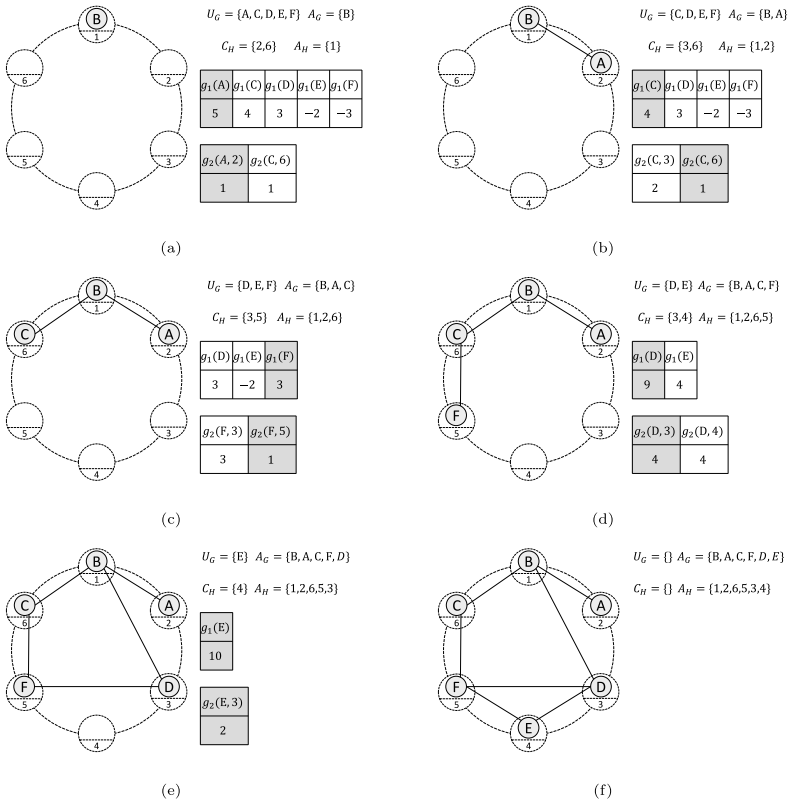


Fig. 2. Example of a step-by-step greedy construction of an initial solution.

resultant partial solution in Fig. 2(b). These steps are repeated until a feasible solution is obtained (see Fig. 2(f)).

3.2. Variable neighborhood search

The solutions generated by our constructive method are further improved by an efficient procedure based on the Variable Neighborhood Search (VNS) metaheuristic. VNS was proposed by Mladenović and Hansen as a general method to solve hard combinatorial optimization problems [28,29]. The basic principle of this methodology is to perform systematic changes in the neighborhood structure to escape from local optima traps. Based on the general idea of VNS, it is possible to find many variants. Some of the best known are: Reduced VNS (RVNS) [30], Basic VNS (BVNS) [28], Variable Neighborhood Descent (VND) [30], General VNS (GVNS) [30], Parallel VNS (PVNS) [31], or Variable Formulation Search (VFS) [14], among others.

In this paper, we propose the use of the BVNS variant as an improving method. The general scheme of BVNS is illustrated in Algorithm 3. Particularly, BVNS receives the input graph (G), a feasible solution (φ) and the largest neighborhood to be explored (k_{max}). The method is conformed by three main steps: a shake procedure which helps to escape from local minima traps (step 5); a local search procedure based on the exploitation of a neighborhood structure (step 6); and the neighborhood change procedure, which determines which neighborhood is explored next (step 7) depending on the improvements found in the current neighborhood. These three steps are repeated until reaching the maximum number of neighborhoods explored, k_{max}. The neighborhoods proposed, the shake procedure, and the local search procedure are detailed next. The neighborhood change procedure follows an standard design (see [29]) which increases the value of k in 1 unit when there is not an improvement in the current iteration and it resets the value of k to 1 when an improvement is found.

Algorithm 3 Basic Variable Neighborhood Search Procedure

1: **Procedure** BVNS (G, φ, k_{max})
2: $\varphi \leftarrow \emptyset$
3: $k \leftarrow 1$
4: **while** $k \leq k_{max}$ **do**
5: $\varphi' \leftarrow$ Shake(φ, k)
6: $\varphi'' \leftarrow$ LocalSearch(G, φ')
7: $\varphi \leftarrow$ NeighborhoodChange(φ, φ'', k)
8: **end while**
9: **return** φ

3.2.1. Neighborhood structures

Any VNS must define the neighborhoods to be explored either in a stochastic or in a systematic way. In this paper, we

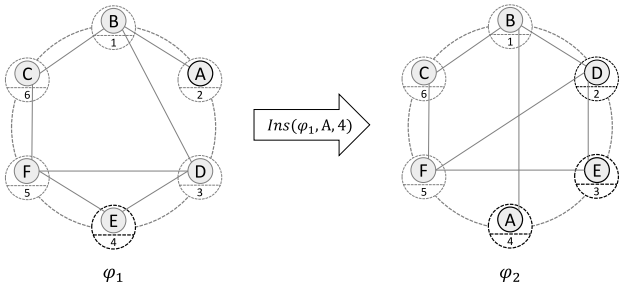


Fig. 3. Example of an embedding φ_1 and the resultant embedding φ_2 obtained after the operation $Ins(\varphi_1, A, 4)$.

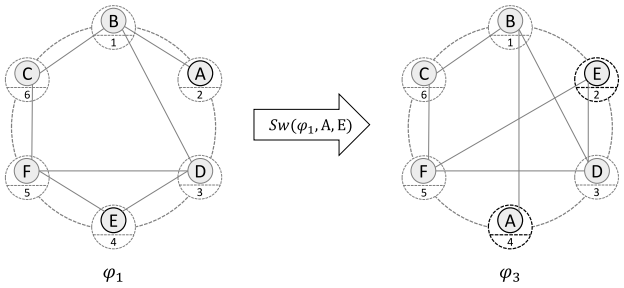


Fig. 4. Example of an embedding φ_1 and the resultant embedding φ_3 obtained after the operation $Sw(\varphi_1, A, E)$.

define two different neighborhood structures for the CBSP: insert neighborhood and swap neighborhood.

The first neighborhood is defined by the *insert* move, formally denoted as $Ins(\varphi, u, v)$. Given a vertex $u \in V_G$, such that $\varphi(u) = v$ (with $v \in V_H$) the *insert* move consists in removing u from its current assignation v and assigning it (i.e., inserting) into another vertex $z \in V_H$ (i.e., $\varphi(u) = z$). Additionally, an *insert* move implies a counterclockwise shift of a set of affected vertices.

Let us illustrate the *insert* move with an example. In Fig. 3, we show an example of an embedding φ_1 of the graph introduced in Fig. 1(a). Then, we show the resultant embedding φ_2 after the *insert* operation $Ins(\varphi_1, A, 4)$. In particular, vertex A of V_G is inserted in vertex 4 of V_H. As it can be observed, to perform this operation, the vertices E and D are shifted in the embedding to make room for vertex A.

Considering the aforementioned *insert* move, we define the associated neighborhood (N_I) as the set of solutions that can be reached by applying the *insert* operator for each $u \in V_G$ and $v \in V_H$. Mathematically:

$$N_I(\varphi) = \{Ins(\varphi, u, v) : \forall\, u \in V_G, \forall\, v \in V_H, v \neq \varphi(u)\}. \tag{8}$$

The second neighborhood proposed in this paper is defined by the *swap* move, and it is formally denoted as $Sw(\varphi, u, w)$. Given two vertices $u, w \in V_G$, such that $\varphi(u) = v$ and $\varphi(w) = z$ (with $v, z \in V_H$) the *swap* move consists in exchanging the assignation of vertices u and w (i.e., $\varphi(u) = z$ and $\varphi(w) = v$).

Again, let us illustrate the *swap* move with an example. In Fig. 4, we show the embedding φ_1 of the input graph depicted in Fig. 1(a) and the embedding φ_3 of the same input graph, obtained after the *swap* operation $Sw(\varphi_1, A, E)$. In φ_1, vertices A and E of V_G are assigned to vertices 2 and 4 of V_H, respectively. Then, in φ_3, vertices A and E exchange their assigned host vertices, resulting in $\varphi_3(A) = 4$ and $\varphi_3(E) = 2$.

Considering the aforesaid *swap* move, we define the associated neighborhood (N_S) as the set of solutions that can be reached by applying the *swap* operator for each $u, w \in V_G$. More formally:

$$N_S(\varphi) = \{Sw(\varphi, u, w) : \forall\, u, w \in V_G, u \neq w\}. \tag{9}$$

3.2.2. Shake procedure

The shake procedure is aimed to perform stochastic moves in a particular neighborhood despite the quality of the solution obtained after the move. The rationale behind this procedure is to let the BVNS to explore other regions of interest further than the current basin of attraction. Notice that the shake procedure (further than the first iteration) is usually run after the local search procedure, which ends in a local optimum with respect to a particular neighborhood.

Particularly, we propose a simple shake procedure based on the swap neighborhood ($N_S(\varphi)$) detailed in Eq. (9). In this case, at each iteration, the method performs k swaps of two random vertices $u, w \in V_G$. Notice that the value of k is updated at each iteration in the `NeighborhoodChange` procedure of Algorithm 3. Particularly, it is increased in one unit every non-improving iteration until k_{max} value is reached. On the other hand, the value of k is reset to 1 every improving iteration.

3.2.3. Local search procedure

The local search strategy systematically explores a particular neighborhood, starting from a given feasible solution and performing predefined moves. Its objective is to traverse the neighborhood structure searching for better solutions than the current one. Once a better solution has been identified in the neighborhood of the current solution, the procedure has to determine if the search should continue from this new solution (following a first improvement strategy) or should explore the whole neighborhood of the current solution, before deciding which solution should be the next one for continuing the search (following a best

improvement strategy). We refer the reader to [32] for further details about these two strategies. The process continues until the local search finds a local optimum (i.e., none of the solutions in the neighborhood of the best solution found improves it).

The local search procedure proposed in this paper explores the insertion neighborhood (N_I) previously introduced. We summarize the steps of this procedure in Algorithm 4. The method receives an input graph and a feasible solution φ as the input parameters. Then, it combines the first improvement and best improvement strategies to explore N_I using the *insert* move. Since the vertices of the solution are randomly scanned one by one (see step 7) the first improvement strategy is related to performing the *insert* move of the first vertex able to improve the current solution. On the other hand, for each explored vertex v, the best improvement strategy is referred to only considering the best *insert* move for v, among all possible insertions (see step 8) in the current iteration of the algorithm. The procedure stops when no further improvements can be performed by applying any *insert* move to the vertices of the current solution. Therefore, the returned solution φ^* is a local minimum with respect to N_I.

Algorithm 4 Local search procedure

1: **Procedure** `LocalSearch` (G, φ)
2: $\varphi^* \leftarrow \varphi$
3: *improved* $\leftarrow$ *true*
4: **while** *improved* **do**
5: *improved* $\leftarrow$ *false*
6: V $\leftarrow$ `shuffle`(V_G)
7: **for** $u \in$ V **do**
8: $\varphi \leftarrow \underset{v \in V_H}{\operatorname{argmin}}\, Ins(\varphi^*, u, v)$
9: **if** $cbs(G, \varphi) < cbs(G, \varphi^\star)$ **then**
10: $\varphi^\star \leftarrow \varphi$
11: *improved* $\leftarrow$ *true*
12: `break`
13: **end if**
14: **end for**
15: **end while**
16: **return** φ^*

The size of this neighborhood is $n \cdot (n-1)$, being n the number of vertices of the input graph (which is also the number of vertices of the host graph). Hence, just considering the computational complexity of evaluating a solution ($O(|E_G|)$, using Eq. (3) as the evaluation function), a straightforward implementation of a scheme that entirely explores this neighborhood (such as the best improvement strategy) would result in an algorithm with a computational complexity of at least $O(n^2|E_G|)$ (only considering the evaluation of all solutions in the neighborhood). Therefore, in those cases where the number of vertices and edges of the input graph are relatively large, it would require a significant computational time to explore the whole neighborhood. To reduce the local search execution time, in the following section, we propose two advanced strategies to speed up this procedure considerably.

3.3. Advanced strategies

Despite the benefits of using local search procedures, in terms of quality of the solutions found, the exploration of a whole neighborhood is usually the most computationally demanding component of a heuristic search procedure.

Inspired by the ideas presented in [3,33–35], we propose two strategies to perform an efficient exploration of the neighborhood N_I. The first strategy aims to transform a movement based on the *insert* move in a sequence of consecutive *swap* moves of vertices assigned to adjacent vertices in the host graph. The second strategy introduces a procedure for efficiently calculating the objective function when a solution is obtained from a previous solution (already evaluated) and the insertion is performed as a sequence of *swap* moves.

3.3.1. An insert move through a chain of swap moves

Let us define the operation denoted as *consecutive swap* (*CS*) as a particular case of the *swap* move, where the input vertices involved in the *swap* are embedded into two adjacent vertices of the host graph. More formally:

$$CS(\varphi, u, w) = \{Sw(\varphi, u, w) : u, w \in V_G \wedge (\varphi(u), \varphi(w)) \in E_H\}. \quad (10)$$

Notice that since the graph is a cycle, for every input vertex embedded there are only two possible *CS*. Then, any *insert* move can be performed with a chain of *consecutive swap* moves. Since there are two possible chains of *CS* to perform the *insert* move, to simplify the notation, we always consider the sequence of *CS* that follows a clockwise order, in the graphical representation. Let us illustrate it with an example. In Fig. 5, we show the steps followed to perform the *insert* move $Ins(\varphi_1, A, 4)$ by a chain of *consecutive swap* moves. Particularly, the operation $Ins(\varphi_1, A, 4)$ is equivalent to the chain of *consecutive swap* moves $\{\varphi'_1 \leftarrow CS(\varphi_1, A, D), \varphi_2 \leftarrow CS(\varphi'_1, A, E)\}$. Therefore, in this case, the *insert* operation can be performed with two *consecutive swaps* and, as we can observe, the obtained solution after the chain of *consecutive swaps* is the same as φ_2, represented in Fig. 3, which was obtained after applying the $Ins(\varphi_1, A, 4)$ move.

The rationale behind using a chain of consecutive swaps instead of an *insert* move introduces two main advantages. First, an efficient calculation of the objective function value of the resultant solution after a consecutive swap. Second, the exploration of the intermediate solutions of the chain of consecutive swaps, when traversing an insert neighborhood, which eases the use of a best improvement strategy.

3.3.2. Efficient evaluation of a solution after a consecutive swap move

A local search procedure typically evaluates a solution after performing a move, to determine if the search direction is adequate to continue the search or not. A naive evaluation would recalculate the value of the objective function from scratch. However, an efficient evaluation can be addressed for certain moves, based on the evaluation of the solution prior to the move. The idea is to isolate the components used in the evaluation of the objective function and to determine which of them have not changed after the move, avoiding its revaluation.

Particularly, based on Eq. (3), the evaluation of a solution for the CBSP is obtained as the sum of the bandwidth bw of each edge of the input graph. Therefore, the solution obtained after a move can be evaluated just by updating the bandwidth of the edges affected by the move.

In this case, when considering a *consecutive swap* move, $CS(\varphi, u, w)$, only the bandwidth bw of the edges with endpoints in u or w are affected. Let E_G^u (analogously E_G^w) be the subset of edges from the input graph with an endpoint in u (i.e., $E_G^u = \{(u, u') \in E_G, \forall\ u' \in V_G\}$). Then, the objective function of a solution $\varphi' \leftarrow CS(\varphi, u, w)$, obtained after the move, can be efficiently calculated as follows:

$$cbs(G, \varphi') = cbs(G, \varphi) + \sum_{e \in \{E_G^u \cup E_G^w\}} (bw(e, \varphi') - bw(e, \varphi)) \quad (11)$$

Observing the computational complexity of Eq. (11), which is $\mathcal{O}(n)$, we noticed that it could be further improved to be $\mathcal{O}(1)$ by storing and updating the relative position in the embedding of the adjacent vertices to any input vertex.

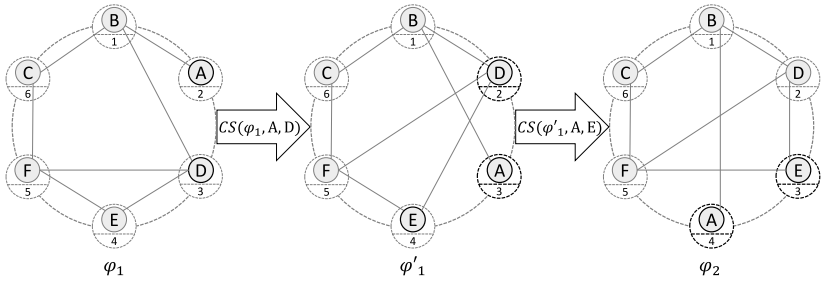


Fig. 5. Example of an embedding φ_1 and the resultant embeddings φ'_1 and φ_2 obtained after the operations $CS(\varphi_1, A, D)$ and $CS(\varphi'_1, A, E)$.

Particularly, for each input vertex, we are interested in identifying the host vertex placed in the antipodal point of the graphical representation of the host graph. Given an input vertex u, let us denote $\overline{\varphi(u)}$ as the host vertex placed at the antipodal point to $\varphi(u)$ in the host graph. More formally,

$$\overline{\varphi(u)} \leftarrow \underset{v \in V_H}{\arg} \left(|p(\varphi(u), v)| = \left\lceil \frac{|V_G|}{2} \right\rceil \right). \tag{12}$$

The host vertex $\overline{\varphi(u)}$ is used to determine the sets of input vertices L^u_G, R^u_G and O^u_G. These sets contain the input vertices adjacent to u, which are embedded at the left, right, and antipodal point host vertices, respectively, in the graphical representation of a solution. These sets are used to perform the efficient evaluation and they are mathematically defined as follows:

$$L^u_G = \{w \in V_G : \varphi(w) \in p(\varphi(u), \overline{\varphi(u)}) \wedge (u, w) \in E_G \wedge \varphi(w) \neq \overline{\varphi(u)}\}. \tag{13}$$

$$R^u_G = \begin{cases} \{w \in V_G : \ \varphi(w) \notin p(\varphi(u), \overline{\varphi(u)}) \wedge \\ (u, w) \in E_G \wedge \varphi(w) \neq \overline{\varphi(u)}\}, & \text{if } |V_G| \textit{ is even}; \\ \{w \in V_G : \varphi(w) \notin p(\varphi(u), \overline{\varphi(u)}) \wedge (u, w) \in E_G\}, & \text{otherwise.} \end{cases} \tag{14}$$

$$O^u_G = \begin{cases} \{w \in V_G : (u, w) \in E_G \wedge \varphi(w) = \overline{\varphi(u)}\}, & \text{if } |V_G| \text{ is even;} \\ \emptyset, & \text{otherwise.} \end{cases} \tag{15}$$

Notice that $p(\varphi(u), \overline{\varphi(u)}) \neq p(\overline{\varphi(u)}, \varphi(u))$, and the path starting at $\varphi(u)$ and ending at $\overline{\varphi(u)}$ considers a clockwise order in the graphical representation of the solution. Moreover, given the set of adjacent vertices to u, denoted as V^u_G, these sets satisfy that $L^u_G \cup R^u_G \cup O^u_G = V^u_G$. It is worth mentioning that each time a move is performed, it is necessary to update these sets. However, this update, despite the first time, can be made in $\mathcal{O}(1)$.

Then, with the previous information at hand, we can simplify the update of the objective function after a *consecutive swap* move. In particular, we calculate the increase/decrease of the objective function, denoted as $\Delta CS(\varphi, u, w)$, as follows:

$$\Delta CS(\varphi, u, w) = |R^u_G| - |L^u_G \setminus \{w\}| - |O^u_G| + |L^w_G| - |R^w_G \setminus \{u\}| - |O^w_G|, \tag{16}$$

with $w \in \{p(\varphi(u), \overline{\varphi(u)})\}$ and $u \notin \{p(\varphi(w), \overline{\varphi(w)})\}$.

Therefore, the evaluation of the objective function of the solution $\varphi' \leftarrow CS(\varphi, u, w)$, introduced in Eq. (11), can be simplified as follows:

$$cbs(G, \varphi') = cbs(G, \varphi) + \Delta CS(\varphi, u, w). \tag{17}$$

Let us illustrate this strategy with an example. Considering the solution φ_1 (previously depicted in Figs. 3, 4, and 5) we show how to efficiently calculate the objective function of the solution $\varphi' \leftarrow Ins(\varphi_1, A, 1)$. In particular, $Ins(\varphi_1, A, 1) = \{CS(\varphi_1, A, D), CS(\varphi'_1, A, E), CS(\varphi'_2, A, F), CS(\varphi'_3, A, C), CS(\varphi_{4'}, A, B)\}$.

Then, in Table 1 we show the evaluation of each *consecutive swap*, obtained through the operation $Ins(\varphi_1, A, 1)$. For each row, we present: the cardinality of the sets L^u_G, R^u_G, O^u_G, L^w_G, R^w_G and O^w_G, before the CS is performed; the increase in the objective function, ΔCS, after the CS is performed; and the resulting value of the objective function, $cbs(G, \varphi'_i)$, after the CS is performed. Focusing on the first row of Table 1, given the solution φ_1, the $cbs(G, \varphi_1) = 9$, we perform the operation $CS(\varphi_1, A, D)$. The resultant solution, φ'_1, is evaluated by aggregating $\Delta CS(\varphi_1, u, w)$ to the objective function value. In this case, $cbs(G, \varphi'_1) = cbs(G, \varphi_1) + \Delta CS(\varphi_1, u, w) = 9 + (1 + 2 - 1) = 9 + 2 = 11$. Similarly, given φ'_1, we evaluate the solution φ'_2, etc.

As a final remark, it is worth mentioning that when performing the move $Ins(\varphi_1, A, 1)$ by means of swaps, all solutions are evaluated in the path from φ_1 to φ' are evaluated. Therefore, we can take advantage of this technique to exhaustively explore a neighborhood (i.e., to follow an efficient best improving strategy for each vertex) in the local search procedure.

4. Experimental results

In this section, we compile the computational experiments carried out in this research. First, we present the sets of instances used in our experiments, then we introduce the preliminary experiments used to tune our MS-BVNS proposal. Finally, we compare our best strategy with the best method identified in the state of the art of the CBSP (DMAB+MA [22]).

All experiments were performed in a virtual CPU AMD EPYC 7282 16-Core and 16 GB of RAM. The operating system used was Ubuntu 20.04.2 64 bit LTS, and the algorithms were implemented in Java 13.

4.1. Instances

The computational tests have been performed over 40 instances previously reported in the literature in the context of the CBSP [19,22]. This set is conformed by topologically diverse graphs that have been often used in similar graph layout problems [1,4]. Particularly, this set includes 16 regular-structured graphs: 2 paths, 2 cycles, 2 wheels, 4 kth powers of a cycle, and 6 Cartesian products of graphs; and 24 sparse matrices from the Harwell-Boeing collection [36].

Since the set of instances can be considered small and the methods compared were able to find the optimal solution for many of them, in addition to the previous data set, we extended

Table 1
Evaluation of the objective function of the solutions obtained by means of *consecutive swap* moves, needed to achieve operation $Ins(\varphi, A, 1)$.

	$\lvert L_G^u\rvert$	$\lvert R_G^u\rvert$	$\lvert O_G^u\rvert$	$\lvert L_G^w\rvert$	$\lvert R_G^w\rvert$	$\lvert O_G^w\rvert$	ΔCS	$cbs(G, \varphi_i')$
$\varphi_1' \leftarrow CS(\varphi_1, A, D)$	0	1	0	2	1	0	2	$9+2=11$
$\varphi_2' \leftarrow CS(\varphi_1', A, E)$	0	1	0	1	1	0	1	$11+1=12$
$\varphi_3' \leftarrow CS(\varphi_2', A, F)$	0	0	1	1	1	1	−2	$12-2=10$
$\varphi_4' \leftarrow CS(\varphi_3', A, C)$	1	0	0	1	1	0	−1	$10-1=9$
$\varphi' \leftarrow CS(\varphi_4', A, B)$	1	0	0	1	2	0	0	$9-0=9$

the previous instances with 66 additional instances derived from the Harwell-Boeing collection [36] for a better understanding of the performance of the methods tested.

To ease future comparisons, all instances have been made publicly available at https://heuristicas.es/problems/cyclic-bandwidth-sum.

4.2. Preliminary testing

The goal of our preliminary testing is to study the performance of the strategies proposed in this paper for the CBSP. In addition, we tried to identify its contribution to the final algorithmic design, which is compared to the best previous state-of-the-art method. Finally, we used the preliminary experiments to find the most effective configuration of the parameters within the MS-BVNS.

All preliminary experiments have been carried out on a reduced subset of instances consisting of 18 graphs (approximately the 17% of all problem instances considered in this paper). Hereinafter, we refer to this subset as the preliminary data set.

To compare the performance of the algorithms proposed, in each experiment, we report the averaged value of the objective function (Avg. *cbs*) for all instances tested, the deviation from the best solution found in the experiment (Dev. (%)), the number of best solutions found in the experiment (#Best) and the running time in seconds (CPU Time (s)).

To start the preliminary experiments, we study the performance of the constructive procedure described in Section 3.1. Particularly, the constructive procedure uses two greedy functions (g_1 and g_2) devoted to select, respectively, the next input vertex to embed, and the associated host vertex for the embedding. This constructive was inspired by a naive version (proposed in [25]) that only considered a greedy function to select the next input vertex. Therefore, we study the influence of the two new strategies proposed in this paper. In Table 2, we start by comparing the impact on the performance of introducing the greedy selection of the host vertex. Despite the fact that the proposed constructive is a greedy algorithm, the ties are randomly broken. Therefore, in Table 2 we report the best result found after performing 1000 executions of our greedy constructive procedure. Furthermore, the parameter m (studied next) has been set to 1 to isolate its influence in the experiment. As we can observe, the introduction of g_2 in the constructive procedure (denoted as $C_{g_1+g_2}$) considerably improves the results obtained by the original constructive method, finding 14 best solutions out of 18 and a deviation of 0.36% when comparing the two procedures.

The next experiment is devoted to test the influence of the value of the parameter m, introduced in this paper, in the greedy selection of the next input vertex to embed. In particular, in Table 3, we report the results obtained after 1000 constructions when fixing the value of this parameter ($m = \{0, 1, 2, 4, 6, 9\}$) and also the results of choosing a random value for each construction ($m \in U(0, 9)$). As we can observe, the original configuration of the procedure (equivalent to $m = 1$) is outperformed in terms of deviation by any other value larger than 1. The reason for this behavior is that when $m > 1$ the procedure favors those unassigned vertices with a larger number of adjacent vertices already embedded. However, since we could not find a clear pattern of the best fixed configuration for this parameter, we tested the random selection of $m \in U(0, 9)$ for each construction, which turned out to be the best overall method (the smaller deviation, the larger number of best solutions found and a similar CPU time).

Table 2
Influence of the greedy selection g_2 of the host vertex, in the performance of the constructive procedure.

	Cg_1 [25]	$Cg_1 + g_2$
Avg. *cbs*	18116.61	16838.72
Dev. (%)	5.93	0.36
# Best	4	14
CPU T. (s)	1.35	1.46

To complement the previous experiments, in Fig. 6 we report the influence of the number of independent executions on the performance of the proposed greedy constructive procedure. In particular, we have reported three of the previous configurations of the constructive method. As we can observe in the figure, the performance of the procedure drastically improves in the first 100 executions for any of the variants. We also observe a moderate improvement for up to 1000 executions and a marginal improvement for a larger number of executions. However, the behavior of the procedure is very similar with the independence of the value of m chosen.

Since an increase in the number of executions also implies an increase in the CPU time, we chose 1000 executions (made in 1.38 s on average) as a balance between quality and computing time to configure our multistart procedure.

Our next preliminary experiment is devoted to testing the performance of the advanced strategies proposed to explore the neighborhood structure (N_I) defined for the local search procedure. In Table 4, we report the comparison between the naive exploration of N_I (denoted as LS − N_I) and the efficient exploration of the neighborhood based on *consecutive swap* moves (denoted as LS − N_{CS}). Both methods started from the same random construction. As expected, both strategies were able to reach the same solutions in terms of quality; however, the naive exploration needed an average of 732.26 s to reach them, while the advanced strategy was able to explore the neighborhood in 2.81 s. Note that this time corresponds only to one construction + one improvement.

The aim of the next experiment is to evaluate the influence of the k_{max} parameter in the proposed BVNS procedure. Again, we started from a random solution and we configured the local search procedure with the advanced strategies included. Particularly, we considered six different values for k_{max} (5, 10, 15, 20, 25, and 30). The results obtained were very similar for $k_{max} \geq 20$, being $k_{max} = 30$ the best configuration, but also the most time consuming. Since the differences between the methods were very small for $k_{max} \geq 20$, and $k_{max} = 20$ was the fastest, we chose this value as a balance between the quality of the solution reached and the CPU time expended.

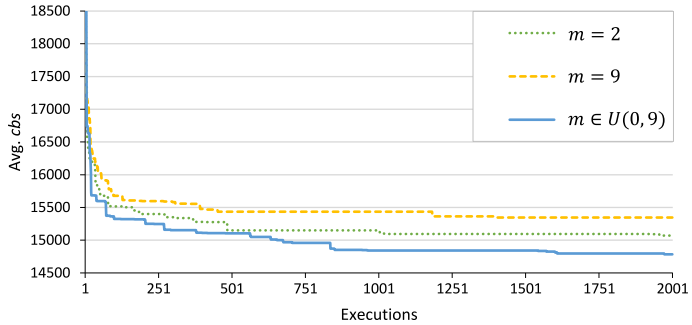


Fig. 6. Evolution of the average objective function value (*cbs*) when increasing the number of independent executions of the proposed greedy constructive procedure.

Table 3
Influence of the value of the parameter m in the performance of the constructive procedure.

	$m = 0$	$m = 1$	$m = 2$	$m = 4$	$m = 6$	$m = 9$	$m \in$ U(0, 9)
Avg. *cbs*	30225.83	16838.72	15151.00	15135.39	14933.78	15435.17	14842.22
Dev. (%)	132.36	25.50	3.77	2.77	2.85	3.79	0.61
# Best	1	6	1	0	2	0	9
CPU T. (s)	1.63	1.46	1.41	1.36	1.52	1.44	1.30

Table 4
Differences in the performance between the naive exploration of the N_l neighborhood and the efficient exploration based on *consecutive swap* moves.

	LS − N_l	LS − N_{CS}
Avg. *cbs*	20340.78	20340.78
Dev. (%)	0.00	0.00
# Best	18	18
CPU T. (s)	732.26	2.81

As a final experiment, we evaluated the contribution of each of the components included in the final algorithmic proposal. In particular, in Table 5, we compare the tuned greedy constructive (*Greedy*), the efficient local search (LS − N_{CS}) and the tuned BVNS procedure (BVNS). The constructive was configured with $m \in$ U(0, 9) and the use of the second greedy function g_2. The method was run for 1000 constructions and the best solution was reported. The local search procedure started from the best solution found by the constructive and uses the advanced exploration strategy based on the *consecutive swap* moves. Notice that the quality of the initial solution provided to LS − N_{CS} (Table 5) is better than the random initial solution provided to LS − N_{CS} in the previous experiment (Table 4). As we can observe, starting the search from a solution obtained with a greedy constructive allows the search to reach much better quality solutions than starting from the solution provided by a random constructive procedure. Finally, the BVNS starts from the same best solution found by the constructive procedure and is configured with $k_{max} = 20$.

As expected, the BVNS is the best method of the comparison, but also the most time-consuming one. Although the difference between LS − N_{CS} and BVNS is significantly smaller than the quality difference of the solution between *Greedy* and LS − N_{CS}, the BVNS can find 14 better solutions than LS − N_{CS}, resulting in a robust and effective method of dealing with CBSP. Moreover, we conclude that all components included in the final configuration

Table 5
Contribution of each component included in the final algorithmic design.

	Greedy $m \in$ U(0, 9)	LS − N_{CS}	BVNS
Avg. *cbs*	14842.22	13324.50	12627.67
Dev. (%)	9.39	2.53	0.00
# Best	0	4	18
CPU T. (s)	1.30	3.33	118.78

of our proposal contribute to the overall performance of the method.

4.3. Comparison with the state of the art

Finally, we compare our tuned procedure with the best previous algorithm in the state of the art, the Dynamic Multi Armed Bandit + Memetic Algorithm (DMAB+MA) [22], that was described in the literature review of this paper. Notice that we have compared our procedure with the original source code implemented by the authors in [22]. To perform the fairest possible comparison, the DMAB+MA procedure was run with the configuration indicated by the authors, and the reported quality values are the best ones obtained after 31 runs. On the other hand, our multistart procedure, denoted as MS-BVNS, has been executed with the configuration established in Section 4.2. Notice that the whole procedure is executed only once per instance and the termination criterion is self-adaptive, being set to 50 consecutive iterations without improvement. To complement this information, we observed that our method stopped, on average, after 56.46 iterations in the experiments reported in this section.

First, we report, in Table 6, the results obtained after executing both procedures over the set of instances previously reported in the state of the art. In this comparison, we reported: the average value of the objective function (Avg. *cbs*); the deviation to the best solution in the experiment (Dev. (%)); the number of best solutions found in the experiment (# Best); and two different running times, the total running time (CPU T. (s)) and the time

Table 6
Comparison of MS-BVNS with the best state-of-the-art method, DMAB+MA [22], over previously studied instances.

		DMAB+MA [22]	MS-BVNS
Regular structure (16)	Avg. *cbs*	2796.81	2796.81
	Dev. (%)	0.00	0.00
	# Best	16	16
	CPU T. (s)	18600.47	116.63
	CPU T. Best (s)	116.59	2.34
Harwell-Boeing (24)	Avg. *cbs*	12022.38	11227.63
	Dev. (%)	7.67	0.00
	# Best	12	24
	CPU T. (s)	18600.43	4834.71
	CPU T Best (s)	7089.77	1220.30
Total (40)	Avg. *cbs*	8332.15	7855.30
	Dev. (%)	4.60	0.00
	# Best	28	40
	CPU T. (s)	18600.45	2947.48
	CPU T. Best (s)	4300.50	733.12

to the best solution found (CPU T. Best (s)). The results in Table 6 are reported separately by graph type (i.e, regular-structured graphs, or Harwell-Boeing graphs) and also grouped altogether (Total).

Based on the results reported in Table 6 we observe that both compared methods, DMAB+MA and MS-BVNS, were able to find all optimal values for graphs with regular structure. However, the time needed by MS-BVNS was two orders of magnitude smaller than that of DMAB+MA in this set of instances. On the other hand, the performance of MS-BVNS is better than that of DMAB+MA on the Harwell-Boeing instances. Particularly, the MS-BVNS algorithm finds the best solutions for all studied graphs in less computational time. Finally, when considering all instances together, the best overall method is MS-BVNS based on the deviation and the number of best solutions found. Particularly, MS-BVNS has a deviation of 0.00%, while the deviation of DMAB+MA reaches the 4.61%. It is also worth mentioning that MS-BVNS is able to find the best solution for all instances tested. As a final remark, we would like to point out that both the total CPU time and the time to the best solution found expended by MS-BVNS is considerably smaller than that used by DMAB+MA.

To complement the previous experiment, we conducted a Wilcoxon Signed-Rank Test. The resulting p-value of 0.002 confirms the better performance of MS-BVNS over DMAB+MA for the tested instances.

Although the statistical tests performed justify the existence of significant differences between the methods compared, to corroborate our findings, we extended the data set of instances used by introducing 66 new hard instances belonging to the Harwell-Boeing collection [36]. The results of both methods, run with the same setup as in the previous experiment, are presented in Table 7. This time, MS-BVNS found the best solutions in 60 out of 66 instances, while DMAB+MA was only able to reach the best solution in 22 out of 66 graphs. The computing time needed by MS-BVNS was again considerably smaller (in both CPU T. (s) and CPU T. Best (s)). Finally, the obtained average deviation of the MS-BVNS with respect to the best solution found in the experiment was 0.17%, while DMAB+MA had a deviation of 7.20%.

Again, we used Wilcoxon's signed rank test to determine if the identified differences between the objective function values of the best solutions found by DMAB+MA and MS-BVNS are significant. The resultant p-value < 0.001 indicates a strong rejection of the null hypothesis with a reasonable level of significance.

To ease future comparisons, in the appendix of this paper we include the best individual results per instance found by the compared methods on the Harwell-Boeing set of instances.

Table 7
Comparison of MS-BVNS with the best state-of-the-art method, DMAB+MA [22], over the new data set of Harwell-Boeing instances.

Harwell-Boeing (66)	DMAB+MA [22]	MS-BVNS
Avg. cbs	91649.53	90653.11
Dev. (%)	7.08	0.17
# Best	22	60
CPU T. (s)	18600.54	5294.63
CPU T. Best (s)	7061.36	1941.50

5. Conclusions

In this paper, we have studied the Cyclic Bandwidth Sum Problem by proposing heuristic strategies to find high-quality solutions for the problem. The CBSP consists in finding an embedding of an input graph into a cycle (denoted as host graph) which minimizes the sum of the bandwidth of each edge of the input graph. This problem has been previously studied from an exact perspective for some types of graphs with regular structure. Additionally, previous heuristic procedures have been proposed for general random graphs.

In this research, we introduced a new algorithm to address the CBSP. The CBSP and other related Graph Layout Problems, have been commonly addressed with traditional optimization procedures such as trajectory-based metaheuristics. However, other novel computational intelligence algorithms such as the ones introduced in [37–39] could be also applied to solve this family of problems. Furthermore, we hope this paper motivates the interest of researchers in the field to propose those and other computational intelligence approaches for the problem. Our proposal has two key features that make it an effective and efficient method for finding high-quality solutions for the CBSP: a new greedy constructive procedure and an efficient local search method. The constructive procedure proposed introduces two novel strategies for the construction of solutions for graph layout problems. Particularly, it makes use of two greedy criteria to determine the next vertex to be embedded in the solution and to determine the most suitable host vertex for that particular input vertex. This strategy can be useful for other related problems, especially those where the host graph is a cycle. The exploration strategy of the local search also introduces a novel contribution, the efficient exploration of the neighborhood defined by the insertion move, through the use of sets of consecutive swap moves. This idea lets the method to drastically reduce the complexity of the exploration, since the evaluation of a new solution obtained from a previous one can be efficiently evaluated. Again, this strategy has general applicability and could be adapted to accelerate other local search procedures in different contexts.

Finally, it is worth mentioning that our best algorithmic variant has been favorably compared to the best previous method in the state of the art, over both previously studied and newly introduced sets of instances. The results of these comparisons have been supported by statistical tests that corroborate the merit of our proposal and establish it as the new state-of-the-art method for the CBSP.

CRediT authorship contribution statement

Sergio Cavero: Conceptualization, Investigation, Data curation, Writing– original draft, Software. **Eduardo G. Pardo:** Conceptualization, Methodology, Investigation, Validation, Writing – original draft. **Abraham Duarte:** Supervision, Writing – review & editing, Funding acquisition. **Eduardo Rodriguez-Tello:** Supervision, Formal analysis, Writing – Review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research has been partially supported by the Ministerio de Ciencia, Innovación *y* Universidades (Grant Ref. PGC2018-095322-B-C22, PID2021-125709OA-C22, and FPU19/04098) and by Comunidad de Madrid and European Regional Development Fund (Grant Ref. P2018/TCS-4566). We would also like to thank E. Rodriguez-Tello et al., authors of the best previous method in the state of the art [22] for sharing their code and their findings with us.

Appendix

See Table A.1.

Table A.1
Individual results per instance obtained by the compared methods over the Harwell-Boeing data set [36].

Instance	DMAB+MA	MS+BVNS	Instance	DMAB+MA	MS+BVNS
494_bus	5405	**4306**	ibm32	**405**	**405**
662_bus	11822	**7930**	impcol_a	5570	**5566**
685_bus	14576	**8851**	impcol_b	**1822**	**1822**
arc130	**14397**	**14397**	impcol_c	3351	**3350**
ash292	6479	**6399**	impcol_d	12192	**12147**
ash85	**913**	**913**	impcol_e	15510	**15489**
bcspwr01	**98**	**98**	jgl011	**141**	**141**
bcspwr02	**148**	**148**	lns__131	**2053**	2067
bcspwr03	663	**662**	lns__511	20197	**19961**
bcspwr04	**4199**	4256	lund_a	**10165**	**10165**
bcspwr05	5082	**4395**	lund_b	**10160**	**10160**
bcsstk01	**936**	**936**	mbeacxc	4099540	**4085555**
bcsstk02	**35937**	**35937**	mcca	**23018**	**23018**
bcsstk04	**29812**	**29812**	nnc261	**8539**	9170
bcsstk05	**11059**	**11059**	nnc666	38620	**37773**
bcsstk06	52096	**51686**	nos1	**624**	**624**
bcsstk20	9531	**4933**	nos2	7688	**2544**
bcsstk22	**811**	**811**	nos4	**1031**	**1031**
bcsstm07	47653	**47176**	nos5	55902	**55649**
can__144	**1776**	**1776**	nos6	15042	**11225**
can__161	**4998**	**4998**	plat362	33789	**33661**
can__292	15117	**15102**	plskz362	6595	**6467**
can__445	26873	**26619**	pores_1	**349**	**349**
can__715	63070	**60314**	pores_3	12434	**11425**
can___24	**182**	**182**	saylr1	2574	**2545**
curtis54	**411**	**411**	saylr3	20494	**18298**
dwt__209	6383	**6350**	sherman4	15970	**14988**
dwt__221	3782	**3774**	shl__200	122410	**119899**
dwt__234	**957**	958	shl__400	125027	**120149**
dwt__245	3899	**3880**	shl____0	118780	**114036**
dwt__310	6455	**6454**	steam1	**23486**	**23486**
dwt__361	12085	**12061**	steam2	158742	**158236**
dwt__419	15335	**14422**	steam3	**1416**	**1416**
dwt__503	37754	**37404**	str__200	**94750**	95772
dwt__592	25729	**25019**	str__600	105541	**105048**
fs_183_1	16779	**16777**	str____0	64232	**63801**
fs_541_1	98614	**88735**	west0132	**4769**	4784
fs_680_1	13995	**7837**	west0156	4605	**4604**
gent113	**5605**	**5605**	west0167	5518	**5517**
gre_216a	7318	**7314**	west0381	100755	**100244**
gre__115	**2130**	**2130**	west0479	71423	**70109**
gre__185	**5153**	**5153**	west0497	47262	**46527**
gre__343	16192	**16148**	west0655	145605	**143167**
gre__512	32433	**31568**	will199	13708	**13705**
hor__131	30625	**30441**	will57	**335**	**335**

References

[1] E.G. Pardo, R. Martí, A. Duarte, Linear layout problems, in: R. Martí, P. Panos, M. Resende (Eds.), Handbook of Heuristics, Springer International Publishing, Cham, 2016, pp. 1–25.
[2] R. Martí, J. Pantrigo, A. Duarte, E. Pardo, Branch and bound for the cutwidth minimization problem, Comput. Oper. Res. 40 (1) (2013) 137–149.
[3] E. Rodriguez-Tello, J.-K. Hao, J. Torres-Jimenez, An improved simulated annealing algorithm for bandwidth minimization, European J. Oper. Res. 185 (3) (2008) 1319–1335.
[4] S. Cavero, E.G. Pardo, M. Laguna, A. Duarte, Multistart search for the cyclic cutwidth minimization problem, Comput. Oper. Res. 126 (2021) 105–116.
[5] E. Rodriguez-Tello, F. Lardeux, A. Duarte, V. Narvaez-Teran, Alternative evaluation functions for the cyclic bandwidth sum problem, European J. Oper. Res. 273 (3) (2019) 904–919.
[6] S. Cavero, E.G. Pardo, A. Duarte, A general variable neighborhood search for the cyclic antibandwidth problem, Comput. Optim. Appl. (2022) 1–31.
[7] G. Ding, B. Oporowski, Some results on tree decomposition of graphs, J. Graph Theory 20 (4) (1995) 481–499.
[8] T.C. Hu, Optimum communication spanning trees, SIAM J. Comput. 3 (3) (1974) 188–195.
[9] M.A. Rodríguez-García, J. Sánchez-Oro, E. Rodriguez-Tello, E. Monfroy, A. Duarte, Two-dimensional bandwidth minimization problem: Exact and heuristic approaches, Knowl.-Based Syst. 214 (2021) 106651.
[10] J. Yuan, Cyclic arrangement of graphs, in: Graph Theory Notes of New York, New York Academy of Sciences, New York, NY, USA, 1995, pp. 6–10.
[11] Y.-D. Chen, J.-H. Yan, A study on cyclic bandwidth sum, J. Comb. Optim. 14 (2) (2007) 295–308.
[12] H. Jianxiu, Cyclic bandwidth sum of graphs, Appl. Math. A J. Chin. Univ. 16 (2) (2001) 115–121.
[13] A. Duarte, R. Martí, M.G.C. Resende, R.M.A. Silva, GRASP with path relinking heuristics for the antibandwidth problem, Networks 58 (3) (2011) 171–189.
[14] E.G. Pardo, N. Mladenović, J.J. Pantrigo, A. Duarte, Variable formulation search for the cutwidth minimization problem, Appl. Soft Comput. 13 (5) (2013) 2242–2252.
[15] A. Lim, J. Lin, F. Xiao, Particle swarm optimization and hill climbing for the bandwidth minimization problem, Appl. Intell. 26 (3) (2007) 175–182.
[16] A. Lim, J. Lin, B. Rodrigues, F. Xiao, Ant colony optimization with hill climbing for the bandwidth minimization problem, Appl. Soft Comput. 6 (2) (2006) 180–188.
[17] M. Lozano, A. Duarte, F. Gortázar, R. Martí, A hybrid metaheuristic for the cyclic antibandwidth problem, Knowl.-Based Syst. 54 (2013) 103–113.
[18] D. Satsangi, K. Srivastava, Gursaran, General variable neighbourhood search for cyclic bandwidth sum minimization problem, in: 2012 Students Conference on Engineering and Systems, 2012, pp. 1–6.
[19] R. Hamon, P. Borgnat, P. Flandrin, C. Robardet, Relabelling vertices according to the network structure by minimizing the cyclic bandwidth sum, J. Complex Netw. 4 (4) (2016) 534–560.
[20] P. Jaccard, The distribution of the flora in the alpine zone. 1, New Phytol. 11 (2) (1912) 37–50.
[21] R. Tarjan, Depth-first search and linear graph algorithms, SIAM J. Comput. 1 (2) (1972) 146–160.
[22] E. Rodriguez-Tello, V. Narvaez-Teran, F. Lardeux, Dynamic multi-armed bandit algorithm for the cyclic bandwidth sum problem, IEEE Access 7 (2019) 40258–40270.
[23] E. Rodriguez-Tello, V. Narvaez-Teran, F. Lardeux, Comparative study of different memetic algorithm configurations for the cyclic bandwidth sum problem, in: A. Auger, C.M. Fonseca, N. Lourenço, P. Machado, L. Paquete, D. Whitley (Eds.), Parallel Problem Solving from Nature – PPSN XV, in: Lecture Notes in Computer Science, Springer International Publishing, Cham, 2018, pp. 82–94.
[24] S. Cavero, E.G. Pardo, A. Duarte, Influence of the alternative objective functions in the optimization of the cyclic cutwidth minimization problem, in: Conference of the Spanish Association for Artificial Intelligence, Springer, 2021, pp. 139–149.
[25] 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.
[26] E. Rodriguez-Tello, J.-K. Hao, J. Torres-Jimenez, An effective two-stage simulated annealing algorithm for the minimum linear arrangement problem, Part Special Issue: Search-based Software Engineering, Comput. Oper. Res. 35 (10) (2008) 3331–3346.
[27] J.J. Pantrigo, R. Martí, A. Duarte, E.G. Pardo, Scatter search for the cutwidth minimization problem, Ann. Oper. Res. 199 (1) (2012) 285–304.
[28] P. Hansen, N. Mladenović, Variable neighborhood search: Principles and applications, European J. Oper. Res. 130 (3) (2001) 449–467.
[29] P. Hansen, N. Mladenović, R. Todosijević, S. Hanafi, Variable neighborhood search: basics and variants, EURO J. Comput. Optim. 5 (3) (2017) 423–454.

[30] P. Hansen, N. Mladenović, Variable neighborhood search, in: Handbook of Heuristics, Springer International Publishing, Cham, 2018, pp. 759–787.
[31] A. Duarte, J.J. Pantrigo, E.G. Pardo, J. Sánchez-Oro, Parallel variable neighbourhood search strategies for the cutwidth minimization problem, IMA J. Manag. Math. 27 (1) (2016) 55–73.
[32] P. Hansen, N. Mladenović, First vs. best improvement: An empirical study, IV ALIO/EURO Workshop on Applied Combinatorial Optimization, Discrete Appl. Math. 154 (5) (2006) 802–817.
[33] A. Herrán, J.M. Colmenar, A. Duarte, An efficient metaheuristic for the K-page crossing number minimization problem, Knowl.-Based Syst. 207 (2020) 106–352.
[34] G. Palubeckis, Fast local search for single row facility layout, European J. Oper. Res. 246 (3) (2015) 800–814.
[35] M. Rubio-Sánchez, M. Gallego, F. Gortázar, A. Duarte, GRASP with path relinking for the single row facility layout problem, Knowl.-Based Syst. 106 (2016) 1–13.
[36] I.S. Duff, R.G. Grimes, J.G. Lewis, Sparse matrix test problems, ACM Trans. Math. Software 15 (1) (1989) 1–14.
[37] A.A. Heidari, S. Mirjalili, H. Faris, I. Aljarah, M. Mafarja, H. Chen, Harris hawks optimization: Algorithm and applications, Future Gener. Comput. Syst. 97 (2019) 849–872.
[38] G.-G. Wang, S. Deb, L.d.S. Coelho, Elephant herding optimization, in: 2015 3rd International Symposium on Computational and Business Intelligence, IEEE, 2015, pp. 1–5.
[39] G.-G. Wang, S. Deb, Z. Cui, Monarch butterfly optimization, Neural Comput. Appl. 31 (7) (2019) 1995–2014.

Chapter 10

Two-Dimensional Bandwidth Minimization Problem

The Two-Dimensional Bandwidth Minimization Problem is the last GLP studied in this Doctoral Thesis and it was previously introduced in Section 2.4. Moreover, the 2DBMP is the only problem studied in which the host graph is a grid instead of a cycle. As a result of the research conducted, one article has been published:

1. S. Cavero, E. G. Pardo, and A. Duarte. Efficient iterated greedy for the two-dimensional bandwidth minimization problem. *European Journal of Operational Research*, 306(3): 1126–1139, 2023 [32].

Moreover, a presentation has been made at a national conference:

2. A. Duarte, S. Cavero, and E. G. Pardo. Heurísticas aplicadas al 2D bandwidth problem. *XXXIX Congreso Nacional de Estadística e Investigación Operativa* (SEIO 2022), in Granada, Spain, 2021 [57].

The article, titled: "*Efficient Iterated Greedy for the Two-Dimensional Bandwidth Minimization Problem*" [32], was published in a JCR journal. Figure 10.1 compiles some information about the journal. The 2DBMP was previously approached from an exact perspective, based on CSP, and from a heuristic perspective, based on VNS. In order to further improve the results obtained by the two previous approaches, we developed an efficient

and effective IG algorithm. The main components of the algorithm proposed to address 2DBMP are summarized next:

- Three-phase **greedy constructive procedure**. The first phase initializes the solution by determining the first assignation of a vertex of the input graph to a vertex of the host graph. Then, the next vertex to be added to the solution is determined in the second phase. This phase is inspired by the weighted greedy criteria proposed in the previous work [33]. Then, in the third and last phase, a greedy function based on the objective function of the problem is used to locate the selected vertex in the host graph.

- **Local search** that explores the swap neighborhood following a first improvement strategy. Additionally, the improvement method incorporates again the three advanced strategies used for the CGLP: an efficient way to evaluate the objective function of neighbor solutions, a tiebreak criterion to deal with "flat landscapes", and a neighborhood reduction technique.

- **IG** is the metaheuristic used to combine the heuristic procedures proposed to tackle this problem. IG incorporates two main phases: the destruction and reconstruction phases, which have been implemented as a way to perturb the incumbent solution and explore other regions of the solution space.

Finally, the best algorithmic variant of our proposal has been compared to the best previous method in the state of the art, over a previously reported set of instances. The obtained results, supported by statistical tests, corroborate the merit of our proposal and establish it as a new state-of-the-art algorithm for the 2DBMP.

To conclude this chapter, we include a copy of the most relevant paper published for the 2DBMP in the context of this Doctoral Thesis.

Efficient IG for the two-dimensional bandwidth minimization problem

Sergio Cavero, Eduardo G. Pardo and Abraham Duarte

European Journal of Operational Research. Volume 306(1), 126-1139, 2023.

`https://doi.org/10.1016/j.ejor.2022.09.004`

Journal Information

Research Areas:

- Operations Research & Management Science

Category Rank:

- Operations Research & Management Science: 17/87 (Q1)

Journal Impact Factor: 6.363

Data obtained from Journal Citation Reports 2021

Figure 10.1 Information related to the publication [32].

European Journal of Operational Research 306 (2023) 1126–1139

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

Efficient iterated greedy for the two-dimensional bandwidth minimization problem

Sergio Cavero, Eduardo G. Pardo*, Abraham Duarte

Universidad Rey Juan Carlos, C/Tulipán, s/n, Mstoles, 28933, Madrid, Spain

ARTICLE INFO

Article history:
Received 21 January 2022
Accepted 5 September 2022
Available online 9 September 2022

Keywords:
Heuristics
Graph layout problem
Iterated greedy
Combinatorial optimization
Bandwidth

ABSTRACT

Graph layout problems are a family of combinatorial optimization problems that consist of finding an embedding of the vertices of an input graph into a host graph such that an objective function is optimized. Within this family of problems falls the so-called Two-Dimensional Bandwidth Minimization Problem (2DBMP). The 2DBMP aims to minimize the maximum distance between each pair of adjacent vertices of the input graph when it is embedded into a grid host graph. In this paper, we present an efficient heuristic algorithm based on the Iterated Greedy (IG) framework hybridized with a new local search strategy to tackle the 2DBMP. Particularly, we propose different designs for the main IG procedures (i.e., construction, destruction, and reconstruction) based on the trade-off between intensification and diversification. Additionally, the improvement method incorporates three advanced strategies: an efficient way to evaluate the objective function of neighbor solutions, a tiebreak criterion to deal with "flat landscapes", and a neighborhood reduction technique. Extensive experimentation was carried out to assess the IG performance over state-of-the-art methods, emerging our approach as the most competitive algorithm. Specifically, IG finds the best solutions for all instances considered in considerably less execution time. Statistical tests corroborate the merit of our proposal.

1. Introduction

The Two-Dimensional Bandwidth Minimization Problem (2DBMP) belongs to a family of combinatorial optimization problems denoted as Graph Layout Problems (GLP). GLPs consist of embedding an input graph (also known as a candidate graph) into a host graph by defining a mathematical function that relates (or assigns) the vertices of the input graph to the vertices of the host graph, optimizing a particular objective function. These problems can be classified according to the structure of the host graph. Particularly, the most studied GLPs are those that consider a regular host graph, such as: a path (Pardo, García-Sánchez, Sevaux, & Duarte, 2020; Pardo, Mladenović, Pantrigo, & Duarte, 2013), a cycle (Cavero, Pardo, Laguna, & Duarte, 2021b; Cavero, Pardo, & Duarte, 2022a), or a grid (Lin & Lin, 2010; Rodríguez-García, Sánchez-Oro, Rodriguez-Tello, Monfroy, & Duarte, 2021), among others. Also, GLPs can be classified according to the optimized objective function. Among the most studied ones, we can find the minimization of: the maximum cutwidth (Cavero et al., 2021b; Pardo et al., 2013), the linear arrangement (Petit, 2004; Rodriguez-Tello, Hao, & Torres-Jimenez, 2008a), the maximum Bandwidth (Ren, Hao, Rodriguez-Tello, Li, & He, 2020; Rodriguez-Tello, Hao, & Torres-Jimenez, 2008b), or the sum of the Bandwidth (Cavero, Pardo, Duarte, & Rodriguez-Tello, 2022b; Rodriguez-Tello, Narvaez-Teran, & Lardeux, 2019), among others. We refer the interested reader to surveys (Díaz, Petit, & Serna, 2002) and (Pardo, Martí, & Duarte, 2016) for further references about GLPs.

In this paper, we deal with the 2DBMP, which consists of minimizing the Bandwidth of the input graph when embedding it into a grid host graph. The 2DBMP has a large interest for the scientific community from either a practical and theoretical perspective. On the one hand, real-world applications have been devised in the literature related to the problem, as compiled in Section 1.2. On the other hand, from a theoretical perspective, there is a wide family of optimization problems related to the embedding of graphs in regular structures (Ren, Hao, & Rodriguez-Tello, 2019; Rodriguez-Tello et al., 2008b). In this sense, the algorithmic strategies proposed for the 2DBMP have interest, not only for this problem, but also for other related variants. In the following sections, we formally define the 2DBMP (Section 1.1), as well as we review the state of the art of the problem (Section 1.2).

* Corresponding author.
E-mail addresses: sergio.cavero@urjc.es (S. Cavero), eduardo.pardo@urjc.es (E.G. Pardo), abraham.duarte@urjc.es (A. Duarte).

https://doi.org/10.1016/j.ejor.2022.09.004

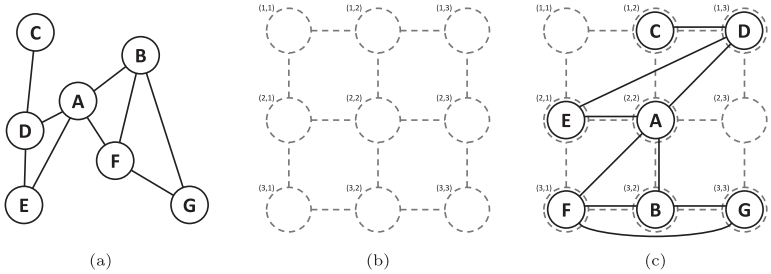


Fig. 1. (a) An input graph, G. (b) A host graph, H. (c) An example of an embedding.

1.1. Problem statement

Before formalizing this problem, we introduce a basic notation. Let $G(V_G, E_G)$ be a connected, unweighted and undirected input graph where the set of vertices is denoted as V_G (with $|V_G| = n$) and its edge set as E_G.

Similarly, let $H = (V_H, E_H)$ be a two-dimensional grid host graph where the set of vertices is denoted as V_H (with $|V_H| = \lceil\sqrt{n}\rceil \cdot \lceil\sqrt{n}\rceil$) and its edge set as E_H. Grid graphs are a common type of lattice graph, whose drawing in a Euclidean space $\mathbb{R}^2$ forms a regular tiling. These graphs satisfy the requirement that each vertex can be represented as a 2-tuple (i, j) that corresponds to a point in the plane, where $1 \leq i, j \leq \lceil\sqrt{n}\rceil$. Two vertices are connected with an edge as long as the corresponding points are at distance 1. Therefore, this particular host graph is a unit-distance, median, and bipartite graph.

In Fig. 1a, we depict an example of an input graph, G, with 7 vertices ($V_G = \{A, B, C, D, E, F, G\}$) and 9 edges ($E_G = \{(A, B), (A, D), (A, E), (A, F), (B, F), (B, G), (C, D), (D, E), (F, G)\}$). Similarly, in Fig. 1b, we show the host graph, H, needed for the embedding of the graph depicted in Fig. 1a (i.e., $|V_H| = 9 = \lceil\sqrt{7}\rceil \cdot \lceil\sqrt{7}\rceil$). In this case, vertices are denoted by their (i, j) coordinates (i.e., $V_H = \{(1, 1), (1, 2), \ldots, (3, 3)\}$), and there is an edge between them if they are at distance 1.

As it was aforementioned, an embedding consists of defining a mathematical function that assigns each vertex of the input graph to a vertex of the host graph. In mathematical terms, let φ be an injective function such that:

$$\varphi : V_G \to V_H,\ \forall\, u \in V_G\ \exists!\ v \in V_H \mid \varphi(u) = v. \tag{1}$$

Since each vertex $v \in V_H$ is defined by a 2-tuple (i, j), for the sake of convenience, we reformulate φ as follows:

$$\varphi : V_G \to V_H,\ \forall\, u \in V_G\ \exists!\ (i, j) \in V_H \mid \varphi(u) = (i, j). \tag{2}$$

with $i, j \in [1, \ldots, \lceil\sqrt{n}\rceil]$.

Then, given an embedding φ of an input graph G, the objective function of the 2DBMP, denoted as BW, is defined as follows:

$$\mathrm{BW}(G, \varphi) = \max_{(u,v)\in E_G} \{d(\varphi(u), \varphi(v))\}, \tag{3}$$

where d is a function that measures the distance between two adjacent vertices. In the related literature, d is computed with the L_1-norm (Lin & Lin, 2010; Rodríguez-García et al., 2021), which is also known as Taxicab norm distance or Manhattan distance (Craw, 2010; Lin & Lin, 2010). That is, the distance between two points $(i, j), (i', j') \in V_H$ is

$$d((i, j), (i', j')) = |i - i'| + |j - j'|. \tag{4}$$

Finally, the Two-Dimensional Bandwidth Minimization Problem (2DBMP) for a graph G consists of finding an embedding $\varphi^\star$ that minimizes Eq. (3). More formally,

$$\varphi^\star \leftarrow \mathrm{2DBMP}(G) = \min_{\varphi\in\Phi} \{\mathrm{BW}(G, \varphi)\}, \tag{5}$$

where Φ represents the set of all possible embeddings of the problem.

In Fig. 1c, we show a possible embedding φ of G (Fig. 1a) in H (Fig. 1b). As can be observed, all vertices of V_G have been assigned to a vertex of V_H through the definition of φ. For example, $\varphi(A) = (2, 2)$ indicates that vertex $A \in V_G$ is assigned to vertex $(2, 2) \in V_H$. Similarly, $\varphi(B) = (2, 3)$ indicates that vertex $B \in V_G$ is assigned to vertex $(3, 2) \in V_H$, and so on.

In order to evaluate the objective function of the example described in Fig. 1, it is required to calculate the distance between each pair of adjacent vertices in V_G (i.e., for each edge of E_G) by using Eq. (4). For instance, considering vertices A and B, with $\varphi(A) = (2, 2)$ and $\varphi(B) = (2, 3)$, the associated distance $|2 - 2| + |2 - 3| = 1$. Similarly, the distance between vertices A and D is 2, since $\varphi(D) = (1, 3)$, and $|2 - 1| + |2 - 3| = 2$. This calculation is performed over the rest of the edges of G. Then, the value of the objective function is the maximum across all distances, which is 3 in this example. Therefore, in mathematical terms, the evaluation of $\mathrm{BW}(G, \varphi)$ in Fig. 1c is computed as follows:

$$\begin{aligned}\mathrm{BW}(G, \varphi) = \max\{&d(\varphi(A), \varphi(B)), d(\varphi(A), \varphi(D)), d(\varphi(A), \varphi(E)),\\ &d(\varphi(A), \varphi(F)), d(\varphi(B), \varphi(F)), d(\varphi(B), \varphi(G)),\\ &d(\varphi(C), \varphi(D)), d(\varphi(D), \varphi(E)), d(\varphi(F), \varphi(G))\}\\ = \max\{&1, 2, 1, 2, 1, 1, 1, 3, 2\} = 3.\end{aligned} \tag{6}$$

1.2. Literature review

The 2DBMP has been widely used to formulate a variety of real-world applications. In particular, it has a direct application in the design of telecommunication architectures, where grid network topologies have gained relevance. These networks are commonly used due to their simple structure, becoming a design that is easy to build and extend (Bezrukov, Chavez, Harper, Röttger, & Schroeder, 1998). The 2DBMP has also been used for very large-scale integration (VLSI) circuit modeling (Bhatt & Thomson Leighton, 1984; Chung, 1988). Indeed, the L_1-norm distance was originally derived from circuit design models where connectors are placed in horizontal or vertical directions, although the paths in the VLSI design cannot overlap each other (Bhatt & Thomson Leighton, 1984; Lin & Lin, 2010). Other practical applications include job scheduling for parallel processing computers, solving systems of equations, or performing matrix decomposition, among others (Lai & Williams, 1999; Rodríguez-García et al., 2021).

The 2DBMP is closely related to other optimization problems belonging to the Graph Layout family, such as the Bandwidth Minimization Problem (BMP) and the Cyclic Bandwidth Problem (CBP). Differences among these three problems reside on the host graph. Specifically, in the BMP, the host graph is a path, while in the CBP it is a cycle, and in the 2DBMP it is a grid.

BMP and CBP have been widely studied by the scientific community. The former was proved to be $\mathcal{NP}$-complete for general graphs in Papadimitriou (1976) and it has been approached from both, exact (Del Corso & Manzini, 1999; Gurari & Sudborough, 1984) and heuristic perspectives (Mladenovic, Urosevic, Pérez-Brito, & García-González, 2010; Rodriguez-Tello et al., 2008b). Similarly, the CBP is also $\mathcal{NP}$-complete for general graphs as proven in Lin (1994). It has been mainly approached by considering special graphs (such as grids, trees, or planar graphs, among others) and determining either lower or upper bounds (Hromkovič, Müller, Sỳkora, & Vrt'o, 1992; Jinjiang & Sanming, 1995). Recently, in Ren et al. (2019, 2020) two advanced metaheuristics have been proposed for the CBP.

In this paper, we focus on the 2DBMP, originally proposed in Chung (1988), that belongs to the $\mathcal{NP}$-complete class (Bhatt & Thomson Leighton, 1984; Lin & Lin, 2010). This optimization problem has been studied from different points of view. In particular, lower bounds for regular-structured graphs were introduced in Lin & Lin (2010, 2011). More recently, three Constraint Satisfaction Programming (CSP) models (Tsang, 2014) were described in Rodríguez-García et al. (2021). The best model, denoted as M3, is able to solve small and regular-structured graphs, providing a lower bound (not necessarily tight) for medium and large instances. The best previous metaheuristic procedure identified in the related literature was introduced in Rodríguez-García et al. (2021). Specifically, the authors described an algorithm based on the combination of the Greedy Randomized Adaptive Search Procedure (GRASP) (Feo & Resende, 1995) and the Basic Variable Neighborhood Search (BVNS) (Hansen & Mladenović, 2006) methodologies. This procedure follows a multi-start strategy, where many initial points are generated with GRASP and then improved with BVNS. The constructive procedure uses a greedy criterion based on the quality of the objective function, while the BVNS, based on the idea of systematic changes of neighborhood the structure within the search, uses two neighborhoods and a random perturbation to escape from local optima. This procedure is currently considered the state of the art for the 2DBMP.

1.3. Our contributions

The main contribution of this work is the proposal of an efficient procedure based on the Iterated Greedy framework. The proposed algorithm includes novel construction/destruction/reconstruction strategies, as well as advanced improvement methods. These techniques are designed from a general perspective, i.e., they are not only valid for the 2DBMP, but also for any other related optimization problem. Specifically, the proposed construction, destruction, and reconstruction strategies vary from totally random to totally greedy approaches. In addition, a straightforward design of a local search for the 2DBMP is enriched by: a tiebreak criterion to distinguish between same-quality solutions; fast evaluation of the objective function; and neighborhood reduction techniques.

The proposed method is configured by a set of preliminary experiments, which allow us tuning the search parameters as well as to evaluate the influence of the proposed mechanisms. Finally, the best identified variant is compared with state-of-the-art algorithms through competitive tests. The merit of the results obtained is supported by statistical tests.

The rest of the paper is organized as follows: Section 2 describes our algorithmic proposal. Section 3 introduces several advanced search strategies. Section 4 presents and analyses the results of the computational experiments carried out. Finally, the conclusions are drawn in Section 5.

2. Algorithmic proposal: Iterated greedy

In this paper, we propose a procedure based on the Iterated Greedy (IG) metaheuristic (Ruiz & Stützle, 2007; Stützle & Ruiz, 2018). IG is a search method where solutions are gradually improved through the repeated application of two main phases: a partial destruction of a solution followed by a reconstruction to reach a new feasible solution. These two phases are usually repeated for a fixed number of iterations, for a maximum number of iterations without finding an improvement, or even for a combination of the two previous criteria.

IG can be easily hybridized with other strategies, such as local search procedures or other metaheuristics. In this case, the destruction and reconstruction phases of IG can be understood as a way to perturb the incumbent solution, similarly to other well-known metaheuristics such as Iterated Local Search (ILS) (Lourenço, Martin, & Stützle, 2003) or Variable neighborhood Search (VNS) (Hansen, Mladenović, Todosijević, & Hanafi, 2017). However, in the IG methodology, an important part of the process, the reconstruction phase, uses greedy decisions rather than stochastic ones. See Stützle & Ruiz (2018) for further details.

In this paper, we propose the hybridization of IG with an efficient local search procedure. The pseudocode of the proposed method is presented in Algorithm 1. The procedure receives as

Algorithm 1 General procedure based on IG algorithm.

```
1:  Procedure IteratedGreedy (G, maxIter, maxNotImprIter)
2:  iter = 0, notImprIter = 0
3:  φ = GreedyConstructive(G)
4:  φ ← LocalSearch(G, φ)
5:  while iter < maxIter do
6:      iter = iter + 1, notImprIter = notImprIter + 1
7:      φ' ← Destruction(notImprIter, φ)
8:      φ'' ← Reconstruction(G, φ')
9:      φ''' ← LocalSearch(G, φ'')
10:     if BW(G, φ''') < BW(G, φ) then
11:         φ ← φ'''
12:         notImprIter = 0
13:     end if
14:     if notImprIter > maxNotImprIter then
15:         break
16:     end if
17: end while
18: return φ
```

parameters: the input graph, G; the maximum number of iterations, *maxIter*; and the number of iterations without improvement *maxNotImprIter*. The algorithm starts by generating an initial solution with the greedy constructive procedure (line 3) that will be introduced in Section 2.1. Then, after obtaining an improved solution through the local search procedure (line 4), described in Section 2.2, the procedure enters a loop (lines 5 to 17). In each iteration, some elements are removed from the current solution using the destruction method (line 7). Next, the solution is greedily reconstructed (line 8) and improved again by the local search procedure (line 9). Both destruction and reconstruction methods, are described in Section 2.3. In each iteration, IG determines (steps 10 to 13) whether the perturbed and improved solution (φ''') is better than the incumbent one (φ). If so, φ and *notImprIter* are updated accordingly. These three last steps (destruction, reconstruction, and local search) are repeated until a maximum number of iterations is reached, unless the procedure is not able to improve the current best solution for a number of iterations (line 14). Once the termination condition is met, IG returns the best solution found (step 18).

2.1. Greedy constructive procedure

Constructing a solution for the 2DBMP from a greedy perspective consists of performing the best possible assignation of the vertices of the input graph to the vertices of the host graph (i.e., defining φ). To do this, we need to answer three questions: #1 which vertices (from either the host or the input graphs) are more suitable to start with; #2 given a partial solution, which vertex (u) of the input graph should be assigned next; and #3 given a partial solution and u, which vertex v of the host graph should be assigned to u. To answer each of these questions, we propose different strategies, which are described below.

We start by selecting a random vertex from the input graph. Then, to perform its assignation, we need to select a vertex from the host graph. In this case, we study three alternatives: a random vertex; a vertex placed in a corner of the grid (i.e., $(1,1), (1, \lceil\sqrt{n}\rceil), (\lceil\sqrt{n}\rceil, 1), (\lceil\sqrt{n}\rceil, \lceil\sqrt{n}\rceil)$; or a vertex placed in the center of the grid (i.e., $(\lceil\frac{\lceil\sqrt{n}\rceil}{2}\rceil, \lceil\frac{\lceil\sqrt{n}\rceil}{2}\rceil)$ if $\lceil\sqrt{n}\rceil$ is odd; or $(\lceil\frac{\lceil\sqrt{n}\rceil}{2}\rceil, \lceil\frac{\lceil\sqrt{n}\rceil}{2}\rceil)$, $(\lceil\frac{\lceil\sqrt{n}\rceil}{2}\rceil + 1, \lceil\frac{\lceil\sqrt{n}\rceil}{2}\rceil)$, $(\lceil\frac{\lceil\sqrt{n}\rceil}{2}\rceil, \lceil\frac{\lceil\sqrt{n}\rceil}{2}\rceil + 1)$, $(\lceil\frac{\lceil\sqrt{n}\rceil}{2}\rceil + 1, \lceil\frac{\lceil\sqrt{n}\rceil}{2}\rceil + 1)$ if n is even).

Given the first assignation, we already have a partial solution. Then, we propose a greedy function, inspired by McAllister et al. (1999) and denoted as g_1, to determine which vertices of the input graph should be assigned in the following steps to a partial solution φ. In other words, g_1 is used to evaluate the "urgency" of any unassigned vertex from the input graph to be assigned next.

Let us introduce some notation before defining g_1. We define the subset $U_G \subset V_G$ as the set of vertices of the input graph that have not been assigned yet in φ. Then, given a vertex $u \in U_G$, we define $A(u)$ as the set of adjacent vertices to u already assigned. More formally, $A(u) = \{v \in V_G : v \notin U_G \wedge (u, v) \in E_G\}$. Similarly, we define $R(u)$ as the set of vertices adjacent to u that remain unassigned. In mathematical terms, $R(u) = \{v \in V_G : v \in U_G \wedge (u, v) \in E_G\}$. It is worth mentioning that $A(u) \cup R(u)$ is the set of adjacent vertices to u. Then, g_1 is defined in Eq. (7) as follows:

$$g_1(u, \varphi) = w_1 \cdot |A(u)| - w_2 \cdot |R(u)|, \tag{7}$$

where w_1 and w_2 are parameters that should be tuned experimentally and satisfy $0 \leq w_1, w_2 \leq 1$ and $w_1 + w_2 = 1$. These two parameters balance the relevance of having a large number of adjacent vertices assigned ($w_1 > w_2$) or a reduced number of adjacent vertices unassigned ($w_1 < w_2$). Notice that if $w_1 = w_2$, then the strategy is equivalent to the original proposal introduced in McAllister et al. (1999). Then, all unassigned vertices from the input graph are evaluated, and the vertex with the largest g_1 value is chosen to be assigned next (with ties broken at random).

Once the vertex of the input graph has been chosen, the second proposed question is answered. Then, an available vertex from the host graph must be selected to embed the selected vertex of the input graph. In this case, we propose two approaches: one based on graphical patterns and the other based on a greedy function. The first approach determines the order in which host vertices are selected on the basis of a graphical pattern. Moreover, in this work we study three patterns: Sequential, Diagonal, and Zigzag, which are illustrated in Fig. 2(a)–(c), respectively. In each of the figures, the sequence is indicated by green arrows and numbers inside each of the host vertices, being the number 1 the vertex selected in the first iteration and number 9 the vertex selected in the last iteration.

The second approach to select a vertex of the host graph is based on a new greedy function, denoted as g_2, that evaluates the variation of the objective function when assigning a vertex of the input graph. As it is well documented in the related literature (Cavero et al., 2021b), some successful strategies in graph layout problems are related to the closeness/remoteness of adjacent vertices. In this case, g_2 is used to place every candidate vertex as close as possible to its adjacent vertices. Let $C_H \subseteq V_H$ be the set of vertices which distance (see Eq. (4)) to the already assigned host vertices is 1. More formally, $C_H(\varphi) = \{(i, j) \in V_H : \forall\, v \notin U_G, d(\varphi(v), (i, j)) = 1\}$. Then, given a vertex $u \in U_G$, a vertex $(i, j) \in C_H$, and a partial solution φ, g_2 is formally defined as:

$$g_2(\varphi, u, (i, j)) = \max_{v \in A(u)} \{d((i, j), \varphi(v))\}. \tag{8}$$

Thus, all unassigned vertices from the host graph are evaluated and the vertex $(i, j) \in C_H(\varphi)$ that minimizes g_2 is selected to host the input graph vertex u, i.e., $\varphi(u) = (i, j)$. The rationale behind this strategy is that the returned value from g_2 corresponds to the contribution of the evaluated vertex to the objective function. Therefore, the best host vertex for an input vertex is the one that minimizes Eq. (8).

In Algorithm 2 we show the pseudocode of a general greedy constructive procedure which can use any of the greedy strategies described in this section. Particularly, it receives an input graph $G = (V_G, E_G)$ and a host graph $H = (V_H, E_H)$ as input parameters. The method starts by selecting a vertex u of the input graph at random (line 2). According to the aforementioned Question #1, in line 3, the procedure identifies the vertex of the host graph to embed u. Then, the first assignation is performed (line 4). The set of unassigned vertices U_G for the partial solution φ is constructed in line 5. While there are still elements in U_G, the greedy constructive procedure selects a vertex according to g_1 (see Question #2) in line 8. Next, this selected vertex is assigned to one vertex in the host graph (see Question #3), either based on patterns (Fig. 2) or based on g_2. Finally, the partial solution and the set of unassigned vertices are updated in lines 10 and 11, respectively. Lines 7 to 12 are repeated until generating a complete feasible solution, which is returned at the end of the procedure (line 13). To complement the pseudocode, we graphically illustrate the steps of the constructive procedure in the flowchart depicted in Fig. 3. This flowchart follows the activity diagram standard described in the Unified Modeling Language (Booch, 2005). Therefore, each rectangle describes a task or activity, while each diamond expresses a condition of the constructive procedure. Once the solution has been fully constructed, the evaluation of the solution is performed from scratch following the procedure described at the end of Section 1.1.

Algorithm 2 Greedy constructive procedure.

```
1: Procedure GreedyConstructive (G, H)
2: u ← random(V_G)
3: (i, j) = GetInitialHostGraphVertex(V_H)   ▷ Answer to question #1
4: φ(u) ← (i, j)
5: U_G ← V_G \ {u}
6: while U_G ≠ ∅ do
7:     u ← GetNextInputGraphVertex(φ, U_G)   ▷ Answer to question #2
8:     (i, j) ← GetNextHostGraphVertex(φ, u)▷ Answer to question #3
9:     φ(u) ← (i, j)
10:    U_G ← U_G \ {u}
11: end while
12: return φ
```

The choice of the first vertex of the input graph might influence deeply the quality of the generated solution. Moreover, our constructive procedure is not fully deterministic since random decisions are also made to break ties in both g_1 and g_2. Therefore, we propose to execute the constructive procedure for a fixed number of iterations, selecting as initial solution the best one among

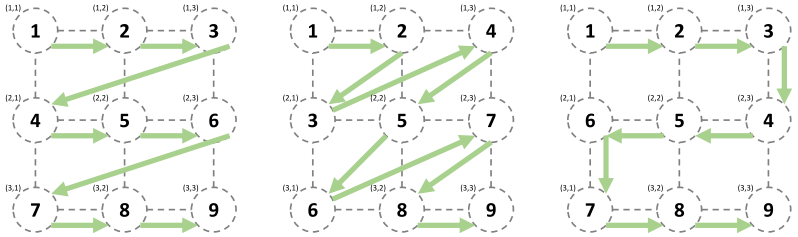


Fig. 2. Examples of the order followed to assign vertices of the input graph to vertices of the host graph using the Sequential (a), Diagonal (b) and Zigzag (c) patterns.

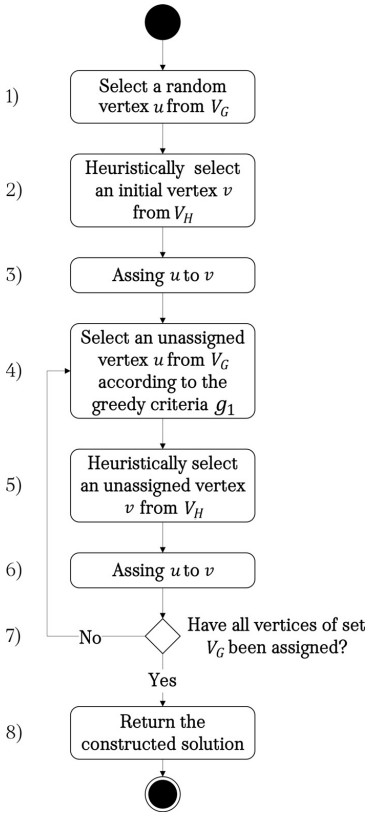


Fig. 3. Activity diagram of the constructive procedure..

them. It is worth mentioning that this strategy has been successfully used not only in IG (Huerta-Muñoz, Ríos-Mercado, & Ruiz, 2017; Stützle & Ruiz, 2018) but also in combination with VNS (López-Sánchez, Sánchez-Oro, & Hernández-Díaz, 2019; Pérez-Peló, Sánchez-Oro, Gonzalez-Pardo, & Duarte, 2021), or TS (Abdinnour-Helm & Hadley, 2000; Delmaire, Díaz, Fernández, & Ortega, 1999), among others. To ensure the construction of diverse solutions, the number of iterations is always set to a value larger than the number of vertices of the input graph, guaranteeing that at least one construction is performed starting from every vertex of the input graph.

2.2. Improvement strategy

Local search is a heuristic method widely used to solve hard optimization problems due to its ability to trade solution quality with computation time. This procedure is based on systematic moves from one solution to another with better quality, until reaching a locally optimal solution (Michiels, Aarts, & Korst, 2018). Given a predefined move operation, the set of feasible solutions reachable by the local search starting from that solution is usually known as neighborhood.

In the related literature, there have been proposed two different neighborhoods for the 2DBMP based on exchange and insert moves (see Rodríguez-García et al., 2021 for further details). However, the definition of these neighborhoods do not include the possibility of exploring the vertices of the host graph that have not been assigned to vertices of the input graph in the construction phase. In this paper, we propose a more flexible move operator, which explores those host vertices not initially assigned, resulting in a larger neighborhood space. Specifically, the number of vertices in the host graph ($|V_H| = \lceil\sqrt{n}\rceil \cdot \lceil\sqrt{n}\rceil$) is always larger than or equal to the number of vertices in the input graph ($|V_G| = n$). Therefore, φ is an injective function, since there may exist vertices in V_H that do not host a vertex of V_G. The move operator considers this property as follows: given a solution φ, a vertex $u \in V_G$ assigned to $(i, j) \in V_H$ (i.e., $\varphi(u) = (i, j)$), and a vertex $(i', j') \in V_H$ the move operator $Move(\varphi, u, (i', j'))$ assigns u to (i', j') (i.e., $\varphi(u) = (i', j')$). In the case that a vertex $v \in V_G$ was assigned to (i', j') prior the move, this move would also perform a new assignment for the vertex v (i.e., $\varphi(v) = (i, j)$). Otherwise, (i, j) will not host any vertex. This move produces a new feasible solution, which is denoted as $\varphi' \leftarrow Move(\varphi, u, (i', j'))$.

Let us illustrate this move with an example. Departing from the solution represented in Fig. 1c, we show in Fig. 4a the situation before performing $Move(\varphi, \mathrm{B}, (2, 3))$, where vertex B of the input graph is assigned to vertex (3,2) prior the move. Then, in Fig. 4b, we depict the resulting solution after the move, where B is assigned to (2,3) leaving (3,2) without any assignation.

Considering the aforementioned move operator, the proposed neighborhood for the 2DBMP is defined as follows:

$$N(\varphi) = \{Move(\varphi, u, (i', j')) : \forall\, u \in V_G, (i', j') \in V_H, \varphi(u) \neq (i', j')\}. \tag{9}$$

The size of $N(\varphi)$ can be determined depending on the number of vertices of the input graph. Specifically, given an input graph with $|V_G| = n$ vertices and the associated host graph with $|V_H| = \lceil\sqrt{n}\rceil \cdot \lceil\sqrt{n}\rceil$ vertices, the size of $N(\varphi)$ is $\frac{1}{2}n(\lceil\sqrt{n}\rceil^2 - 1)$.

In this research, we study two well-known strategies to explore the proposed neighborhood: *best improvement* and *first improvement* (Hansen & Mladenović, 2006). If a local search follows the *best improvement* strategy it selects, at each iteration, the best pos-

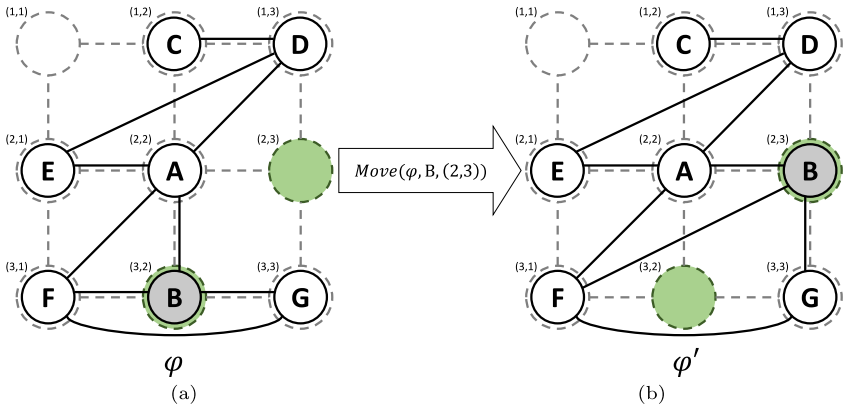


Fig. 4. (a) Example of an embedding φ. (b) The resultant embedding φ' obtained after the operation $Move(\varphi, B, (2, 3))$ (b).

sible move which produces an improvement of the current neighborhood; otherwise, in the *first improvement* strategy, it selects the first solution that improves the current solution. Any of them stop the search when no further improving moves can be performed. We study the effectiveness of both strategies in the experiments reported in Section 4.1.

2.3. Destruction and reconstruction procedures

The two main procedures of any IG algorithm are the destruction and reconstruction phases. The IG proposed in the context of 2DBMP, first removes a determined number of assignations of vertices of the input graph to vertices of the host graph, during the destruction phase. Then, the reconstruction phase applies a greedy heuristic to reassign the unassigned vertices until reaching a new feasible solution.

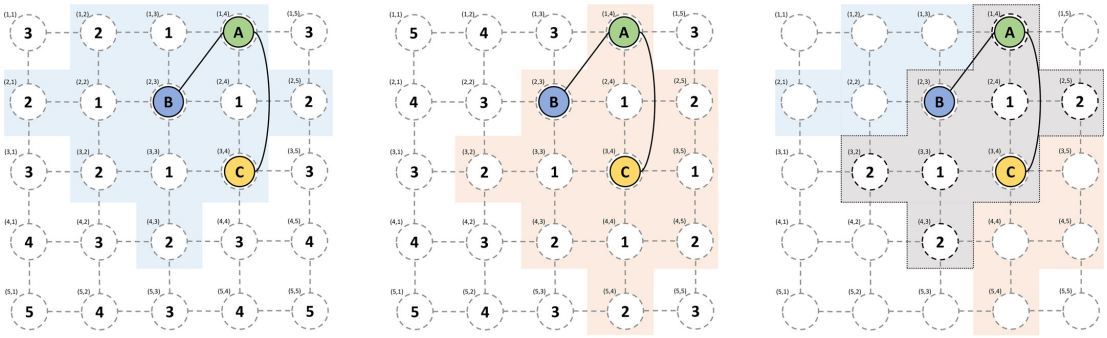
The number of assignations that should be removed is dynamically defined depending on the current number of iterations without improvement (see lines 6 and 7 of Algorithm 1). The rationale behind this decision is to have a trade-off between search intensification and diversification. In particular, when the number of iterations without improvement is large, it is expected that the procedure gets stuck in a "deep basin of attraction". Therefore, a large number of assignations are removed (with the corresponding reconstruction steps), with the aim of moving to rather distant solutions in the solution space. On the contrary, when the number of iterations without improvement is small, a few assignations are removed, which leads to a more localized search Stützle & Ruiz (2018). In addition, to avoid the complete destruction of the solution, we set a maximum number of assignations that can be removed. Specifically, this value is set to 25% of the vertices of the instance under consideration.

In this paper, we propose three different destruction strategies. The first one, denoted as *fully randomized destruction*, is a straightforward adaptation of the IG framework. Specifically, it consists of randomly selecting and then removing a determined number of assignations of vertices of the input graph to vertices of the host graph. The second strategy, denoted as *random area destruction*, focuses on a specific area of the host graph. More precisely, it selects a vertex of the host graph at random and then removes its assignation, and also the assignation of all adjacent host graph vertices (i.e., those at distance 1 to the selected initial vertex, according to Eq. (4)). This strategy keeps on removing host adjacent vertices to the unassigned area following a proximity criterion (i.e., first those at distance 2, then those at distance 3, etc.) until reaching the expected number of unassigned vertices. The third strategy, denoted as *greedy destruction*, focuses on those vertices that determine the value of the objective function. Notice that the 2DBMP consists of minimizing a maximum value; then, the objective function is usually determined by a reduced number of assignations. This destruction strategy removes the assignation of the vertex that determines the value of the objective function, also removing the assignations of all its adjacent host graph vertices. As in the second strategy, it keeps on removing vertices and their adjacent vertices in the host graph following a proximity criterion, until reaching the expected number of unassigned vertices.

As far as the reconstruction strategy is concerned, the Iterated Greedy framework Stützle & Ruiz (2018) suggests that the reconstruction should be governed by a greedy heuristic and, typically, deterministic (except random tiebreaking). We therefore do not explore any random reconstruction strategy. Specifically, given a partial solution obtained as the result of a destruction phase, the reconstruction of the solution is performed by following a greedy strategy. To this end, we propose the use of three strategies, based on the greedy criteria presented in Section 2.1: 1) unassigned vertices of the input graph are selected according to g_1 function and then are randomly assigned to any of the available host graph vertices; 2) unassigned vertices of the input graph are randomly selected and then assigned to its best host graph vertex according to g_2 function; and 3) the best vertex of the input graph is selected according to g_1 and it is assigned to the best host vertex selected according to g_2 function.

3. Advanced search strategies for exploring the neighborhoods

As it is well documented in the related literature, most of the computing time of a heuristic algorithm is spent by the local search procedure. Particularly, a local search heuristic selects, at each iteration, a move to a neighbor solution, if it results in an improvement of the objective function. Then, it needs to explore multiple neighbor solutions, evaluating each of them, to determine which move should be done next. In this section, we provide three new advanced strategies devoted to increasing the efficiency of the proposed local search: first, we introduce an efficient strategy to reduce the number of solutions to explore in a given neighborhood (see Section 3.1); second, once the number of moves has been reduced, we propose a technique to speed up the search by optimizing the calculation of the objective function after a move (see



(a) S^{φ}_{B} (b) S^{φ}_{C} (c) $\mathrm{PSV}(\mathrm{A}, \varphi)$

Fig. 5. Definition of the PSV of A for a solution φ, with $\mathrm{BW}(G, \varphi) = 2$.

Section 3.2); finally, when the move results in a tie, in terms of the objective function value, we propose a way of distinguishing between two solutions with the same quality (see Section 3.3). It is worth mentioning that some of these strategies can be applied or adapted with a few modifications, not only for other Graph Layout Problems, but also for other optimization problems.

3.1. Neighborhood reduction strategy

The neighborhood proposed in Section 2.2 for the 2DBMP has a size of $\frac{1}{2}n(\lceil\sqrt{n}\rceil^2 - 1)$, being n the number of vertices of the input graph. In the worst case, an exhaustive exploration requires to traverse the whole neighborhood. In this section, we propose a strategy to focus the search on promising solutions, avoiding to waste time in the evaluation of solutions which produce a deterioration in the objective function.

For each vertex of the input graph $u \in V_G$, we can define a set of vertices of the host graph, denoted as Promising Set of Vertices (PSV) which can host u satisfying that the distance d (in the host graph) from u to any of its neighbors in the input graph is equal or smaller to the objective function value.

Given a solution φ, and an input vertex $v \in V_G$, let us define S^{φ}_{v} as the set of host vertices which satisfy that the distance d from $\varphi(v)$ to any of them is smaller or equal to the $\mathrm{BW}(G, \varphi)$. More formally:

$$S^{\varphi}_{v} = \{(i, j) \in V_H : d(\varphi(v), (i, j)) \leq \mathrm{BW}(G, \varphi)\}. \tag{10}$$

Then, once we have defined the set S^{φ}_{v} for a single input vertex, we can define the set PSV for an input vertex $u \in V_G$ as the intersection of the sets S^{φ}_{v} for all adjacent vertices to u in the input graph (i.e., $v \in A(u)$). More formally:

$$\mathrm{PSV}(u, \varphi) = \bigcap_{v \in A(u)} S^{\varphi}_{v}. \tag{11}$$

In Fig. 5 we show an example of the definition of PSV of the vertex A. Particularly, vertices B and C are adjacent to A in the input graph. For the sake of clarity, the rest of the input vertices, not adjacent to A, have not been represented in the figure. Moreover, let us suppose that the value of the objective function for the solution φ, represented in the figure is 2 (i.e., $\mathrm{BW}(G, \varphi) = 2$).

In Fig. 5a we have highlighted with light blue color the set of host vertices placed at a distance 2 or minor from the host vertex (2,3), which hosts B. Specifically, $S^{\varphi}_{\mathrm{B}} = \{(1, 2), (1, 3), (1, 4), (2, 1), (2, 2), (2, 3), (2, 4), (2, 5), (3, 2), (3, 3), (3, 4), (4, 3)\}$. Additionally, in Fig. 5a we have indicated, with a number inside each host vertex, the distance d to (2,3). Similarly, in Fig. 5b we have highlighted in light orange the set of host vertices placed at a distance 2 or minor from the host vertex (3,4), which hosts C. Particularly, $S^{\varphi}_{\mathrm{C}} = \{(1, 4), (2, 3), (2, 4), (2, 5), (3, 2), (3, 3), (3, 4), (3, 5), (4, 3), (4, 4), (4, 5), (5, 4)\}$. Finally, in Fig. 5c we show the set PSV of A, obtained as the intersection of the two previous sets, which contains the host vertices placed at distance 2 or minor to any of the adjacent vertices to A. More formally, $\mathrm{PSV}(\mathrm{A}, \varphi) = S^{\varphi}_{\mathrm{B}} \cap S^{\varphi}_{\mathrm{C}} = \{(1, 4), (2, 3)(2, 4), (2, 5), (3, 2), (3, 3), (3, 4), (4, 3)\}$. Therefore, if A is assigned to any of the vertices in PSV(A), the maximum distance with respect to its adjacent vertices will be equal or smaller than the BW of φ.

Finally, we formally define $N_R(\varphi)$, as the restricted neighborhood of $N(\varphi)$ as follows:

$$N_R(\varphi) = \{Move(\varphi, u, (i, j)) : \forall\, u \in V_G, (i, j) \in \mathrm{PSV}(u), \varphi(u) \neq (i, j)\}. \tag{12}$$

Then, we propose the exploration of the reduced neighborhood instead of the whole neighborhood defined by the move operator introduced in Section 2.2.

3.2. Efficient move calculation

For the 2DBMP, a naive straightforward evaluation of a solution after a move consists in recalculating the value of the objective function from scratch. This means that the contribution of every input vertex has to be updated. However, an intelligent evaluation could avoid reevaluating the whole solution by just updating the elements that have been affected by the move. Therefore, we propose an efficient local search method, based on the move operator defined in Section 2.2, which applies an efficient evaluation after a move. Given an input graph G, a host graph H, and a solution φ, this move considers the vertex $u \in V_G$ (hosted in vertex $(i, j) \in V_H$), and assigns it to vertex $(i', j') \in V_H$. As it was aforementioned, if there were a vertex v hosted in (i', j'), i.e., $\varphi(v) = (i', j')$, this move would also assign v to (i, j). Therefore, an efficient objective function revaluation just needs to update the distance assigned to those edges with u (also v when existing) as an endpoint.

To define the Efficient Bandwidth calculation, denoted as EBW, of an input graph G and a solution φ' obtained by a move $Move(\varphi, u, (i, j))$, we first define the set of edges $I^{\varphi}_{u,(i,j)}$ involved in the move denoted as follows:

$$I^{\varphi}_{u,(i,j)} = \{(u, w) \in E_G : \forall w \in V_G\} \cup \{(v, w) \in E_G : \forall\, v, w \in V_G \wedge\ \varphi(v) = (i, j)\}. \tag{13}$$

Then, the EBW$(G, Move(\varphi, u, (i', j')), I)$ can be computed as:

$$\begin{aligned}\text{EBW}\quad & (G, Move(\varphi, u, (i', j')), I^{\varphi}_{u,(i,j)}) = \\ & \max\{\underbrace{\max_{(w,z)\in E_G \setminus I^{\varphi}_{u,(i,j)}} \{d(\varphi(w), \varphi(z))\}}_{\text{Not updated}}, \underbrace{\max_{(w,z)\in I^{\varphi}_{u,(i,j)}} \{d(\varphi(w), \varphi(z))\}}_{\text{Updated}}\}.\end{aligned} \tag{14}$$

Notice that the distances of the edges that do not need to be updated in Eq. (14) can be easily evaluated by storing the distance associated with each edge before the move. Specifically, we use an array of sets, where the range of the array is determined by the possible distance values, while in each set it is stored all edges with a the same distance value. When we consider together the use of the aforementioned data structure and the efficient move calculation, the running time can be reduced in two orders of magnitude, on average, as we will show in the computational experience.

Let us illustrate this with an example. Particularly, we consider again the move $\varphi' \leftarrow Move(\varphi, \text{B}, (2, 3))$ depicted in Fig. 4. In order to evaluate the objective function of the resulting solution φ' it is just needed to update the distance associated to the edges with an endpoint in B, since (2,3) is not hosting any vertex of the input graph. More precisely, the edges with an endpoint in B are (A, B), (B, F), and (B, G). Then, only the distance of those edges needs to be re-evaluated, being the BW(G, φ') calculated as follows:

$$\begin{aligned}\text{BW}(G, \varphi') &= \max\{d(\varphi(\text{A}), \varphi(\text{D})), d(\varphi(\text{A}), \varphi(\text{E})), d(\varphi(\text{A}), \varphi(\text{F})), \\ &\quad d(\varphi(\text{C}), \varphi(\text{D})), d(\varphi(\text{D}), \varphi(\text{E})), d(\varphi(\text{F}), \varphi(\text{G})), \\ &\quad \boldsymbol{d(\varphi'(\text{A}), \varphi'(\text{B}))}, \boldsymbol{d(\varphi'(\text{B}), \varphi'(\text{F}))}, \boldsymbol{d(\varphi'(\text{B}), \varphi'(\text{G}))}\} \\ &= \max\{\underbrace{2, 1, 2, 1, 3, 2}_{\text{Not updated}}\underbrace{\mathbf{1}, \mathbf{3}, \mathbf{1}}_{\text{Updated}}\} = 3.\end{aligned} \tag{15}$$

Notice that, in this particular example, the number of edges evaluated is 3, while evaluating the whole solution requires 9 updates. Reasonably, the impact of this strategy is larger when dealing with instances composed by many vertices.

3.3. Tiebreak criterion for solutions with the same objective function value

An optimization problem consists of maximizing or minimizing a particular objective function. In some cases, this mathematical function consists of computing either the maximum or minimum value of a set of elements. Therefore, regardless the size of this set, the maximum or the minimum value, which determines the value of the objective function, is usually reached in more than one element. When the goal of a problem is to minimize a function based on a maximum value, we denote it as min-max problem. Similarly, when the goal of a problem is to maximize a minimum function, it is denoted as max-min problem. Max-min and min-max problems are quite common in optimization and become a challenge for heuristic methods because there may be many different solutions with the same objective function value, despite they are different solutions. This fact is usually known as "*flat landscape*" (Martí, Pantrigo, Duarte, & Pardo, 2013; Pardo et al., 2013). When this happens, it is difficult to determine the search direction since there is no way, according to the objective function, to determine which solution is more promising. To mitigate this problem, researchers have opted for tiebreaking criteria or alternative objective functions (Cavero, Pardo, & Duarte, 2021a; Cavero et al., 2021b).

The 2DBMP is a min-max optimization problem and preliminary experiments corroborate the existence of flat landscapes throughout the solution space. Furthermore, for an input graph with $n = |V_G|$ vertices and a host graph with $m = \lceil\sqrt{n}\rceil \cdot \lceil\sqrt{n}\rceil$ vertices, the number of solutions in the search space is upper bounded by $m!/(m-n)!$, but the range of values obtained as the evaluation with the objective function for that solutions are integer numbers in the interval $[1, 2 \cdot (\lceil\sqrt{n}\rceil - 1)]$. Let us remember that the evaluation of the objective function of the 2DBMP is directly related to the distance of adjacent input vertices measured in the host graph (see Eq. (4)). Then, the number of possible objective function values is considerably smaller than the number of solutions. Therefore, there might be many solutions with the same value of the objective function. To overcome this difficulty, we propose a tiebreak criterion based on the frequency of a particular distance in a solution. More formally, given an input graph $G(V_G, E_G)$ and a solution φ, we define f_l as the number of edges (u, v) of E_G with an associated distance in the host graph equal to l:

$$f_l = |\{(u, v) \in E_G : d(\varphi(u), \varphi(v)) = l\}|. \tag{16}$$

Let $l_{\max}$ be the maximum distance among all adjacent vertices in the graph. It trivially holds that $l_{\max}$ corresponds to the objective function value of the problem for a particular solution (i.e., $l_{\max} = \text{BW}(G, \varphi)$). Then, given an input graph G and an embedding φ, we propose a tiebreaking function t defined as follows:

$$t(G, \varphi) = \sum_{l=1}^{l_{max}} n^l \cdot f_l. \tag{17}$$

This equation is inspired by previous works related to circular layout problems (Cavero et al., 2021a; Cavero et al., 2021b). It takes into consideration not only the maximum distance of an embedding, but also additional semantic information related to promising solutions. Specifically, when the objective function value of two solutions is equal, both solutions are evaluated using the function t. The solution with the lower value of t is then chosen as the most promising one. The rationale behind this decision is to penalize those solutions with many edges with an associated distance close to the value of the objective function. It is worth mentioning that if the t value for two solutions is the same, they are considered as equivalent (in terms of the tiebreak criterion).

Let us illustrate the use of the tiebreak criterion with an example. To do so, we consider three different solutions φ_1, φ_2, and φ_3. Let us assume that the objective function for each solution is $\text{BW}(G, \varphi_1) = \max\{1, 3, 1, 2, 3\} = 3$, $\text{BW}(G, \varphi_2) = \max\{1, 2, 1, 2, 3\} = 3$, and $\text{BW}(G, \varphi_3) = \max\{2, 2, 1, 2, 3\} = 3$, respectively. Then, these three solutions are equal according to the objective function of the problem (i.e., the maximum value across all elements, which is 3). However, if we evaluate them with the tiebreak criterion, we appreciate differences among the solutions:

$$\begin{aligned}t(G, \varphi_1) &= 5^1 \cdot 2 + 5^2 \cdot 1 + 5^3 \cdot 2 = 10 + 25 + 250 = 285. \\ t(G, \varphi_2) &= 5^1 \cdot 2 + 5^2 \cdot 2 + 5^3 \cdot 1 = 10 + 50 + 125 = 185. \\ t(G, \varphi_3) &= 5^1 \cdot 1 + 5^2 \cdot 3 + 5^3 \cdot 1 = 5 + 75 + 125 = 205.\end{aligned}$$

In this case, since $t(G, \varphi_2) < t(G, \varphi_3) < t(G, \varphi_1)$, we would consider φ_2 as the most promising one. Similarly, we consider φ_3 more promising than φ_1.

4. Computational results

In this section, we present the experiments carried out to empirically evaluate the algorithmic proposals introduced in this paper. Particularly, we first propose a set of preliminary experiments to configure the best variant of our algorithm and to illustrate the influence of the advanced search strategies. Then, our best variant is compared with the best previous algorithm identified in the state of the art.

Table 1
Influence of the initial host vertex (answer to Question #1 in Section 2.1).

	Random	Corner	Center
Avg. OF	13.23	13.00	u8
CPU Time (s)	0.37	0.37	0.38
Dev. (%)	6.92	5.02	11.55
#Best	7	8	6

Table 2
Influence of the host vertex selected after the first assignation based on the weights w_1 and w_2 in g_1 (answer to Question #3 in Section 2.1).

	Diagonal	Sequential	ZigZag	g_2
Avg. OF	34.23	21.46	18.77	10.08
CPU Time (s)	4.68	4.51	4.48	5.79
Dev. (%)	283.19	138.11	89.66	6.15
#Best	0	0	1	12

The computational tests have been performed over 90 instances previously reported in the related literature on the 2DBMP (Rodríguez-García et al., 2021). These instances are grouped into two different subsets which include: 45 topologically diverse small graphs (with $|V_G| \in [5, 21]$ and $|E_G| \in [6, 190]$); and 45 representative graphs from the Harwell-Boeing collection (with $|V_G| \in [48, 960]$ and $|E_G| \in [78, 7442]$). To ease future comparisons, all instances have been made publicly available at https://www.heuristicas.es/.

All experiments have been performed on an AMD EPYC 7282 16-core virtual CPU with 16GB of RAM. The operating system used was Ubuntu 20.04.2 64 bit LTS, and all algorithms were implemented in Java 16.

4.1. Preliminary experiments

In this section, we identify the best configuration of the components of the Iterated Greedy procedure proposed in this paper. Also, we illustrate the merit of the proposed advanced search strategies. The preliminary experiments have been performed over a reduced set of instances consisting of 15% of the total considered instances (i.e., 13 graphs). We will refer to this subset of instances as the preliminary set.

For each of the experiments carried out, we report the following metrics: the average value of the objective function (Avg. OF), the total execution time in seconds (CPU Time (s)), the average deviation to the best solution found in the experiment (Dev. (%)) and the number of best solutions found in the experiment (#Best).

The first set of experiments performed is devoted to configure the best variant of the greedy constructive procedure described in Section 2.1, by trying to find the best answer to the questions raised in that section. Notice that the parameters are studied one by one, varying the values for the selected parameter and fixing the value for the rest of parameters.

Particularly, in Table 1 we analyse the proposed strategies to determine which is the most suitable host vertex to perform the first assignation (denoted as the answer to Question #1 in Section 2.1). The results reported in Table 1 correspond to a 100 of constructions where the input vertex has been chosen at random, g_1 is configured with $w_1 = 0.5$ and $w_2 = 0.5$ (i.e., both have the same weight), and g_2 is used as the criterion to determine the host vertices in the following assignations. With this configuration, starting the construction from a corner host vertex seems to be the best alternative, since the constructive procedure is able to reach the largest number of best solutions and the smallest deviation to the best solution.

In this case, we do not need to perform an experiment to select the most suitable input vertex to start the construction. This issue has been solved by starting the construction, at least, once from every input vertex.

Then, we analyse the influence of the parameters w_1 and w_2 in g_1 which determine the selection of the following input vertices further than the first assignation (denoted as the answer to Question #2 in Section 2.1). Let us remember, that w_1 and w_2 balance the influence of the adjacent assigned/unassigned vertices respectively, for every input vertex being evaluated with $g1$. Particularly, in Fig. 6 we depict the average performance of the constructive procedure for five different configurations of these two parameters, when the number of constructions increases from 1 to 2500. Notice that in this experiment the input/host vertices of the first assignation are selected following the best configuration found in the previous experiment. As we can observe in the figure, the combination $w_1 = 1.00, w_2 = 0.00$ is systematically the best configuration and therefore it will be selected for future experiments. Since the sum of w_1 and w_2 equals 1, the selected configuration indicates that, for this problem, the value of g_1 is fully determined by the already assigned adjacent vertices to the vertex being evaluated. Furthermore, the benefits obtained by performing multiple constructions do not improve significantly after 1500 constructions.

Finally, we analyze the influence of the strategies proposed to select the host vertex in every assignation but the first (denoted as the answer to Question #3 in Section 2.1). Particularly, in Table 2, we evaluate the four strategies proposed for this task. The reported results are obtained as the average of the best solutions found for each instance after 1500 constructions. Again, the input/host vertices of the first assignation are selected following the best configuration found with the criteria previously defined, and g_1 is configured with $w_1 = 1$ and $w_2 = 0$.

On the one hand, according to the selection of the vertices of the host graph, g_2 is easily recognized as the best strategy since it finds the best quality solutions (lower average of the objective function, lower deviation, and larger number of best solutions found). Among the pattern-based strategies, zigzag is the most prominent.

To sum up, the final configuration of our constructive procedure has been set to be executed for 1500 constructions, and the best overall solution is selected. Each construction starts from a different initial input vertex (if all vertices have been used at least once, the procedure selects a repeated vertex to start with). The initial host vertex is set to be one of the corner vertices of the grid. The following input vertices are selected one by one with g_1 configured with $w_1 = 1$ and $w_2 = 0$. Finally, g_2 is selected as the method to determine the host vertices for any assignation performed after the first one.

Our next preliminary experiment is devoted to test the influence of the advanced strategies proposed in Section 3 in the local search procedure described in Section 2.2. First, we evaluated the exploration of the neighborhood defined by the move operator following both: a *first improvement* and a *best improvement* strategy. We found that both strategies reached the same average quality of the objective function (32.54) for the preliminary data set. However, the CPU time of the local search using a best improvement strategy was 5 times larger than using a *first improvement* strategy. Then, we configured our local search procedure with a *first improvement* strategy.

In Table 3, we report the results obtained when incorporating each of the three proposed advanced strategies to the local search procedure (i.e., the tiebreak criterion (T), the efficient move calculation (E), and the neighborhood reduction strategy (R)). We also include in the comparison the original local search procedure in isolation (LS). The results provided in the table are obtained

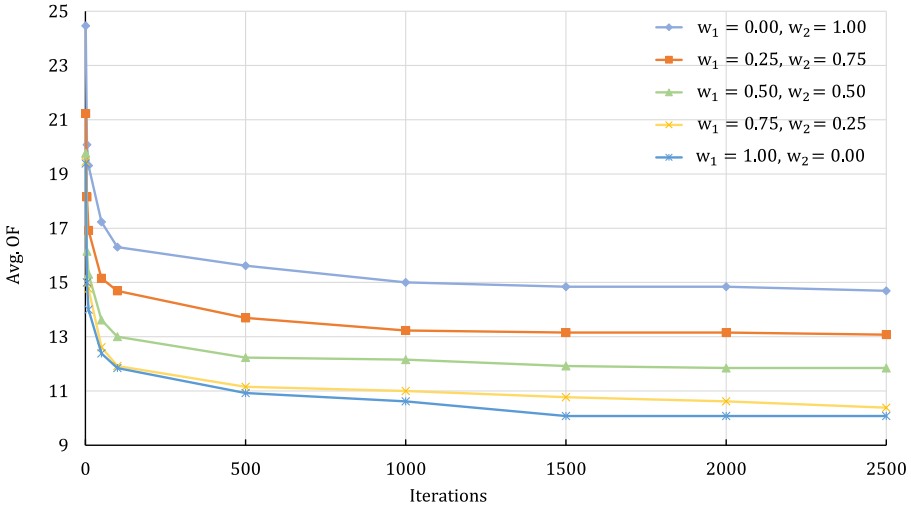


Fig. 6. Evolution of the average objective function value when increasing the number of constructions for different values of w_1 and w_2 in g_1.

Table 3
Contribution of advanced strategies to the local search.

	LS	LS+T	LS+T+E	LS+T+E+R
Avg. OF	32.54	7.77	7.77	7.85
CPU Time (s)	72.51	7873.28	22.76	2.18
Dev. (%)	350.04	4.62	4.62	7.05
#Best	0	8	8	9

after a single execution of each method, where the initial solution was the same for all compared methods and it was randomly constructed.

As we can observe in Table 3 the inclusion of the tiebreak criterion (LS+T) drastically improves the quality of the solutions obtained with respect to the original local search (LS) although the time increases considerably. This increase in the time needed by the method is due to the larger exploration of solutions performed by the LS+T. As expected, LS+T and LS+T+E were able to reach the same solutions in terms of quality, however, then the efficient move calculation reduces the time needed to reach them in 99.71%. Finally, the method including the proposed neighborhood reduction strategy, LS+T+E+R, is able to reduce by one order of magnitude the time needed by LS+T+E, slightly deteriorating the average quality of the solutions obtained. Both behaviors are explained by the fact that the number of solutions explored is considerably smaller. We consider that LS+T+E+R is the most promising combination as a balance between computing time and quality.

Next, we study the best procedures for the destruction and reconstruction phase. In this experiment, we analyse all possible combinations of the strategies proposed in Section 2.3. Specifically, we evaluate three destruction strategies: random assignations (random), assignations of random areas (random area), and assignations of areas contributing to the objective function of the solution (greedy area). Additionally, as far as the reconstruction phase is concerned, we evaluate three proposals: g_1 + random, random + g_2, and $g_1 + g_2$. Each proposal includes two strategies to determine the next assignation. The first strategy selects an input unassigned vertex, while the second strategy selects an available host vertex.

In Table 4, we present the results of this experiment. Particularly, each Iterated Greedy configuration has been executed for a maximum of 300 iterations, with the additional condition that the method is halted if it does not find an improvement of the best solution found in the last 150 iterations. The best configuration is obtained when the destruction is made greedily ("Greedy area" in the table) and the reconstruction is made by using the "random + g_2" criterion. The second best variant is the one where the destruction is made at random ("Random" in the table) and the reconstruction uses "random + g_2". However, this variant finds very similar solutions in terms of quality in half time. Therefore, as a trade-off between quality and time, we have selected this second configuration for our final proposed procedure.

Finally, it is important to remark that we performed a fine-tuning experiment to adjust the parameters: maximum number of iterations (*maxIter*) and the maximum number of iterations without improving (*maxNotImprIter*) introduced in the Algorithm 1. Particularly, we tested different values of *maxIter* in the range [100, 1000] in steps of 50. Similarly, we studied the behavior of *maxNotImprIter*) with different percentages (0.25, 0.5, and 0.75) of the *maxIter*. For the sake of brevity, we do not include all the values of this experiment in here. However, among the proposed configurations, we selected $maxIter = 300$ and $maxNotImprIter = 0.75 \cdot maxIter = 225$ for our final design, as a balance of quality and CPU time.

To conclude the preliminary experiments, we compare our three main algorithmic proposals to verify if an increase in the complexity of the method also results in an improvement in the obtained results. Specifically, we propose two executing scenarios: a single run of each method and running each method iteratively for 100 seconds. Notice that in this experiment, the solution produced by the greedy constructive is provided to the local search and to the Iterated Greedy procedure. The results obtained are reported in Table 5. As expected, in the single execution scenario, the IG is the best method in terms of average of the objective function, deviation and # Best solutions found. However, it is also the most time-consuming procedure. On the other hand, when running all methods for 100 seconds, the differences among the results obtained are reduced, but IG is still the best overall method.

Table 4
Influence of the destruction and reconstruction strategies in the performance of the greedy constructive procedure.

Destruction	Reconstruction	Avg. OF	CPU Time (s)	Dev. (%)	#Best
Random	g_1 + random	5.46	154.21	9.29	8
	random + g_2	5.23	60.20	3.85	11
	$g_1 + g_2$	5.46	43.13	7.37	8
Random area	g_1 + random	5.38	145.25	6.41	9
	random + g_2	5.31	110.44	4.81	10
	$g_1 + g_2$	5.46	65.79	7.37	8
Greeedy area	g_1 + random	5.31	147.90	4.81	10
	random + g_2	5.15	119.62	0.96	12
	$g_1 + g_2$	5.38	63.62	6.09	9

Table 5
Behaviour of the proposed strategies in a single execution and running for 100 seconds.

	Single execution			100 seconds		
	Constructive	LS+T+E+R	IG	Constructive	LS+T+E+R	IG
Avg. OF	10.08	6.00	5.23	9.08	5.38	5.23
CPU Time (s)	6.05	6.43	84.71	102.79	106.86	106.90
Dev. (%)	95.69	13.32	0.00	75.62	4.49	1.92
#Best	0	5	13	0	10	12

Table 6
Results obtained by the compared methods on the 41 instances of the Small graphs data set with a known optimal, and on the 45 instances of the Harwell-Boeing data set.

	Small graphs (41)			Harwell-Boeing (45)	
	M3 (Rodríguez-García et al., 2021)	BVNS (Rodríguez-García et al., 2021)	IG	BVNS (Rodríguez-García et al., 2021)	IG
Avg. OF	2.17	2.32	2.17	7.11	4.84
CPU Time (s)	1957.65	3.64	0.02	439.63	102.01
Dev. (%)	0.00	12.20	0.00	51.77	0.00
#Best	41	35	41	5	45

4.2. Final experiments

In this section, we compare our best Iterated Greedy (IG) variant with the best previous algorithms in the state of the art: the best Constraint Satisfaction Programming (CSP) model proposed in Rodriguez-Tello et al. (2019) (denoted M3) and the Basic Variable Neighborhood Search (BVNS) proposed in Rodríguez-García et al. (2021). Both procedures were described in the literature review. Notice that we have compared our procedure with the original source code implemented and provided by the authors. To make the fairest comparison possible, instead of directly using the results reported in Rodríguez-García et al. (2021), the BVNS procedure was run again with the configuration indicated by the authors, in the same execution environment as the one used for our code.

In Table 6 we report the quality indicators presented in the preliminary experiment: the average deviation, the average execution time, and the number of best solutions found for each of the subsets of instances studied: the diverse small graph subset and the Harwell-Boeing subset.

In particular, on the left side of Table 6, we compare our IG with the BVNS and M3 over the set of diverse small graphs. Note that M3 was unable to complete the search for 4 instances out of 45 within the established time limit (72h). Therefore, we have removed those instances from this comparison to fairly illustrate the behavior of the proposed algorithm. Additionally, in Table A.1 we include the individual results per instance for each of the 45 instances of the complete subset. We observe in Table 6 that M3 and IG were able to reach the optimal solution for the 41 instances studied, followed by BVNS with 35. However, the time required by IG was 5 orders of magnitude shorter than M3 and 2 orders of magnitude shorter than BVNS.

Similarly, on the right side of Table 6, we compare IG and BVNS over the Harwell-Boeing subset. In this case, M3 was not able to finish within the maximum time limit and therefore it has been excluded from this comparison. Again, to ease future comparisons, we include the individual of each instance in Table A.2. Based on the results reported in Table 6 we observe that IG finds the best solution for all the graphs studied (45) in less computational time (102.01 s) than the BVNS procedure (439.63 s). Consequently, the average value of the objective function is lower in the solutions obtained by IG than in the solutions obtained by BVNS. Finally, we highlight that BVNS has a 51.77% deviation from the best solutions found, obtaining only five best solutions out of 45 instances.

To complement the previous experiment, we conducted a Wilcoxon signed rank test. The resulting p-value < 0.00001 confirms the significance of the results obtained when comparing the methods for the tested instances.

5. Conclusions

In this paper, we tackle the Two-Dimensional Bandwidth Minimization Problem by proposing several efficient heuristic strategies to find high-quality solutions for the problem. The 2DBMP belongs to the graph layout family of problems, and it has been previously approached from an exact perspective, based on Constraint Satisfaction Programming, and from a heuristic perspective, based on the Variable Neighborhood Search metaheuristic.

We have developed an efficient and effective Iterated Greedy algorithm to deal with the 2DBMP, including an exhaustive study of multiple greedy criteria at the destruction and reconstruction steps within the IG framework. In addition, we introduce a novel local search procedure based on swap moves of vertices, which includes three advanced enhancement strategies. It is worth mentioning that several of the strategies proposed in this paper have further

applicability to other optimization problems, especially those related to Graph Layout Problems.

The results obtained in this paper emphasize the importance of using a tiebreak criterion to guide the search through flat landscape regions. This is a key strategy when the objective function is not useful in distinguishing between two solutions with the same objective function value. Also, we identified that classical move operators applied to Graph Layout Problems, such as the 2DBMP, usually drive to extensive neighborhoods. In this kind of scenario, local search procedures might be inefficient when the time limit is short. To overcome this drawback, we propose two general strategies with applicability to other problems: a speed-up technique to evaluate the objective function of neighbor solutions; and a neighborhood reduction technique based on the exploration of the most promising neighbor solutions. Moreover, the graph structure of either the input and host graphs is key in determining the best heuristic strategy in the context of GLPs. Particularly, the number of adjacent vertices of each vertex tends to contribute to relevant information at the time of constructing new solutions.

Finally, we would like to highlight that the best algorithmic variant of our proposal has been compared to the best previous method in the state of the art, over a previously reported set of instances. The obtained results, supported by statistical tests, corroborate the merit of our proposal and establish it as a new state-of-the-art algorithm for the 2DBMP.

Acknowledgment

This research has been partially supported by the Ministerio de Ciencia, Innovación y Universidades (Grant Ref. PGC2018-095322-B-C22, PID2021-125709OA-C22 and FPU19/04098) and by the Comunidad de Madrid and the European Regional Development Fund (Grant Ref. P2018/TCS-4566). We also thank M.A. Rodrguez et al., authors of the previous most competitive method in the state of the art Rodríguez-García et al. (2021) for sharing their code with us.

Appendix A. Individual results per instance

In Table A.1 and Table A.2 we report the individual results per instance for the Small and Harwell-Boeing data sets. These values were used to calculate the values presented in Table 6.

Table A.1
Individual results per instance obtained from the small data set. Note that a symbol "-" in the table indicates that the algorithm was not able to solve the instance in a maximum CPU time of 72h.

		M3			BVNS			IG		
Instance	Best	OF	CPU Time (s)	Dev. (%)	OF	CPU Time (s)	Dev. (%)	OF	CPU Time (s)	Dev. (%)
p2p3	**1**	1	0.20	0.00	2	0.11	1.00	1	0.01	0.00
p3p3	**1**	1	0.26	0.00	2	0.38	1.00	1	0.01	0.00
p4p5	**1**	1	0.22	0.00	2	7.84	1.00	1	0.03	0.00
p2c3	**2**	2	0.24	0.00	2	0.16	0.00	2	0.01	0.00
p3c3	**2**	2	0.20	0.00	2	0.54	0.00	2	0.01	0.00
p4c5	**2**	2	0.53	0.00	3	7.55	0.50	2	0.03	0.00
c3c3	**2**	2	0.24	0.00	2	0.83	0.00	2	0.02	0.00
c3c4	**2**	2	0.28	0.00	2	2.13	0.00	2	0.03	0.00
c4c5	**2**	2	0.28	0.00	3	9.26	0.50	2	0.04	0.00
k3k4	**3**	3	0.55	0.00	3	2.59	0.00	3	0.02	0.00
k4k5	**4**	4	74192.71	0.00	4	20.00	0.00	4	0.05	0.00
c3k4	**3**	3	0.60	0.00	3	2.83	0.00	3	0.03	0.00
c4k5	**3**	3	2.18	0.00	3	17.86	0.00	3	0.05	0.00
p3k4	**2**	2	0.27	0.00	2	2.58	0.00	2	0.02	0.00
p4k5	**3**	3	1.93	0.00	3	14.81	0.00	3	0.04	0.00
path10	**1**	1	0.30	0.00	1	0.44	0.00	1	0.01	0.00
path15	**1**	1	0.35	0.00	2	1.32	1.00	1	0.02	0.00
path20	**1**	1	0.31	0.00	1	4.04	0.00	1	0.02	0.00
cycle10	**1**	1	0.24	0.00	1	0.40	0.00	1	0.01	0.00
cycle15	**2**	2	0.50	0.00	2	1.19	0.00	2	0.02	0.00
cycle20	**1**	1	0.28	0.00	1	4.30	0.00	1	0.03	0.00
wheel5	**2**	2	0.21	0.00	2	0.12	0.00	2	0.00	0.00
wheel7	**2**	2	0.23	0.00	2	0.35	0.00	2	0.01	0.00
wheel10	**2**	2	0.41	0.00	2	1.17	0.00	2	0.02	0.00
wheel15	**3**	3	4444.86	0.00	3	4.11	0.00	3	0.02	0.00
wheel20	3	-	-	-	3	11.93	0.00	4	0.03	0.33
cyclePow10-2	**2**	2	0.25	0.00	2	1.12	0.00	2	0.02	0.00
cyclePow15-2	**2**	2	0.28	0.00	2	3.58	0.00	2	0.02	0.00
cyclePow20-2	**2**	2	0.30	0.00	2	9.15	0.00	2	0.04	0.00
cyclePow10-10	**4**	4	0.21	0.00	4	3.74	0.00	4	0.02	0.00
cyclePow15-10	6	-	-	-	6	15.01	0.00	6	0.04	0.00
cyclePow20-10	6	-	-	-	6	20.00	0.00	6	0.08	0.00
bipartite3-3	**2**	2	0.26	0.00	2	0.18	0.00	2	0.01	0.00
bipartite3-4	**3**	3	0.33	0.00	3	0.28	0.00	3	0.02	0.00
bipartite4-4	**3**	3	0.33	0.00	3	0.56	0.00	3	0.02	0.00
bipartite5-5	**3**	3	0.78	0.00	3	1.46	0.00	3	0.02	0.00
bipartite7-8	**4**	4	1050.31	0.00	4	9.03	0.00	4	0.04	0.00
bipartite10-10	5	-	-	-	5	20.00	0.00	5	0.21	0.00
petersen	**2**	2	0.31	0.00	2	0.77	0.00	2	0.01	0.00
complete5	**2**	2	0.28	0.00	2	0.12	0.00	2	0.01	0.00
complete10	**4**	4	14.03	0.00	4	4.15	0.00	4	0.02	0.00
tree2-2	**1**	1	0.31	0.00	1	0.11	0.00	1	0.01	0.00
tree2-3	**2**	2	0.31	0.00	2	0.99	0.00	2	0.01	0.00
tree3-2	**2**	2	0.25	0.00	2	1.25	0.00	2	0.02	0.00
tree2-4	**2**	2	546.79	0.00	2	5.93	0.00	2	0.04	0.00

Table A.2
Individual results per instance obtained from the Harwell-Boeing data set.

		BVNS (Rodríguez-García et al., 2021)			IG		
Instance	Best	OF	CPU Time (s)	Dev. (%)	OF	CPU Time (s)	Dev. (%)
bcsstk01	5	5	48.00	0.00	5	0.35	0.00
can___62	2	3	62.00	50.00	2	0.22	0.00
nos4	4	4	100.01	0.00	4	1.13	0.00
bcspwr03	3	4	118.01	33.33	3	0.97	0.00
bcsstk04	8	9	132.01	12.50	8	33.24	0.00
bcsstk22	3	4	138.01	33.33	3	1.76	0.00
can__144	4	5	144.01	25.00	4	2.84	0.00
bcsstk05	7	7	153.01	0.00	7	13.82	0.00
can__161	4	6	161.01	50.00	4	4.31	0.00
dwt__198	4	5	198.01	25.00	4	6.49	0.00
dwt__209	5	6	209.01	20.00	5	12.95	0.00
dwt__221	4	5	221.01	25.00	4	7.26	0.00
can__229	5	7	229.01	40.00	5	11.65	0.00
dwt__234	4	4	234.01	0.00	4	11.46	0.00
nos1	3	4	237.01	33.33	3	4.68	0.00
dwt__245	4	6	245.02	50.00	4	8.69	0.00
lshp_265	3	6	265.00	100.00	3	9.31	0.00
bcspwr04	4	7	274.02	75.00	4	13.05	0.00
ash292	4	6	292.00	50.00	4	9.84	0.00
can__292	6	7	292.00	16.67	6	31.82	0.00
dwt__307	5	7	307.00	40.00	5	25.23	0.00
dwt__310	4	5	310.00	25.00	4	10.78	0.00
dwt__361	5	8	361.01	60.00	5	25.45	0.00
plat362	7	8	362.01	14.29	7	110.03	0.00
bcsstk07	6	9	420.01	50.00	6	298.42	0.00
bcspwr05	5	5	443.01	0.00	5	29.52	0.00
can__445	7	9	445.01	28.57	7	59.74	0.00
bcsstk20	4	6	485.01	50.00	4	39.75	0.00
494_bus	5	6	494.01	20.00	5	41.65	0.00
dwt__503	6	8	503.01	33.33	6	83.16	0.00
lshp_577	5	8	577.00	60.00	5	63.92	0.00
dwt__607	5	9	607.00	80.00	5	107.66	0.00
662_bus	5	7	662.01	40.00	5	56.83	0.00
nos6	5	14	960.01	180.00	5	49.94	0.00
685_bus	5	8	685.01	60.00	5	51.32	0.00
can__715	8	11	715.01	37.50	8	698.93	0.00
nos7	6	10	729.01	66.67	6	174.52	0.00
dwt__758	5	7	758.01	40.00	5	127.15	0.00
lshp_778	4	9	778.01	125.00	4	142.28	0.00
bcsstk19	6	9	817.00	50.00	6	399.65	0.00
dwt__878	5	9	878.00	80.00	5	192.83	0.00
gr_30_30	2	9	900.01	350.00	2	45.41	0.00
dwt__918	6	9	918.01	50.00	6	431.28	0.00
nos2	4	6	957.01	50.00	4	106.14	0.00
nos3	7	14	960.01	100.00	7	1032.95	0.00

References

Abdinnour-Helm, S., & Hadley, S. W. (2000). Tabu search based heuristics for multi--floor facility layout. *International Journal of Production Research, 38*(2), 365–383.

Bezrukov, S. L., Chavez, J. D., Harper, L. H., Röttger, M., & Schroeder, U. P. (1998). Embedding of hypercubes into grids. In L. Brim, J. Gruska, & J. Zlatuka (Eds.), *Mathematical foundations of computer science 1998*. In *Lecture notes in computer science* (pp. 693–701). Berlin, Heidelberg: Springer.

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

Booch, G. (2005). *The unified modeling language user guide*. Pearson Education India.

Cavero, S., Pardo, E. G., & Duarte, A. (2021a). Influence of the alternative objective functions in the optimization of the cyclic cutwidth minimization problem. In *Advances in artificial intelligence*. In *Lecture notes in computer science* (pp. 139–149). Cham: Springer International Publishing.

Cavero, S., Pardo, E. G., Laguna, M., & Duarte, A. (2021b). Multistart search for the cyclic cutwidth minimization problem. *Computers & Operations Research, 126*, 105116.

Cavero, S., Pardo, E. G., & Duarte, A. (2022a). A general variable neighborhood search for the cyclic antibandwidth problem. *Computational Optimization and Applications, 81*(2), 657–687.

Cavero, S., Pardo, E. G., Duarte, A., & Rodriguez-Tello, E. (2022b). A variable neighborhood search approach for cyclic bandwidth sum problem. *Knowledge-Based Systems, 246*, 108680.

Chung, F. (1988). Labelings of graphs. *Selected topics in graph theory, 3*, 151–168.

Craw, S. (2010). *Manhattan distance* (pp. 639–639)). Boston, MA: Springer US.

Del Corso, G. M., & Manzini, G. (1999). Finding exact solutions to the bandwidth minimization problem. *Computing, 62*(3), 189–203.

Delmaire, H., Díaz, J. A., Fernández, E., & Ortega, M. (1999). Reactive grasp and tabu search based heuristics for the single source capacitated plant location problem. *INFOR: Information Systems and Operational Research, 37*(3), 194–225.

Díaz, J., Petit, J., & Serna, M. (2002). A survey of graph layout problems. *ACM Computing Surveys (CSUR), 34*(3), 313–356.

Feo, T. A., & Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization, 6*(2), 109–133.

Gurari, E. M., & Sudborough, I. H. (1984). Improved dynamic programming algorithms for bandwidth minimization and the mincut linear arrangement problem. *Journal of Algorithms, 5*(4), 531–546.

Hansen, P., & Mladenović, N. (2006). First vs. best improvement: An empirical study. *Discrete Applied Mathematics, 154*(5), 802–817.

Hansen, P., Mladenović, N., Todosijević, R., & Hanafi, S. (2017). Variable neighborhood search: Basics and variants. *EURO Journal on Computational Optimization, 5*(3), 423–454.

Hromkovič, J., Müller, V., Sỳkora, O., & Vrt'o, I. (1992). On embedding interconnection networks into rings of processors. In *International conference on parallel architectures and languages Europe* (pp. 51–62). Springer.

Huerta-Muñoz, D. L., Ríos-Mercado, R. Z., & Ruiz, R. (2017). An iterated greedy heuristic for a market segmentation problem with multiple attributes. *European Journal of Operational Research, 261*(1), 75–87.

Jinjiang, Y., & Sanming, Z. (1995). Optimal labelling of unit interval graphs. *Applied Mathematics, 10*(3), 337–344.

Lai, Y.-L., & Williams, K. (1999). A survey of solved problems and applications on bandwidth, edgesum, and profile of graphs. *Journal of Graph Theory, 31*(2), 75–94.

Lin, L., & Lin, Y. (2010). Two models of two-dimensional bandwidth problems. *Information Processing Letters, 110*(11), 469–473.

Lin, L., & Lin, Y. (2011). Square-root rule of two-dimensional bandwidth problem. *RAIRO-Theoretical Informatics and Applications, 45*(4), 399–411.
Lin, Y. (1994). The cyclic bandwidth problem. *Journal of Systems Science and Complexity, 7*(3), 282–288. Cited By 12
Lourenço, H. R., Martin, O. C., & Stützle, T. (2003). Iterated local search. In *Handbook of metaheuristics* (pp. 320–353). Springer.
López-Sánchez, A., Sánchez-Oro, J., & Hernández-Díaz, A. (2019). GRASP and VNS for solving the p-next center problem. *Computers & Operations Research, 104*, 295–303.
Martí, R., Pantrigo, J. J., Duarte, A., & Pardo, E. G. (2013). Branch and bound for the cutwidth minimization problem. *Computers & Operations Research, 40*(1), 137–149.
McAllister, A. et al. (1999). A new heuristic algorithm for the linear arrangement problem.
Michiels, W., Aarts, E. H., & Korst, J. (2018). Theory of local search. In *Handbook of heuristics* (pp. 299–339). Springer.
Mladenovic, N., Urosevic, D., Pérez-Brito, D., & García-González, C. G. (2010). Variable neighbourhood search for bandwidth reduction. *European Journal of Operational Research, 200*(1), 14–27.
Papadimitriou, C. H. (1976). The np-completeness of the bandwidth minimization problem. *Computing, 16*(3), 263–270.
Pardo, E. G., García-Sánchez, A., Sevaux, M., & Duarte, A. (2020). Basic variable neighborhood search for the minimum sitting arrangement problem. *Journal of Heuristics, 26*(2), 249–268.
Pardo, E. G., Martí, R., & Duarte, A. (2016). Linear layout problems. In R. Martí, P. Panos, & M. G. Resende (Eds.), *Handbook of heuristics* (pp. 1–25). Cham: Springer International Publishing.
Pardo, E. G., Mladenović, N., Pantrigo, J. J., & Duarte, A. (2013). Variable formulation search for the cutwidth minimization problem. *Applied Soft Computing, 13*(5), 2242–2252.
Pérez-Peló, S., Sánchez-Oro, J., Gonzalez-Pardo, A., & Duarte, A. (2021). A fast variable neighborhood search approach for multi-objective community detection. *Applied Soft Computing, 112*, 107838.
Petit, J. (2004). Experiments on the minimum linear arrangement problem. *ACM Journal of Experimental Algorithmics, 8*, 2.3.
Ren, J., Hao, J.-K., & Rodriguez-Tello, E. (2019). An iterated three-phase search approach for solving the cyclic bandwidth problem. *IEEE Access, 7*, 98436–98452.
Ren, J., Hao, J.-K., Rodriguez-Tello, E., Li, L., & He, K. (2020). A new iterated local search algorithm for the cyclic bandwidth problem. *Knowledge-Based Systems, 203*, 106136.
Rodriguez-Tello, E., Hao, J.-K., & Torres-Jimenez, J. (2008a). An effective two-stage simulated annealing algorithm for the minimum linear arrangement problem. *Computers & Operations Research, 35*(10), 3331–3346.
Rodriguez-Tello, E., Hao, J.-K., & Torres-Jimenez, J. (2008b). An improved simulated annealing algorithm for bandwidth minimization. *European Journal of Operational Research, 185*(3), 1319–1335.
Rodriguez-Tello, E., Narvaez-Teran, V., & Lardeux, F. (2019). Dynamic multi-armed bandit algorithm for the cyclic bandwidth sum problem. *IEEE Access, 7*, 40258–40270.
Rodríguez-García, M. A., Sánchez-Oro, J., Rodriguez-Tello, E., Monfroy, E., & Duarte, A. (2021). Two-dimensional bandwidth minimization problem: Exact and heuristic approaches. *Knowledge-Based Systems, 214*, 106651.
Ruiz, R., & Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research, 177*(3), 2033–2049.
Stützle, T., & Ruiz, R. (2018). *Iterated greedy* (pp. 547–577)). Cham: Springer International Publishing.
Tsang, E. (2014). *Foundations of constraint satisfaction: The classic text*. Books on Demand.

Chapter 11

Other related publications

This chapter collects some additional publications, and papers under review, which are not part of this Doctoral Thesis, but they are closely related and has being performed during the period that the Ph.D. candidate was performing his Doctoral Thesis.

These articles and publications provide relevant information on related topics and can be a useful complement to the main conclusions of the Doctoral Thesis, even if they are not directly related to the research conducted. The purpose of this chapter is to provide the reader with a sample of these articles and publications for further reading and reference. Next, we present the references to the articles along with a brief description.

1. M. Robles, S. Cavero, and E. G. Pardo. BVNS for the Minimum Sitting Arrangement problem in a cycle. In *Advances in Artificial Intelligence: 9th International Conference on Variable Neighborhood Search (ICVNS2022), in Abu Dhabi, U.A.E., 2022.* In press.

In this research, we study a particular CGLP where the input graph is denoted as a "signed graph". This kind of graphs is characterized by adding a weight to the edges of the graph (in this case, the weight can be +1/-1). This input graph is embedded in a cycle host graph trying to minimize the number of negative adjacent vertices closer than a positive adjacent vertex, for each vertex of the input graph. Specifically, when such a situation occurs, it is denoted as "error". Finally, the objective is to minimize the sum of errors of an embedding. This problem is a variant of the Minimum Sitting Arrangement Problem [190].

This work is a preliminary investigation and was submitted to the *9th International Conference on Variable Neighborhood Search*, (ICNVS), held in Abu Dhabi (U.A.E). The article has been accepted for publication and it is currently in press at the journal *Advances in Artificial Intelligence* as part of the collection *Lecture Notes in Computer Science* (SJR / Q2).

2. S. Cavero, E. G. Pardo, F. Glover and R. Martí. SOS Tabu Search for Improved Hierarchical Graph Drawing. *Expert Systems With Applications*. Under review.

As mentioned in Section 1.2.3, one of the main applications of GLPs is the drawing of graphs. During the international stay at the University of Colorado, a collaboration with two renowned researchers on hierarchical graph drawings was performed. This problem, although it does not belong to the GLP family, is closely related to them and many of the strategies and algorithms can be transferred to this area.

In particular, in this research we tackle the drawing of hierarchical graphs that consist of representing a graph based on the alignment of its edges, while reducing the number of crossing. The work developed was collected in an article that has been submitted to the journal *Expert Systems With Applications* (JCR / Q1) and, at the time of writing this dissertation, is under review.

3. R. Martín-Santamaría, S. Cavero, A. Herrán, A. Duarte, and J. M. Colmenar. A practical methodology for reproducible experimentation: an application to the double-row facility layout problem. *Evolutionary Computation*, 1–35, 2022 [161].

In this research, we do not work on any problem belonging to the GLP family, but we propose a methodology that promotes reproducible experimentation. Therefore, despite the fact that from the point of view of the problems approached, it does not have a close relationship with GLPs, from the methodological point of view it is of great interest and the knowledge learned in this research was applied in the projects developed later.

This work has been developed with colleagues from the group GRAFO and it proposes a practical methodology to favor reproducibility in optimization problems tackled with stochastic methods, such as the one addressed in this Doctoral Thesis. This methodology is organized into three main steps, in which the researcher is helped by specific tools that

use state-of-the-art methods for this process. In particular, in this research, we work on concepts that in this Doctoral Thesis have been given special importance such as parameter adjustment, and the selection of a preliminary set of instances or metrics to compare the performance of various algorithms.

The article has been published in "*Evolutionary Computation journal*", indexed in the JCR index under the categories "*Computer Science, Artificial Intelligence*" (Q2) and "*Computer Science, Theory and Methods*" (Q1) [161].

Part III

Appendix

Appendix A

Example of solution visualizations

There are two main objectives when visualizing a solution in the context of this Doctoral Thesis. The first one is to graphically analyze the process of construction and improvement of solutions until the final solution is reached. For example, Figure A.1 illustrates the evolution of the search process when moving from the solution obtained from greedy constructive procedure until the best solution found for the CCMP is reached.

The second objective is two to illustrate the solutions that may be optimal, or may be thought to be optimal, in order to analyze their structure and find possible properties that may be useful for proposing more effective methods. Figure A.2 illustrates the best solutions found for three instances of the CAB.

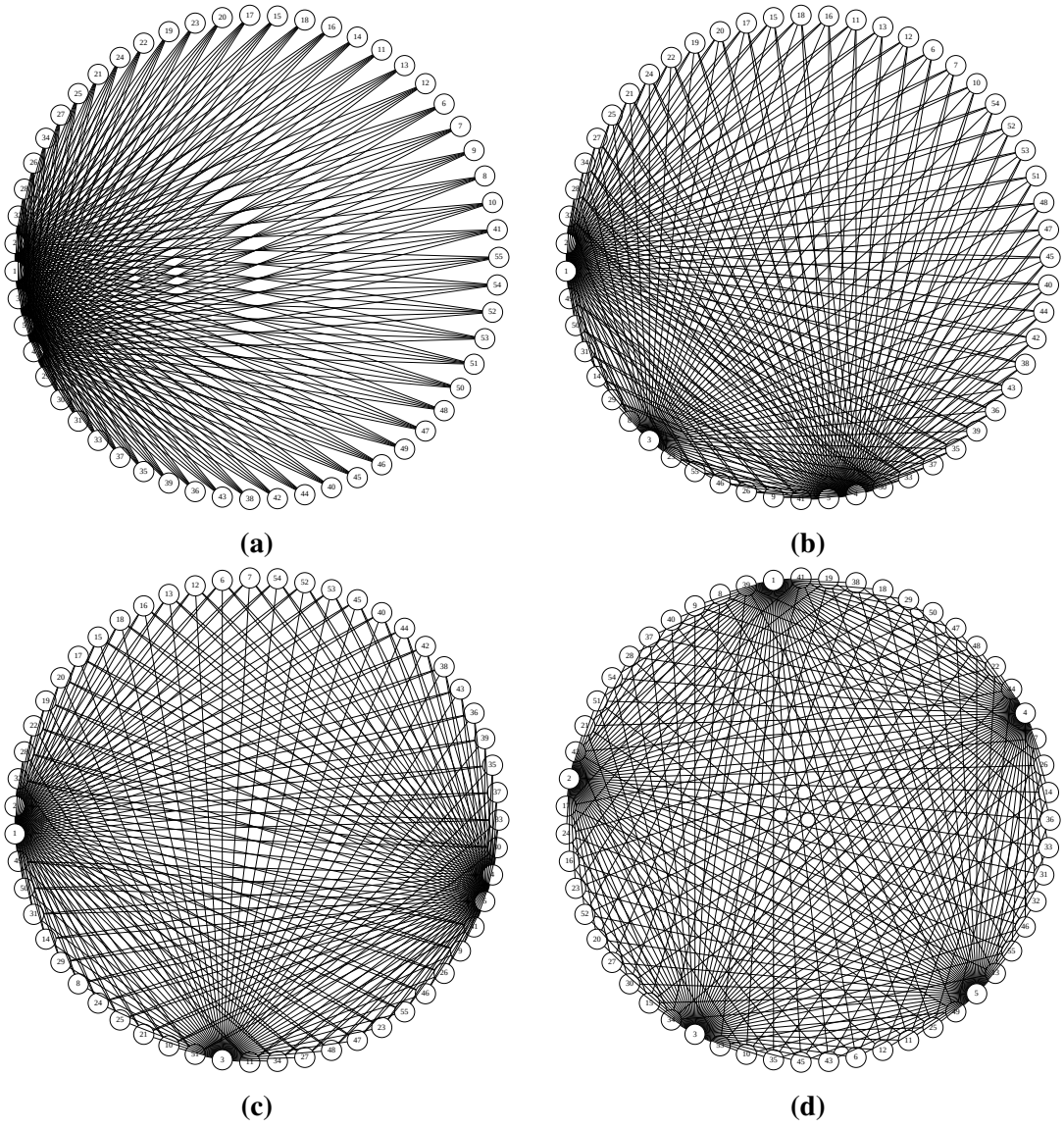


Figure A.1 Solutions obtained after the construction phase (a) and during the improvement process (b), (c), and (d) of the *CompleteSplit* graph when solving the Cyclic Cutwidth Minimization Problem. This graph has 55 vertices and 260 edges. The best objective function value (*ccw*) found is 68 corresponding to solution (d).

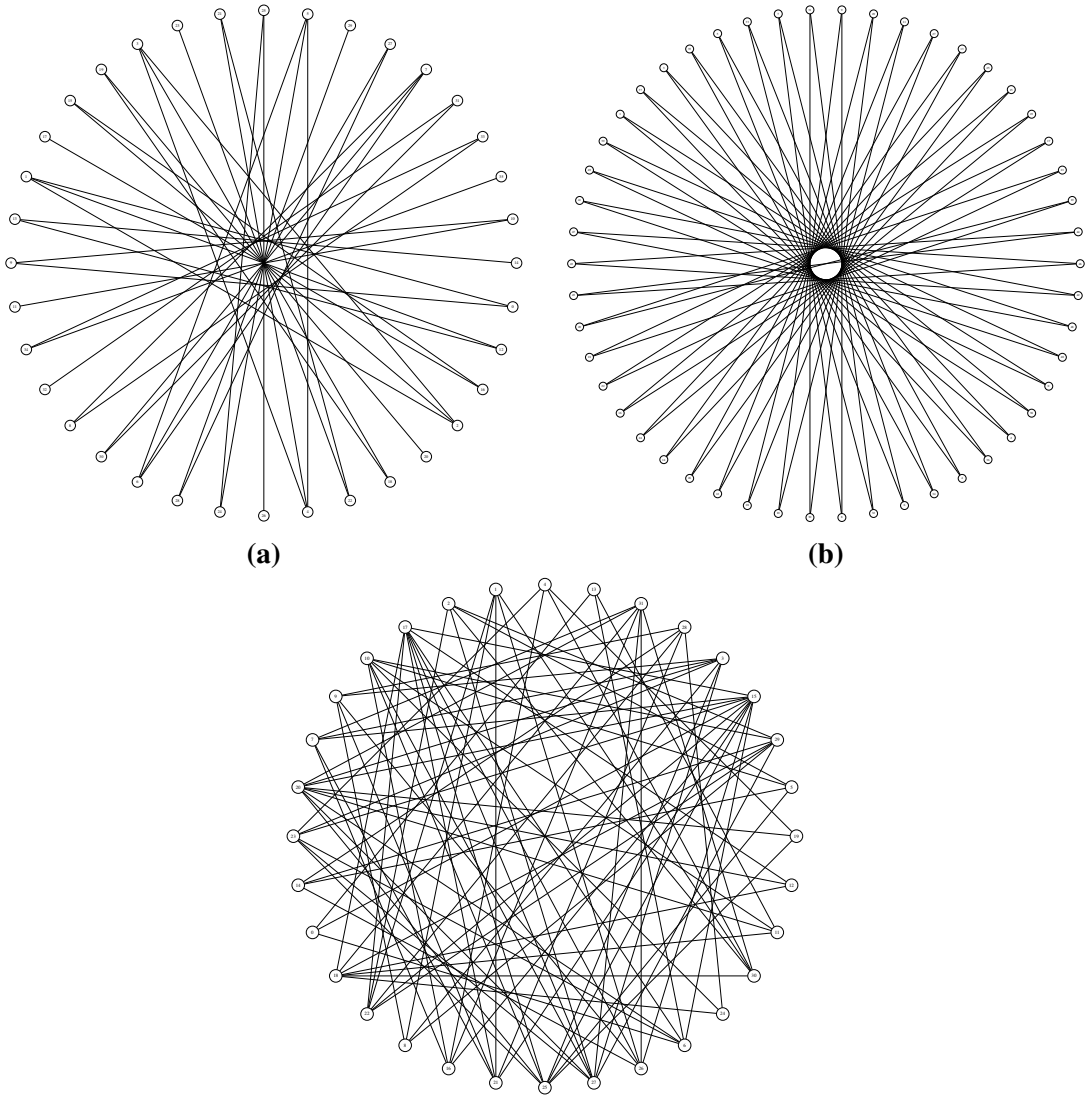


Figure A.2 Example of solutions for the caterpillar_9_4 (a), cycle_50 (b) and ibm32 (c) graphs when solving the Cyclic Antibandwidth Problem.

Appendix B

Resumen en castellano

El desarrollo de esta Tesis Doctoral se enmarca en la disciplina de la optimización heurística. Esta disciplina aborda la búsqueda de la mejor solución posible a un problema de optimización, de manera aproximada. Un problema de optimización se define como la maximización o minimización de una o varias funciones objetivo, donde las soluciones satisfacen un conjunto de restricciones. La optimización heurística tiene especial interés en la resolución de problemas computacionalmente difíciles (como son aquellos pertenecientes a la clase de complejidad NP-difícil) con aplicación práctica en industria, transporte, ingeniería y otras disciplinas. El tratamiento de estos problemas requiere un modelado previo para poder ser tratados computacionalmente. En este sentido, muchos de los problemas de optimización son modelados mediante grafos, que son estructuras de datos comúnmente utilizadas en medios computacionales para modelar sistemas de la vida real y otras abstracciones. Entre los problemas de optimización modelados mediante grafos se encuentra una familia de problemas denominada como problemas de embebido o etiquetado de grafos (GLP, del inglés, *Graph Layout Problem*) [65], los cuales han sido ampliamente estudiados en la literatura debido al gran número de aplicaciones prácticas que tienen. Ejemplo de estas aplicaciones son: el diseño de circuitos, la gestión del espectro de retransmisión de redes inalámbricas, el dibujado de grafos o la migración de redes de telecomunicaciones [191]. Sin embargo, dado que esta familia de problemas pertenece a la clase de problemas NP-difícil, la resolución de los mismos mediante técnicas exactas, cuando el tamaño de la entrada es grande, resulta completamente ineficiente. Por esa razón, esta Tesis Doctoral

se centra en el estudio de los problemas de embebidos de grafos desde un punto de vista aproximado, concretamente mediante algoritmos heurísticos y metaheurísticos.

A continuación, en la Sección B.1 se realiza una introducción a los problemas de embebido de grafos. En la Sección B.2 se recogen los trabajos más relevantes del área. Seguidamente, en la Sección B.3, se presentan los objetivos y la hipótesis planteada. La metodología propuesta en la Sección B.4 y los elementos principales de la propuesta algorítmica planteada en la Sección B.5. Por último, en la Sección B.6, se recogen los resultados más relevantes de esta investigación, concluyendo, en la Sección B.7, con las principales conclusiones.

B.1 Introducción

Esta Tesis Doctoral se centra en el estudio de problemas de optimización combinatoria utilizando técnicas heurísticas. En concreto, se estudian los denominados, *Graph Layout Problem* (GLP), consistentes en proyectar un grafo de entrada en otro grafo, generalmente denominado grafo huésped. Esta proyección es generalmente conocida como embebido o etiquetado, y consiste en definir dos funciones matemáticas. La primera de ellas relaciona los vértices del grafo de entrada con los vértices del grafo huésped. La segunda asigna a cada arista del grafo de entrada un camino en el grafo huésped (es decir, un conjunto de aristas). Pese a que cualquier grafo puede ser utilizado como grafo huésped, son aquellos que presentan una topología conocida (líneas, ciclos, árboles, rejillas, etc.) los que han despertado un mayor interés en la comunidad científica.

Formalmente, un grafo de entrada es denotado como $G = (V_G, E_G)$ donde V_G y E_G representan el conjunto de vértices y aristas respectivamente. De manera similar, un grafo huésped es denotado como $H = (V_H, E_H)$ donde V_H y E_H representan el conjunto de vértices y aristas, respectivamente.

A continuación, se ilustran los conceptos explicados hasta el momento. En concreto, la Figura B.1(a) muestra un grafo de entrada G, con $V_G = \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}, \mathrm{E}\}$ y $E_G = \{(\mathrm{A}, \mathrm{B}), (\mathrm{A}, \mathrm{C}), (\mathrm{A}, \mathrm{E}), (\mathrm{B}, \mathrm{C}), (\mathrm{C}, \mathrm{D}), (\mathrm{C}, \mathrm{E}), (\mathrm{D}, \mathrm{E})\}$. Por otro lado, las Figuras B.1(b), B.1(c) y B.1(d) muestran, respectivamente, ejemplos de grafos con estructura de camino, ciclo y rejilla que podrían ser utilizados como grafos huésped para el grafo de entrada en la Figura

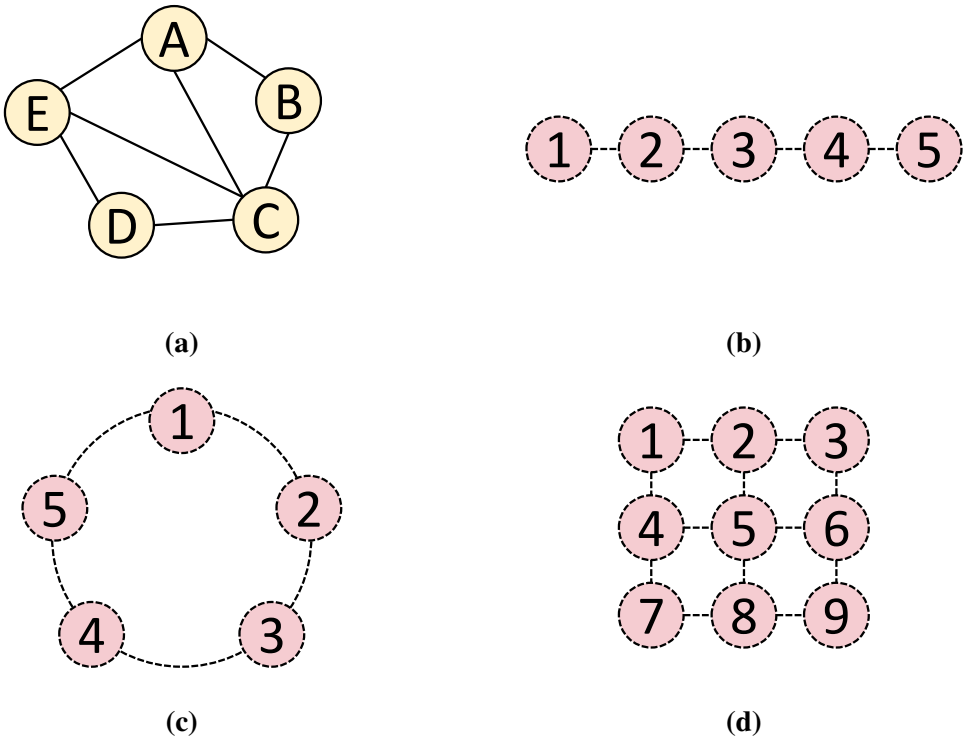


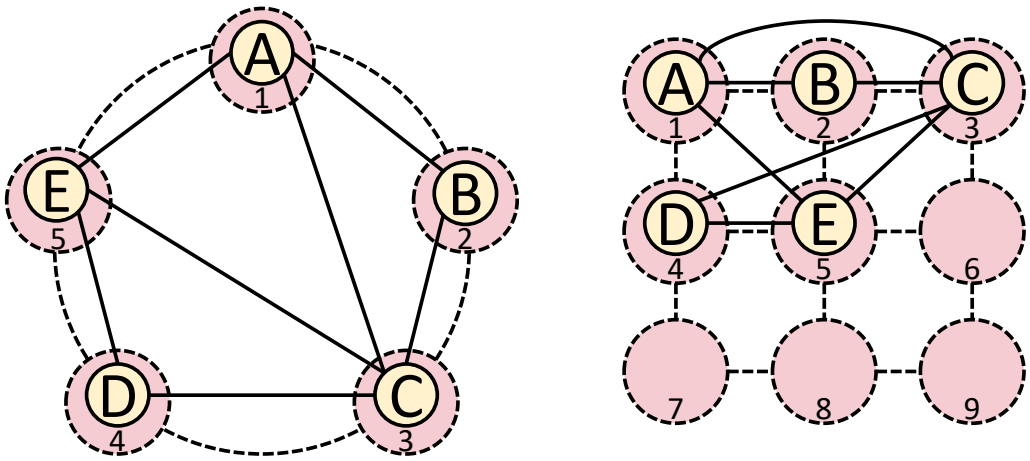
Figura B.1 (a) Ejemplo de un grafo de entrada G formado por 5 vértices y 7 aristas. (b) Ejemplo de un grafo huésped camino, P_H. (c) Ejemplo de un grafo huésped ciclo, C_H. (d) Ejemplo de un grafo huésped rejilla, G_H.

B.1(a). En particular, el grafo huésped camino, denotado como H_P, es ilustrado en la Figura B.1(b) y está formado por $V_{H_P} = \{1,2,3,4,5\}$ y $E_{H_P} = \{(1,2),(2,3),(3,4),(4,5)\}$. Del mismo modo, el grafo huésped de ciclo, denotado como H_C, representado en B.1(c) está formado por $V_{H_C} = \{1,2,3,4,5\}$ y $E_{H_C} = \{(1,2),(2,3),(3,4),(4,5),(5,1)\}$. Por último, el grafo huésped rejilla, denotado H_G, representado en B.1(d) está formado por $V_{H_G} = \{1,2,3,4,5,6,7,8,9\}$ y $E_{H_C} = \{(1,2),(1,4),(2,3),(2,5),(3,6),(4,5),(4,7),(5,6),(5,8),(6,9),(7,8),(8,9)\}$.

Dado un grafo de entrada, G y un grafo huésped H, un embebido o proyección consiste en definir dos funciones matemáticas. La primera función relaciona los vértices del conjunto V_G con los vértices del conjunto V_H, de modo que, a cada vértice del conjunto V_G le corresponde un único vértice del conjunto V_H y viceversa. Formalmente φ es una función biyectiva tal que $\varphi : V_G \rightarrow V_H$. La segunda función asigna aristas del conjunto E_G

a caminos del grafo huésped, denotados como P_H. Formalmente, esta segunda función, denotada como ψ es una función inyectiva tal que $\psi : E_G \rightarrow P_H$. La definición de la función ψ depende del problema a abordar. Generalmente, ψ asigna una arista $(u, v) \in E_G$, al camino en P_H con extremos en $\varphi(u)$ y $\varphi(v)$, que tiene una menor cardinalidad.

En la Figura B.2 se muestran dos posibles embebidos del grafo G de la Figura B.1(a), en un grafo ciclo, Figura B.2(a), y en un grafo rejilla, Figura B.2(b). La definición de φ en ambos ejemplos es equivalente: $\varphi = \{\varphi(\mathrm{A}) = 1, \varphi(\mathrm{B}) = 2, \varphi(\mathrm{C}) = 3, \varphi(\mathrm{D}) = 4, \varphi(\mathrm{E}) = 5\}$. La definición de ψ depende del problema estudiado. A modo de ejemplo, ψ asignará el camino con menor cardinalidad a cada arista del grafo de entrada. Considerando el grafo huésped ciclo, la arista (C,E) es asignada al camino más corto entre los vértices $\varphi(\mathrm{C}) = 3$ y $\varphi(\mathrm{E}) = 5$. Concretamente, $\psi(\mathrm{C},\mathrm{E}) = \{(3,4),(4,5)\}$. Considerando el grafo huésped rejilla, en este caso hay dos posibles caminos con la menor cardinalidad entre los vértices asignados a C y E, $\psi(\mathrm{C},\mathrm{E}) = \{\{(3,6),(6,5)\},\{(3,2),(2,5)\}\}$.



(a) **(b)**

Figura B.2 Ejemplo de un embebdido en un grafo huésped ciclo y un grafo huésped de rejilla (b).

Esta Tesis Doctoral se centra en el estudio de varios GLPs donde el embebido se realiza en ciclos. De esta manera, al centrar la investigación en un conjunto de problemas que guardan una estrecha relación es posible detectar propiedades similares y aplicar el

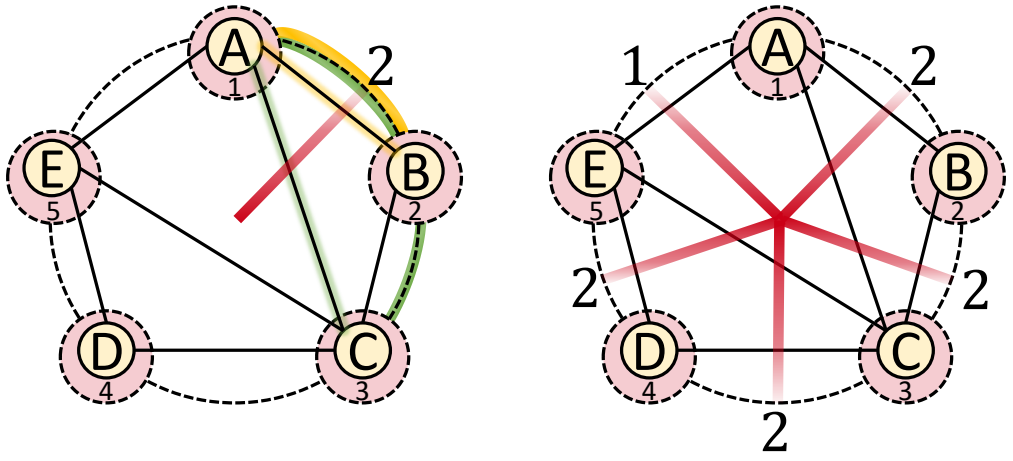
conocimiento adquirido a medida que avanza el proceso de investigación. Además, las estrategias o algoritmos propuestos pueden ser adaptados con mayor facilidad a problemas pertenecientes a la misma familia. Para determinar los problemas a abordar en la Tesis Doctoral se ha realizado un estudio previo del estado del arte de todos los problemas enmarcados en la familia de los GLP cuyo grafo huésped es un ciclo. Posteriormente, se han seleccionado aquellos problemas con mayor interés y relevancia en la literatura. En concreto, se han estudiado tres problemas de embebidos en ciclos los cuales se especifican a continuación.

- ***Cyclic Cutwidth Minimization Problem*** (CCMP): consiste en la minimización del número de aristas del grafo candidato que concurren con una arista del grafo huésped. Más formalmente, el problema se puede formular como:

$$\text{CCMP} = \underset{(\varphi,\psi)\ \in\ \Phi}{\arg\min}\ \{\ \underset{(w,z)\ \in\ E_H}{\arg\max}\ \{\mathit{cut}(\varphi,\psi,(w,z))\}\}, \tag{B.1}$$

  donde $\mathit{cut}(\varphi,\psi,(w,z))$ es una función que calcula el número de aristas del grafo de entrada que concurren en la arista (w,z) del grafo huésped, y Φ es el conjunto de todas las soluciones (φ,ψ).

  En la Figura B.3 se ilustra la evaluación de la solución presentada previamente en la Figura B.2(a) del grafo de entrada ilustrado en la Figura B.1(a). La evaluación de una solución para el CCMP, implica el cálculo del corte (es decir, de la función *cut*) asociado a cada arista del grafo huésped. Por ejemplo, para calcular el corte de la arista $(1,2)$ primero es necesario obtener los caminos asociados a las aristas del grafo de entrada que contienen la arista $(1,2)$. En este ejemplo, hay dos caminos que cumplen esta condición: $\psi(\mathrm{A},\mathrm{B}) = \{(1,2)\}$ y $\psi(\mathrm{A},\mathrm{C}) = \{(1,2),(2,3)\}$. La Figura B.3 resalta la arista (A,B) y su camino asociado en amarillo. Del mismo modo, la arista (A,C) se ha resaltado en verde. Por tanto, $\mathit{cut}((1,2),\varphi,\psi) = |\{(\mathrm{A},\mathrm{B}),(\mathrm{A},\mathrm{B})\}| = 2$. Esta función es calculada para cada una de las aristas del grafo huésped (Figura B.3(b)). Finalmente, el valor de la función objetivo para el CCMP es el máximo de todos los cortes. Para esta solución, $\max\{2,2,2,2,1\} = 2$.



(a) **(b)**

Figura B.3 (a) Evaluación del corte de la arista $(1,2) \in E_H$. (b) Evaluación de la función objetivo de una solución para el CCMP.

- ***Cyclic Antibandwidth Problem*** **(CAB)**: consiste en la maximización de la distancia (también conocida como *bandwidth*, en inglés) a la que se encuentran los vértices adyacentes del grafo de entrada, medida en el ciclo. Más formalmente, el problema se puede formular como:

$$\text{CAB} = \underset{(\varphi,\psi) \in \Phi}{\arg\max} \{ \underset{(u,v) \in E_G}{\arg\min} \{bw(\varphi,\psi,(u,v)\}\}, \tag{B.2}$$

donde $bw(\varphi,\psi,(u,v))$ es una función que calcula la cardinalidad del camino asignado a la arista $(u,v) \in E_G$.

Considerando nuevamente la solución presentada anteriormente en la Figura B.2(a) del grafo de entrada ilustrado en la Figura B.1(a), en la Figura B.4 se representa la evaluación de la función objetivo de este problema para dicha solución. En concreto, para calcular el *bandwidth* de una arista, se calcula la cardinalidad del camino que tiene asignado a través de la función ψ. Por ejemplo, en la Figura B.4(a), el *bandwidth* de la arista (A,C) se calcula como $|\psi(\text{A},\text{C})| = |\{(1,2),(2,3)\}| = 2$. Tanto la arista como su camino asignado aparecen resaltados en verde en la Figura B.4. De manera similar, el *bandwidth* de la arista $|\psi(\text{D},\text{E})| = |\{(4,5)\}| = 1$, es representado

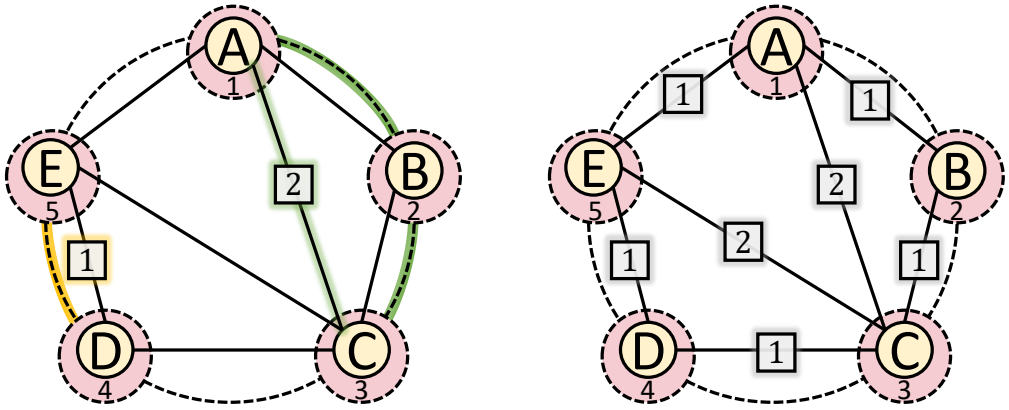


Figura B.4 (a) Evaluación del *bandwidth* de las aristas $(\mathrm{A},\mathrm{C}),(\mathrm{D},\mathrm{E}) \in E_G$. (b) Evaluación del *bandwidth* de todas las aristas del grafo de entrada en el grafo huésped ciclo.

en amarillo en la misma figura. Por último, en la Figura B.4(b), se indica el valor del *bandwidth* de cada arista, necesario para calcular la función objetivo. Concretamente, el valor de la función objetivo es el mínimo de todas del *bandwidth* de todas las aristas, que es 1 en este ejemplo. En términos matemáticos: $\min\{1,2,1,1,1,2,1\} = 1$.

- ***Cyclic Bandwidth Sum Problem*** **(CBS)**: consiste en la minimización de la suma de la distancia a la que se encuentran los vértices adyacentes del grafo de entrada, medida en el ciclo. Más formalmente, el problema se puede formular como:

$$\mathrm{CBS} = \underset{(\varphi,\psi)\ \in\ \Phi}{\arg\min}\ \{ \sum_{(u,v)\ \in\ E_G} bw(\varphi,\psi,(u,v)\}. \tag{B.3}$$

  De nuevo, se hace uso de la Figura B.4. En este caso, la función objetivo del CBS se calcula como la suma del *bandwidth* de cada arista del grafo de entrada. En términos matemáticos: $1+2+1+1+1+2+1 = 9$.

Como se comentó anteriormente, el objetivo principal de esta Tesis Doctoral se centra en la propuesta de algoritmos heurísticos y metaheurísticos para los problemas de embebidos en ciclos. Sin embargo, tras abordar con éxito los tres problemas anteriormente mencionados, se decidió extender la aplicación de los conocimientos aprendidos a problemas

de embebidos en otros grafos huésped. Concretamente, se optó por abordar un problema cuyo embebido es realizado en un grafo huésped rejilla el cual se define a continuación:

- ***Two-Dimensional Bandwidth Minimization Problem* (2DBMP)**: consiste en la minimización de la suma de la distancia a la que se encuentran los vértices adyacentes, medida en la rejilla. Más formalmente, el problema se puede formular como:

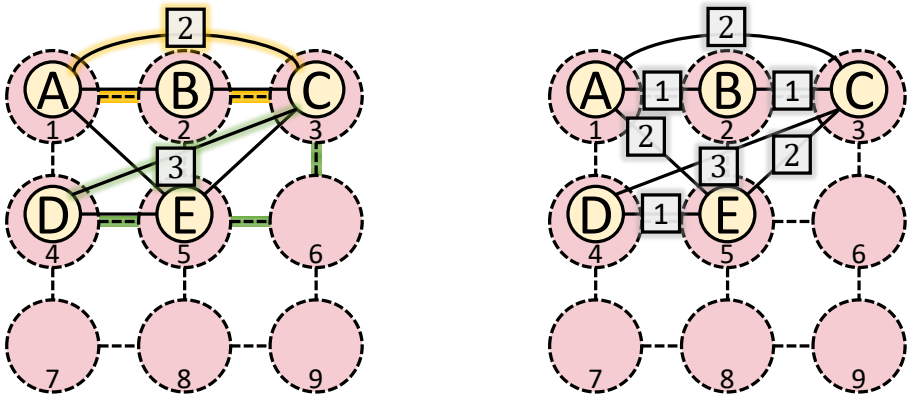
$$\text{2DBMP} = \underset{(\varphi,\psi)\,\in\,\Phi}{\arg\min}\,\{\max_{(u,v)\,\in\,E_G} bw(\varphi,\psi,(u,v)\} \tag{B.4}$$

  A continuación, se muestra un ejemplo del cálculo de la función objetivo del 2DBMP haciendo uso de la solución presentada en la Figura B.2(b). Para evaluar la función objetivo de una solución es necesario calcular la distancia entre cada para cada arista de E_G por medio de la función *bandwidth*. Por ejemplo, se considera la arista (C,D). Tres caminos pueden ser asignados a esa arista por medio de la función ψ. Para este problema, dado que la distancia de los tres caminos es la misma, la elección de uno de ellos no influye en el cálculo del *bandwidth*. Por lo tanto, se selecciona uno de ellos al azar. En concreto, $\psi(\mathrm{C},\mathrm{D}) = \{(3,6),(6,5),(5,4)\}$, que aparece resaltado en verde en la Figura B.5(a). Por tanto, $bw(\varphi,\psi,(\mathrm{C},\mathrm{D})) = 3$. Del mismo modo, el *bandwidth* de la arista (A,C), resaltada en amarillo en la misma figura, es $bw(\varphi,\psi,(\mathrm{A},\mathrm{C})) = \{(1,2),(1,3)\} = 2$. Este cálculo se realiza sobre el resto de las aristas de G (Figura B.5(b)). Finalmente, el valor de la función objetivo es el máximo de todas las distancias, que es 3 en este ejemplo. En términos matemáticos: $\max\{1,2,2,1,3,2,1\} = 3$.

En la siguiente sección se recogen los estudios previos que se han realizado sobre los problemas de embebido de grafos. Además, se hace especial hincapié en aquellos trabajos que abordan los cuatro problemas estudiados en esta Tesis Doctoral.

B.2 Antecedentes

Los grafos se usan comúnmente para representar aplicaciones reales mediante puntos conectados con líneas. Además de visualizar sistemas del mundo real, también se utilizan



(a) **(b)**

Figura B.5 (a) Evaluación del *bandwidth* de las aristas $(\mathrm{A},\mathrm{C}),(\mathrm{C},\mathrm{D}) \in E_G$. (b) Evaluación del *bandwidth* de todas las aristas del grafo de entrada en el grafo huésped rejilla.

para modelar problemas de optimización que pueden ser resueltos por computadora. En los últimos años, ha habido un creciente interés en estudiar los problemas de embebidos de grafos, especialmente en el contexto del diseño de circuitos VLSI [44, 65].

Tras la primera motivación, basada en el diseño de VLSI, los GLP han surgido como una familia de problemas combinatorios cuyo objetivo principal es proyectar o embeber un grafo en un grafo huésped predefinido. Esta proyección también se ha denotado como *layout*. [63, 191], etiquetado [42, 130], numeración [41, 191], u ordenación [25, 81, 194, 206].

La mayoría de los trabajos abordan los GLP desde un punto de vista teórico: estudiando su complejidad computacional, proponiendo ecuaciones para determinar el valor de la solución óptima, determinando los límites inferiores y superiores, o encontrando relaciones entre problemas, etc. Por lo general, este tipo de trabajos se centran en el estudio de una función objetivo para un grafo huésped. Por otro lado, los investigadores también estudian estos problemas desde una perspectiva más general, para lo que proponen algoritmos exactos y aproximados para cualquier tipo de grafo de entrada. A continuación, se recogen los trabajos más relevantes para cada uno de los problemas estudiados.

El CCMP ha sido ampliamente estudiado desde un punto de vista teórico. Generalmente, los trabajos se centran en el estudio de la solución óptima para grafos de entrada con topología conocida. Entre estas investigaciones se destacan las siguiente: [1, 8, 27, 34, 43, 68, 125, 128, 203, 223, 224, 222]. Hasta el momento, el CCMP no ha sido resultado de manera exacta, sin embargo, sí que se pueden encontrar varios trabajos que lo abordan desde un punto de vista aproximado, específicamente heurístico: [28, 34, 110, 124].

El CAB, el segundo problema abordado en esta Tesis Doctoral, ha sido trabajado, al igual que el problema anterior, de manera teórica [54, 146, 172, 199, 237] y aproximada [12, 31, 157], ya que, por el momento, no ha sido resuelto por algoritmos exactos.

El CBS tampoco ha sido abordado mediante algoritmos exactos, aunque sí que se pueden encontrar estudios teóricos [37, 126, 259] y propuestas de algoritmos heurísticos [33, 98, 208, 209, 221].

Por último, el 2DBMP es el único problema estudiado en esta investigación cuyo embebido no es realizado en un grafo huésped ciclo. Este problema ha sido estudiado de manera teórica [42, 149, 150], desde una perspectiva exacta [211] y desde un punto de vista heurístico [32, 211].

Entre los trabajos teóricos recogidos en esta sección, cabe destacar aquellos que pretenden demostrar la complejidad de los problemas. En concreto, el trabajo clave sobre la complejidad de los GLP se centra en demostrar que el *Cutwidth Minimization Problem* (CMP), variante del CCMP en el que el embebido es realizado en un grafo huésped camino, es NP-completo [82, 83, 84]. Estudios posteriores de la complejidad computacional del CMP condujeron a la demostración de que es NP-completo para algún grafo de entrada específico [64, 159, 177]. Una vez demostrada la pertenencia de un GLP a la clase de complejidad NP-completo, otros investigadores demostraron la pertenencia de problemas similares, en los que el embebido se realiza en grafo caminos, a esa misma clase de complejidad [187]. Como puede verse, la complejidad de los problemas cuyo embebido se realiza en grafos camino ha sido ampliamente estudiada. Sin embargo, apenas se encuentran trabajos de este tipo cuando se pasa a otros GLP, donde el grafo huésped no es un camino. Generalmente, los autores abordan esta cuestión simplemente relacionando la complejidad de un problema concreto con otro cuya clase de complejidad es conocida. Sin embargo, sí es posible encontrar estudios para grafos específicos.

B.3 Hipótesis y objetivos

De manera más concreta, una vez identificado el conjunto de problemas que se pretenden abordar en esta Tesis Doctoral, y revisado el estado del arte de los mismos, se enuncia una hipótesis general válida para todos ellos.

> "*Los algoritmos heurísticos, utilizados conjuntamente con técnicas metaheurísticas, son métodos capaces de encontrar soluciones de alta calidad, potencialmente cercanas a la solución óptima, a problemas de optimización modelados por grafos que consisten en embeber un grafo candidato en un grafo huésped con estructura circular.*"

Derivado de la hipótesis anterior, el objetivo principal de esta Tesis Doctoral se enuncia de la siguiente manera:

> "*Diseñar e implementar algoritmos heurísticos, utilizados junto con técnicas metaheurísticas, para abordar problemas de optimización modelados mediante grafos consistentes en el embebido de grafos de tipo ciclo*".

Para alcanzar el objetivo principal planteado, es necesario cubrir los siguientes objetivos específicos:

- Revisar y analizar el estado del arte de los GLP, haciendo especial hincapié en aquellos trabajos cuyo embebido se realiza en un grafo huésped ciclo. Este objetivo también incluye el estudio de las estrategias, algoritmos y procedimientos utilizados para abordar problemas relacionados que puedan transferirse a los problemas en los que se centra esta Tesis Doctoral. Por último, se recopilarán los conjuntos de instancias sobre los que se han probado los algoritmos propuestos.

- Identificar las propiedades y características estructurales de cada problema. Dado que los procedimientos heurísticos son dependientes del problema, conocer las características específicas de estos puede ayudar a proponer técnicas más avanzadas y eficientes.

- Modelar el problema de forma que pueda ser abordado computacionalmente. Para ello se utilizará el lenguaje de programación Java [7].

- Diseñar y desarrollar un algoritmo heurístico para resolver el problema. Para ello se utilizarán técnicas heurísticas y metaheurísticas. Este objetivo incluye la identificación de los algoritmos heurísticos y metaheurísticos más apropiados para cada problema.

- Comparar experimentalmente el algoritmo propuesto con los algoritmos del estado del arte. Se ejecutarán los algoritmos del estado del arte identificados para realizar una comparación exhaustiva y justa, sobre un conjunto de instancias previamente utilizadas.

- Elaborar un documento que recoja el trabajo realizado y las conclusiones de los resultados obtenidos. En la fase final de la investigación se redactará la memoria de la Tesis Doctoral, en la que se describirán los problemas identificados y abordados.

- Difundir los resultados mediante su publicación en foros de investigación tales como: revistas, congresos o jornadas científicas. Los resultados del trabajo de investigación serán sometidos a un proceso de revisión por instituciones independientes que culminará con su publicación en revistas o congresos de prestigio nacional e internacional en el área.

B.4 Metodología

El desarrollo de esta Tesis Doctoral se fundamenta en el método científico como metodología para generar nuevo conocimiento. Concretamente, este método se basa en la observación, medición, experimentación, formulación y modificación de una hipótesis. Además, este método puede ser aplicado a cualquier área científica cuyo fin sea el de aportar conocimiento nuevo. En este caso, el método científico es utilizado como guía para proponer algoritmos heurísticos y metaheurísticos para abordar un problema de optimización combinatoria como son los pertenecientes a la familia de los GLP.

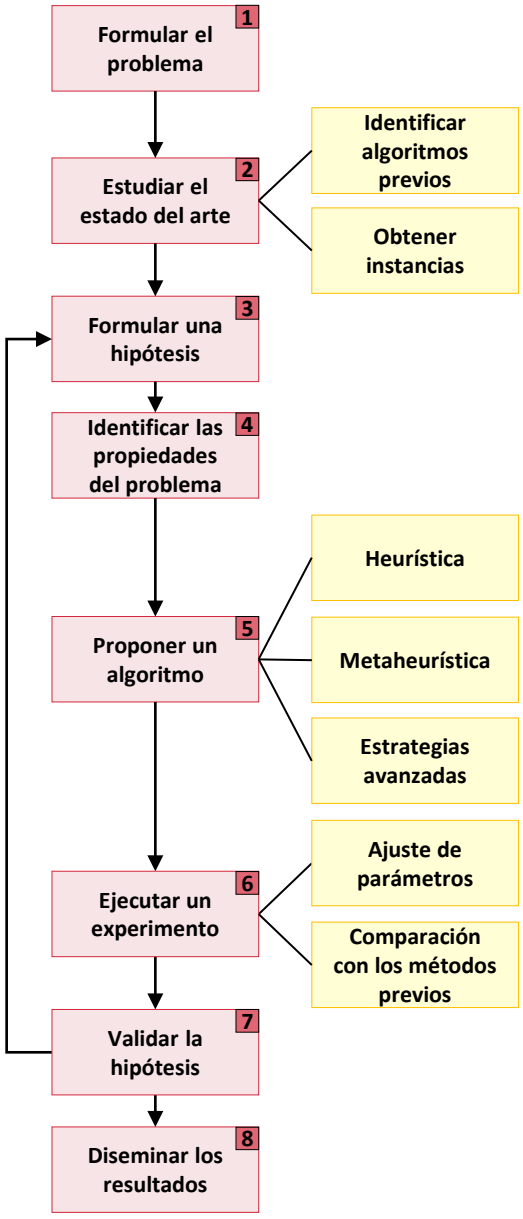


Figura B.6 Representación de la adaptación del método científico al contexto de la Tesis Doctoral.

En la Figura B.6, se representa el proceso de investigación propuesto en la Tesis Doctoral. El proceso comienza con el planteamiento del problema (fase 1). A continuación, se lleva a cabo el estudio del estado del arte, es decir, se recogen los avances más recientes del problema en cuestión (fase 2). En esta misma etapa se destacan las tareas de identificación de los algoritmos previos y obtención de las instancias (también conocidas como entradas o casos de prueba) puesto que son fundamentales para el correcto desarrollo de la investigación. Las siguientes cinco fases del proceso de investigación, detallado en la Figura B.6, son repetidas iterativamente (fases 3, 4, 5, 6 y 7). Partiendo del planteamiento de una hipótesis (fase 3), se realiza un estudio del problema que permita la extracción de características y su modelado (fase 4). A continuación, se propone y se implementa un algoritmo para abordar el problema en cuestión. Generalmente, los algoritmos implementados serán: algoritmos heurísticos, algoritmos metaheurísticos, o estrategias avanzadas que aumenten la eficiencia o robustez del algoritmo propuesto (fase 5). Tras la implementación de un algoritmo es necesario analizar su rendimiento mediante un conjunto de experimentos (fase 6). Estos experimentos son clasificados en: experimentos preliminares, si el objetivo es establecer la mejor configuración de los parámetros del algoritmo propuesto; o bien en experimentos competitivos, destinados a comparar el algoritmo propuesto con los algoritmos del estado del arte. Finalmente, se valida la hipótesis de partida (fase 7). Los resultados más relevantes de la investigación realizada son diseminados por medio de publicaciones en revistas o conferencias o *workshops* (fase 8).

Cabe desacatar que cada una de las fases del proceso de investigación descrito en la Figura B.6 se relacionan con uno o varios de los objetivos planteados en la Sección B.3. Concretamente las fases 1 y 2 se corresponden con el objetivo 1. De manera similar, el objetivo 2 se corresponde con la fase 4, los objetivos 5, 6 y 7 con la fase 5, el objetivo 7 con la fase 6 y, finalmente, los objetivos 8 y 9 se corresponden con la fase 8. Las fases 3 y 7, consistentes en el planteamiento y validación de la hipótesis, condicionan el diseño general del proceso de investigación.

B.5 Propuesta algorítmica

En esta Tesis Doctoral, se proponen diferentes procedimientos heurísticos y metaheurísticos para abordar cuatro problemas de embebidos de grafos.

Los algoritmos heurísticos son métodos de resolución de problemas que se basan en la experiencia y el conocimiento más que en principios formales. Suelen utilizarse cuando no se puede encontrar una solución óptima o se tarda demasiado en calcularla, o cuando un problema es demasiado complejo para resolverlo con métodos exactos. La heurística se suele utilizar para encontrar buenas soluciones rápidamente y puede ser muy eficaz en situaciones prácticas. Algunas heurísticas y metaheurísticas habituales son los procedimientos constructivos y los de mejora. Mientras que las metaheurísticas son estrategias o enfoques de nivel superior que guían el uso de heurísticas para resolver un problema. A continuación, se recogen los aspectos más relevantes de los algoritmos heurísticos propuestos (es decir, los procedimientos constructivos y de mejora) y las metaheurísticas empleadas.

Los procedimientos constructivos son los métodos utilizados para la generación de soluciones, que se utilizarán como punto de partida de los métodos de mejora. Generalmente, los procedimientos constructivos parten de una solución parcial vacía y terminan con una solución completa, es decir, una solución factible.

La construcción de una solución para un GLP consiste en definir las funciones matemáticas φ y ψ. Vale la pena recordar que la función φ asigna a cada vértice del grafo de entrada a un vértice del grafo huésped, mientras que la función ψ asigna a cada arista del grafo de entrada un camino del grafo huésped. Generalmente, la función ψ depende de la función φ ya que el camino asignado a cada arista es arbitrario. Por lo canto, cualquier definición de la función φ que satisfaga que cada vértice del grafo de entrada esté asignado a un vértice del grafo huésped será una solución factible del problema.

En este trabajo, más allá de proponer un algoritmo constructivo específico para cada uno de los problemas estudiados. se propone un esquema o marco general para la definición de cualquier algoritmo constructivo para la familia de problemas de embebido de grafos. En concreto, la definición de un algoritmo constructivo para un GLP debe considerar los siguientes pasos:

1. Generar un conjunto de vértices candidatos a partir de los vértices del grafo de entrada que todavía no han sido asignados.
2. Generar un conjunto de vértices candidatos a partir de los vértices del grafo huésped.
3. Seleccionar un vértice del conjunto de vértices candidatos del grafo de entrada.
4. Seleccionar un vértice del conjunto de vértices candidatos del grafo huésped.
5. Asignar el vértice del grafo huésped al vértice del grafo de entrada.
6. Actualizar los conjuntos de vértices candidatos.

Estos pasos son repetidos hasta obtener una solución factible. Para cada uno de los problemas abordados se proponen criterios específicos en cada uno de los pasos. Estos criterios son generalmente funciones matemáticas que dependen de la adyacencia de los vértices del grafo de entrada y de la función objetivo del problema. Cada uno de los criterios planteados se puede encontrar en los artículos que se adjuntan a este documento en la Parte II.

En general, el objetivo principal de un procedimiento constructivo es generar soluciones de la mejor calidad posible, es decir, generar la solución óptima del problema o acercarse lo más posible a ella para que otros métodos de intensificación, como la búsqueda local, puedan alcanzarla rápidamente. Sin embargo, a veces puede ser conveniente aplicar repetidamente el procedimiento de construcción para obtener soluciones de partida diversas. En este sentido, la diversificación podría proporcionar soluciones de menor calidad pero más diversas, permitiendo explorar más regiones del espacio de soluciones. Este tipo de estrategias en las que se fomenta la diversidad son especialmente interesantes cuando se combinan con metaheurísticas multiarranque o poblacionales.

Para fomentar la diversidad de las soluciones generadas, se utilizan varias de las ideas o principios de la metodología GRASP, que es una metaheurística multiarranque que combina decisiones voraces y aleatorias en cada una de las soluciones construidas [71, 72].

Dada una solución, esta es generalmente mejorada por un procedimiento de intensificación. Sin duda, el más relevante es la búsqueda local. Este procedimiento ha sido implementado en todas las investigaciones desarrolladas dentro de esta Tesis Doctoral y consiste

en la exploración de vecindarios para realizar movimientos sistemáticos a soluciones de mejor calidad.

El planteamiento de una búsqueda local implica la definición de una o varias estructuras de vecindad. Una vecindad es un conjunto de soluciones que pueden ser obtenidas por medio de un movimiento, una ligera variación en algún aspecto de la solución. En el contexto de la optimización combinatoria en grafos, dos de los movimientos más utilizados son la inserción y el intercambio. Ambos movimientos han sido implementados para los problemas estudiados.

El movimiento de inserción consiste en quitar un vértice del grafo de entrada de su asignación actual y asignarlo (es decir, insertarlo) a otro vértice del grafo huésped. Este movimiento implica un "desplazamiento" de algunos vértices para "hacer sitio" al vértice que se va a insertar. Por otro lado, el movimiento de intercambio consiste en intercambiar la asignación de dos vértices.

Dada una estructura de vecindad, un algoritmo de búsqueda local intenta encontrar una solución que mejore la calidad de la mejor solución encontrada hasta el momento. Si es capaz de encontrar una mejor, sustituye la mejor solución encontrada por la nueva. Este proceso se repite hasta alcanzar una solución óptima con respecto a su vecindad, denominada comúnmente óptimo local. A la hora de explorar un vecindario, dos estrategias son comúnmente utilizadas: la estrategia de primera mejora (en inglés, *first improvement*), y la estrategia de mejor mejora (en inglés, *best improvement*). La primera de las estrategias realiza el primer movimiento que mejora la calidad de la mejor solución encontrada hasta el momento. Por el contrario, la estrategia de mejor mejora, realiza el mejor movimiento posible de la vecindad.

Los procedimientos de mejora o búsqueda presentados en esta sección pueden ser mejorados si son combinados con tres estrategias avanzadas. Pese a que estas estrategias hayan sido diseñadas en el contexto de la resolución de GLP, pueden ser aplicadas a otros problemas. Concretamente, se plantean tres estrategias: el cálculo eficiente de la función objetivo, un criterio de desempate cuando dos soluciones presentan el mismo valor de función objetivo y técnicas para reducir el tamaño de los vecindarios.

La primera estrategia explora formas para realizar un cálculo eficiente de la función objetivo tras realizar un movimiento. Esto es de especial importancia cuando se trabaja con

vecindarios extensos, o bien cuando computar la función objetivo es costoso. Por ello, los investigadores han propuesto una evaluación inteligente o eficiente de la función objetivo que evite revaluar toda la solución mediante la actualización de aquellos elementos que se han visto afectados por el movimiento [157].

La segunda de las estrategias trata de abordar los conocidos como horizontes de búsqueda planos por medio de funciones objetivo alternativas o criterios de desempate. Los horizontes de búsqueda planos o mesetas ocurren, generalmente, en problemas de optimización formulados como problemas max-min (o min-max), donde la función objetivo consiste en maximizar un valor mínimo (o minimizar un valor máximo). Generalmente, este tipo de problemas tiene numerosas soluciones asociadas al mismo valor de la función objetivo. Cuando esto ocurre, es difícil determinar cuál de las soluciones comparadas es más prometedora para continuar la búsqueda. En ese caso, el valor de la función objetivo por sí solo no proporciona información suficiente para encontrar direcciones de búsqueda eficaces. *A priori*, dos soluciones con la misma función objetivo son iguales para un procedimiento de búsqueda local, pero su estructura puede ser diferente y puede haber ciertas propiedades que determinen que una solución es mejor que otra. Algunas de las propuestas que se pueden encontrar en la literatura para mitigar el impacto de este problema consisten en utilizar una o varias funciones objetivo como criterios de desempate [186, 192, 207]. En esta investigación se proponen varios criterios para determinar, en caso de empate en la función objetivo principal, qué solución es más prometedora.

Por último, teniendo en cuenta que para los problemas de interés los vecindarios son generalmente de tamaño inmenso, parece necesario considerar técnicas más avanzadas para la exploración de estos vecindarios. En concreto, para cada uno de los problemas estudiados se propone técnicas de reducción de las vecindades. Idealmente, una técnica de reducción se centra en soluciones prometedoras, evitando la pérdida de tiempo en la evaluación de soluciones que producen un deterioro en la función objetivo. Sin embargo, en la práctica, dada la complejidad de los vecindarios, algunas técnicas se limitan a reducir su tamaño estableciendo un porcentaje o un número fijo de soluciones a explorar.

Los algoritmos heurísticos presentados (los procedimientos constructivos y los procedimientos de mejora) son comúnmente combinados en esquemas más inteligentes, como son los algoritmos metaheurísticos. Dado que las metaheurísticas son estrategias genéricas

que no dependen del problema, los algoritmos que a continuación se presentan han podido ser utilizados para para abordar múltiples problemas en el contexto de esta Tesis Doctoral. Concretamente, se han utilizado procedimientos multiarranque, búsqueda tabú (en inglés, *Tabu Search*), búsqueda en vecindades variables (en inglés, *Variable Neighborhood Search*) y procedimientos voraces iterativos (en inglés, *Iterated Greedy*).

Las estrategias multiarranque se utilizan para escapar de los óptimos locales, puntos del espacio de búsqueda donde los algoritmos heurísticos quedan atrapados. Se basan en la idea de aplicar una estrategia de intensificación a diferentes puntos de partida en el espacio de búsqueda. Son, por lo tanto, estrategias donde la construcción de soluciones juega un papel destacado y donde, en cada iteración se construye una nueva solución que actuará como punto de partida para la búsqueda.

Tabu Search (TS) es una metaheurística introducida por F. Glover en 1986 como una forma de mejorar los algoritmos tradicionales de búsqueda local [88]. La idea principal de TS es combinar la búsqueda local con estrategias basadas en la memoria para explorar el espacio de búsqueda de forma más eficiente y escapar de los óptimos locales. El algoritmo funciona manteniendo una lista tabú, que es una lista de soluciones que se consideran "tabú" o prohibidas durante un cierto número de iteraciones. La lista tabú actúa como una memoria de soluciones que han sido visitadas y ayuda al algoritmo a evitar volver a visitarlas. Esto permite al algoritmo escapar de los óptimos locales y seguir explorando el espacio de búsqueda. Muchas ideas y extensiones de este método se discuten en [87, 90, 91].

Variable Neighborhood Search (VNS) fue propuesto por N. Mladenović y P. Hansen como un método general para resolver problemas difíciles de optimización combinatoria [99, 100, 102]. El principio básico de esta metodología es realizar cambios sistemáticos en la estructura de vecindad para escapar de los óptimos locales. En el momento de su aparición (1997), el uso de diferentes estructuras de vecindad era una idea novedosa, ya que los procedimientos clásicos de búsqueda generalmente operaban con una única estructura de vecindad. A partir de estas ideas es posible encontrar muchas variantes de VNSen la literatura. Algunas de las más relevantes son: *Reduced Variable Neighborhood Search*, *Basic Variable Neighborhood Search*, *Variable Neighborhood Descent*, *General Variable Neighborhood Search*, *Parallel Variable Neighborhood Search*, o *Variable Formulation Search*, entre otras [59, 99, 101, 192].

Por último *Iterated Greedy* (IG) es una metaheurística basada en la aplicación repetida de dos fases principales: una destrucción parcial de una solución seguida de una reconstrucción para alcanzar una nueva solución factible. Estas dos fases suelen repetirse hasta que se cumplen los criterios de terminación [216, 235]. En la fase de destrucción, se destruye parcialmente una solución para crear una nueva solución factible. La fase de reconstrucción recibe una solución parcialmente destruida, y la restaura para crear una nueva solución factible que sea mejor que la actual. Generalmente, esta estrategia es combinada con procedimientos de mejora como la búsqueda local.

B.6 Resultados

Esta sección resume y sintetiza la experimentación llevada a cabo en esta Tesis Doctoral. Analizar el rendimiento de un algoritmo es esencial para evaluar su eficacia y eficiencia en la resolución de un problema. Sin embargo, no suele ser suficiente basarse únicamente en un enfoque teórico para evaluarlos. Los algoritmos heurísticos y metaheurísticos buscan soluciones buenas pero no necesariamente óptimas. Sin embargo, su desempeño puede ser afectado por factores difíciles de identificar mediante análisis teóricos.

Para evaluar adecuadamente los algoritmos heurísticos y metaheurísticos propuestos ha sido necesario realizar un análisis empírico, en el que el algoritmo se ejecuta en una serie de casos de prueba y su rendimiento se mide utilizando métricas de rendimiento adecuadas. Por lo tanto, en primer lugar ha sido necesario recopilar y analizar los conjuntos de prueba que se utilizarán para analizar los algoritmos. Estos casos de prueba son conocidos como "instancias" y se corresponden con los grafos de entrada de los GLP.

Una vez identificados los conjuntos de prueba, se realiza una experimentación preliminar con el fin de ajustar los diversos parámetros de los algoritmos propuestos. Además, dado que los valores de los parámetros serán los mismos para todas las instancias, resultan determinantes en el rendimiento final del algoritmo. El ajuste de parámetros de un algoritmo de este estilo es tan complejo que puede ser considerado un problema de optimización en sí mismo.

Los investigadores, utilizan dos tipos de técnicas para abordar el ajuste de parámetros: el ajuste de parámetros manual y el automático. Ambas aproximaciones han sido usadas en

el trascurso de esta investigación. La primera de ellas se fundamenta en el "diseño factorial de experimentos" [18]. Aunque esta estrategia manual requiere de tiempo y no garantiza encontrar la mejor configuración del algoritmo, es muy importante desde el punto de vista científico, ya que proporciona un mayor control y comprensión del proceso de ajuste y de las estrategias propuestas. Por otro lado, el ajuste de parámetros automático no requiere la intervención del investigador durante el proceso. En concreto, en esta investigación se ha utilizado la herramienta software *irace* [153].

Tras obtener el mejor algoritmo posible para el conjunto de datos preliminar se realiza una experimentación competitiva. Esta experimentación se utiliza para evaluar la eficacia de distintos enfoques o estrategias. En concreto, consiste en enfrentar diferentes algoritmos entre sí para determinar cuál es el más eficiente. En el contexto de esta tesis, las pruebas competitivas comparan el rendimiento de los mejores algoritmos del estado del arte a la hora de encontrar las mejores soluciones para un conjunto de datos de entrada.

Para concluir este apartado, en la Tabla B.1 se presenta un resumen de las pruebas competitivas realizadas en cada problema estudiado, comparando el mejor algoritmo del estado del arte en ese momento con nuestra propuesta. Nótese que para el CAB se comparan dos algoritmos del estado del arte, y el objetivo era maximizar el valor de una función objetivo. Para cada problema se compara el promedio del valor de la función objetivo ($\overline{ccmp}$, $\overline{cab}$, $\overline{cbs}$ y $\overline{2dbmp}$, respectivamente), la desviación con respecto a las mejores soluciones encontradas (Desv. (%)), el número de mejores soluciones encontradas (#Mejores ($\ll$nº total de instancias$\gg$) y el tiempo de cómputo (T. CPU (s)).

En general, los resultados obtenidos por cada uno de los algoritmos propuestos para cada problema sugieren que nuestra propuesta es superior al algoritmo del estado del arte en términos de calidad de la solución. Estos resultados llevan a afirmar que el algoritmo propuesto es un enfoque prometedor para la resolución del problema abordado. Esta afirmación es respaldada por pruebas estadísticas que pueden ser consultadas en los artículos correspondientes.

		Estado del arte		Nuestra propuesta
CCMP	$\overline{ccmp}$		57.94	57.24
	Desv. (%)		3.54	0.69
	#Mejores (179)		141	157
	T. CPU (s)		1434.01	177.42
CAB	$\overline{cab}$	149.39	164.70	164.09
	Desv. (%)	21.11	7.57	5.45
	#Mejores (267)	99	94	167
	T. CPU (s)	150.00	150.00	150.00
CBS	$\overline{cbs}$		60209.01	59408.65
	Desv. (%)		6.14	0.10
	#Mejores (106)		50	100
	T. CPU (s)		6019.53	1485.50
2DBMP	$\overline{2dbmp}$		4.82	3.57
	Desv. (%)		32.90	0.00
	#Mejores (86)		40	86
	T. CPU (s)		231.77	53.39

Tabla B.1 Resumen de la comparación de los mejores algoritmos propuestos con los mejores algoritmos del estado del arte para el CCMP, CAB, CBS y 2DBMP, respectivamente.

B.7 Conclusiones

En esta Tesis Doctoral se han abordado cuatro problemas de optimización pertenecientes a la familia de problemas de embebido de grafos, más conocidos como GLP. Estos problemas consisten en encontrar un embebido de un grafo de entrada en un grafo huésped, de forma que se optimice una función objetivo dada. En particular, tres de los problemas estudiados realizan el embebido en un grafo huésped ciclo, mientras que el otro lo hace en un grafo huésped de rejilla. Además, las funciones objetivo a optimizar se basan en dos funciones matemáticas, *bandwidth* y *cutwidth*.

La familia de problemas GLP es una clase amplia y diversa de problemas de optimización que tiene muchas aplicaciones prácticas, incluyendo el diseño de redes, la asignación de recursos y el dibujado de grafos. Resolver estos problemas de forma eficiente y exacta es un reto importante, ya que la mayoría de ellos podrían incluirse en la categoría NP-difícil. Dado que los algoritmos exactos son ineficientes para resolver este tipo de problemas, es necesario proponer otro tipo de técnicas que, aunque no sean capaces de certificar si las soluciones encontradas son óptimas, obtengan soluciones de alta calidad en tiempos de computación reducidos. Entre las posibles técnicas para abordar estos problemas, los algoritmos heurísticos y metaheurísticos han demostrado ser muy eficaces. En consecuencia, en esta investigación se proponen algoritmos heurísticos y metaheurísticos para abordar los GLP estudiados.

Estas ideas son las que dan lugar a la hipótesis de partida planteada en esta Tesis Doctoral. En concreto, la hipótesis se ha basado en la propuesta de técnicas heurísticas y metaheurísticas para la resolución de problemas de embebido de grafos en grafos huésped con estructura cíclica. Esta hipótesis ha sido validada satisfactoriamente para los tres problemas estudiados: el *Cyclic Cutwidth Minimization Problem*, el *Cyclic Antibandwidth Problem* y el *Cyclic Bandwidth Sum Problem*. Posteriormente se amplía esta hipótesis modificando el grafo huésped en el que se realiza el embebido, en concreto, se propone esta misma hipótesis para grafos huésped de rejilla. Del mismo modo, esta hipótesis es validada para el *Two-Dimensional Bandwidth Minimization Problem*.

Para corroborar que la hipótesis inicial es cierta, se han planteado diferentes objetivos. Estos objetivos han sido la guía de esta investigación, marcando cada uno de los pasos

a realizar para los problemas abordados. En conjunto, se puede afirmar que los objetivos planteados al inicio de esta Tesis Doctoral han sido logrados. Además, los resultados más relevantes han sido de interés para la comunidad científica, ya que han sido difundidos en diferentes foros científicos. En concreto, se destacan las publicaciones realizadas en revistas indexadas en el índice JCR:

- S. Cavero, E. G. Pardo, M. Laguna, and A. Duarte. Multistart search for the cyclic cutwidth minimization problem. *Computers & Operations Research*, 126:105116, 2021 [34].
- S. Cavero, E. G. Pardo, and A. Duarte. A general variable neighborhood search for the cyclic antibandwidth problem. *Computational Optimization and Applications*, 81(2):657–687, 2022 [31].
- S. Cavero, E. G. Pardo, A. Duarte, and E. Rodriguez-Tello. A variable neighborhood search approach for cyclic bandwidth sum problem. *Knowledge-Based Systems*, 246:108680, 2022 [33].
- S. Cavero, E. G. Pardo, and A. Duarte. Efficient iterated greedy for the two-dimensional bandwidth minimization problem. *European Journal of Operational Research*, 306(3): 1126–1139, 2023 [32].

Cabe mencionar que, en la Parte II de esta Tesis Doctoral, se adjunta una copia de cada una de las publicaciones mencionadas.

Independientemente del cumplimiento de los objetivos de la investigación, el planteamiento de los algoritmos ha permitido obtener conocimientos y experiencia que pueden ser aplicados a otros problemas o en futuras investigaciones, no necesariamente ligadas a los GLP. Las conclusiones más relevantes extraídas de esta investigación se resumen a continuación

- Este trabajo presenta una formalización general de los problemas de embebido de grafos, que implica definir el problema en términos precisos y matemáticos.
- Los horizontes de búsqueda en problemas max/min o min/max requieren funciones objetivo alternativas.

- Existen estrategias comunes que pueden aplicarse a múltiples problemas relacionados. Esto sugiere que puede ser posible desarrollar algoritmos de propósito general o enfoques que sean eficaces en una gama de diferentes problemas de embebido de grafos.

- La estrategia para construir una solución inicial es clave en todos los algoritmos propuestos en esta investigación.

- La combinación de un constructivo voraz con un esquema multiarranque es un enfoque eficaz para resolver problemas de optimización, en particular cuando se combina con la búsqueda local.

- Los vecindarios clásicos, como los intercambios y las inserciones, son eficientes en la resolución de GLP. Además, su combinación con el cálculo eficiente de la función objetivo o las técnicas de reducción de vecindades los convierten en elementos clave para abordar problemas de estas características.

- En general, no es necesario explorar exhaustivamente un vecindario para encontrar una buena solución. En concreto, suele ser suficiente explorar un subconjunto de soluciones potenciales para encontrar una mejora.

- Los vecindarios planteados tienen un gran tamaño, esto se debe, en gran medida, a la existencia de muchas soluciones equivalentes.

- Las estrategias de búsqueda avanzada, como las que se proponen en esta investigación, son un aspecto importante de la resolución de problemas de optimización y pueden considerarse una combinación de conocimientos de programación y experiencia científica.

- Se carece de algoritmos exactos para la resolución de GLP. Dada la complejidad de los problemas, parece razonable que apenas se haya propuesto ninguno para resolver este tipo de problemas. Sin embargo, es interesante considerar otras perspectivas para comprobar la calidad de las soluciones encontradas.

- Por último, la representación de las soluciones mediante su dibujo es fundamental para analizar el correcto funcionamiento del algoritmo y proponer estrategias más efectivas.

Para finalizar, este resumen en español se recogen algunas líneas de investigación abiertas, que pueden ser un punto de partida alternativo para futuras investigaciones. Como resultado del proceso de investigación iniciado con esta Tesis Doctoral, se proponen los siguientes trabajos futuros:

- La investigación futura puede centrarse en el desarrollo de métodos exactos para resolver algunas instancias y problemas. Esto puede implicar el desarrollo de algoritmos de "ramificación y poda" (*branch-and-bound*, en inglés) o modelos matemáticos. Estas técnicas se pueden combinar con algoritmos heurísticos que acaben dando lugar a algoritmos mateheurísticos.

- Realizar una revisión exhaustiva de la literatura para identificar las principales contribuciones y tendencias en el contexto de los problemas de embebidos de grafos para otras estructuras de grafo huésped, y sintetizar esta información en una visión coherente del estado actual del campo.

- Estudiar otros problemas de embebidos de grafos. Esto puede implicar explorar diferentes tipos de grafos de entrada y funciones objetivo, así como examinar la viabilidad de embeber grafos en otros grafos huésped.

- Proponer un algoritmo de caja negra o similar que recoja y combine los conocimientos adquiridos en investigaciones anteriores con el fin de crear una solución más completa y eficaz, para grafos de entrada y huésped más generales.

- Plantear un enfoque multiobjetivo para abordar múltiples funciones objetivo en el contexto de los problemas de embebidos de grafos.

- Estudiar cómo las soluciones equivalentes pueden afectar al rendimiento y eficacia de las soluciones de los algoritmos. Esto podría implicar la investigación en el uso de técnicas como la ruptura de simetría o el uso de restricciones adicionales que

podrían reducir significativamente el tamaño de los vecindarios y, consecuentemente, del espacio de búsqueda.

- Por último, desarrollar y publicar un *software* de dibujado de grafos. El desarrollo de esta herramienta combinaría los conocimientos adquiridos por los científicos en este ámbito y permitiría a un público general visualizar soluciones grafos genéricos.

Bibliography

[1] H. L. Abbott. Hamiltonian circuits and paths on the n-cube. *Canadian Mathematical Bulletin*, 9(5):557–562, 1966.

[2] B. Acharya and M. Gill. On the index of gracefulness of a graph and the gracefulness of two-dimensional square lattice graphs. *Indian J. Math*, 23(81-94):14, 1981.

[3] G. Alway and D. Martin. An algorithm for reducing the bandwidth of a matrix of symmetrical configuration. *The Computer Journal*, 8(3):264–272, 1965.

[4] A. R. Amaral. A mixed 0-1 linear programming formulation for the exact solution of the minimum linear arrangement problem. *Optimization Letters*, 3(4):513–520, 2009.

[5] N. Andersen, J. G. Bramness, and I. O. Lund. The emerging covid-19 research: dynamic and regularly updated science maps and analyses. *BMC Medical Informatics and Decision Making*, 20(1):1–7, 2020.

[6] N. H. Anderson. Scales and statistics: parametric and nonparametric. *Psychological bulletin*, 58(4):305, 1961.

[7] K. Arnold, J. Gosling, and D. Holmes. *The Java programming language*. Addison Wesley Professional, 2005.

[8] R. Aschenbrenner. A proof for the cyclic cutwidth of Q5. Technical report, California State University, 2001.

[9] D. Auber. Tulip—a huge graph visualization framework. In *Graph drawing software*, pages 105–126. Springer, 2004.

[10] S. Baluja. Population-based incremental learning. a method for integrating genetic search based function optimization and competitive learning. Technical report, Carnegie Mellon University, 1994.

[11] R. Bansal and K. Srivastava. Memetic algorithm for the antibandwidth maximization problem. *Journal of Heuristics*, 17(1):39–60, 2011.

[12] R. Bansal and K. Srivastava. A memetic algorithm for the cyclic antibandwidth maximization problem. *Soft Computing*, 15(2):397–412, 2011.

[13] R. Bar-Yehuda, G. Even, J. Feldman, and J. Naor. *Computing an optimal orientation of a balanced decomposition tree for linear arrangement problems*, pages 387–413. World Scientific, 2004.

[14] S. T. Barnard, A. Pothen, and H. Simon. A spectral algorithm for envelope reduction of sparse matrices. *Numerical linear algebra with applications*, 2(4):317–334, 1995.

[15] R. S. Barr, B. L. Golden, J. P. Kelly, M. G. Resende, and W. R. Stewart. Designing and reporting on computational experiments with heuristic methods. *Journal of heuristics*, 1(1):9–32, 1995.

[16] M. Basseur and A. Goëffon. Climbing combinatorial fitness landscapes. *Applied Soft Computing*, 30:688–704, 2015.

[17] R. Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[18] D. Berengut. Statistics for experimenters: Design, innovation, and discovery. *The American Statistician*, 60(4):341–342, 2006.

[19] J. Blythe, C. McGrath, and D. Krackhardt. The effect of graph layout on inference from social network data. In *International symposium on graph drawing*, pages 40–51. Springer, 1995.

[20] A. Bodaghi, S. Goliaei, and M. Salehi. The number of followings as an influential factor in rumor spreading. *Applied Mathematics and Computation*, 357:167–184, 2019.

[21] J. A. Bondy, U. S. R. Murty, et al. *Graph theory with applications*, volume 290. Macmillan London, 1976.

[22] G. Booch. *The unified modeling language user guide*. Pearson Education India, 2005.

[23] P. Briest, D. Brockhoff, B. Degener, M. Englert, C. Gunia, O. Heering, T. Jansen, M. Leifhelm, K. Plociennik, H. Röglin, et al. Experimental supplements to the theoretical analysis of eas on problems from combinatorial optimization. In *International Conference on Parallel Problem Solving from Nature*, pages 21–30. Springer, 2004.

[24] Broadcom. Dónde colocar el appliance en su red para obtener mejores resultados. `https://techdocs.broadcom.com`. Accessed: 27-12-2022.

[25] V. Campos, F. Glover, M. Laguna, and R. Martí. An experimental evaluation of a scatter search for the linear ordering problem. *Journal of Global Optimization*, 21(4):397–414, 2001.

[26] A. Caprara, A. N. Letchford, and J.-J. Salazar-González. Decorous lower bounds for minimum linear arrangement. *INFORMS Journal on Computing*, 23(1):26–40, 2011.

[27] C. Castillo. A proof for the cyclic cutwidth of q6. *REU Project, Cal State Univ., San Bernardino*, 2003.

[28] S. Cavero, E. G. Pardo, and A. Duarte. Influence of the alternative objective functions in the optimization of the cyclic cutwidth minimization problem. In *Advances in Artificial Intelligence: 19th Conference of the Spanish Association for Artificial Intelligence, CAEPIA 2020/2021, Málaga, Spain, September 22–24, 2021, Proceedings*, page 139–149. Springer, 2021.

[29] S. Cavero, E. G. Pardo, and A. Duarte. Influence of the alternative objective functions in the optimization of the cyclic cutwidth minimization problem. *XIX Conferencia de la Asociación Española para la Inteligencia Artificial* (CAEPIA 2021), in Malaga, Spain, 2021.

[30] S. Cavero, E. G. Pardo, and A. Duarte. A vns approach for a variant of the antibandwidth problem. *8th International Conference on Variable Neighborhood Search* (ICVNS2021), in Abu Dhabi, U.A.E., 2021.

[31] S. Cavero, E. G. Pardo, and A. Duarte. A general variable neighborhood search for the cyclic antibandwidth problem. *Computational Optimization and Applications*, 81(2):657–687, 2022.

[32] S. Cavero, E. G. Pardo, and A. Duarte. Efficient iterated greedy for the two-dimensional bandwidth minimization problem. *European Journal of Operational Research*, 306(3):1126–1139, 2023.

[33] S. Cavero, E. G. Pardo, A. Duarte, and E. Rodriguez-Tello. A variable neighborhood search approach for cyclic bandwidth sum problem. *Knowledge-Based Systems*, 246:108680, 2022.

[34] S. Cavero, E. G. Pardo, M. Laguna, and A. Duarte. Multistart search for the cyclic cutwidth minimization problem. *Computers & Operations Research*, 126:105116, 2021.

[35] T. Charitou, K. Bryan, and D. J. Lynn. Using biological networks to integrate, visualize and analyze genomics data. *Genetics Selection Evolution*, 48(1):1–12, 2016.

[36] J. D. Chavez and R. Trapp. The cyclic cutwidth of trees. *Discrete Applied Mathematics*, 87(1-3):25–32, 1998.

[37] Y.-D. Chen and J.-H. Yan. A study on cyclic bandwidth sum. *Journal of Combinatorial Optimization*, 14(2):295–308, 2007.

[38] R. Cheng, M. Gent, and T. Tosawa. Genetic algorithms for designing loop layout manufacturing systems. *Computers & Industrial Engineering*, 31(3-4):587–591, 1996.

[39] S. Cheng, Y. Shi, and Q. Qin. On the performance metrics of multiobjective optimization. In *International Conference in Swarm Intelligence*, pages 504–512. Springer, 2012.

[40] M. Chimani, C. Gutwenger, M. Jünger, G. W. Klau, K. Klein, and P. Mutzel. The Open Graph Drawing Framework (OGDF). *Handbook of graph drawing and visualization*, 2011:543–569, 2013.

[41] P. Z. Chinn, J. Chvátalová, A. K. Dewdney, and N. E. Gibbs. The bandwidth problem for graphs and matrices — a survey. *Journal of Graph Theory*, 6(3):223–254, 1982.

[42] F. R. Chung. Labelings of graphs. *Selected topics in graph theory*, 3(25):151–168, 1988.

[43] D. W. Clarke. *The cyclic cutwidth of mesh cubes*. PhD thesis, California State University, 2002.

[44] J. P. Cohoon and S. Sahni. Heuristics for backplane ordering. *Journal of VLSI and computer systems*, 2(1-2):37–60, 1987.

[45] W. Commons. The VLSI VL82C106 is a I/O chip for x86 computers. `https://commons.wikimedia.org/wiki/File:VLSI_Chip.jpg`, 2009. Accessed: 29-11-2022.

[46] S. Craw. *Manhattan Distance*, pages 639–639. Springer US, Boston, MA, 2010.

[47] G. B. Dantzig. *Origins of the Simplex Method*, page 141–151. Association for Computing Machinery, New York, NY, USA, 1990.

[48] R. C. de Andrade, T. d. O. e Bonates, M. Campêlo, and M. da Silva Ferreira. A compact quadratic model and linearizations for the minimum linear arrangement problem. *Discrete Applied Mathematics*, 323:134–148, 2022.

[49] M. G. de Carvalho Resende and D. V. Andrade. Method and system for network migration scheduling, Mar. 20 2012. US Patent 8,139,502.

[50] M. De Domenico, M. A. Porter, and A. Arenas. Muxviz: a tool for multilayer analysis and visualization of networks. *Journal of Complex Networks*, 3(2):159–176, 2015.

[51] K. A. De Jong. An analysis of the behavior of a class of genetic adaptive systems. Technical report, University of Michigan, 1975.

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

[53] R. Diestel. Graph theory. *Graduate texts in mathematics*, 173:33, 2005.

[54] S. Dobrev, R. Královič, D. Pardubská, L. Török, and I. Vrt'o. Antibandwidth and cyclic antibandwidth of Hamming graphs. *Discrete Applied Mathematics*, 161(10):1402–1408, 2013.

[55] M. Dorigo. *Optimization, learning and natural algorithms*. PhD thesis, Politecnico di Milano, 1992.

[56] M. Dorigo, M. Birattari, and T. Stutzle. Ant colony optimization. *IEEE computational intelligence magazine*, 1(4):28–39, 2006.

[57] A. Duarte, S. Cavero, and E. G. Pardo. Heurísticas aplicadas al 2d bandwidth problem. *XXXIX Congreso Nacional de Estadística e Investigación Operativa* (SEIO 2022), in Granada, Spain, 2021.

[58] A. Duarte, R. Martí, M. G. Resende, and R. M. Silva. GRASP with path relinking heuristics for the antibandwidth problem. *Networks*, 58(3):171–189, 2011.

[59] A. Duarte, J. J. Pantrigo, E. G. Pardo, and J. Sánchez-Oro. Parallel variable neighbourhood search strategies for the cutwidth minimization problem. *IMA Journal of Management Mathematics*, 27(1):55, 2016.

[60] A. Duarte, J. Sánchez-Oro, N. Mladenović, and R. Todosijević. *Variable Neighborhood Descent*, pages 341–367. Springer International Publishing, Cham, 2018.

[61] I. S. Duff, R. G. Grimes, and J. G. Lewis. *User's guide for the Harwell-Boeing sparse matrix collection (release 1)*. RAL, Chilton, 1992.

[62] T. Duff. *Plutarch's Lives: exploring virtue and vice*. Oxford University Press, 1999.

[63] V. Dujmović and D. R. Wood. On linear layouts of graphs. *Discrete Mathematics and Theoretical Computer Science*, 6(2):339–358, 2004.

[64] J. Díaz, M. D. Penrose, J. Petit, and M. Serna. Approximating layout problems on random geometric graphs. *Journal of Algorithms*, 39(1):78–116, 2001.

[65] J. Díaz, J. Petit, and M. Serna. A survey of graph layout problems. *ACM Computing Surveys (CSUR)*, 34(3):313–356, 2002.

[66] J. Edmonds. Matroids and the greedy algorithm. *Mathematical programming*, 1(1):127–136, 1971.

[67] J. Ellson, E. Gansner, L. Koutsofios, S. C. North, and G. Woodhull. Graphviz—open source graph drawing tools. In *International Symposium on Graph Drawing*, pages 483–484. Springer, 2001.

[68] J. Erbele, J. D. Chavez, and R. Trapp. The cyclic cutwidth of qn. *Manuscript, California State University, San Bernardino USA*, 2003.

[69] H. Eves. *An introduction to the history of mathematics*. Saunders College Publishing, 1983.

[70] O. Ezenwoye. What language?-the choice of an introductory programming language. In *2018 IEEE Frontiers in Education Conference (FIE)*, pages 1–8. IEEE, 2018.

[71] T. A. Feo and M. G. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations research letters*, 8(2):67–71, 1989.

[72] T. A. Feo and M. G. Resende. Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2):109–133, 1995.

[73] E. Ferrara, P. De Meo, S. Catanese, and G. Fiumara. Detecting criminal organizations in mobile phone networks. *Expert Systems with Applications*, 41(13):5733–5750, 2014.

[74] A. Field. *Discovering statistics using IBM SPSS statistics*. SAGE Publications Ltd, 4th edition, 2013.

[75] M. Fischetti and M. Fischetti. Matheuristics. In *Handbook of heuristics*, pages 121–153. Springer, 2018.

[76] L. J. Fogel. Toward inductive inference automata. In *IFIP Congress*, volume 62, pages 395–400, 1962.

[77] M. Fourment and M. R. Gillings. A comparison of common programming languages used in bioinformatics. *BMC bioinformatics*, 9(1):1–9, 2008.

[78] T. M. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991.

[79] E. Gansner, E. Koutsofios, and S. North. Drawing graphs with dot. Technical report, AT&T Research, 2006.

[80] E. R. Gansner. Drawing graphs with Graphviz. Technical report, Graphviz, 2009.

[81] C. G. Garcia, D. Pérez-Brito, V. Campos, and R. Martí. Variable neighborhood search for the linear ordering problem. *Computers & operations research*, 33(12):3549–3565, 2006.

[82] M. R. Garey. A guide to the theory of NP-Completeness. *Computers and intractability*, 1979.

[83] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete problems. In *Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 47–63, 1974.

[84] F. Gavril. Some NP-complete problems on graphs. In *Proceedings of the eleventh conference on information sciences and systems*, pages 91–95, 1977.

[85] A. Ghavasieh and M. De Domenico. Statistical physics of network structure and information dynamics. *Journal of Physics: Complexity*, 3(1):011001, 2022.

[86] N. E. Gibbs, W. G. Poole, Jr, and P. K. Stockmeyer. An algorithm for reducing the bandwidth and profile of a sparse matrix. *SIAM Journal on Numerical Analysis*, 13(2):236–250, 1976.

[87] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision sciences*, 8(1):156–166, 1977.

[88] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5):533–549, 1986.

[89] F. Glover, V. Campos, and R. Martí. Tabu search tutorial. A graph drawing application. *Top*, 29(2):319–350, 2021.

[90] F. Glover and M. Laguna. Tabu search. In *Handbook of combinatorial optimization*, pages 2093–2229. Springer, 1998.

[91] F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. *Control and cybernetics*, 29(3):653–684, 2000.

[92] F. W. Glover and G. A. Kochenberger. *Handbook of metaheuristics*, volume 57. Springer Science & Business Media, 2006.

[93] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995.

[94] T. F. Gonzalez. *Handbook of approximation algorithms and metaheuristics*. Chapman and Hall/CRC, 2007.

[95] J. L. Gross, J. Yellen, and M. Anderson. *Graph theory and its applications*. Chapman and Hall/CRC, 2018.

[96] S. Gueye, S. Michel, and M. Moeini. Adjacency variables formulation for the minimum linear arrangement problem. In *International Conference on Operations Research and Enterprise Systems*, pages 95–107. Springer, 2014.

[97] Gurobi Optimization LLC. *Gurobi Optimizer Reference Manual*, 9th edition, 2020.

[98] R. Hamon, P. Borgnat, P. Flandrin, and C. Robardet. Relabelling vertices according to the network structure by minimizing the cyclic bandwidth sum. *Journal of Complex Networks*, 4(4):534–560, 2016.

[99] P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European journal of operational research*, 130(3):449–467, 2001.

[100] P. Hansen and N. Mladenović. *Variable Neighborhood Search*, pages 211–238. Springer US, Boston, MA, 2005.

[101] P. Hansen and N. Mladenović. *Variable Neighborhood Search*, pages 759–787. Springer International Publishing, Cham, 2018.

[102] P. Hansen, N. Mladenović, R. Todosijević, and S. Hanafi. Variable neighborhood search: basics and variants. *EURO Journal on Computational Optimization*, 5(3):423–454, 2017.

[103] J. Hao. Maximum cutwidth problem for graphs. *Applied Mathematics-A Journal of Chinese Universities*, 18(2):235–242, 2003.

[104] M. C. Hao, U. Dayal, M. Hsu, T. Sprenger, and M. H. Gross. Visualization of directed associations in e-commerce transaction data. In *Data Visualization 2001*, pages 185–192. Springer, 2001.

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

[106] L. H. Harper. Optimal numberings and isoperimetric problems on graphs. *Journal of Combinatorial Theory*, 1(3):385 – 393, 1966.

[107] J. P. Hart and A. W. Shogan. Semi-greedy heuristics: An empirical study. *Operations Research Letters*, 6(3):107–114, 1987.

[108] M. M. Hassan. Machine layout problem in modern manufacturing facilities. *The International Journal of Production Research*, 32(11):2559–2584, 1994.

[109] D. M. Hawkins. The problem of overfitting. *Journal of chemical information and computer sciences*, 44(1):1–12, 2004.

[110] M. He, Q. Wu, and Y. Lu. Breakout local search for the cyclic cutwidth minimization problem. *Journal of Heuristics*, 28(5):583–618, 2022.

[111] T. L. Heath et al. *The thirteen books of Euclid's Elements*. Courier Corporation, 1956.

[112] A. Hejlsberg, S. Wiltamuth, and P. Golde. *C# language specification*. Addison-Wesley Longman Publishing Co., Inc., 2003.

[113] Heuristic. *The Merriam-Webster Dictionary*. Merriam-Webster, 11th edition, 2019.

[114] Heurística. *Diccionario de la lengua española*. Real Academia Española, 23.6 edition, 2022.

[115] D. S. Hochba. Approximation algorithms for NP-hard problems. *ACM Sigact News*, 28(2):40–52, 1997.

[116] J. H. Holland. Outline for a logical theory of adaptive systems. *Journal of the ACM (JACM)*, 9(3):297–314, 1962.

[117] T. Hoskin. Parametric and nonparametric: Demystifying the terms. In *Mayo Clinic*, volume 5, pages 1–5, 2012.

[118] D. C. Howell. *Statistical methods for psychology*. PWS-Kent Publishing Co, 1992.

[119] J. Hromkovič, V. Müller, O. Sỳkora, and I. Vrt'o. On embedding interconnection networks into rings of processors. In *International Conference on Parallel Architectures and Languages Europe*, pages 51–62. Springer, 1992.

[120] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer, 2011.

[121] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle. Paramils: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009.

[122] IBM Corp. *IBM ILOG CPLEX Optimization StudioCPLEX User's Manual*, version 12 release 8 edition, 2017.

[123] L. W. Jacobs and M. J. Brusco. Note: A local-search heuristic for large set-covering problems. *Naval Research Logistics (NRL)*, 42(7):1129–1140, 1995.

[124] P. Jain, K. Srivastava, and G. Saran. Minimizing cyclic cutwidth of graphs using a memetic algorithm. *Journal of Heuristics*, 22(6):815–848, 2016.

[125] B. James. The cyclical cutwidth of the three-dimensional and fourdimensional cubes. *Cal State Univ., San Bernardino McNair Scholar's Program Summer Research Journal*, 1996.

[126] H. Jianxiu. Cyclic bandwidth sum of graphs. *Applied Mathematics-A Journal of Chinese Universities*, 16(2):115–121, 2001.

[127] Y. Jinjiang and Z. Sanming. Optimal labelling of unit interval graphs. *Applied Mathematics*, 10(3):337–344, 1995.

[128] M. Johnson. The linear and cyclic cutwidth of the complete bipartite graph. *REU Project, Cal State Univ., San Bernardino*, 2003.

[129] M. Jünger and P. Mutzel. *Graph drawing software*. Springer Science & Business Media, 2012.

[130] M. Juvan and B. Mohar. Optimal linear labelings and eigenvalues of graphs. *Discrete Applied Mathematics*, 36(2):153–168, 1992.

[131] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. IEEE, 1995.

[132] B. W. Kernighan and D. M. Ritchie. *The C programming language*. Pearson Education, 2006.

[133] S. Khot, G. Kindler, E. Mossel, and R. O'Donnell. Optimal inapproximability results for max-cut and other 2-variable csps? *SIAM Journal on Computing*, 37(1):319–357, 2007.

[134] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[135] B. Koohestani. On the solution of the graph bandwidth problem by means of search methods. *Applied Intelligence*, pages 1–17, 2022.

[136] Y. Koren and D. Harel. A multi-scale algorithm for the linear arrangement problem. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 296–309. Springer, 2002.

[137] M. Koutrouli, E. Karatzas, D. Paez-Espino, and G. A. Pavlopoulos. A guide to conquer the biological network era using graph theory. *Frontiers in bioengineering and biotechnology*, page 34, 2020.

[138] J. R. Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and computing*, 4(2):87–112, 1994.

[139] J. R. Koza. *Genetic programming II: automatic discovery of reusable programs*. MIT press, 1994.

[140] M. Laguna. Tabu search. In *Handbook of heuristics*, pages 741–758. Springer, 2018.

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

[142] P. Larrañaga and J. A. Lozano. *Estimation of distribution algorithms: A new tool for evolutionary computation*, volume 2. Springer Science & Business Media, 2001.

[143] E. L. Lawler and D. E. Wood. Branch-and-bound methods: A survey. *Operations research*, 14(4):699–719, 1966.

[144] C. K. Leung, V. V. Kononov, A. G. Pazdor, and F. Jiang. Pyramidviz: visual analytics and big data visualization for frequent patterns. In *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*, pages 913–916. IEEE, 2016.

[145] C. K.-S. Leung, P. P. Irani, and C. L. Carmichael. Wifisviz: effective visualization of frequent itemsets. In *2008 eighth IEEE international conference on data mining*, pages 875–880. IEEE, 2008.

[146] J. Y.-T. Leung, O. Vornberger, and J. D. Witthoff. On Some Variants of the Bandwidth Minimization Problem. *SIAM Journal on Computing*, 13(3):650–667, 1984.

[147] R. R. Levary and S. Kalchik. Facilities layout—a survey of solution procedures. *Computers & Industrial Engineering*, 9(2):141–148, 1985.

[148] A. Lim, J. Lin, and F. Xiao. Particle swarm optimization and hill climbing for the bandwidth minimization problem. *Applied Intelligence*, 26(3):175–182, 2007.

[149] L. Lin and Y. Lin. Two models of two-dimensional bandwidth problems. *Information Processing Letters*, 110(11):469–473, 2010.

[150] L. Lin and Y. Lin. Square-root rule of two-dimensional bandwidth problem. *RAIRO-Theoretical Informatics and Applications*, 45(4):399–411, 2011.

[151] Y. Lin. The cyclic bandwidth problem. In *Chinese Science Abstracts Series A*, volume 2, 1995.

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

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

[154] M. C. López-Locés, N. Castillo-García, H. J. F. Huacuja, P. Bouvry, J. E. Pecero, R. A. P. Rangel, J. J. G. Barbosa, and F. Valdez. A new integer linear programming model for the cutwidth minimization problem of a connected undirected graph. In *Recent Advances on Hybrid Approaches for Designing Intelligent Systems*, pages 509–517. Springer, 2014.

[155] H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated local search. In *Handbook of metaheuristics*, pages 320–353. Springer, 2003.

[156] M. Lozano, A. Duarte, F. Gortázar, and R. Martí. Variable neighborhood search with ejection chains for the antibandwidth problem. *Journal of Heuristics*, 18(6):919–938, 2012.

[157] M. Lozano, A. Duarte, F. Gortázar, and R. Martí. A hybrid metaheuristic for the cyclic antibandwidth problem. *Knowledge-Based Systems*, 54:103–113, 2013.

[158] J. Luttamaguzi, M. Pelsmajer, Z. Shen, and B. Yang. Integer programming solutions for several optimization problems in graph theory. Technical report, Center for Discrete Mathematics and Theoretical Computer Science, DIMACS, 2005.

[159] F. S. Makedon, C. H. Papadimitriou, and I. H. Sudborough. Topological bandwidth. *SIAM Journal on Algebraic Discrete Methods*, 6(3):418–444, 1985.

[160] O. Martin, S. W. Otto, and E. W. Felten. *Large-step Markov chains for the traveling salesman problem*. Citeseer, 1991.

[161] R. Martín-Santamaría, S. Cavero, A. Herrán, A. Duarte, and J. M. Colmenar. A practical methodology for reproducible experimentation: an application to the double-row facility layout problem. *Evolutionary Computation*, pages 1–35, 2022.

[162] V. G. Martins Santos and A. M. Moreira de Carvalho. Tailored heuristics in adaptive large neighborhood search applied to the cutwidth minimization problem. *European Journal of Operational Research*, 2019.

[163] R. Martí. Multi-start methods. In *Handbook of metaheuristics*, pages 355–368. Springer, 2003.

[164] R. Martí, V. Campos, and E. Piñana. A branch and bound algorithm for the matrix bandwidth minimization. *European Journal of Operational Research*, 186(2):513–528, 2008.

[165] R. Martí, M. Laguna, F. Glover, and V. Campos. Reducing the bandwidth of a sparse matrix with tabu search. *European Journal of Operational Research*, 135(2):450–459, 2001.

[166] R. Martí, J. A. Lozano, A. Mendiburu, and L. Hernando. Multi-start methods. In *Handbook of heuristics*, pages 155–175. Springer, 2018.

[167] R. Martí, J. J. Pantrigo, A. Duarte, and E. G. Pardo. Branch and bound for the cutwidth minimization problem. *Computers & Operations Research*, 40(1):137 – 149, 2013.

[168] A. J. Mcallister. A new heuristic algorithm for the linear arrangement problem. Technical report, New Brunswick, CA: University of New Brunswick, 1999.

[169] C. McCreary, C. Combs, D. Gill, and J. Warrenz. An automated graph drawing system using graph decomposition. In *ALCOM International Workshop on Graph Drawing*, page 80, 1993.

[170] R. D. Meller and K.-Y. Gau. The facility layout problem: recent and emerging trends and perspectives. *Journal of manufacturing systems*, 15(5):351–366, 1996.

[171] W. Michiels, E. H. L. Aarts, and J. Korst. *Theory of Local Search*, pages 299–339. Springer International Publishing, Cham, 2018.

[172] Z. Miller and D. Pritikin. On the separation number of a graph. *Networks*, 19(6):651–666, 1989.

[173] A. K. Mishra, J. Choi, M. F. Rabbee, and K.-H. Baek. In silico genome-wide analysis of the atp-binding cassette transporter gene family in soybean (glycine max l.) and their expression profiling. *BioMed research international*, 2019, 2019.

[174] N. Mladenović. A variable neighborhood algorithm-a new metaheuristic for combinatorial optimization. In *Papers Presented at Optimization Days*, volume 112, 1995.

[175] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & operations research*, 24(11):1097–1100, 1997.

[176] N. Mladenović, D. Urosevic, D. Pérez-Brito, and C. G. García-González. Variable neighbourhood search for bandwidth reduction. *European Journal of Operational Research*, 200(1):14–27, 2010.

[177] B. Monien and I. H. Sudborough. Min cut is np-complete for edge weighted trees. *Theoretical Computer Science*, 58(1-3):209–229, 1988.

[178] E. Montero, M.-C. Riff, and B. Neveu. A beginner's guide to tuning methods. *Applied Soft Computing*, 17:39–51, 2014.

[179] P. Moscato et al. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, 826:1989, 1989.

[180] F. Neri, C. Cotta, and P. Moscato. *Handbook of memetic algorithms*, volume 379. Springer, 2011.

[181] A. R. Newton. Computer-aided design of vlsi circuits. *Proceedings of the IEEE*, 69(10):1189–1199, 1981.

[182] M. Oliveira and J. Gama. An overview of social network analysis. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(2):99–115, 2012.

[183] D. B. Owen. The power of student's t-test. *Journal of the American Statistical Association*, 60(309):320–333, 1965.

[184] M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100, 1991.

[185] G. Palubeckis. A branch-and-bound algorithm for the single-row equidistant facility layout problem. *OR Spectrum*, 34(1):1–21, Jan 2012.

[186] J. J. Pantrigo, R. Martí, A. Duarte, and E. G. Pardo. Scatter search for the cutwidth minimization problem. *Annals of Operations Research*, 199(1):285–304, 2012.

[187] C. H. Papadimitriou. The NP-completeness of the bandwidth minimization problem. *Computing*, 16(3):263–270, 1976.

[188] C. H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[189] E. G. Pardo, S. Cavero, and A. Duarte. Un enfoque metaheurístico para problemas deordenación circular. *XXXIX Congreso Nacional de Estadística e Investigación Operativa* (SEIO 2022), in Granada, Spain, 2021.

[190] E. G. Pardo, A. García-Sánchez, M. Sevaux, and A. Duarte. Basic variable neighborhood search for the minimum sitting arrangement problem. *Journal of Heuristics*, 26(2):249–268, 2020.

[191] E. G. Pardo, R. Martí, and A. Duarte. Linear layout problems. In *Handbook of Heuristics*, pages 1025–1049. Springer, 2018.

[192] E. G. Pardo, N. Mladenović, J. J. Pantrigo, and A. Duarte. Variable formulation search for the cutwidth minimization problem. *Applied Soft Computing*, 13(5):2242–2252, 2013.

[193] G. A. Pavlopoulos, M. Secrier, C. N. Moschopoulos, T. G. Soldatos, S. Kossida, J. Aerts, R. Schneider, and P. G. Bagos. Using graph theory to analyze biological networks. *BioData mining*, 4:1–27, 2011.

[194] J. Petit. Experiments on the minimum linear arrangement problem. *Journal of Experimental Algorithmics (JEA)*, 8, 2003.

[195] R. Poli, J. Kennedy, and T. Blackwell. Particle swarm optimization. *Swarm intelligence*, 1(1):33–57, 2007.

[196] L. Prechelt. An empirical comparison of seven programming languages. *Computer*, 33(10):23–29, 2000.

[197] H. C. Purchase, R. F. Cohen, and M. James. Validating graph drawing aesthetics. In *International Symposium on Graph Drawing*, pages 435–446. Springer, 1995.

[198] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, 2022.

[199] A. Raspaud, H. Schröder, O. Sýkora, L. Torok, and I. Vrt'o. Antibandwidth and cyclic antibandwidth of meshes and hypercubes. *Discrete Mathematics*, 309(11):3541–3552, 2009.

[200] A. Raspaud, O. Sýkora, and I. Vrt'o. Congestion and Dilation, Similarities and Differences: a Survey. In *Proceedings of the 7th International Colloquium on Structural Information and Communication Complexity*, page 14, 2000.

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

[202] J. Ren, J.-K. Hao, E. Rodriguez-Tello, L. Li, and K. He. A new iterated local search algorithm for the cyclic bandwidth problem. *Knowledge-Based Systems*, 203:106136, 2020.

[203] F. Rios. Complete graphs as a first step toward finding the cyclic cutwidth of the n-cube. *Cal State Univ., San Bernardino McNair Scholar's Program Summer Research Journal*, 1996.

[204] E. Rodriguez-Tello and L. C. Betancourt. An improved memetic algorithm for the antibandwidth problem. In *International Conference on Artificial Evolution (Evolution Artificielle)*, pages 121–132. Springer, 2011.

[205] E. Rodriguez-Tello, J.-K. Hao, and J. Torres-Jimenez. Memetic algorithms for the minla problem. In *International Conference on Artificial Evolution (Evolution Artificielle)*, pages 73–84. Springer, 2005.

[206] E. Rodriguez-Tello, J.-K. Hao, and J. Torres-Jimenez. An effective two-stage simulated annealing algorithm for the minimum linear arrangement problem. *Computers & Operations Research*, 35(10):3331–3346, 2008.

[207] E. Rodriguez-Tello, F. Lardeux, A. Duarte, and V. Narvaez-Teran. Alternative evaluation functions for the cyclic bandwidth sum problem. *European Journal of Operational Research*, 273(3):904–919, 2019.

[208] E. Rodriguez-Tello, V. Narvaez-Teran, and F. Lardeux. Comparative Study of Different Memetic Algorithm Configurations for the Cyclic Bandwidth Sum Problem. In A. Auger, C. M. Fonseca, N. Lourenço, P. Machado, L. Paquete, and D. Whitley, editors, *Parallel Problem Solving from Nature – PPSN XV*, Lecture Notes in Computer Science, vol 11101, pages 82–94, Cham, 2018. Springer International Publishing.

[209] E. Rodriguez-Tello, V. Narvaez-Teran, and F. Lardeux. Dynamic Multi-Armed Bandit Algorithm for the Cyclic Bandwidth Sum Problem. *IEEE Access*, 7:40258–40270, 2019.

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

[211] M. A. Rodríguez-García, J. Sánchez-Oro, E. Rodriguez-Tello, E. Monfroy, and A. Duarte. Two-dimensional bandwidth minimization problem: Exact and heuristic approaches. *Knowledge-Based Systems*, 214:106651, 2021.

[212] J. Rolim, O. Sỳkora, and I. Vrt'o. Optimal cutwidths and bisection widths of 2-and 3-dimensional meshes. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 252–264. Springer, 1995.

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

[214] F. Rossi, P. Van Beek, and T. Walsh. *Handbook of constraint programming*. Elsevier, 2006.

[215] D. Rouvray and D. Bonchev. *Chemical graph theory: introduction and fundamentals*. Abacus Press, 1991.

[216] R. Ruiz and T. Stützle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European journal of operational research*, 177(3):2033–2049, 2007.

[217] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, 2nd edition, 2003.

[218] J. Sánchez-Oro, J. J. Pantrigo, and A. Duarte. Combining intensification and diversification strategies in vns. an application to the vertex separation problem. *Computers & Operations Research*, 52:209–219, 2014.

[219] M. Saravanan and S. Ganesh Kumar. Different approaches for the loop layout problems: a review. *The International Journal of Advanced Manufacturing Technology*, 69(9):2513–2529, 2013.

[220] SAS Institute Inc. *SAS/STAT 15.1 User's Guide*, 15.1 edition, 2018.

[221] D. Satsangi, K. Srivastava, and Gursaran. General variable neighbourhood search for cyclic bandwidth sum minimization problem. In *2012 Students Conference on Engineering and Systems*, pages 1–6, 2012.

[222] H. Schröder, O. Sỳkora, and I. Vrt'o. Cyclic cutwidths of the two-dimensional ordinary and cylindrical meshes. *Discrete applied mathematics*, 143(1):123–129, 2004.

[223] H. Schröder, O. Sỳykoa, and I. Vrt'o. Cyclic cutwidth of the mesh. In *International Conference on Current Trends in Theory and Practice of Computer Science*, pages 449–458. Springer, 1999.

[224] V. Sciortino, J. D. Chavez, and R. Trapp. The cyclic cutwidth of a p2× p2× pn mesh. *REU Project, Cal State Univ., San Bernardino*, 2002.

[225] T. D. Seeley. *The wisdom of the hive: the social physiology of honey bee colonies*. Harvard University Press, 2009.

[226] S. C. Shapiro. Generalized augmented transition network grammars for generation from semantic networks. *American Journal of Computational Linguistics*, 8(1):12–26, 1982.

[227] E. A. Silver, R. Victor, V. Vidal, and D. de Werra. A tutorial on heuristic methods. *European Journal of Operational Research*, 5(3):153–162, 1980.

[228] H. D. Simon and S.-H. Teng. How good is recursive bisection? *SIAM Journal on Scientific Computing*, 18(5):1436–1445, 1997.

[229] S. S. Skiena. *The algorithm design manual*, volume 2. Springer, 1998.

[230] K. Smith-Miles and L. Lopes. Measuring instance difficulty for combinatorial optimization problems. *Computers & Operations Research*, 39(5):875–889, 2012.

[231] K. Sörensen and F. Glover. Metaheuristics. *Encyclopedia of operations research and management science*, 62:960–970, 2013.

[232] K. Sörensen, M. Sevaux, and F. Glover. A history of metaheuristics. In *Handbook of heuristics*, pages 791–808. Springer, 2018.

[233] H. Stegherr, M. Heider, and J. Hähner. Classifying metaheuristics: Towards a unified multi-level classification system. *Natural Computing*, pages 1–17, 2020.

[234] B. Stroustrup. *The C++ programming language*. Pearson Education India, 2000.

[235] T. Stützle and R. Ruiz. *Iterated Greedy*, pages 547–577. Springer International Publishing, Cham, 2018.

[236] K. Sugiyama and K. Misue. A simple and unified method for drawing graphs: Magnetic-spring algorithm. In *International Symposium on Graph Drawing*, pages 364–375. Springer, 1994.

[237] O. Sýkora, L. Torok, and I. Vrt'o. The Cyclic Antibandwidth Problem. *Electronic Notes in Discrete Mathematics*, 22:223–227, 2005.

[238] E.-G. Talbi. *Metaheuristics: from design to implementation*. John Wiley & Sons, 2009.

[239] R. Tamassia, G. Di Battista, and C. Batini. Automatic graph drawing and readability of diagrams. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(1):61–79, 1988.

[240] D. W. H. Ten, S. Manickam, S. Ramadass, and H. A. Al Bazar. Study on advanced visualization tools in network monitoring platform. In *2009 Third UKSim European Symposium on Computer Modeling and Simulation*, pages 445–449. IEEE, 2009.

[241] D. M. Thilikos, M. Serna, and H. L. Bodlaender. Cutwidth ii: Algorithms for partial w-trees of bounded degree. *Journal of Algorithms*, 56(1):25–49, 2005.

[242] Q. D. Truong, Q. B. Truong, and T. Dkaki. Graph methods for social network analysis. In *International Conference on Nature of computation and Communication*, pages 276–286. Springer, 2016.

[243] G. Van Rossum and F. L. Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[244] V. V.Cernỳ. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of optimization theory and applications*, 45(1):41–51, 1985.

[245] J. Venn. I. on the diagrammatic and mechanical representation of propositions and reasonings. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 10(59):1–18, 1880.

[246] M. Vidoni. Beyond hard and soft or: operational research from a software engineering perspective. *Journal of the Operational Research Society*, 73(4):693–715, 2022.

[247] C. Voudouris. Guided local search—an illustrative example in function optimisation. *BT Technology Journal*, 16(3):46–50, 1998.

[248] C. Voudouris and E. Tsang. Function optimization using guided local search. *University of Essex, Technical Report CSM-249, Colchester, UK*, 1995.

[249] Y. Weili, L. Xiaoxu, and Z. Ju. Dual bandwidth of some special trees. *Journal-zhengzhou University Natural Science Edition*, 35(3):16–19, 2003.

[250] R. Wiese, M. Eiglsperger, and M. Kaufmann. yfiles—visualization and automatic layout of graphs. In *Graph Drawing Software*, pages 173–191. Springer, 2004.

[251] F. Wilcoxon. Individual comparisons by ranking methods. In *Breakthroughs in statistics*, pages 196–202. Springer, 1992.

[252] D. P. Williamson and D. B. Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

[253] A. Wolff. Drawing subway maps: A survey. *Informatik-Forschung und Entwicklung*, 22:23–44, 2007.

[254] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[255] D. H. Wolpert, W. G. Macready, et al. No free lunch theorems for search. Technical report, Technical Report SFI-TR-95-02-010, Santa Fe Institute, 1995.

[256] X. Yao, Y. Liu, and G. Lin. Evolutionary programming made faster. *IEEE Transactions on Evolutionary computation*, 3(2):82–102, 1999.

[257] X. Ying. An overview of overfitting and its solutions. In *Journal of physics: Conference series*, volume 1168. IOP Publishing, 2019.

[258] Y. Yonezawa and T. Kikuchi. Ecological algorithm for optimal ordering used by collective honey bee behavior. In *MHS'96 Proceedings of the Seventh International Symposium on Micro Machine and Human Science*, pages 249–256. IEEE, 1996.

[259] J. Yuan. Cyclic arrangement of graphs. In *Graph Theory Notes of New York*, pages 6–10, New York, NY, USA, 1995. New York Academy of Sciences.

[260] S. H. Zanakis and J. R. Evans. Heuristic "optimization": Why, when, and how to use it. *Interfaces*, 11(5):84–91, 1981.

[261] S. H. Zanakis, J. R. Evans, and A. A. Vazacopoulos. Heuristic methods and applications: a categorized survey. *European Journal of Operational Research*, 43(1):88–110, 1989.

[262] W. Zeng, C.-W. Fu, S. M. Arisona, A. Erath, and H. Qu. Visualizing mobility of public transportation system. *IEEE transactions on visualization and computer graphics*, 20(12):1833–1842, 2014.

[263] P. Zhang and Y. Itan. Biological network approaches and applications in rare disease studies. *Genes*, 10(10):797, 2019.

[264] S. Zhou. Bounding the bandwidths for graphs. *Theoretical computer science*, 249(2):357–368, 2000.

[265] D. W. Zimmerman and B. D. Zumbo. Relative power of the wilcoxon test, the friedman test, and repeated-measures anova on ranks. *The Journal of Experimental Education*, 62(1):75–86, 1993.

Glossary

Symbols

A

B

C

T

U

V

W