Universidad Rey Juan Carlos

Tesis Doctoral

Graph layout problems: a metaheuristic approach

Autor:

Jesús Sánchez-Oro Calvo

Directores:

Abraham Duarte Muñoz
Rafael Martí Cunquero

2016

Esta tesis doctoral ha sido desarrollada bajo la financiación de los fondos asociados al proyecto TIN2012-35632-C02-02 a través de la beca de formación de profesorado con referencia BES-2013-062658 del Ministerio de Economía y Competitividad.

Abraham Duarte Muñoz, Profesor Titular de Universidad del Departamento de Ciencias de la Computación, Arquitectura de la Computación, Lenguajes y Sistemas Informáticos y Estadística e Investigación Operativa de la Universidad Rey Juan Carlos, y Rafael Martí Cunquero, Profesor Catedrático del Departamento de Estadística e Investigación Operativa de la Universitat de València, directores de la Tesis Doctoral *Graph layout problems: a metaheuristic approach*, realizada por el doctorando Jesús Sánchez-Oro Calvo, hacen constar que esta Tesis Doctoral reúne los requisitos necesarios para su defensa y aprobación.

En Móstoles, a 3 de noviembre de 2016

Fdo: Abraham Duarte Muñoz   Fdo: Rafael Martí Cunquero

Agradecimientos

Esta tesis es el resultado de cuatro largos años de trabajo, en el que han participado una gran cantidad de personas. Todas ellas, de una u otra manera, me han marcado y me han hecho evolucionar, tanto en lo profesional como en lo personal, por lo que no puedo dejar de agradecer cada una de sus aportaciones en las diferentes etapas que me han llevado a finalizar este trabajo.

Gracias a Leti, mi persona, la que consigue que cada día sea mejor que el anterior, que siempre está ahí pase lo que pase. Si he llegado hasta aquí, es gracias a ti, a que siempre me has apoyado para hacer aquello que me gusta. Llevas media vida conmigo, y nunca me has fallado desde que te conocí en el colegio. Una cosa que poca gente puede decir es que el momento más feliz de su vida se produjo durante la tesis, aunque en mi caso, gracias a ti, es totalmente cierto. Siempre encuentras las palabras justas que me hacen falta para seguir adelante y olvidarme de todo lo malo, incluso cuando tú misma no te encuentras bien. Sin duda puedo decir que he encontrado el óptimo global en el problema de optimización de mi vida.

Gracias a mis padres, por haberme educado y enseñado el camino correcto, y por los esfuerzos realizados para que siempre, pasase lo que pasase, tuviera la oportunidad de dedicarme a lo que más me gusta. Gracias también a mi hermana, porque nunca ha asumido el rol de hermana pequeña, sino que se ha mostrado fuerte para ayudarnos siempre que ha hecho falta. Gracias a Dani, que en vez de cuñado ha sido siempre mi hermano mayor. Gracias también a mi segunda familia: Belén, Luis, Ángel, Carmen, Arturo (padre), Bea, Irene, Víctor, Peter, Arturo (hijo), ... Os conozco desde los 14 años y ya no puedo hablar de "segunda" familia, porque es como si sólo tuviera una muy grande.

Gracias también a Juan Carlos. Aunque la guerra te llevó demasiado pronto, los pocos años que te conocí fueron más que suficientes para saber que eres de los mejores amigos que he tenido. Espero que, estés donde estés, veas que, por fin, termino aquello que empecé cuando te conocimos.

Por supuesto, gracias a mis tutores, Abraham y Rafa, sin ellos esta Tesis Doctoral habría sido imposible. Tal y como dice Borges, *todo encuentro casual es una cita*, y aunque el primer trabajo que comencé con vosotros fue de casualidad,

resultó en el descubrimiento de un área de trabajo hasta entonces inexistente para mi, pero que se ha convertido en una afición más que en un trabajo. Gracias por la confianza depositada para ese primer trabajo, y también gracias por darme la oportunidad de formarme con vosotros y por seguir enseñándome día a día, y más teniendo en cuenta la cantidad de estudiantes que daría lo que fuera por tener mi suerte. Además del campo profesional, también agradeceros la cercanía en el trato recibido desde el principio, lo que os ha convertido no solo en mis directores de tesis, sino también en mis amigos. Nunca olvidaré los viajes a congresos y las estancias, donde hemos pasado lo mejor y lo peor, pero que siempre han merecido la pena, no solo en lo profesional sino en las experiencias personales que de otra manera sería imposible tener. Creo que pocos estudiantes de doctorado pueden agradecer a sus directores que le hayan enseñado el maravilloso mundo de la cerveza.

Gracias a Eduardo, mi tutor en la sombra, te conocí en el último año de carrera y desde entonces no he parado de aprender de ti. Creo que ninguna otra persona habría sido capaz de poner un poco de orden en el “Chiringuito” del laboratorio 132, y de recuperar trabajos en principio perdidos gracias a tu constancia. Espero que algún día se den cuenta del profesional que han dejado escapar.

Gracias a Antonio y Juanjo, que me recogisteis recién salido de la carrera, sin tener nada claro ni mi futuro ni mis posibilidades, y me descubristeis el campo de la investigación, enseñándome a qué me quería dedicar y luchando por obtener financiación que me permitiera dedicarme a aquello que más me gustaba.

Gracias a Rulo, Mica y Patxi, por aceptarme como alumno de Proyecto Fin de Carrera. Con vosotros aprendí más durante el desarrollo del proyecto que en varios años de carrera. Puedo decir que soy muy afortunado habiendo trabajado con los mejores en sus respectivos campos, en aquel *binding* de Java para OpenCL que tanto trabajo costó, pero que tan buenos resultados produjo.

Gracias a David, por haber pasado de ser mi profesor de OpenCV a uno de mis mejores amigos, espero que esas cenas en VIPS se sigan repitiendo cada vez con más frecuencia. Gracias también a Noelia, porque sé que ella ha sido tu principal apoyo en convertirte en lo que eres y, por transitividad, también me ha ayudado a mi mientras te apoyaba a ti. Los cuatro hemos pasado juntos algunos de los momentos más importantes de nuestra vida, incluyendo dos bodas, y estoy seguro de que lo mejor está por llegar.

Gracias a Concha, porque ya van más de cinco años trabajando codo con codo, sin que pase ningún día en el que aprenda un nuevo dato de la *Conchapedia.* Aún no entiendo (y creo que nunca lo haré) como es posible que te quepan tantos datos y tan inverosímiles en la cabeza. Gracias también por esas tardes de Bloodborne, Dark Souls y derivados en los que todos los problemas se reducen a adivinar cómo conseguimos matar al jefe de la zona que nos toca jugar.

Gracias a Colli y Zenko, que completan el grupo de los becarios originales, que me acogieron primero en el laboratorio de GAVAB y luego en el despacho 130, batiendo récords de ocupación con cinco personas en el mismo despacho. Siempre recordaré los ratos muertos viendo vídeos en la "pantalla cara". En esos años quedó claro que cinco cabezas piensan mejor que una cuando nos enfrentábamos a problemas que al principio parecían imposibles de resolver.

No puedo olvidarme del equipo que me encontré en el segundo viaje en Colorado. Cuando llegué, asumía que estaría, como diría Abraham, *sólo como una rata* durante toda la estancia. Sin embargo, a los tres días de estar allí volvéis de uno de vuestros viajes Ana y Laura, llenando de esa gracia andaluza los cubículos en los que trabajábamos, y dando un giro de 180 grados a mi estancia. Pocos días después aparece Anna recién llegada de Valencia, y nos descubre aspectos que nunca habría imaginado (la escena de baile en el License No 1 se me ha quedado grabada a fuego). En el breve período que pasamos allí me habéis demostrado mucho más que algunas personas que conozco desde hace años. Nunca podré olvidar la ruta *happy hour* por Pearl Street, ni mi variado número de roles dependiendo de la situación. El grupo se completó con Cristóbal, Raquel, y Rocío, dejando la marca de nuestra tierra por todo Boulder.

Gracias al grupo de investigación GRAFO (Abraham, Alf, Eduardo, Patxi, Chema y Mica), por hacerme sentir uno más del grupo, incluso cuando llegué nuevo y sin experiencia. Gracias al antiguo GAVAB, por apoyarme desde el principio y ayudarme en todo lo que ha hecho falta y más: Abraham, Alf, Ángel, Antonio, Belén, César, José Miguel, Juanjo, Mica, Patxi, Soto y Vélez. Gracias también al nuevo departamento CALE, porque aún habiéndose formado por grupos muy heterogéneos, hemos sabido actuar como un único grupo siempre que ha sido necesario.

Gracias a Manuel Laguna, no solo por permitirme realizar la estancia con él en Colorado y por el aprendizaje profesional, sino por la experiencia personal de haberme hecho sentir como uno más de la familia las veces que he estado allí y las que (seguro) estaré. Gracias también a Marc Sevaux y a André Rossi, por permitirme realizar con ellos una estancia muy productiva en Lorient. Aunque fue una estancia de una única semana, pude aprender mucho de vosotros gracias a vuestra total disponibilidad y paciencia.

Contents

Part I

PhD dissertation

Chapter 1

Introduction

The word optimization, when used in informal language, is considered as a synonym of improvement. However, in the scientific context, it is defined as the process of searching the best solution for a particular problem, among all possible solutions. It can be said that an optimization problem always presents two different features: there are several solutions (usually a huge number of them) and there is a clear method for comparing them [38]. The quality of each solution is computed with the objective function of the problem.

Let us define an optimization problem (OP) as the minimization (or maximization) of the value of an objective function $f(\varphi)$, being φ a solution of the problem, and subject to a set of constraints [109]. In mathematical terms,

$$OP = \begin{cases} \text{Minimize} & f(\varphi) \\ \text{subject to} & \varphi \in \Phi \end{cases} \tag{1.1}$$

where Φ is the set of feasible solutions (those which satisfy every problem constraint). A feasible solution is optimum if and only if its objective function value is the smallest (in minimization problems) or the largest (in maximization problems) among all feasible solutions.

The relevance of optimization problems in several areas of the industry and science has lead the scientific community to put considerable effort in the design of new algorithms and methodologies in order to be able to solve this kind of problems. These new algorithms have been designed with the goal of solving real-world problems emerging in different areas, such as vehicle routing, storage or communication antennas location, selection of the best distribution for electronic components of an integrated circuit, or the prediction of energy demand in different countries, among others.

Solving an optimization problem basically consists on finding the best values of a set of decision variables in terms of an objective function, while satisfying some given constraints. Optimization has been studied within differents areas of knowledge, being Mathematical Programming and Computer Science two of the most prominent. A classical clasification of the different optimization problems according to their type of variables and mathematical expression of the objective and constraints, breaks down Mathematical Programming into three well established fields: Linear Programming, in which the variables take real (continuous) numbers and both the objetive and constraint are linear; Integer Linear Programming, in which additionally the variables are integer, and Non-linear Programming also known as Global Optimization, in which the variables take real numbers and there is no limitation in the expression of objective and constraints. This classification assumes that the mathematical expression of the objective and constraints is known, which is not always the case. As a matter of fact, we can find many real problems in industry and business that are very difficult to formulate in mathematical terms and are difficult to solve due to their combinatorial nature. In this doctoral thesis we consider combinatorial optimization problems (COP), with solutions taking integer numbers [7, 109]. Without loss of generality, a solution φ is represented as a vector $\varphi = (x_1, x_2, \ldots, x_n)$, being n the dimensionality of the problem. Then, each variable x_i of the COP receives a value v_i belonging to the domain D_i (with $i = 1 \ldots n$), so that solution φ fulfills all the problem constraints. More formally,

$$COP = \begin{cases} \varphi = \{x_1, \ldots, x_n\} \\ D_1, \ldots, D_n \\ f : D_1 \times D_2 \times \ldots \times D_n \to \mathbb{R} \\ \varphi = \{\{(x_1, v_1), \ldots, (x_n, v_n)\} | v_i \in D_i\} \end{cases} \tag{1.2}$$

Optimization problems can also be classified according to their computational complexity (how hard it is to be solved by a computer). This taxonomy establishes complexity classes, where each class contains a family of problems that shares a certain attribute related to its complexity. Some problems can be solved in polynomial time, which means that the time taken to solve the problem in terms of the steps or operations performed by the algorithm, can be expressed, or upper bounded, as a polynomial function of the input size of the problem. We denote this time as $O(n^p)$ where p is the maximum exponent in the polynomial function, and all the problems that share this characteristic belong to $\mathcal{P}$. Examples of this kind of problems are the Minimum Spanning Tree (solved with either the Kruskal [83] or Prim [115] algorithms), the Shortest Path Problem (solved with either the Dijsktra [34] or Bellman-Ford [10] algorithms), and the Network Flow Problem (solved with the Ford-Fulkerson [49] algorithm).

Unfortunately, we can find many problems for which a polynomial algorithm is not known. There are some problems in this group however, for which it is possible to determine (in polynomial time) if a given value is a solution for the problem. These problems belong to the $\mathcal{NP}$ class. It is important to remark that all the problems belonging to $\mathcal{P}$ also belong to $\mathcal{NP}$, because it can be determined in polynomial time if a given solution is a solution for the problem. Although the theory of the complexity is a subtle field in which there are not only optimization but also decision problems and reductions from a problem into another, it is well accepted that the term $\mathcal{NP}$-hard is used to refer to difficult problems that are worth to investigate and constitute a challenge to optimization algorithms.

Most of the real-world problems belong to this set, and the need for solving them has motivated the development of efficient methods for finding high quality solutions for the problem, although without any guarantee of optimality. These methods for solving hard problems that consider that the speed of the search process is as important as its quality are called heuristic methods. The work heuristic comes from the Greek word *heuriskein*, which means to find or to discover. On the contrary of exact methods, those which find the optimum solution for the considered problem [109, 106], heuristic methods provide good solutions, but not necessarily the optimum ones. It is worth mentioning that the time needed by an exact method to find the optimum solution is usually orders of magnitude higher than the time needed by a heuristic method.

In addition to the computing time, there exist more reasons for using a heuristic method instead an exact one. For instance, there is no exact method known for some problems (i.e., when considering optimization of simulations [4]). Furthermore, heuristic methods are more flexible than exact ones, so that it is easier to add new conditions which are rather difficult to model in exact methods.

Metaheuristics belong to a new and more complex class of algorithms that emerged with the objective of improving the results obtained by traditional heuristic methods. Conceptually, metaheuristic algorithms have been designed to guide the search of heuristic methods. The term "metaheuristic" was originally introduced by Glover in 1986 [56], with the following definition:

> *A metaheuristic refers to a master strategy that guides and modifies other heuristics to produce solutions beyond those that are normally generated in a quest for local optimality. The heuristics guided by such a meta-strategy may be high level procedures or may embody nothing more than a description of available moves for transforming one solution into another, together with an associated evaluation rule.*

Therefore, metaheuristics are considered as a family of approximated algorithms that leads the search of a traditional heuristic procedure combining intelligently both exploitation (intensification) and exploration (diversification) of the search space of the problem considered to be solved. On the one hand, the exploitation consists of intensively exploring a limited but promising region of the search space looking for improvements in the incumbent solution, and it is traditionally related to local search methods. On the other hand, exploration consists of exploring a large portion of the search space in order to increase diversification. These algorithms have been successfully used in several optimization problems, providing high quality solutions (without guaranteeing optimality) in a reasonable computational time. Although there exists a large number of metaheuristic strategies, some of them have become renowned due to their high success rate when applied to optimization problems. It is worth mentioning, among others, *Greedy Randomized Adaptive Search Procedure*, GRASP [45]; *Iterated Local Search*, ILS [96]; *Scatter Search*, SS [88]; *Simulated Annealing*, SA [139]; *Tabu Search*, TS [61]; and *Variable Neighborhood Search*, VNS [103] .

This PhD dissertation is intended to develop algorithms based on some of these metaheuristics in order to provide high quality solutions for optimization problems with real-world applications. This manuscript summarizes the research performed and it is structured as follows:

- **Part I** contains the summary about the developed research, including the problems considered, the metaheuristics used, and the results obtained for each of the problems, as well as the conclusions derived from them.

  - **Chapter 1** introduces the dissertation and describes its main contributions.
  - **Chapter 2** presents graph layout problems, which are the type of problems considered in this dissertation. Furthermore, it thoroughly defines each of the considered problems, supported with graphical examples, and introduces the best previous works found in the state of the art. Finally, the practical application of each problem is also introduced, in order to contextualize them.
  - **Chapter 3** describes the methodologies successfully applied to the collection of problems described in the previous chapter. The origin of each metaheuristic is described, as well as the basic principle of operation. Additional details like extensions or applications are also included.
  - **Chapter 5** discusses the results obtained by the proposed algorithms for each considered problem. This chapter also includes a summary of the experimental comparison in which the performance of each proposed algorithm can be analyzed. Furthermore, the main contributions made are also highlighted in each section of the chapter.
  - **Chapter 6** ends the dissertation with the conclusions derived from the research, presenting possible future lines of work.

- **Part II** contains the publications associated with this dissertation, with additional information about the journal in which it was published. Furthermore, additional publications derived from the research are also included.

- **Part III** presents the abstract and conclusions of the dissertation in Spanish language.

Chapter 2

Graph layout problems

In recent years graph layout problems are becoming more interesting for the scientific community. The term layout problem comes from an original application to the optimal layout of circuits in the context of Very Large Scale Integration (VLSI) design. In this context, an electronic circuit is modeled as a graph, where the edges represent the wires and the vertices represent each module of the circuit. The objective in VLSI design is to place the modules on a board without overlapping, and wiring together the terminals of the different modules.

From a theoretical point of view, graph layout problems consist of projecting an original graph $G = (V_G, E_G)$ into a host graph $H = (V_H, E_H)$. The projection is performed by defining a function that maps the vertices of G into the vertices of H, associating a path in H for each edge of G. The quality of this projection depends on the dilation (length of the largest path in H), the congestion (the largest number of paths that share an edge of H), and the load (the maximum number of vertices of G that are mapped to a vertex of H) [33].

Let $G = (V, E)$ be a finite and undirected graph where V and E represent the set of vertices and edges, respectively ($|V| = n$ and $|E| = m$). Figure 2.1 depicts an example graph $G = (V, E)$ with vertex set $V = \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{E}, \mathtt{F}, \mathtt{G}\}$ and edge set $E = \{(\mathtt{A}, \mathtt{B}), (\mathtt{A}, \mathtt{C}), (\mathtt{A}, \mathtt{D}), (\mathtt{A}, \mathtt{G}), (\mathtt{B}, \mathtt{F}), (\mathtt{B}, \mathtt{G}), (\mathtt{C}, \mathtt{D}), (\mathtt{C}, \mathtt{E}), (\mathtt{C}, \mathtt{F}), (\mathtt{D}, \mathtt{E}), (\mathtt{E}, \mathtt{F})\}$, with $n = 7$ and $m = 11$.

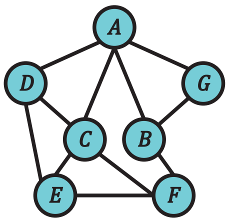


Figure 2.1: Graph G with seven vertices and eleven edges.

Problems considered in this work refers to the case in which a graph with n vertices is projected over a path graph with a load equals to 1. More formally, it consists of defining a bijective function, φ, where each vertex $v \in V$ and edge $(v, u) \in E$ has a unique correspondence in H. This function $\varphi : V \rightarrow \{1, \ldots, n\}$ assigns a different integer number in the range $[1, n]$ to each vertex $v \in V$. The path graph H is usually represented by aligning its vertices in an horizontal line, where each vertex v is located at position $\varphi(v)$. This graphical representation is known as linear layout, but it can be found with different names in the literature: linear ordering [1], linear arrangement [134], numbering [23], embedding in the line [111], or labeling [78]. Figure 2.2 presents an example of a layout for graph G depicted in Figure 2.1. In this layout, vertex D occupies the first position ($\varphi(\mathtt{D}) = 1$), vertex B the second one ($\varphi(\mathtt{B}) = 2$), and so on. For the sake of simplicity, the linear layout is usually represented as an array $\varphi = \{\mathtt{D}, \mathtt{B}, \mathtt{F}, \mathtt{E}, \mathtt{A}, \mathtt{G}, \mathtt{C}\}$, where the position of each vertex $v \in V$ in this array corresponds to $\varphi(v)$.

Several problems belonging to the family of graph layout problems, with practical applications in different areas, have been proven to be $\mathcal{NP}$-hard. In this dissertation we have considered four different graph layout problems. Specifically, Vertex Separation Problem (Section 2.1), Profile Minimization Problem (Section 2.2), Cutwidth Minimization Problem (Section 2.3), and

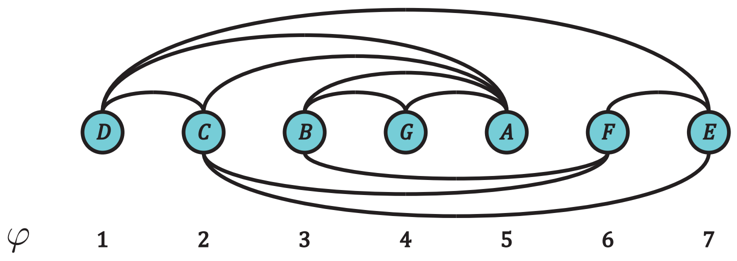


Figure 2.2: A possible linear layout of graph G.

Bandpass Problem (Section 2.4). The remaining of this chapter is devoted to introduce and define each one of the considered problems.

2.1 Vertex Separation

Before formally defining the Vertex Separation Problem, we need to introduce some additional definitions. Let $L(G, \varphi, p)$ be the set of vertices with a position lower than or equal than p in the layout φ of the graph G (i.e., the set of left vertices with respect to position p). Symmetrically, we define $R(G, \varphi, p)$ as the set of vertices with a position larger than p in layout φ (i.e., the set of right vertices with respect to position p). These two sets are formally described in Equation 2.1.

$$\begin{aligned} L(G, \varphi, p) &= \{v \in V : \varphi(v) \leq p\} \\ R(G, \varphi, p) &= \{v \in V : \varphi(v) > p\} \end{aligned} \tag{2.1}$$

We define the separation value at position p of layout φ, $Sep(G, \varphi, p)$, as the number of vertices in $L(G, \varphi, p)$ with one or more adjacent vertices in $R(G, \varphi, p)$. Equation 2.2 contains the mathematical definition of the separation value of a particular position.

$$Sep(G, \varphi, p) = |\{u \in L(G, \varphi, p) : \exists v \in R(G, \varphi, p) \wedge (u, v) \in E\}| \tag{2.2}$$

Finally, the vertex separation value (VS) of layout φ is defined as the maximum of all the separation values among all positions $1 \leq p \leq n$ in φ, as defined in Equation 2.3.

$$VS(G, \varphi) = \max_{1 \leq p < n} Sep(G, \varphi, p) \tag{2.3}$$

The Vertex Separation Problem (VSP) consists of finding a layout $\varphi^\star$ with the minimum vertex separation value among all possible layouts Φ in the graph G. Equation 2.4 formally define the layout with the minimum vertex separation value.

$$\varphi^{\star} = \arg\min_{\varphi \in \Phi} VS(G, \varphi) \tag{2.4}$$

Figure 2.3 shows the evaluation of the vertex separation value of the layout φ depicted in Figure 2.2. The *Sep*-value of each position is highlighted with a dashed line. As it can be seen in the figure, the first position of the layout does not have an associated *Sep*-value, since it is equal to 0 in all cases (no adjacent vertex to the first one can be placed in a lower position in any layout). Therefore, the *Sep*-value starts being evaluated in the second position.

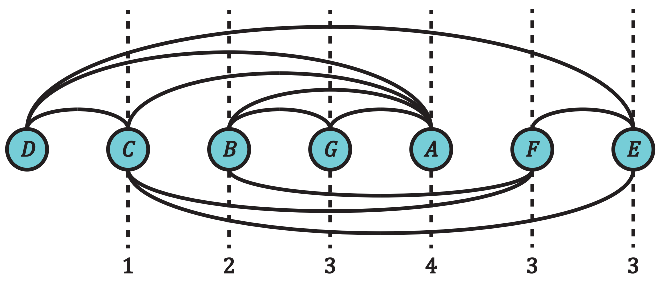


Figure 2.3: Evaluation of the vertex separation value in each position of the layout depicted in Figure 2.2.

The *Sep*-value of second position is equal to one, since $L(G, \varphi, 2) = \{\mathtt{D}\}$, $R(G, \varphi, 2) = \{\mathtt{C}, \mathtt{B}, \mathtt{G}, \mathtt{A}, \mathtt{F}, \mathtt{E}\}$, and there is only one vertex in $L(G, \varphi, 2)$, vertex D, having, at least, one adjacent vertex in $R(G, \varphi, 2)$. Similarly, the *Sep*-value of the third position, where $L(G, \varphi, 3) = \{\mathtt{D}, \mathtt{C}\}$ and $R(G, \varphi, 3) = \{\mathtt{B}, \mathtt{G}, \mathtt{A}, \mathtt{F}, \mathtt{E}\}$, is 2, since there are two vertices in $L(G, \varphi, 3)$ with at least one adjacent vertex in $R(G, \varphi, 3)$. The same evaluation is performed for each position, obtaining a maximum vertex separation value of 4 in the fifth position of the ordering φ. Therefore, the objective function value of the example considered is $VS(G, \varphi) = 4$.

The VSP appears in the context of finding "good separators" for graphs [95] where a separator is a set of vertices or edges whose removal divides the graph into disconnected subgraphs. This optimization problem has several applications in Very Large Scale Integration (VLSI) design for partitioning circuits into smaller subsystems, with a small number of components on the boundary between the subsystems [89]. The decisional version of the VSP consists of finding a Vertex Separation value larger than a given threshold. It has applications on computer language compiler design, where the code to be compiled can be represented as a directed acyclic graph (DAG), with vertices representing the input values to the code as well as the values computed by the operations within the code. An edge from vertex u to vertex v in this DAG represents the fact that value u is one of the inputs to operation v. A topological ordering of the vertices of this DAG represents a valid reordering of the code, and the number of registers needed to evaluate the code in a given ordering is precisely the Vertex Separation number of the ordering [15]. The decisional version of VSP has also applications in graph theory [48]. Specifically, if a graph has a Vertex Separation value, say w, then it is possible to find the maximum independent set of G in time $O(2^w n)$. Other practical applications include Graph Drawing and Natural Language Processing. See [41] and [125] for further details.

VSP has been widely studied from both, exact and heuristic perspectives. Although the optimal value for trees can be obtained by using a linear algorithm, a $O(n \log n)$ algorithm must be considered in order to depict the optimal layout [43]. Later, this algorithm was improved in order to find the optimal layout in linear time [135]. A different algorithm for computing the optimal VS-value of trees was also proposed [112]. A $O(n \log n)$ algorithm for solving the VSP in unicyclic graphs (i.e., trees with an extra edge) was proposed in [42]. Authors in [14] proposed a polynomial-time algorithm for solving the Path Width problem, which is equivalent to the VSP. However, this algorithm cannot be considered from a practical point of view, since the bound on its time complexity is $\Omega(n)$ [42]. The VSP for n-dimensional grids, co-graphs and permutational graphs can be optimally solved in polynomial time using the algorithms proposed in [17, 14, 18], respectively. The VSP has been also studied with approximation algorithms (i.e., those that guarantee a proximity to the optimum). Specifically, a $O(\log^2 n)$-approximation algorithm for general graphs and $O(\log n)$-approximation algorithm for planar graphs were proposed in [16]. Similar results were presented for binomial random graphs in [33].

2.2 Profile Minimization

Given a linear ordering φ for a graph $G = (V, E)$, the profile value of a vertex $v \in V$ located at position p is defined as $\delta_v = p - \omega(v)$, where $\omega(v)$ is the smallest position $q < p$, with $\varphi(u) = q$ such that u is adjacent to v. It is worth mentioning that if $\omega(v)$ does not exists, then $\omega(v) = p$ (i.e., if there is no adjacent vertex in a position smaller than p, then the profile must be 0). Then, the profile value of an ordering φ, denoted as $P(G, \varphi)$ for the graph G, is evaluated as the sum of the profile values of all its vertices. More formally,

$$P(G, \varphi) = \sum_{v \in V} \delta_{\varphi(v)} \tag{2.5}$$

The Profile Minimization Problem (PMP) then consists of finding an ordering $\varphi^\star$ over the set Φ of all possible permutations with the minimum profile value.

Equation 2.6 mathematically defines the best solution for PMP given a graph G.

$$\varphi^\star = \arg\min_{\varphi\in\Phi} P(G,\varphi) \tag{2.6}$$

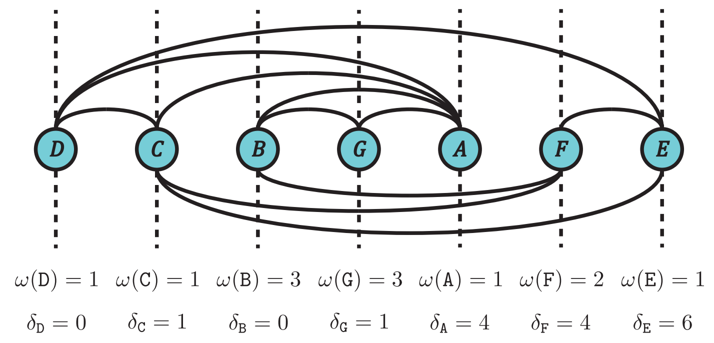


Figure 2.4: Evaluation of the profile value in each position of the layout depicted in Figure 2.2.

Figure 2.4 depicts the evaluation of the profile value in each position of the layout introduced in Figure 2.2. Regarding the first position with vertex D, its profile value is 0 since $\omega(\mathtt{D}) = 1$, because there is no vertex adjacent to D with a smaller position than D. The value $\omega(\mathtt{C})$ for vertex C is equal to 1 since the adjacent vertex to C located in a previous position, which is also smaller than the position of C, is vertex D, located at position 1. Therefore, $\delta_\mathtt{C} = 2 - 1 = 1$. The evaluation is performed for each vertex in the layout, until reaching the last vertex, E, where $\omega(\mathtt{E}) = 1$ since the adjacent vertex to E that presents the smallest position is D, located at position 1. Then, $\delta_\mathtt{E} = 7 - 1 = 6$. Finally, the profile value of the considered layout is evaluated as the sum of the profile values of each position, resulting in the evaluation presented in Equation 2.7.

$$\begin{aligned} P(G,\varphi) &= \delta_\mathtt{D} + \delta_\mathtt{C} + \delta_\mathtt{B} + \delta_\mathtt{G} + \delta_\mathtt{A} + \delta_\mathtt{F} + \delta_\mathtt{E} \\ &= 0 + 1 + 0 + 1 + 4 + 4 + 6 \\ &= 16 \end{aligned} \tag{2.7}$$

The PMP was originally proposed as an approach to reduce the space requirements for storing sparse matrices [137]. In this context, the PMP is equivalent to the SumCut problem [118]. One of the main applications of the PMP is the reduction of the space requirements to store systems of equations. Additionally, the PMP enhances the performance of operations on systems of non-linear equations, such as the Cholesky factorization [125]. Another interesting application of the PMP is described by Karp [81] in the context of the

Human Genome Project. The main goals of that project are to identify all genes in human DNA (between 20 and 25 thousand, approximately) and determine the sequences of the approximately 3 billion chemical base pairs associated with human DNA.

The PMP is also useful in archeology [82], where it has been used in connection with the problem of organizing items such as fossils, tools, and jewels according to a specific order. This process is known as "seriation", and consists of placing several items from the same culture in a chronological order determined by a method of establishing dates that are relative to each other. This results in an optimization problem for which the reordering of the rows and columns of a matrix is required, which is equivalent to the projection of a graph in a line. Other applications of the PMP can be found in information retrieval [19] and fingerprinting [81].

Lin and Yuan [94] proved that the PMP for an arbitrary graph is equivalent to the interval graph completion problem, which was shown to be $\mathcal{NP}$-complete [51]. However, special classes of graphs can be optimally solved in polynomial time. For example, Lin and Yuan [94] proposed several polynomial-time algorithms to find the optimal solution of the PMP for paths, wheels, complete bipartite graphs and D4-trees (trees with diameter 4). Likewise, Guan and Williams [69] developed an algorithm to find the optimal solution of the PMP for triangulated graphs.

The SumCut problem [31, 32, 33] and the PMP are equivalent in the sense that a solution to one problem provides a solution to the other problem by reversing the corresponding permutation. Consequently, the optimum of one problem corresponds to the optimum of the other [118]. The PMP is also related to the bandwidth minimization problem (BMP), which consists of finding a permutation of the rows and the columns in a matrix, such that all the nonzero elements are confined to a band that is the closest to the main diagonal.

Algorithmically, the PMP has been tackled from the late sixties. To the best of our knowledge, the first heuristic in the literature consisted of a constructive procedure proposed by Cuthill and McKee [29] known as the Reverse Cuthill–McKee algorithm (RCM). The procedure is based on constructing a level structure of the vertices. Poole et al. [54] improved the RCM algorithm by changing the selection of the root node within a more general level structure. The method is known in the literature as GPS. Gibbs [53] developed an algorithm called GK, which uses a pseudodiameter to produce a new level structure. GK outperforms GPS in solution quality but it requires more computational time. Lewis [90] introduced new implementations that improve the performance of both GK and GPS. His experimental results confirm the superiority of GK over GPS in terms of solution quality.

The best heuristic for the PMP in the literature belongs to Lewis [91]. It uses the simulated annealing (SA) methodology in combination with existing constructive procedures. The SA search starts from a solution constructed with RCM or GK, whichever is better according to the objective function value. In typical SA fashion, neighborhood exploration is performed with moves that are randomly generated. Improving moves are always executed and non-improving moves are only executed with a probability that depends on the current temperature. A so-called cooling schedule controls the systematic reductions of the temperature and the search ends when the temperature reaches a predefined minimum level.

2.3 Cutwidth Minimization

The Cutwidth Minimization Problem (CMP) is an $\mathcal{NP}$-hard [52] layout problem. CMP consists of finding an ordering of the vertices of a graph on a line that minimizes the maximum number of edges between each pair of consecutive vertices. This problem has been formulated in both, combinatorial [113] and mathematical programming [97] ways, but this work is only focused on the former.

Given a graph $G = (V, E)$ and an ordering for the vertices of the graph φ, the cutwidth value of a vertex v with respect to the ordering φ, denoted as $CW(G, \varphi, v)$ is calculated as the number of edges (u, w) satisfying $\varphi(u) \leq \varphi(v) < \varphi(w)$ (see Equation 2.8).

$$CW(G, \varphi, v) = |\{(u, w) \in E : \varphi(u) \leq \varphi(v) < \varphi(w)\}| \tag{2.8}$$

The cutwidth of a given ordering φ over a graph G, denoted as $CW(G, \varphi)$ is computed by using the cutwidth of a vertex. Specifically, it is evaluated as the maximum of all cutwidth values among its vertices, as can be seen in Equation 2.9

$$CW(G, \varphi) = \max_{v \in V} CW(G, \varphi, v) \tag{2.9}$$

The CMP then consists of finding a labeling $\varphi^\star$ with the minimum cutwidth value among the set of all possible layouts Φ for a given graph G. This definition is formally described in Equation 2.10.

$$\varphi^\star = \arg\min_{\varphi \in \Phi} CMP(G, \varphi) \tag{2.10}$$

Figure 2.5 shows the evaluation of the cutwidth value in each position of the layout presented in Figure 2.2. Each cutwidth value is represented with a number below the dashed line that separates each pair of vertices. Starting with position 1, the obtained cutwidth is equal to 3, since there are three edges with an endpoint in a position lower then or equal to 1 (the position under evaluation) and with the other endpoint in a position larger than 1. Specifically, those edges are (`D`, `C`), (`D`, A), and (D, E). The same evaluation is performed for each position until reaching the last position, where the edges involved in the cutwidth evaluation are (D, E), (C, E), and (F, E), resulting in a cutwidth value of 3. Finally, the cutwidth value of this solution is the maximum of all cutwidth values, which corresponds to 8 (cutwidth value in positions three and four).

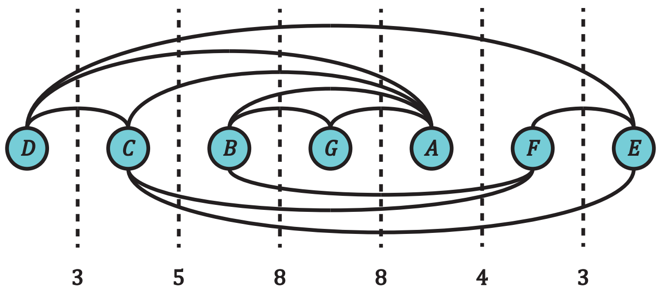


Figure 2.5: Evaluation of the cutwidth value in each position of the layout depicted in Figure 2.2.

Several names have been used for referring the CMP in the literature, such as Minimum Cut Linear Arrangement [30, 136] or Network Migration Scheduling [5, 6]. The Board Permutation Problem was proposed as a generalization of the CMP for hypergraphs [25, 26]. Solving the CMP is relevant for several areas, including circuit design [1, 26, 98], network reliability [80], information retrieval [19], automatic graph drawing [105, 133] or protein engineering [13]. There is also an industrial patent where the CMP is applied to networks migration [119].

The CMP has been the spotlight of the scientific community since the eighties, in order to produce approximate and exact solutions for the problem. Cohoon and Sahni [26] were the first authors proposing heuristic algorithms for the generalized version of the problem. These authors proposed several constructive and local search procedures, embedding them in a Simulated Annealing metaheuristic. Twenty years later [5, 6] a GRASP procedure hybridized with a Path Relinking method was proposed. The most recent approach is based on the Scatter Search metaheuristic [108] which outperformed previous results in the state of the art.

As far as the exact approaches are concerned, most of them are centered in particular types of graphs, like hypercubes [75], special cases of trees [24, 140, 138], and grids [124]. All these algorithms present polynomial-time complexity.

Additionally, some exact methods for general graphs have been also proposed. It is worth mentioning an Integer Linear Programming formulation for the CMP [97], and a Branch & Bound algorithm to find exact solutions to general graphs [100]. However, these approaches can only solve relatively small instances.

2.4 Bandpass

The bandpass problem (BP) emerged in the context of early-generation DWDM (Dense Wavelength Division Multiplexing) systems in fiber optic networks [45, 68, 117]. The transmission of several wavelengths in a single fiber-optic cable is supported by the band concept used in this technology. OADMs (Optical Add/Drop Multiplexers) are devices designed to act as an entrance/exit for wavelengths in the fiber-optic cables [68, 79]. This behavior is controlled by special electronic cards inside the OADMs which manage the wavelengths, identifying which ones should be allowed to continue or not through the device. Several wavelengths are able to pass through the same OADM in most networks, and they may be handled as a block (if they are consecutive). Each considered block is known as a bandpass.

Without considering this wavelength grouping in blocks, each wavelength would need a card. However, a single card can handle the entire block of wavelengths embedded in a bandpass. The bandpass number, B, is defined as the number of wavelengths contained in the block of the bandpass. Then, the creation of a block results in a reduction of the number of cards needed and, as a consequence, a cost reduction not only in the design but also in the maintenance of the fiber-optic network. In this context, it is interesting to assign wavelengths in order to maximize the number of bandpasses with a predefined bandpass number B.

The BP basically consists of creating blocks of wavelengths that need to be sent from a set of origins to a set of destinations. The goal of the problem is to reduce the number of devices needed for the transmission of the wavelengths, resulting in a decrease of both, installation and maintenance costs. The telecommunication network is modeled as a $m \times n$ binary matrix A, where a_{ij} is equal to 1 if wavelength i must be delivered to destination j.

The bandpass number B is given when considering the basic version of the problem. A bandpass is formed when there are B consecutive 1s in the same column of the telecommunication matrix. Each value of 1 can only belong to one bandpass and therefore two distinct bandpasses in the same column cannot have a common element.

Figure 2.6.a shows a network with a set W of 7 wavelengths (`A,B,C,D,E,F`, and `G`) and a set T of 5 destinations (1,2,3,4, and 5). The network shows that, for instance, wavelength 1 must be delivered to all but the third destination. Assuming that $B = 3$, the matrix in Fig. 2.6.a contains 4 bandpasses, as indicated by the highlighted blocks.

The bandpass for column 1 in Fig. 2.6.a, is shown as rows `A, B`, and `C`. However, the bandpass for that column could also be formed by rows `B,C,D` or `C,D,E`. Regardless of which one is chosen, there can only be one bandpass in

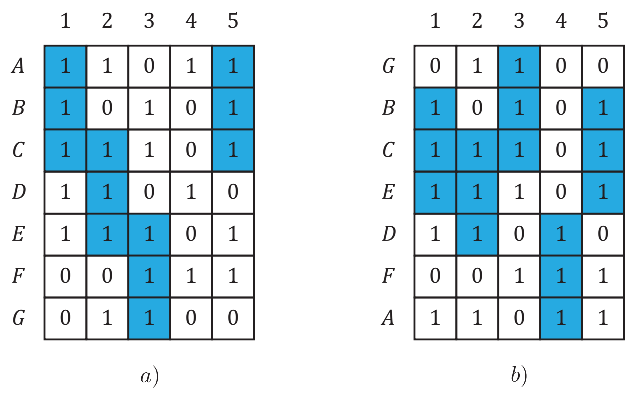


Figure 2.6: Two possible assignments of wavelengths to rows of the telecommunication matrix. Bandpasses are highlighted in light blue.

column 1, because no row can be shared in two different bandpasses. In order to increase the number of bandpasses, the assignment of wavelengths to rows could be changed. This assignment could also be thought as a reordering of the rows in the matrix. For instance, Figure 2.6.b shows the matrix that results by assigning the following wavelength: `G` to row 1, `A` to row 7, `E` to row 4, and `D` to row 5. In other words, wavelengths `A` , `G`, and `D`, `E` have exchanged positions from the original lexicographical order shown in Figure 2.6.a. The reassignment results in an increase of bandpasses from 4 to 5.

The reassignment of rows allows us to consider the BP as a layout problem, where each element of the layout refers to a wavelength and the position of each element in the layout represents the row in which that wavelength is assigned. For example, for Figure 2.6.a, the resulting layout is $\varphi_a = (\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{E}, \mathtt{F}, \mathtt{G})$, since wavelength `A` is assigned to row 1, wavelength `B` is assigned to row 2, and so on. On the other hand, the layout for Figure 2.6.b is $\varphi_b = (\mathtt{G}, \mathtt{B}, \mathtt{C}, \mathtt{E}, \mathtt{D}, \mathtt{F}, \mathtt{A})$, following the description given above.

The literature on the bandpass problem is limited to seven articles [8, 12, 70, 84, 92, 93, 107]. Mathematical models for the original problem and a variant are presented in [8] along with a $\mathcal{NP}$-hard proof. Two solution procedures were proposed: an exact polynomial algorithm with $|T| = 2$ and a heuristic for the general case [8]. The authors also report optimal solutions for the original problem instances with $|W| \leq 32$ and $|T| \leq 8$, found by solving a mathematical programming model. A description of a generator of problem instances is provided and used to produce 90 instances with $m = 64$ and 96 and $n = 8, 12, 16$, and 25. These instances are divided into two groups of 45 instances each, one for which the optimal solutions are known (OS instances) and one for which the optimal solutions are not known (BKS instances). Collectively, these problems are referred to as the Library of Bandpass Problems (BPLIB) [1]. No experiments are performed for the variant in [8]. Additional mathematical proofs for special cases of the original version are provided in [92, 93].

Berberler and Gürsoy [12] developed four variants of a genetic algorithm and applied them to the 45 OS instances in BPLIB. Optimality gaps are reported for all variants in all problem instances. Several GA variants are also developed and tested in [84]. This article introduces a new set of problems for a new variant of the problem, called Multi Bandpass Problem (MBP). The data include both the B_j values and the binary A matrix [2]. Finally, some of the same authors in [8, 84] are co-authors of [107] where mathematical models are presented. Some of these models already appeared in [8, 84] and some are new. No additional experimentation or results are included in [107].

[1]The Library of Bandpass Problems can be found here `http://sci.ege.edu.tr/~math/BandpassProblemsLibrary`

[2]Problem instances introduced in [84] are found here `http://fen.ege.edu.tr/arifgursoy/mopt`

Chapter 3

Metaheuristics

Metaheuristics (MHs) are among the most prominent and successful techniques to solve a large amount of complex and computationally hard combinatorial and numerical optimization problems arising in human activities, such as economics (e.g., portfolio selection), industry (e.g., scheduling or logistics), or engineering (e.g., routing), among many others. MHs can be seen as general algorithmic frameworks that require relatively few modifications to be adapted to tackle a specific problem. They constitute a very diverse family of optimization algorithms including methods such as Simulated Annealing, Tabu Search, Multi-start Methods, Iterated Local Search, Variable Neighbourhood Search, GRASP, Memetic Algorithms, Scatter Search, Evolutionary Algorithms, or Ant Colony Optimization.

Nowadays, MHs have become an interdisciplinary research area intertwining disciplines such as Computer Science, Operations Research, Engineering, etc. They have received enormous attention, as witnessed by thousands of journal and conference papers, hundreds of authored and edited books, and dedicated conference series like the Metaheuristics International Conference (MIC), the IEEE Conference on Evolutionary Computation (CEC), or the Genetic and Evolutionary Computation Conference (GECCO).

This Chapter is devoted to describe the MHs considered in this thesis for solving the $\mathcal{NP}$-hard problems described in Chapter 2. Specifically, we present algorithms based on Variable Neighborhood Search (Section 3.1), Scatter Search (Section 3.2), and Path Relinking (Section 3.3). Finally, Section 3.4 describes the parallel strategies and technologies used for improving the performance of the classical sequential metaheuristics.

3.1 Variable Neighborhood Search

Variable Neighborhood Search (VNS) is a relatively recent metaheuristic [103], also considered a framework for building metaheuristics, which tries to avoid getting stuck in local optima by means of systematic changes of neighborhoods both in descent phase (searching for a local optima), and in perturbation phase to escape from the corresponding local optimum. Contrary to other trajectory-based metaheuristic algorithms, VNS not only follows a trajectory, but also explores distant neighborhoods of the incumbent solution, updating it in case of improvement.

VNS was originally proposed by Hansen and Mladenović [102, 103], where the three main variants (Basic VNS, Variable Neighborhood Descent and Reduced VNS) were originally presented. A more complete survey on VNS was later introduced [72], including different extensions and a detailed review of several areas of application. Finally, the principles and applications of the metaheuristic were gathered in a new volume [73].

Most of the metaheuristics based on local search procedures that can be found in the literature are usually based on a single neighborhood structure although, in some cases (like Tabu Search), the neighborhood structure can be dynamic [56, 61, 67]. VNS metaheuristic introduces the management of a certain set of neighborhood structures as a novel idea. Therefore, for each solution φ belonging to the solution search space Φ, a set of neighborhoods $\mathcal{N} = N_1(\varphi), N_2(\varphi), \ldots, N_k(\varphi)$ is defined, where $1 \leq k \leq k_{\max}$. Each neighborhood can be generated using one or more metrics that are usually problem dependent.

The VNS metaheuristic is based in the following ideas [72]:

1. A local optimum with respect to a neighborhood $N_i(\varphi)$ does not need to be optimum with respect to another neighborhood $N_j(\varphi)$, with $i \neq j$.

2. A global optimum is a local optimum with respect to all possible neighborhood structures.

3. For many problems, local optima with respect to one or several neighborhood structures are relatively closed to each other.

The last idea is based on empirical results but it implies that in some situations a local optimum can provide some information about the global optimum. For instance, a local optimum can contain several components of the global optimum, although in most cases it is not possible to know which are those components [61].

For a certain optimization problem, it is only possible to say that a solution is a local optimum exclusively with respect to a considered neighborhood structure. However, this statement does not need to be valid with respect to other neighborhood structures. Therefore, the neighborhood structure determines the landscape of the search space. These concepts can be summed up in the expression *One operator, one landscape* [77].

Figure 3.1 shows a typical behavior of a VNS algorithm when considering three neighborhoods of the incumbent solution φ. Figure 3.1.a presents the exploration of the first neighborhood $N_1(\varphi)$, selecting the best solution φ' found in it to continue the search. The search continues in Figure 3.1.b, where no improvement is found after exploring the first neighborhood again. Therefore, the next neighborhood to be explored is $N_2(\varphi)$, selecting the best solution (that outperforms the incumbent one). After finding an improvement, the search starts again from the first neighborhood of the new incumbent solution, as can be seen in Figure 3.1. However, since no improvement is found neither in $N_1(\varphi)$ nor in $N_2(\varphi)$, VNS resorts to the third and last neighborhood, $N_3(\varphi)$, finding a better solution. The search continues after no improvement can be found in none of the considered neighborhoods.

VNS has evolved in recent years, resulting in a large variety of strategies. The first work on VNS [103] originally proposed three different variants: Reduced VNS (RVNS), Variable Neighborhood Descent (VND) and Basic VNS (BVNS). The main difference among them is the relevance of the randomness during the search.

In the first variant, RVNS, the exploration of the search space is performed in a stochastic manner. Given a certain neighborhood, RVNS randomly selects a solution belonging to it, without performing any local optimization (i.e., no local search procedure is applied). This implementation is particularly useful when the computational effort to perform the local optimization is relatively large. RVNS is related to Monte-Carlo method, although RVNS is more systematic [72]. Several implementations of this variant can be found in the literature [72, 74, 104].

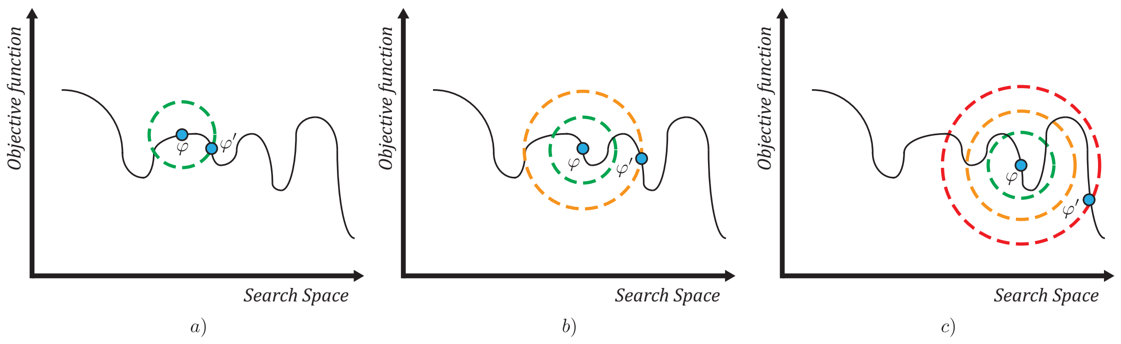


Figure 3.1: Landscape of the search space for two different neighborhood structures.

On the other hand, VND considers a totally deterministic exploration. This variant explores different neighborhoods in a detterministic and exhaustive way. In other words, when the search reaches a local optimum, the neighborhood is changed with the objective of escaping from the local optimum. It is worth mentioning that the solution obtained using this variant is a local optimum with respect to all considered neighborhoods. This variant has been successfully applied in several occasions [20, 71].

Finally, the third strategy, BVNS, combines deterministic and stochastic aspects. In this strategy, a random solution in the neighborhood is selected, as in RVNS, but instead of continuing the search with the next neighborhood, the incumbent solution is improved using a local search strategy. See [20, 71, 121, 122] for some successful results using BVNS.

The successful application of VNS to several $\mathcal{NP}$-hard problems has resulted in new strategies that consider new types of combinations between stochastic and deterministic changes of neighborhood. General Variable Neighborhood Search (GVNS) [72] emerges as a combination of BVNS and VND. Specifically, GVNS follows the same scheme than BVNS, but replacing the local search procedure with a complete VND algorithm. Therefore, the GVNS locally optimizes each random solution selected in a given neighborhood not only with a single neighborhood as in BVNS, but also with a set of different neighborhoods, which eventually leads to better solutions. This implementation has been recently used in several problems with successful results [130, 101, 132].

There exist more complex implementations, like Variable Neighborhood Decomposition Search (VNDS) [72, 71]. This variant is based on the extension of the methodology to two levels. The main difference with respect to BVNS lies largely on the application of the local search procedure. VNDS locally optimizes only a part of the solution, instead of optimizing the complete solution as in BVNS. Other well-known implementations are, for example: Skewed VNS

(SVNS) [72], which provides more flexible and powerful acceptation criteria or the Parallel VNS (PVNS) [72], which will be thoroughly explained in Section 3.4.

3.2 Scatter Search

Scatter Search (SS) is a metaheuristic procedure based on strategies and formulations introduced in the seventies. The basis of SS relies on the combination of high quality solutions, focusing on obtaining better solutions than the original ones. Therefore, SS can be considered as an evolutionary algorithm, since it always maintains a population of solutions [99]. However, unlike evolutionary algorithms, SS is not based on the randomization of a large set of solutions, but on systematic and strategic decisions performed over a small set of solutions.

The basic concepts and principles were proposed at the beginning of the last decade, based on combination strategies for decision rules [99]. The first original description of the metaheuristic [66] establishes the basis of SS, and in 1998 the methodology was formally described [58]. The methods that inspired the ideas of SS were initially used for scheduling problems [55], where several strategies for combining decision rules and constraints were introduced [66]. These methods were able to obtain new rules by means of the lineal combination of other previously used rules [55]. Experimental results showed that this combination produced better results than the original application of the initial rules isolated.

SS is designed to work with a set of small solutions called reference set (*RefSet*), whose main feature is that it contains good solutions. It is worth mentioning that good solutions in the context of SS refers to both high quality solutions (in terms of the value of objective function) or high diversity solutions (which means that solutions are considerable different among them). Notice that finding the subset of most diverse solutions is an $\mathcal{NP}$-hard problem itself, so the most diverse solutions are usually selected in a heuristic way.

A linear combination refers to an expression constructed from a set of two or more vectors where each one can be multiplied by an scalar value. In the context of SS, solution combinations are a more generalized version of linear combination in the euclidean space. However, for most of combinatorial problems it is not easy to apply the linear combination of solutions. Specifically, in the context of graph layout problems (those considered in this dissertation) two solutions cannot be linearly combined. Therefore, in order to use SS in the considered problems, it is necessary to extend the concept of linear combination, introducing new strategies like crossing or voting based combinations [88].

One of the most important features of SS is the inclusion of both improvement and combination methods. The former are usually simple algorithms like local search methods, but there is no limitation for using more complex improving methods like Tabu Search [56], while the latter considers different methods to combine two or more solutions that have been previously improved. The application of the improvement stage ensures that the considered solutions always present a high quality, at least in the given neighborhood.

Algorithmically, the SS metaheuristic basically consists of five different procedures: diversification generation method, improvement method, *RefSet* update method, subset generation method, and solution combination method. Starting with the diversification generation method, its main objective is to generate a set P (with $|P| = n$) of diverse solutions. Once the solutions have been generated, they are improved using the considered improvement method and, finally, a small subset of b good solutions is selected (the so-called *RefSet*).

The goal of the improvement method is basically find a local optimum for each solution with respect to a certain neighborhood. Typically, a local search method is considered, but any trajectory-based metaheuristic can be used. Notice that the method must be able to obtain a feasible solution even when the original one is unfeasible and, then, improve its quality. If the method is not able to find an improvement, then the original solution is considered as the resulting one.

The *RefSet* update method is devoted to construct and maintain the reference set, *RefSet*, which consist of the best b (with $b << n$) solutions found iduring the search. The initial *RefSet* is obtained from the initial population of solutions provided by the diversification generation method, balancing the quality and diversity of the selected solutions. Finally, the solutions in the *RefSet* are usually sorted by quality. Typically, the *RefSet* is created with the best $b/2$ solutions. Then, the remaining $b/2$ solutions are selected maximizing the distance with the solutions already in the *RefSet* (the distance function will depend on the problem).

The new solutions that outperforms those in the *RefSet* must be included in it, removing one solution for each new one included. Therefore, the size of the *RefSet* remains constant, while improving the quality of the solutions in it. There exist two classical strategies for performing the *RefSet* update: static and dynamic. The former stores all the new solutions generated during a combination and, then, it decides if any of the generated solutions is able to enter in the *RefSet*. The latter updates the *RefSet* every time a new solution (better that at least one in the *RefSet*) is generated.

The subset generation method specifies how to construct all the subsets of solutions that will be later considered for combination. The most frequent implementation is based on limiting the subsets to pairs of solutions. The combination method is then applied to each pair generated.

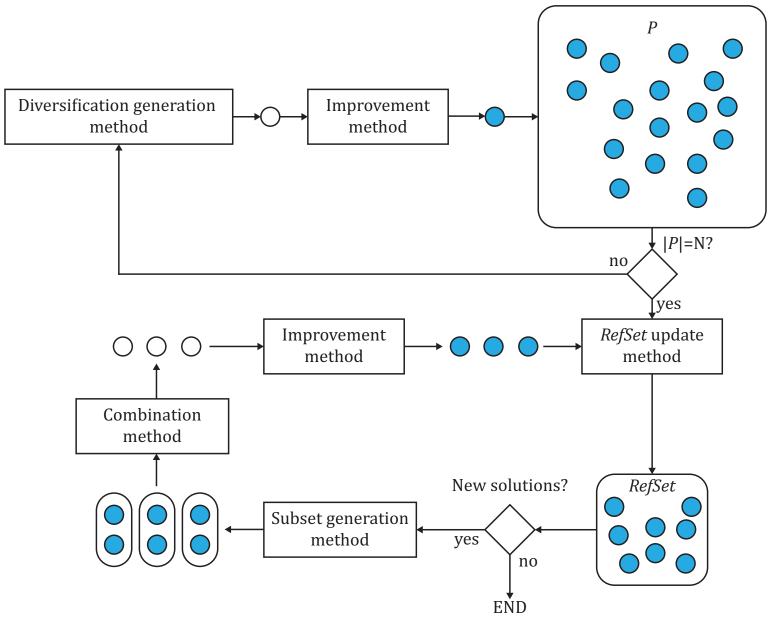


Figure 3.2: Scatter Search methodology scheme.

Figure 3.2 depicts the classical Scatter Search scheme. The algorithm starts by creating solutions with the diversification generation method and improving them with the local optimization procedure. These steps are repeated until generating an initial population P with n different solutions. Then, the *RefSet* update method creates the *RefSet* with the $b/2$ best solutions found in P. Then, the remaining $b/2$ solutions are those from P that present the maximum distance with respect to the solutions already in *RefSet*. The method iterates until no new solutions are included in the *RefSet* as follows. The subset generation method creates the pairs of solutions to be combined with the combination method. The output of the combination method for each pair of solutions combined is the best solution found during the combination, which is improved with the local optimization method. Finally, the *RefSet* update method decides whether the new solution should enter or not in the *RefSet*. Specifically, a new solution is able to enter in the *RefSet* if and only if it outperforms the solution with the lowest quality (in terms of objective function) currently in the *RefSet*.

The flexibility and modularity of the SS methodology allow us to introduce different extensions for the procedures that conform the metaheuristic. Several modifications for the SS methods have been proposed [88, 99, 62, 63, 64, 65], resulting in an increment in the performance of the metaheuristic. The last part of the section is devoted to describe the most promising modifications.

The first considered modification is based on reconstructing the *RefSet*, when the method is not able to generate new solutions that outperforms those already included in the *RefSet*. When reaching this situation, the *RefSet* is partially reconstructed introducing new solutions adding diversity, replacing (typically) $b/2$ solutions for more diverse ones.

Another well-known modification is the inclusion of memory in the search. Frequency based memory usually increment the performance of the metaheuristic, basically due to assuring diversification. This strategy avoid solutions with repeated components or that previously led to low quality solutions.

The Scatter Search metaheuristic has been successfully applied to a large number of problems. See [129, 128, 76, 86] for some successful results using SS.

3.3 Path Relinking

Path Relinking (PR) is a metaheuristic which usually acts as an intensification phase based on the exploration of trajectories or paths that connect two or more high quality solutions which have been obtained by other heuristic or metaheuristic algorithms [57]. It is usually included in the context of Scatter Search, as an extension of its combination method. In this case, PR creates a path between the two solutions selected for combination instead of directly generating a new solution which results from the combination of the two original ones.

PR was originally proposed by Glover [57] as an intensification strategy to generate paths that connect elite solutions provided by a Tabu Search or Scatter Search algorithm [59, 61, 62]. PR was later adapted by Laguna and Martí [87] as a form of intensification in the context of GRASP, by finding a path between a solution found with GRASP and a chosen elite solution. PR is usually applied to two different solutions: the initial solution, and the guiding solution. The algorithm explores one or more paths connecting these solutions looking for better ones. It is worth mentioning that the best solution found in each path does not guarantee local optimality with respect to any neighborhood, so a local search procedure is usually applied to it, in order to find a local optimum.

Let φ_i be the initial solution and φ_g be the guiding one. The exploration of the path that connects φ_i to φ_g is restricted to those solutions in the neighborhood of φ_i that are more similar to φ_g than φ_i itself (i.e., the path generates solutions further from φ_i and closer to φ_g in each step). This is performed by including in φ_i attributes that are already present in φ_g. In this respect, the objective of PR may be viewed as incorporating attributes of the guiding solution in the solution under exploration of the path. In order to select an attribute to be included in the path, the most common strategy consists of selecting the attribute that results in the best-quality solution.

Figure 3.3 depicts an example of the application of the PR algorithm to a pair of solutions for any graph layout problem. For the sake of simplicity, let $f(\varphi)$ the objective function that evaluates a layout φ. Considering that the strategy is being used for a minimization problem, the initial solution, $\varphi_i = (\mathtt{A}, \mathtt{C}, \mathtt{E}, \mathtt{B}, \mathtt{D})$ is better than the guiding one, $\varphi_g = (\mathtt{A}, \mathtt{E}, \mathtt{B}, \mathtt{D}, \mathtt{C})$ since $f(\varphi_i) < f(\varphi_g)$.

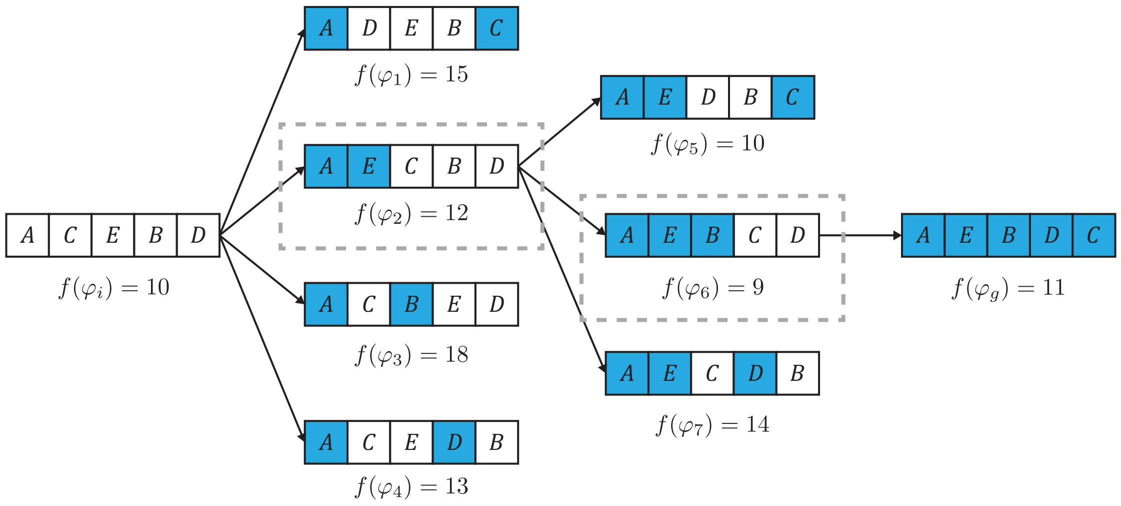


Figure 3.3: Example of Path Relinking between an initial (φ_i) and a guiding (φ_g) solution.

The inclusion of an attribute of the guiding solution into the current one in the context of graph layout problems usually refers to locating a vertex of the current solution in the position that it holds in the guiding solution. This can be accomplished by performing different types of movements. In this example, we have considered the interchange movement, in which given two vertices, they exchange their position in the layout.

Starting from the initial solution φ_i, the strategy can generate four solutions in this neighborhood to start the path to the guiding solution φ_g. Specifically, φ_1 results from the interchange of vertices `D` and `C` in order to locate `C` in the position that it holds in φ_g, which is 5; φ_2 results from the interchange of vertices `C` and `E`, which reallocates `E` in the correct position, and so on. It is worth mentioning that vertex `A` is already located in the same position as in φ_g so it is not considered for the path. Assuming that the corresponding layout problem looks for the minimization of the objective function, the intermediate solution selected in this first step is φ_2, since it presents the minimum objective function among the generated solutions, although it is still larger than the objective function of φ_i. In the next step, solutions φ_5, φ_6, and φ_7 are explored. In this step, the solution with the minimum objective function value is φ_6, which is the one considered for continuing the path. Finally, from solution φ_6 we can directly obtain the guiding solution with only one interchange, which closes the path between φ_i and φ_g. The algorithm returns the best solution found during the path exploration (i.e., φ_6).

There exists a large number of alternatives for PR that have been proposed in recent works. Given two solutions φ_1 and φ_2, the forward implementation considers that the initial solution $\varphi_i = \varphi_1$ and the guiding solution $\varphi_g = \varphi_2$. On the contrary, when considering the backward implementation, $\varphi_i = \varphi_2$ and $\varphi_g = \varphi_1$. The back-and-forward strategy emerges as a combination of the previous implementations, by performing first backward Path Relinking and, then, the forward variant. In PR, the neighborhood of the initial solution is more thoroughly explored than the neighborhood of the guiding one. As a consequence, backward Path Relinking usually performs better than forward Path Relinking. Moreover, back-and-forward Path Relinking obtains, at least, the same results as forward or backward PR isolated, but requiring twice of the computing time approximately.

Mixed PR [65, 123] follows an implementation in which given the initial and guiding solutions φ_i and φ_g, two paths are started simultaneously, one starting at φ_i and the other at φ_g. The method relies on the idea that both paths will meet at some point in an intermediate solution φ_j. The mixed variant explores the neighborhoods of φ_i and φ_g, like back-and-forward implementation, but it usually requires less computing effort.

Resende et al. [120] performed an experimentation to check whether the best solutions found when applying PR are close to the initial and guiding solutions or not. The initial hypothesis (best solutions are close to the extremes of the path) was confirmed empirically by using a back-and-forward implementation for the max-min diversity problem [114]. The obtained results confirm that exploring subpaths near the extremities produces solutions about as good as those found in the complete path, because the concentration of good solutions is higher when moving close to the initial or guiding solutions. Following this idea, it is easy to adapt PR to explore only the path closest to the extremes. This can be done by using a parameter which defines the size of the path explored.

Regarding the path generation, it is important to study the minimum distance between two solutions required for successfully applying PR. If the number of components in which the initial and guiding solutions differs, $\Delta(\varphi_i, \varphi_g)$ is equal to one , then there is a path that directly connects both solutions without generating any intermediate solution.

Assuming that both solutions are locally optimal, any solution found in their corresponding neighborhoods will present lower quality. Therefore, when $\Delta(\varphi_i,$

$\varphi_g) = 2$, any possible generated path will consist in only one intermediate solution which is neighbor of both φ_i and φ_g. Then, considering the local optimality of both φ_i and φ_g, the intermediate solution of any path generated under this assumption will be worse than both φ_i and φ_g. This can be also applied when considering $\Delta(\varphi_i, \varphi_g) = 3$, since there will be two intermediate solutions: a neighbor of φ_i, called φ_1 and a neighbor of φ_g, φ_2, generating a path $\varphi_i \rightarrow \varphi_1 \rightarrow \varphi_2 \rightarrow \varphi_g$. Being φ_i and φ_g locally optimal, it is not possible neither that φ_1 is better than φ_i nor φ_2 is better than φ_g. As a result, it is possible to conclude that it is only interesting to apply PR when $\Delta(\varphi_i, \varphi_g) \geq 4$.

PR can also be randomized in order to increase the portion of the search space explored during the path generation. This can be done by selecting, at each step of the path, a random solution, instead of selecting the best solution among all. This greedy randomized adaptive PR [44] is not constrained to explore a single path, so it can be applied more than once to the same pair of initial and guiding solutions.

Despite the widespread application of PR in combinatorial optimization, almost all PR implementations only consider the between form of PR (Interior Path Relinking). Glover [60] introduced the beyond form of PR, called Exterior Path Relinking (EPR). This new strategy, instead of introducing in the guiding solution attributes of the initial one, is based on introducing in the guiding solutions some attributes that are not present in the initial one. This strategy then obtains intermediate solutions that are further away from both the initial and the guiding solutions.

The relevance of those paths that go beyond the initial and guiding solutions was studied by Glover [57]. The scope of strategies made available by PR is significantly affected by the fact that the term neighborhood has a broader meaning in tabu search than it typically receives in the popular literature on search methods. Often, the neighborhood terminology refers solely to methods that progressively transform one solution into another. Such neighborhoods are called transition neighborhoods in tabu search, and are considered as merely one component of a collection of neighborhoods that also include those operating in regions beyond solutions previously visited. The EPR strategy has been successfully applied in a recent work [40] for solving the differential dispersion minimization problem.

3.4 Parallel Metaheuristics

The application of parallelism to a metaheuristic must be designed in order to reduce the computational time or to increase the exploration of the search space. It is important to remark that the parallelization of a metaheuristic usually involves a complete redesign of the algorithm in order to leverage the resources of the computer. For this reason, several parallel designs have been proposed for the most used metaheuristics [2]. This dissertation is only focused on the parallel versions of Variable Neighborhood Search.

Parallel computing is an ever-evolving discipline which involves both hardware and software. Although this work is focused on software, it is necessary to know the hardware in which the parallel algorithm is being executed in order to make the most of it. Several classification schemes for parallel computer have been proposed, but the most used taxonomy is the one introduced by Flynn in 1972 [47]. Notwithstanding, this taxonomy is not able to describe all possible parallel architecture currently in use, so it is necessary to present some extensions that are able to spread the number of architectures covered by the taxonomy.

The model of Flynn is based on the concept of instruction and data streams, resulting in four different combinations: Single Instruction Single Data stream (SISD), which refers to a traditional sequential computer; Single Instruction Multiple Data stream (SIMD), in which the same instruction is executed by several processors over different data; Multiple Instruction Single Data stream (MISD), where multiple instructions are executed over the same data; and, finally, Multiple Instruction Multiple Data stream (MIMD) which loads different data and programs in each processor.

Most of the parallel computers of nowadays belongs to the MIMD class. However, it does not difference between a single address space shared or a distributed one among modules. The extension of this taxonomy divides MIMD systems into two classes: multiprocessors and multicomputers (distributed systems). In the former, all the processors have direct access to the complete memory, while in the latter each processor has its own local memory and accessing remote memory modules require from additional mechanisms like messaging-passing. Furthermore, multiprocessors can present uniform memory access (if the access time to memory is always constant) or or not. Likewise, distributed systems consists of several computers interconnected, each one with its own processor and local memory. Depending of the type of connection between computers we can difference between cluster of workstations, which is composed of common computers connected through a communication network (that limits the number of processors) or massively parallel processors, which are composed of thousand of processors.

All the algorithms proposed in this dissertation have been developed in a multiprocessor, so the tools considered are all designed for a shared memory model. See [3] for a thoroughly review on parallel tools for multicomputers using distributed memory.

There are three main parallel tools in the context of multiprocessors for developing parallel algorithms, which are Pthreads, Java threads and OpenMP. A thread is defined as the minimum processing unit that can be scheduled by an operating system. Threads have been used by most operating systems since the 1990s, resulting in the multithreading programming model. Although it has been always possible to develop parallel programs using different operating system resources, the use of threads clearly provide some advantages like fast context switching, lower use of resources, and concurrent programming, among others.

Several operating systems of the last decade (most based on Unix) provided their own libraries for multithreading programming, resulting in code that cannot be shared among different computers. Pthreads (POSIX threads) is a standardized library for multithreading programming that emerged as a solution for this problem, providing a set of routines for C programming language in order to increase portability of parallel programs. The Pthreads library contains the necessary routines for performing thread management and synchronization and it is available for most Unix systems.

The advantages of multithreading programming have been reflected not only in the modification of the operating system code but also in the functionality provided by some modern languages. The Java language supplies a multithreading programming library with the same features previously described in Pthreads. However, it offers the advantages of Java code portability, becoming independent of the operating system, and a new programming model adapted to the object-oriented features of Java language.

OpenMP is a set of compiler directives used for expressing shared-memory parallelism. Its programming interface was developed by the major vendors of high performance computing hardware and software, in order to ease the parallelization of algorithms. This parallelization is achieved by adding some compiler directives to sequential programs, indicating the compiler which parts of the program can be executed in parallel, allowing the programmer to establish synchronization points.

All the parallel algorithms presented in this dissertation have been developed using Java threads, since all the sequential code is written in Java and it offers more flexibility than OpenMP for producing new parallel designs for the traditional sequential metaheuristics. Furthermore, the algorithms presented are all parallel versions of Variable Neighborhood Search methodology, previously introduced in Section 3.1.

The parallelization of VNS metaheuristic is done by distributing some steps of the algorithm among all available processors. On the one hand, the efficiency can be increased by performing the same steps than the sequential algorithm by requiring less computing time. On the other hand, the search space can be widely explored by performing more steps than the sequential algorithm in, at most, the same computing time [50]. In particular, four parallel VNS strategies

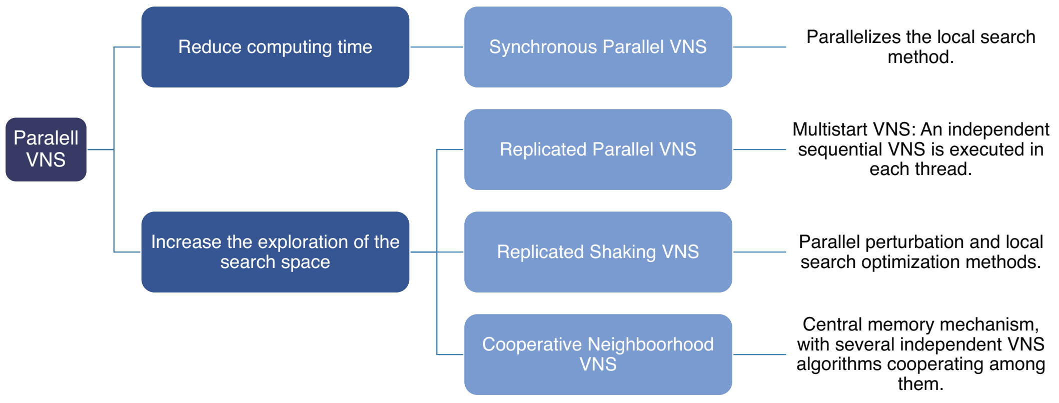

Paralell VNS
Reduce computing time
Increase the exploration of the search space
Synchronous Parallel VNS
Replicated Parallel VNS
Replicated Shaking VNS
Cooperative Neighboorhood VNS
Parallelizes the local search method.
Multistart VNS: An independent sequential VNS is executed in each thread.
Parallel perturbation and local search optimization methods.
Central memory mechanism, with several independent VNS algorithms cooperating among them.

Figure 3.4: Taxonomy of the proposed parallel algorithms depending on its main objective.

have been considered, classified in Figure 3.4. The first one, called Synchronous Parallel VNS (SPVNS) [50], is focused on parallelizing the local search method, which is usually the most time consuming part. The second variant, named Replicated Parallel VNS (RPVNS) [50] increases the portion of the search space explored by using a multistart approach that runs a complete sequential VNS procedure in each available processor. The third variant, called Replicated Shaking VNS (RSVNS) [50], also performs a wider exploration of the search space by parallelizing the perturbation and local optimization methods. Finally, Cooperative Neighborhood Search (CNVNS) [28] considers a central memory mechanism, with several independent VNS algorithms cooperating among them by exchanging information about the best solution found in the search.

It is worth mentioning that the parallelization of a VNS search which is focused on increasing the portion of the search space explored may obtain different (or even worse) results than the sequential version. This is mainly because the increment in the search space explored can guide the search for different paths and, therefore, resulting in different final solutions whose quality is not guaranteed. Furthermore, the creation and management of processes in a parallel algorithm involves additional (but small) computing time. Therefore, a parallel algorithm for reducing the computing time may only be considered when the time required by the sequential version is considerably large, compared with the time needed to manage the processes.

Chapter 4

Justification and objectives

Solving hard optimization problems has become an area of interest for the scientific community, since they are usually a challenge derived from the technological advance of several human activities. The problems usually model real-life situations, which implies a high number of decision variables and constraints. Therefore, it is necessary to propose efficient and effective algorithms in order to solve these problems. Metaheuristics have become one of the most extended strategies for dealing with hard optimization problems, becoming the state-of-the-art methods for a large number of problems that cannot be optimally solve in reasonable computing time.

4.1 Justification

The relevance of an optimization problem can be estimated by considering two criteria. On the one hand, the interest expressed by the scientific community (in terms of previous publications). On the other hand, the problem must have some theoretical or practical applications that justify its study.

The Vertex Separation Problem [36, 131] has several applications in very large scale integration design, computer language compiler design, exponential algorithms, graph theory, graph drawing, and natural language processing, as it was previously described in Section 2.1. The problem has been studied from both exact and heuristic points of view but considering only approximation algorithms for the heuristic approach.

The Profile Minimization Problem was originally proposed for reducing the space requirements for storing sparse matrices, but it presents several applications in difference scientific areas. Specifically, it can be used to enhance the performance of operations on non-linear equations systems, for helping in identifying all genes in human DNA (in the context of Human Genome Project) or for indexing artifacts found in archeology. The best methods found in the state of the art for this problem are traditional heuristic methods and a basic Simulated Annealing algorithm presented in the early '90s.

The applications of the Cutwidth Minimization Problem vary from circuit design, or network reliability to protein engineering. The large number of applications is one of the reason for which the problem has been so extensively studied in recent years. This intensive study makes more and more difficult to find new improvements in terms of quality. Therefore, we propose a different approach for improving the results on the problem. In this case, the main goal is to propose a parallelization of the best previous method that is able to either reduce the computational time required for finding good solutions or to increase the quality of the solutions found.

Finally, the Bandpass problem emerges in the context of telecommunication networks, where the information must be conducted through several nodes minimizing the cost of deploying and maintaining the network. The difficult of this problem makes it suitable for testing the quality of a new Path Relinking approach, called Exterior Path Relinking, which is a novel strategy in the context of solution combination. Furthermore, it is also interesting to include exact and heuristic commercial solvers, such as Gurobi and LocalSolver, in the comparison, in order to empirically analyze the differences between generic and specific algorithms, and elucidate when the effort of developing a specific algorithm is necessary.

There exists a large variety of metaheuristics, as well as different taxonomies to classify them. The most extended taxonomy classifies MHs according to the number of solutions used to scan the search space. Specifically, population based metaheuristics, illustrated in this dissertation by the Scatter Search framework, maintains a set of solutions during the search, while the trajectory-based metaheuristics, illustrated in this dissertation by the Variable Neighborhood Search methodology, maintains a single solution during the search. Finally, it is also interesting to analyze how the inclusion of a combination mechanism affects to the metaheuristic performance, in terms of both solution quality and computing time. For this purpose, different Path Relinking strategies are considered in this research, hybridized with the aforementioned metaheuristic algorithms.

4.2 Objectives

The main goal of this thesis is the development of metaheuristic algorihtms that are able to provide high quality solutions for a certain set of hard combinatorial optimization problems. Specifically, the problems considered are Vertex Separation Problem (Section 2.1), Profile Minimization Problem (Section 2.2), Cutwidth Minimization Problem (Section 2.3), and Bandpass Problem (Section 2.4). The algorithms proposed are based on two different methodologies: Scatter Search and Variable Neighborhood Search. Furthermore, the effect of algorithm parallelization in terms of both quality and computing time is also analyzed. This main objective can be divided into the following milestones:

- **Study the literature related to the considered problems.** It is necessary to review thoroughly those works focused on the studied problems in order to analyze the current best algorithm for each problem, as well as the main contributions of each work. Furthermore, it is necessary to perform a deep analysis on the current scene in metaheuristic research, gathering different techniques for solving hard optimization problems.

- **Design and develop metaheuristic algorithms for the problems.** In this stage, one or more algorithms are developed for each problem, using the studied metaheuristic techniques, and leveraging previous ideas that has led to high quality solutions.

- **Validation of the algorithms.** Each proposed algorithm must be validated, in order to check the feasibility of the proposed solutions. Moreover, a deep analysis on the algorithm will highlight the relevance of each of the included components, as well as their contribution to the final results.

- **Experimental comparison.** The results obtained must be compared with the ones provided by previous algorithms for the same problems, with generic algorithms for combinatorial optimization problems, or with exact or heuristic commercial solvers. This comparison will highlight the strengths and weaknesses of each proposed algorithm.

- **Submit the obtained results to well reputed international journals.** The results derived from the research will be sent to journals and conferences of renowned prestige for their possible publication. Additionally, presenting these results in conferences will allow us to start joint works with recognized researchers in this area.

4.3 Main contributions

This section is intended to present the relation between each publication with the current dissertation. The publications are sorted in ascending order with respect to the publication date, and divided into two parts. Firstly, the publications that resulted from the research in the Combinatorial Optimization problems are presented. Then, those articles that, although they do not tackle the considered problems, were conducted as a consequence of the studies presented in this document are described. During the research, 13 articles indexed in the Journal of Citation Reports (JCR) were published, 6 of them in journals of the first quartile, 3 in the second quartile, 3 in the third quartile and, finally, 1 in the fourth quartile. Additionally, 2 more articles not indexed in the JCR and 4 book chapters were published. Finally, the scientific results obtained were presented in 15 national and international conferences. We now present only those articles indexed in the JCR.

The results of the research performed in the Vertex Separation Problem (Section 2.1 of Chapter 2) have been presented in two different articles: *Variable neighborhood search for the Vertex Separation Problem* (Chapter 7) and *Combining intensification and diversification strategies in VNS. An application to the Vertex Separation Problem* (Chapter 9). The Profile Minimization Problem research resulted in the article *Scatter Search for the profile minimization problem* (Chapter 8), and the results on the parallelization of the Cutwidth Minimization Problem were described in the article *Parallel VNS strategies for the Cutwidth Minimization Problem* (Chapter 10). Finally, the research on the

Bandpass Problem is described in the article *Scatter search for the bandpass problem* (Chapter 11).

Additional publications

- *Radar-based road-traffic monitoring in urban environments* [127]: This paper is devoted to solve a problem related with the road traffic detection and tracking in the field of computer vision. The algorithm proposed is based on Particle Filter, which is a methodology closely related to the Variable Neighborhood Search. Specifically, in both methodologies the random perturbation of the incumbent solution (shake method in VNS and dispersion stage in Particle Filter) are one of the most important parts of the search, increasing the diversification of the method. Therefore, the ideas and results obtained in the previous article were rather relevant for the development of the algorithm presented in this work.

- *Optimization Procedures for the Bipartite Unconstrained 0-1 Quadratic Programming Problem* [37]: The problem considered in this work is not a graph layout problem, since it consists of selecting a subset of vertices of a complete bipartite graph in order to maximize the sum of the weights associated to the edges that connect them. In this work an Iterated Local Search algorithm was proposed, which considers an adaptation of the neighborhoods introduced in [36]. Therefore, although this work does not propose any algorithm based on the metaheuristics considered in this dissertation, it leverages the low-level detail aspects of the constructive and local search methods proposed in the previous work.

- *GRASP with Path Relinking for the Orienteering Problem* [21]: The problem tackled in this work is closely related to graph layout problems. However, in this case, a subset of vertices is selected, instead of considering the complete set of vertices. The algorithm proposed is based on the GRASP methodology, but the solutions generated are combined by means of a Path Relinking algorithm, based on the ideas presented in the Section 3.3 of this dissertation.

- *GRASP with Exterior Path Relinking for differential dispersion minimization* [35]: This work is focused on solving the differential dispersion minimization problem, which consists of selecting the most diverse subset of vertices following a given distance measure. In this work the idea of Exterior Path Relinking was firstly applied, which was afterwards used for solving the Profile Minimization Problem, embedded in the Scatter Search framework.

- *Robust total energy demand estimation with a hybrid Variable Neighborhood Search – Extreme Learning Machine algorithm* [126]: This work was proposed for solving a problem that emerges in the energy field. Specifically, the problem consists of estimating the energy demand of a given country for the next year by using a set of macroeconomic variables obtained in the previous years. The problem was previously tackled by using a Harmony Search algorithm coupled with a neural network. However, in this work we propose to use the ideas of VNS proposed in the works for the Vertex Separation Problem in order to improve the obtained results. Specifically, the macroeconomic variable selection is performed using a novel VNS algorithm, while the neural network uses those selected variables to estimate the energy demand. This work shows the easy adaptation of VNS to problems which are not related to the operations research area.

- *General Variable Neighborhood Search for computing graph separators* [130]: The problem tackled in this work can be seen in the context of the Vertex Separation Problem, since both problems share the objective of selecting good graph separators. However, the problem considered in this work is oriented to select a subset of vertices that separates the remaining ones in two subsets, minimizing the vertices included in this subset. Although this is not an equivalent problem to the VSP, the ideas used in the papers related to the VSP were adapted, proposing a General Variable Search algorithm for solving the vertex separator problem.

- *Parallel variable neighborhood search for the min-max order batching problem* [101]: This paper is intended to solve a problem that emerges in the context of the picking process in large storages, where the order of the product selection drastically affects to the time needed to accomplish the complete order. The ideas of parallel VNS proposed in the work for the Cutwidth Minimization Problem included in this dissertation were adapted to the considered problem, resulting in an effective and efficient algorithm for solving the min-max order batching problem.

- *Efficient Greedy Randomized Adaptive Search Procedure for the Generalized Regenerator Location Problem* [116]: The Generalized Regenerator Location problem consists of selecting a set of relevant vertices in a given random graph that minimizes an objective function based on the edge weights that connects the selected vertices. The similarities of this problem with respect to the Vertex Separation Problem lead us to adapt some of the successful ideas previously presented in the works for the VSP included in this dissertation. Specifically, the same notion of neighborhood was used, but this time embedded in a GRASP algorithm.

Chapter 5

Joint discussion of results

This chapter is intended to present a summary of the different proposals described in this dissertation, as well as a brief discussion of the results obtained in each one of them. It is worth mentioning that, in order to ease the comparison, all tables report the same metrics. Specifically, the average objective function value, Avg.; the average deviation with respect to the best solution found in the experiment, Dev (%); the computing time required to execute the algorithm in seconds, Time (s); and the number of times that the algorithm matches the best solution in the experiment.

Results on Vertex Separation

In the first work related to the Vertex Separation Problem [36], included in Chapter 7 of Part II, we proposed both a pure 0-1 optimization model and an algorithm based on the Variable Neighborhood Search metaheuristic that can be applied in general graphs. The proposed mathematical model has $O(mn^2)$ binary variables and $O(mn^3)$ constraints, which makes it impractical for relatively large instances.

We also propose a metaheuristic algorithm following a Basic VNS (BVNS) scheme, which is executed for a maximum predefined time, which is controlled by a parameter of the algorithm, $t_{\max}$. Specifically, when the BVNS algorithm reaches the largest neighborhood, instead of stopping the search, it starts again from the first neighborhood using the best solution found as the initial one.

The initial solution is constructed by using one of the proposed constructive procedures. The first one is a greedy procedure which starts with an empty solution. The vertices are added to the solution following a greedy function that prioritizes the vertices with many vertices labeled and a small number of vertices unlabeled. The second constructive procedure is based on the creation of level structures, which divides the set of vertices into different sets called levels. Each level contains the vertices adjacent to the ones in the previous level that are not present in any of them. The solution is then constructed by exploring the level structure following a breadth-first search approach.

Additionally, an incremental updating of the objective function is proposed, which drastically reduces the complexity of evaluating an solution. Specifically, the vertex separation value of a given position can be evaluated with respect to the previous one, excepting the first position, whose vertex separation value is always equal to one.

The neighborhood structure proposed leverages the incremental objective function computation to reduce its exploration. In particular, the first neighborhood contains all feasible solutions that can be reached by performing an exchange of two vertices in the incumbent solution.

The VSP is a min-max problem where the value that corresponds to the objective function appears in several positions of the solution. These kind of problems presents a flat landscape which is usually a real challenge for classical local search methods, since most of the performed moves do not result in any improvement. Therefore, a new extended objective function is proposed, where two different solutions with the same vertex separation value can be compared.

The local search strategy traverses all vertices in the solution in descending order with respect to its vertex separation value (worst positions first). Each vertex is exchanged with the remaining vertices in the solution, performing the move that obtains the best solution (only improving moves are considered). The local search ends when no improvement is found.

A second paper on the Vertex Separation Problem [131], included in Chapter 9 of Part II was published intended to review how the balance between intensification and diversification affects to the results of the algorithm proposed. In this work, a new neighborhood, based on the insertion of a vertex in a given position is introduced, additionally to the previous one based on interchanges of two vertices.

Four shake procedures are also proposed. The first one focuses on the diversification, where a set of k randomly selected vertices (being k the current neighborhood) are inserted in random positions of the solution. The second shake procedure is focused on the intensification of the search. Specifically, it selects the k vertices with the highest vertex separation value, inserting them in the best position that generates the best solution. Finally, the third and fourth shake methods are designed as a compromise between intensification and diversification. Specifically, the former selects k vertices at random, inserting them in the best position, while the latter selects the k vertices with the highest vertex separation value, inserting them in random positions.

The algorithm uses the same extended objective function proposed in [36]. Furthermore, the neighborhood based on insertions is implemented in a more efficient way by using these extended objective function, reducing the complexity of the evaluation after performing a move.

Three different algorithms are proposed, namely: a multi-start VND, MS-VND (intensification); a Reduced VNS (diversification), RVNS; and a General VNS, GVNS (balanced). All of them are compared with the BVNS previously proposed in order to analyze the influence of the diversification / intensification in the search.

The computational experiments were performed over a set of 173 instances extracted from three different types of graphs: Harwell-Boeing (which are general graphs), grids, and trees. For the last two set of instances the optimum value is known by construction. The preliminary experimentation examines the performance of the local search when considering the incremental objective function computation, as well as selects the best combination of neighborhood structures for the final VNS algorithm. Table 5.1 shows the results obtained by each variant in the Harwell-Boing challenging instances.

The main conclusion obtained from these results is that the balance between diversification and intensification is a key feature when designing a metaheuristic. In this case, the GVNS, which focuses in this balance, is the one obtaining the best results, over the MS-VND method focused in the intensification and the RVNS mainly centered in diversification. Furthermore, GVNS obtains better results than the previous best method found in the state of the art, which was a

	Avg.	Dev. (%)	Time (s)	# Best
RVNS	27.00	10.85	700.03	28
MS-VND	26.77	11.91	702.89	27
GVNS	24.60	2.07	482.08	53
BestPrev	26.11	6.90	705.60	34

Table 5.1: Comparison of the intensification and diversification balance through different VNS approaches with the best previous method, BestPrev.

BVNS algorithm.

All the results were supported by the non-parametric Friedman test, which ranked the algorithms as follows: GVNS (1.88), BestPrev (2.61), MS-VND (2.75), and RVNS (2.76). The resulting p-value lower than 0.0001 indicates that these results are statistically significant. A pairwise Wilcoxon test was also performed in order to check the differences between the two best algorithms, GVNS and BestPrev. The resulting p-value, lower than 0.0001, confirms the superiority of the proposed GVNS algorithm.

Results on Profile Minimization

We study of the Profile Minimization Problem (PMP) throughout the Scatter Search methodology [128]. The proposed algorithm exploits the network structure of the PMP, including strategies that produce computationally efficient and agile search.

The diversification generation method uses a greedy constructive procedure to construct each solution from scratch, by adding new vertices to it in each iteration. Specifically, the first vertex is the one with the smallest degree and, then, the remaining ones are added following a greedy function that prioritizes the vertices with the highest number of labeled adjacent vertices and the lowest number of adjacent vertices unlabeled. In order to favor diversification, the vertices are added following the Greedy Randomized Adaptive Search Procedure methodology [46], where the next labeled vertex is selected at random among those with the highest greedy function value, instead of greedily selecting the best one. The second constructive procedure proposed inverts the greedy and random stages of the previous one. In particular, the procedure selects, at random, a set of candidate vertices, selecting the one that presents the highest greedy function value among them. An alternative greedy function that takes into account the distance to the labeled adjacent vertices is proposed, resulting in two additional constructive methods.

The local search methods proposed involves the use of two different neighborhoods: one based on swap moves and the other based on interchanges. The evaluation of the objective function after performing a swap move is efficiently calculated by considering only the positions involved in the move. This optimization drastically reduces the computing time needed to execute the algorithm, which lead us to implement the insertions based on a sequence of swap moves.

The combination method included in the Scatter Search methodology is based on Path Relinking. Specifically, two different combination methods are proposed. The first one consists of creating a path between two solutions, the initial and the guiding one. The path is created by locating the vertices of the guiding solution in the position that they occupies in the initial one, throughout insertion moves and selecting, in each step, the best intermediate solution to continue the path. The second path relinking variant introduces the randomization concept, selecting the intermediate solutions at random instead of in a greedy manner. This variant reduces the computing time of creating a path, since it is not necessary to explore the complete set of intermediate solutions at a given step to select the best one.

The computational results have been performed over a set of 73 general graphs derived from the Harwell-Boeing sparse matrix collection, 98 K-graphs (bipartite graphs), and 91 D4-Trees (trees with diameter 4).

The experimentation performed is intended to select the best neighborhood structure as well as the best combination method. This selection is performed in a full-factorial manner, comparing each possible combination of constructive procedure, local search method and combination procedure. The rationale behind this experiment is to confirm the hypothesis that the iterative selection of the best constructive method first, the best local search method, and finally the best combination procedure obtains similar results than performing a computational exhaustive full factorial experiment.

The best Scatter Search variant is then compared with the best heuristic algorithms found in the state of the art, named RCM (heuristic method) and SA (Simulated Annealing). Additionally, we have included the results obtained by LocalSolver [11] and OptQuest [85], two general purpose algorithms, and TS-BMP, an adaptation of the Tabu Search algorithm by Campos et al. [22] for the Bandwidth Problem, which is related to the PMP when considering a matrix representation of the solutions. Table 5.2 shows the comparison among all the aforementioned methods divided in three parts, depending on the instances size. The results only reflect the results over the Harwell-Boeing instances, which are the most challenging instances.

The first conclusion that emerges from the results is that the proposed SS

Experiments with 26 HB instances with $n \leq 200$				
	Avg.	Dev. (%)	Time (s)	# Best
OptQuest	1466.54	46.7	36.9	2
LocalSolver	1030.04	2.4	60.0	23
TS-BMP	1596.83	42.7	30.8	1
RCM	1109.58	20.4	97.3	2
SA	1110.92	14.5	4.0	4
SS	1007.58	1.1	2.8	15
Experiments with 27 HB instances with $200 < n \leq 500$				
	Avg.	Dev. (%)	Time (s)	# Best
OptQuest	8923.07	61.1	211.9	0
LocalSolver	7492.30	25.5	220.0	8
TS-BMP	10204.82	73.4	212.1	0
RCM	7491.85	34.0	335.6	4
SA	7542.48	33.01	39.5	1
SS	5701.48	2.2	78.5	17
Experiments with 20 HB instances with $n > 500$				
	Avg.	Dev. (%)	Time (s)	# Best
OptQuest	27156.00	105.4	1518.4	1
LocalSolver	26078.30	90.0	1575.0	1
TS-BMP	25226.29	61.4	714.7	0
RCM	19256.05	28.5	744.1	1
SA	19246.65	29.2	1367.3	0
SS	15629.05	1.0	674.8	17

Table 5.2: Results obtained by the compared methods divided with respect to the instances size.

algorithm outperforms the remaining ones in terms of quality and computing time. These results are supported by non-parametric statistical test. Specifically, both Friedman test and pairwise Wilcoxon test results in a p-value lower than 0.0001, confirming the superiority of the proposal.

Notwithstanding, it is worth mentioning the performance of the LocalSolver method even being a general purpose algorithm, outperforming the previous problem-specific heuristics in small and medium size instances. Finally, the results obtained by the TS-BMP confirms that, although the representation of both problems may be similar, a specific algorithm for the Bandwidth Problem does not need to be appropriate for the PMP.

Results on Cutwidth Minimization

We tackle the Cutwdith Minimization Problem (CMP) following a parallel perspective [39]. In particular, several parallel designs based on the Variable Neighborhood Search methodology have been tested, comparing them with the best previous algorithm, which consists of a new sequential VNS variant called Variable Formulation Search [110].

The proposed algorithms are based on the parallel designs originally proposed by García et al. [50] and Crainic et al. [28]. Specifically, six variants of parallel VNS algorithms are presented, differing on the stages that are actually parallelized. The corresponding six variants are grouped into three different templates.

The first one is focused on the parallelization of the complete VNS algorithm following a multi-start parallel design. The second one parallelizes the shake method as well as the local search procedures. The last template consists of the parallel exploration of the considered neighborhoods. All the variants are parallel versions of the Variable Formulation Search strategy proposed by Pardo et al. [110], with the aim of testing the efficiency of the parallel schemes versus the sequential one. All the proposed parallel algorithms belong to the multiple search class of parallelization[27].

We propose two variants of the Replicated Parallel VNS, differing in if there exists cooperation among the different VNS procedures. We also present two variants for the Replicated Shaking VNS, where the difference lies in the selection of the best solution for each iteration: while the first algorithm considers a best improvement strategy (it selects the best solution among all threads), the second one considers a first improvement strategy (it selects the first solution obtained in a thread that outperforms the incumbent one). Finally, we propose two Cooperative Neighborhood VNS, where the first one randomly explores the neighborhoods, and the second one explores a given sub-range of neighborhoods in order (from the first to the last one).

The instances used in the computational experiments are the same previously used in [110], which facilitates the comparison among different methods. The set of instances comprise a total of 101 graphs derived from the Harwell-Boeing sparse matrix collection, with vertices ranging from 30 to 3025 and edges ranging from 103 to 8904.

According to Craininc and Toulouse [27], the classical performance measure named speedup [9] is not adequate for comparing parallel metaheuristics since the asynchronous interactions among processes produce different behavior for each execution. Therefore, parallel and sequential metaheuristic should be considered as different algorithms. Furthermore, it is worth mentioning that the parallelization of a methodology does not need to be focused only in reducing the computing time, but also in producing better results by means of a wider exploration of the search space. Notice that each algorihtm has been executed for the same computing time as the best previous algorithm, VFS, in order to have an illustrative comparison.

The preliminary experimentation performed compares each pair of variants for each strategy, selecting the best one for each comparison. It is worth mentioning that different number of processes are considered, selecting the best one for each variant. In particular, 2, 4, 8 and 16 simultaneous processes have been tested. Furthermore, a throughout analysis of the waiting time for each thread is performed, in order to observe the time that each thread needs to wait for the completion of the remaining ones before continuing the search. The results show that CN-VNS is the algorithm that spends the minimum computing time waiting for synchronization, while RP-VNS uses about a 10% of the time in the synchronization stage.

Finally, the best variant of each parallel design is compared with the VFS algorithm. The results of this comparison are presented in Table 5.3. Notice that the processors used for each variant (those with the best results in the preliminary experimentation) are indicated in parenthesis. In this case, the computing time has been removed from the table, since all the algorithms has been executed for the same computing time.

The results clearly shows the superiority of the Replicated Shaking VNS variant (RS-VNS1) over the other competitors, with a deviation and number of best solutions found that compares favorably with respect to the other algorithms. It is important to remark that although CN-VNS2 obtains the worst deviation, it only finds one less better solution than the state-of-the-art method. Even more, considering the average objective function value, CN-VNS2 ranks second, improving RP-VNS1 and VFS.

Non-parametric test have been performed in order to confirm these results.

	Avg.	Dev. (%)	# Best
VFS	285.46	0.89	84
RP-VNS1(4)	285.01	0.64	86
RS-VNS1 (4)	284.64	0.28	92
CN-VNS2 (16)	284.87	1.52	83

Table 5.3: Comparison among parallel algorithms and sequential VFS. The number of processes used in each variant is indicated in parenthesis for each parallel version.

Specifically, the Friedman test resulted in a p-value equal to 0.028, indicating that there are statistical differences among algorithms. The associated rankings for the considered algorithms are: 2.20 (RS-VNS1), 2.45 (RP-VNS1), 2.47 (VFS), and 2.88 (CN-VNS2). Finally, the pairwise Wilcoxon test was performed comparing the best method, RS-VNS1, with the best previous method in the state of the art, VFS. The resulting p-value of 0.024 confirms the superiority of the parallel version.

Results on Bandpass

We use the Scatter Search framework for proposing a metaheuristic algorithm for the Bandpass Problem, combining the solutions with Path Relinking [129].

The diversification generation method follows a semi-greedy strategy that starts with an empty solution. In each step, a vertex is selected at random and, then, inserted in the best position among the ones already located in the solution. In particular, in the first iteration, the selected vertex is assigned to the first position. Then, in the i-th iteration, there are $i-1$ wavelengths (vertices) already added to the solution, and the selection of the best position for the next vertex consists of inserting it in the $i-1$ available positions, starting at position 1. The constructive method ends when all the vertices have been assigned. The random selection of the vertex in each step enables the generation of a diverse population for the Scatter Search framework.

The second diversification method uses a greedy function that evaluates the number of potential bandpasses that can be composed when adding a certain vertex. A potential bandpass is defined as the number of previous consecutive vertices in the permutation that can be grouped in the same wavelength as the considered vertex. The first vertex is selected at random, in order to increase the diversity of the method, while the following vertices are selected following a greedy randomized adaptive search strategy. Specifically, the next vertex is selected at random among the most promising vertex with respect to the greedy function (i.e., the one with the largest number of potential bandpasses).

The search in an improvement method in Scatter Search is local, meaning that it stops as soon as no improved solution is found in the neighborhood of the current solution. Four improvement methods have been proposed for the Bandpass Problem. The first two improvement methods are based on an exhaustive exploration of the neighborhoods generated by insertion and swap moves, respectively. The third local search method is named block merging, and it is based on shifting sets of consecutive vertices in order to form new bandpasses. The last local search method proposed combines the neighborhood based on swaps and the block merging procedure into a Variable Neighborhood Descent algorithm.

Two combination methods for the Bandpass Problem have been proposed: a traditional Interior Path Relinking (IPR) and a novel Exterior Path Relinking (EPR). The IPR algorithm starts by considering two solutions: an initial solution and a guiding one. Then, the vertices of the initial solution are located in the position occupied in the guiding solution, one at each step.

The second combination method, EPR, follows a recently proposed methodology focused on the diversification. EPR searches for intermediate solutions of the path that move away from the guiding solution, instead of moving the initiating solution toward the guiding solution. In the context of the Bandpass Problem, the method inserts each vertex of the initial solution in a random position that must be different than the one held in the guiding position. The combination ends when all the vertices of the initial solution are located in a position different than in the guiding solution.

Regarding the large number of components that must be selected in order to conform a Scatter Search metaheuristic for the Bandpass Problem, a two-factor factorial design is employed, fixing the population size to 100 and the *RefSet* size to 10. The ANOVA associated with the experiment (p-value smaller than 10^{-9}) revealed that both main effects are significant. However, there is no significant integration between the combination method and the improvement method, as indicated with a p-value of 0.881.

The results of the proposed SS algorithm for the Bandpass Problem are compared with the best previous method, named Different Start Rows (DSR) [8] and four genetic algorithms (GA1 to GA4) [12]. Since there is no information about the computing time required for the previous methods, the results are presented without including the computing time. Notwithstanding, for the sake of completeness, the proposed algorithms have been executed for an average time of 875 seconds per instance. An additional column has been added in the results, called GAP, which indicates the average optimality gap. Notice that the optimal value of the instances is known by construction and, then, the # Best column is replaced by the # Opt column, which indicates the number of optimal values found for each algorithm.

Table 5.4 presents the results obtained for each aforementioned algorithm over the set of instances where the optimum value is known. The results indicate that there is little difference in performance among the genetic algorithm variants. DSR produces the lowest quality results, while SS1 dominate all competitors. In order to support this conclusion, a Friedman test was performed, resulting in a *p*-value lower than 0.001.

	Avg.	GAP	# Opt
DSR	37.00	0.3407	0
GA1	40.53	0.2516	3
GA2	40.53	0.2550	3
GA3	40.24	0.2613	2
GA4	40.29	0.2575	3
SS1	46.07	0.1027	16

Table 5.4: Results obtained by each algorithm for the Bandpass Problem.

Chapter 6

Conclusions

This Chapter is devoted to present the conclusions derived during the research performed in this dissertation. Each section provides the conclusions for each one of the considered problems. The chapter is finished with a section describing future lines of research.

Conclusions on the Vertex Separation

The Vertex Separation Problem has been studied following the Variable Neighborhood Search methodology. Firstly, a mathematical model, together with a Basic VNS algorithm for solving the VSP were presented, becoming the first heuristic procedure for the VSP.

The second work on the VSP was intended to study the influence of the balance between diversification and intensification in the context of VNS, supported by an extensive experimental section. A constructive procedure, four perturbation methods, two neighborhood structures and an extended version of the objective function were proposed. The latter allows the comparison of two solutions that present the same original objective function value.

The proposed methods are embedded in a Reduced VNS, representing diversification; a Variable Neighborhood Descent, representing intensification; and a General VNS, which balances both diversification and intensification. The experimental results show that the proposed algorithms outperforms the best algorithms found in the literature, emerging GVNS as the best algorithm for the Vertex Separation Problem.

Of particular interest in this work has been testing the combination of diversification and intensification strategies in the context of VNS. Through extensive experimentation, we have been able to determine the benefits of this combination. We purposefully added these mechanisms in order to measure their effects and studied the combinations that resulted in effective solution procedures with improved outcomes. We believe that our findings can be translated to other combinatorial problems and it will help in the development of more elaborated VNS methods.

Two papers derived from the research on the VSP have been published: *Variable neighborhood search for the Vertex Separation Problem* [36], and *Combining intensification and diversification strategies in VNS. An application to the Vertex Separation Problem* [131], included in Chapters 7 and 9 of Part II, respectively.

Conclusions on the Profile Separation

The goal of this work was to develop an algorithm that becomes the state-of-the-art solution for the Profile Minimization Problem. This goal was accomplished by proposing a Scatter Search procedure that emerges as the best algorithm in the literature, as confirmed by the throughout experimentation.

The proposed mechanisms for the PMP can be also useful in similar problem settings. Specifically, the efficient move evaluation in the improvement method can be adapted to other problems with similar characteristics, mainly in graph layout problems.

Furthermore, the preliminary experimentation devoted to select the best Scatter Search variant shows that the sequential selection of the best components of the algorithms obtains similar results than a full-factorial design. This idea can be considered when the factorial design is computationally intractable.

The research on the PMP is presented in the paper *Scatter Search for the profile minimization problem* [128], included in Chapter 8 of Part II.

Conclusions on the Cutwidth Minimization

In this work six parallel strategies embedded in the Variable Neighborhood Search scheme for solving the Cutwidth Minimization Problem were proposed. The strategies can be classified in three different templates. The first one is oriented to parallelize the complete method, the second one parallelizes the shake and local search procedures and, finally, the third template simultaneously explores the set of proposed neighborhoods.

These strategies were used to parallelize a recent new VNS strategy, called Variable Formulation Search. The experimental computation performs a comparison among all the variants when applied to the Cutwidth Minimization Problem. A preliminary experimentation was designed in order to select the best variant for each template. Then, the selected three algorithms are compared with the sequential version of the method.

Experimental results show that two of the proposed parallel methods, Replicated Shaking VNS and Replicated Parallel VNS are able to outperform the previous best method in term of quality. The results are supported by statistical tests that confirm the superiority of the proposal, emerging Replicated Shaking VNS as the best algorithm for the CMP.

More details of the parallelization of VNS for the CMP are presented in the paper *Parallel VNS strategies for the Cutwidth Minimization Problem* [39], included in Chapter 10 of Part II.

Conclusions on the Bandpass

The bandpass problem is somewhat new to the OR literature in the sense that, although it was originally introduced at a conference more than 10 years ago, the first publication did not appear until 2009. This problem has derived in different variants in subsequent publications along with additional test data. One of our goals in this project was to gain understanding of the current state of knowledge associated with the bandpass problem and to identify avenues for advancing this area. Through our investigation, we identified two problem classes and a total of three problem variants within those classes. We then designed a common Scatter Search solution framework and developed the individual strategies for each problem class.

The meticulous scientific testing performed provided some interesting insights on the behavior and interaction of the search components. These experiments produced new benchmarks that will help future researchers test solution methods that could prove even more effective than those described here. We were also able to show that although the first variant and the Multi-Bandpass one belong to the same problem class, the effectiveness of the search strategies designed for the former diminish when applied to the latter. Nonetheless, competitive testing showed that the proposed method applied (without customization) to both variants produces results that are significantly better than those produced by the existing procedures. Finally, our experiments show that the adaptation of the algorithm to the second variant exhibits a robust behavior when compared to commercial software.

The research on the BP is presented in the paper *Scatter search for the bandpass problem* [129], included in Chapter 11 of Part II.

Future work

This dissertation has been focused on heuristically solving graph layout problems. Future lines of research includes the development of more exact methods that can be compared with the heuristic ones, leveraging the advances in technology. Another line of research comprises the hybridization of the heuristic methods with commercial solvers, in order to obtain better solutions, which is known as “matheuristics”, an area which is increasing in operations research in the recent years.

Finally, the similarity between some of the conclusions obtained in the developed works have lead us to study the development of a general purpose framework for solving graph layout problems. The main idea behind this algorithm is to provide high quality solutions without including any specific information of the problem.

The study of the considered problems has also derived in the research of problems not belonging to the graph layout class, but that are also interesting due to its practical applications in different areas of knowledge.

Part II

Publications

Chapter 7

Variable neighborhood search for the Vertex Separation Problem

- A. Duarte, L.F. Escudero, R. Martí, N. Mladenović, J.J. Pantrigo, and J.Sánchez-Oro. Variable neighborhood search for the Vertex Separation Problem. Computers & Operations Research, 39(12):3247 – 3255, 2012.
  - DOI: `https://doi.org/10.1016/j.cor.2012.04.017`
  - Impact Factor (JCR 2012): 1.909

Subject Category	**Ranking**	**Quartile**
Operations Research and Management Science	10 / 79	Q1
Engineering, industrial	6 / 44	Q1
Computer Science, interdisciplinary applications	33 / 102	Q2

Computers & Operations Research 39 (2012) 3247–3255

Contents lists available at SciVerse ScienceDirect

Computers & Operations Research

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

Variable neighborhood search for the Vertex Separation Problem

Abraham Duarte [a],*, Laureano F. Escudero [b], Rafael Martí [c], Nenad Mladenovic [d], Juan José Pantrigo [a], Jesús Sánchez-Oro [a]

[a] Dpto. de Ciencias de la Computación, Universidad Rey Juan Carlos, Móstoles, Madrid, Spain
[b] Dpto. de Estadística e Investigación Operativa, Universidad Rey Juan Carlos, Móstoles, Madrid, Spain
[c] Dpto. de Estadística e Investigación Operativa, Universidad de Valencia, Valencia, Spain
[d] Department of Mathematics, SISCM, London, United Kingdom

ARTICLE INFO

Available online 19 April 2012

Keywords:
Combinatorial optimization
Metaheuristics
Variable neigborhood search
Layout problems

ABSTRACT

The Vertex Separation Problem belongs to a family of optimization problems in which the objective is to find the best *separator* of vertices or edges in a generic graph. This optimization problem is strongly related to other well-known graph problems; such as the Path-Width, the Node Search Number or the Interval Thickness, among others. All of these optimization problems are NP-hard and have practical applications in VLSI (Very Large Scale Integration), computer language compiler design or graph drawing. Up to know, they have been generally tackled with exact approaches, presenting polynomial-time algorithms to obtain the optimal solution for specific types of graphs. However, in spite of their practical applications, these problems have been ignored from a heuristic perspective, as far as we know. In this paper we propose a pure 0-1 optimization model and a metaheuristic algorithm based on the variable neighborhood search methodology for the Vertex Separation Problem on general graphs. Computational results show that small instances can be optimally solved with this optimization model and the proposed metaheuristic is able to find high-quality solutions with a moderate computing time for large-scale instances.

1. Introduction

Let $G(V,E)$ be an undirected graph where V $(n=|V|)$ and E $(m=|E|)$ are the sets of vertices and edges, respectively. A linear layout φ of the vertices of G is a bijection or mapping $\varphi : V \rightarrow \{1,2,\ldots,n\}$ in which each vertex receives a unique and different integer between 1 and n. For vertex u, let $\varphi(u)$ denote its position or label in layout φ. Let $L(p,\varphi,G)$ be the set of vertices in V with a position in the layout φ lower than or equal to position p. Symmetrically, let $R(p,\varphi,G)$ be the set of vertices with a position in the layout φ larger than position p. In mathematical terms,

$$L(p,\varphi,G)=\{v \in V : \varphi(v) \le p\} \quad \text{and} \quad R(p,\varphi,G)=\{v \in V : \varphi(v) > p\}.$$

Since layouts are usually represented in a straight line, where the vertex in position 1 comes first, $L(p,\varphi,G)$ can be simply called the set of left vertices with respect to position p and, $R(p,\varphi,G)$ the set of right vertices w.r.t. p.

The *Cut*-value at position p of layout φ, $Cut(p,\varphi,G)$, is defined as the number of vertices in $L(p,\varphi,G)$ with one or more adjacent vertices in $R(p,\varphi,G)$, then,

$$Cut(p,\varphi,G)=|\{u \in L(p,\varphi,G) : \exists v \in R(p,\varphi,G) \cap N(u)\}|,$$

where $N(u)=\{v \in V : (u,v) \in E\}$. The Vertex Separation value (VS) of layout φ is the maximum of the *Cut*-value among all positions in layout φ : $\mathrm{VS}(\varphi,G)=\max_p Cut(p,\varphi,G)$.

The Vertex Separation Problem (VSP) consists of finding a layout, say φ^*, minimizing the VS in graph G. Despite of its practical applications, there is no previous heuristic or metaheuristic algorithm to find good solutions in short computing times. We propose a variable neighborhood search (VNS) [18] approach whose performance is assessed by a broad testbed of instances.

The remainder of this paper is organized as follows. Section 2 presents the Vertex Separation Problem, including its application domain. Section 3 formalizes the mathematical optimization model. Section 4 presents our algorithmic approach based on the VNS metaheuristic framework. Section 5 reports on an extensive computational experience to validate the proposed algorithm by (1) comparing its performance and computing time versus the optimization of the mathematical model for small instances and (2) analyzing its performance for large instances. Finally, Section 6 summarizes the main conclusions of our research.

2. Description, related problems and applications

The Vertex Separation Problem (VSP) in graph G consists of finding a layout, say φ^*, that minimizes $\mathrm{VS}(G,\varphi)$, where $\mathrm{VS}(G,\varphi)$ is

* Corresponding author.
E-mail address: abraham.duarte@urjc.es (A. Duarte).

0305-0548/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cor.2012.04.017

the maximum *Cut*-value in graph G for layout φ, i.e., $\mathrm{VS}(G,\varphi)=\max_p Cut(p,\varphi,G)$. For the sake of simplicity, we denote the optimum value $\mathrm{VS}(G,\varphi^*)$ as VS*.

Fig. 1(a) shows an illustrative example of an undirected graph G with seven vertices and nine edges. Fig. 1(b) depicts a solution (layout) φ of this graph and the *Cut*-value of each position p, $Cut(p,\varphi,G)$. For example, $Cut(1,\varphi,G)=1$ because $L(1,\varphi,G)=\{\mathtt{D}\}$ and $R(1,\varphi,G)=\{\mathtt{A,F,G,E,B,C}\}$ and there is one vertex in L having an adjacent vertex in R. Similarly, $Cut(3,\varphi,G)=2$ where $L(3,\varphi,G)=\{\mathtt{D,A,F}\}$ and $R(3,\varphi,G)=\{\mathtt{G,E,B,C}\}$. The objective function value, computed as the maximum of these cut values, is $\mathrm{VS}(G,\varphi)=3$ whose related position is $p=4$.

The decisional version of the VSP was proved to be NP-complete for general graphs [26]. It is also known that the problem remains NP-complete for planar graphs with maximum degree of three [31], as well as for chordal graphs [15], bipartite graphs [16], grid graphs and unit disk graphs [6].

We can find many different graph problems that, although stated in different terms, are equivalent to the VSP in the sense that a solution to one problem provides a solution to the other one. Some of them are the Path-Width problem [20], the Interval Thickness problem [22], the Node Search Number [23] and the Gate Matrix Layout [21]. The equivalence between these problems is a consequence of the results presented in [13,20,23]. For any graph G let $\mathrm{VS}(G)$, $\mathrm{PW}(G)$, $\mathrm{IT}(G)$, $\mathrm{SN}(G)$ and $\mathrm{GML}(G)$ be the objective function value of the optimal solution for the Vertex Separation, Path-Width, Interval Thickness, Node Search Number and Gate Matrix Layout problems, respectively. These values verify the following relations:

$$\mathrm{VS}(G)=\mathrm{PW}(G)=\mathrm{IT}(G)=\mathrm{SN}(G)-1=\mathrm{GML}(G)+1.$$

The VSP appears in the context of finding "good separators" for graphs [28] where a separator is a set of vertices or edges whose removal separates the graph into disconnected subgraphs. This optimization problem has applications in VLSI design for partitioning circuits into smaller subsystems, with a small number of components on the boundary between the subsystems [25]. The decisional version of the VSP consists of finding a Vertex Separation value larger than a given threshold. It has applications on computer language compiler design and exponential algorithms. In compiler design, the code to be compiled can be represented as a directed acyclic graph (DAG) where the vertices represent the input values to the code as well as the values computed by the operations within the code. An edge from node u to node v in this DAG represents the fact that value u is one of the inputs to operation v. A topological ordering of the vertices of this DAG represents a valid reordering of the code, and the number of registers needed to evaluate the code in a given ordering is precisely the Vertex Separation number of the ordering [4]. The decisional version of VSP has also applications in graph theory [14]. Specifically, if a graph has a Vertex Separation value, say w, then it is possible to find the maximum independent set of G in time $O(2^w n)$. Other practical applications include Graph Drawing and Natural Language Processing [9,29].

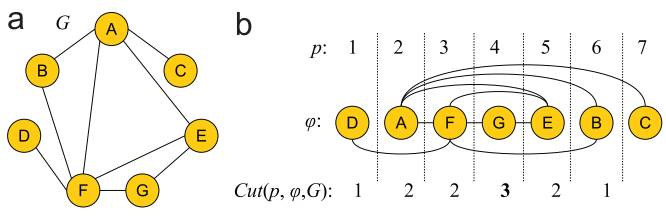


Fig. 1. (a) Graph illustrative example. (b) A layout φ for its VSP.

3. Pure 0-1 optimization model

Before presenting the model, let us define the 0-1 variables that are required:

- x_u^p, whose value is 1 if vertex u is placed in position p in a given feasible layout, i.e. $p=\varphi(u)$ and 0, otherwise (i.e., $p\neq\varphi(u)$), for $u,p=1,2,\ldots,n$.
- $y_{u,v}^{p,q}$, whose value is 1 if $p=\varphi(u)$ and $q=\varphi(v)$ in the given layout φ and 0, otherwise. The inconvenience of this type of variable is that the number of y variables is mn^2 and, then, the cardinality of the set of vertices in graph G must be small in order to obtain the optimal solution of the model by an exact 0-1 solver. On the other hand, $y_{u,v}^{p,q}$ avoids the quadratic expression $x_u^p x_v^q$.
- z_{pc}, whose value is 1 if the vertex in position p (i.e., $p=\varphi(u)$ and, then, $x_u^p=1$) is connected with the vertex in any position, say q (i.e., $q=\varphi(v)$ and, then, $x_v^q=1$) (what implies that $y_{u,v}^{p,q}=1$) that is larger than position c and 0, otherwise.

We propose the following integer programming formulation for the Vertex Separation Problem:

$$\mathrm{VS}^*=\min \mathrm{VS}, \tag{1}$$

subject to

$$\sum_{p\in\{1,\ldots,n\}} x_u^p=1 \quad \forall u=1,\ldots,n, \tag{2}$$

$$\sum_{u\in\{1,\ldots,n\}} x_u^p=1 \quad \forall p=1,\ldots,n, \tag{3}$$

$$y_{u,v}^{p,q}\le x_u^p \quad \forall u,v,p,q=1,\ldots,n, \tag{4}$$

$$y_{u,v}^{p,q}\le x_v^q \quad \forall u,v,p,q=1,\ldots,n, \tag{5}$$

$$x_u^p+x_v^q\le y_{u,v}^{p,q}+1 \quad \forall u,v,p,q=1,\ldots,n, \tag{6}$$

$$z_{pc}\le \sum_{q=c+1}^{n}\sum_{u=1}^{n}\sum_{v=1}^{n} y_{u,v}^{p,q}\le Mz_{pc} \quad \forall p,c=1,\ldots,n-1,\ p\le c, \tag{7}$$

$$\sum_{p=1}^{c} z_{pc}\le \mathrm{VS} \quad \forall c=1,\ldots,n-1, \tag{8}$$

$$x_u^p,y_{u,v}^{p,q},z_{pc}\in\{0,1\} \quad \forall u,v,p,q=1,\ldots,n. \tag{9}$$

The assignment constraints (2) and (3) ensure that each vertex is only assigned to one position, and each position is only assigned to one vertex, respectively. Subsystems (4)–(6) define the variable $y_{u,v}^{p,q}$ as the product of the variables x_u^p and x_v^q in the traditional way. So, notice that for a position p, $\sum_{q=c+1}^{n} y_{u,v}^{p,q}$ computes the number of edges from the vertex in position p, say $p=\varphi(u)$, to any vertex in a position q, say $q=\varphi(v)$ (since $x_v^q=1$ for $y_{u,v}^{p,q}=1$) that is bigger than c, for $c=q+1,\ldots,n$ for a given layout φ.

Constraints (7) compute the 0-1 value of the z_{pc} variable from the $y_{u,v}^{p,q}$ variables (since $x_u^p=1$ for $y_{u,v}^{p,q}=1$), where M is the standard big-M parameter that should be computationally small enough to allow any feasible layout φ, in our case $M=n-1$.

The left hand side (*lhs*) of constraints (8) gives the number of vertices in positions $\{p\}$ that, on the one hand, are lower than or equal to position c, in any feasible layout φ or equal to c and, on the other hand, they are connected with vertices whose positions are larger than position c. So, there is a position c in layout φ that has the greatest *lhs* in the constraints (8), which is VS*. The objective function of the model (1) minimizes that value, being VS* the optimal value.

We can observe that this model has $O(mn^2)$ 0-1 variables and $O(mn^3)$ constraints, what makes it impractical for relatively large instances.

4. Algorithmic approach: variable neighborhood search

In spite of the difficulty of the model presented above we can find efficient exact approaches to solve the VSP on special classes of graphs. A linear algorithm to compute the optimal Vertex Separation of a tree is proposed in [10] as well as an $O(n \log n)$ algorithm for finding the corresponding optimal layout. The algorithm was improved in [36] with a linear time procedure to find the optimal layout. In [33] an alternative method to compute the Vertex Separation of trees was proposed. Ref. [12] proposes an $O(n \log n)$ algorithm to compute the Vertex Separation of unicyclic graphs (i.e., trees with an extra edge). A polynomial-time algorithm to compute the Path-Width (what is identical to VSP) is proposed in [2]. However, the algorithm cannot be considered from a practical point of view, since the bound on its time complexity is $\Omega(n)$, see [12]. Ref. [5] proposes a polynomial time algorithm for optimally solving the VSP for n-dimensional grids. Co-graphs and permutational graphs can also be optimally solved as it was proposed in [1,2], respectively.

Approximation algorithms have been also proposed for the VSP. Specifically, [3] proposes a polynomial time $O(\log^2 n)$-approximation algorithm for general graphs and a $O(\log n)$-approximation algorithm for planar graphs. Similar results for binomial random graphs are presented in [7].

VNS is a metaheuristic for solving optimization problems based on a systematic change of neighborhood structures, without guaranteeing the solution's optimality. In recent years, a large variety of VNS strategies have been proposed. We can highlight the Variable Neighborhood Descent (VND), Reduced VNS (RVNS), Basic VNS (BVNS), Skewed VNS (SVNS), General VNS (GVNS), Variable Neighborhood Decomposition Search (VNDS) and Reactive VNS, among others. We refer the reader to [18] for a complete review of this methodology and its different variants. In this paper, we focus on the Basic VNS variant, see [30] for the details, which combines deterministic and stochastic changes of neighborhood as shown in the BVNS pseudo-code depicted in Algorithm 1.

Algorithm 1. BasicVNS ($k_{\max}$, $t_{\max}$).

```
1.  $\varphi \leftarrow Construct()$
2.  repeat
3.      $k \leftarrow 1$
4.      repeat
5.          $\varphi' \leftarrow Shake(\varphi, k)$
6.          $\varphi'' \leftarrow LocalSearch(\varphi')$
7.          $NeighborhoodChange(\varphi, \varphi'', k)$
8.      until $(k = k_{max})$
9.      $t \leftarrow Time()$
10. until $(t = t_{max})$
```

The method starts by constructing a feasible solution (step 1), using one of the constructive procedures described in Section 4.1. The BVNS implementation is executed for a predefined computing time, $t_{\max}$ (steps 2–10). The search process starts with the first neighborhood of the constructed solution, $N_1(\varphi)$ (step 3). Then, BVNS performs stochastic changes of neighborhood structures until reaching the largest predefined neighborhood $k_{\max}$ (steps 4–8). VNS has three main strategies within the main loop, namely, shaking (step 5), improvement (step 6) and neighborhood change (step 7). In the shaking stage a solution, say φ', is generated within $N_k(\varphi)$, where $k = 1, \dots, k_{\max}$ identifies a neighborhood structure within a set of predefined neighborhoods (see Section 4.3 for additional details). Then, the improvement method (Section 4.5) is applied to φ' in order to find a local optimum, say φ'', in the corresponding neighborhood. Finally, the neighborhood change stage (Section 4.4), analyzes if φ'' is better than φ (see in Section 4.5 the extended definition of the concept of improvement move). If so, φ is replaced with φ'' and k is set to one. Otherwise, k is incremented by one unit. And, in any case, we repeat the procedure.

We now describe with more detail the main strategies of our BVNS approach for solving the Vertex Separation Problem.

4.1. Constructive procedures

We have designed two greedy constructive algorithms, say C1 and C2, for the VSP. The constructive procedure, C1, starts by creating a set of unlabeled vertices U (initially $U = V$), and a set of labeled vertices $L = V \backslash U$. The vertex with the minimum degree is selected as the first node u (ties are broken at random). The vertex u is labeled with 1. Then, sets U and L are properly updated (i.e., $U = U \backslash \{u\}$ and $L = L \cup \{u\}$). Once the first label is assigned to vertex u, C1 evaluates the greedy function as follows:

$$g(v) = |N_L(v)| - |N_U(v)| \quad \forall v \in U,$$

where $N_L(v)$ is the set of vertices adjacent to v that has been already labeled, and $N_U(v)$ is the set of vertices adjacent to v not labeled yet. The constructive procedure selects the vertex with the maximum g-value and assigns the next label to it. The procedure ends when all the vertices of the graph have a label (i.e., $U = \emptyset$).

The second constructive procedure, C2, is based on the creation of the so-called level structures [27] which means that the set of vertices V is partitioned into different sets $L_1, L_2, \dots, L_\lambda$ called levels. The first level, L_1, contains only one vertex. The rest of levels L_l with $l = 2, 3, \dots, \lambda$ (where λ indicates the number of levels) contains all the vertices adjacent to some vertex in L_{l-1} that are not present in any L_j with $1 \leq j < l$. The number of levels λ exclusively depends on the graph and the vertex placed in L_1.

The level structure created in this way guarantees that the vertices in alternative levels are not adjacent. In order to construct this level structure we use a breadth first search approach, performing the search once for each vertex of the graph. Therefore, if the graph has n vertices, we construct n different level structures. In the context of the VSP, it is desirable to obtain a level structure with λ as large as possible (i.e., a structure with the largest number of levels). Once we have identified the most suitable level structure, C2 explores it performing a breadth-first search and assigning levels in an incremental way, thus obtaining a solution for the VSP.

4.2. Updating the objective function

Let φ_p be a partial solution where p vertices have been placed in positions from 1 to p. For the sake of simplicity, we denote v_p as the vertex placed in position p, i.e., $p = \varphi(v_p)$. We maintain this notation for the rest of the paper. Given a graph G and a partial solution φ_p, the *Cut*-value of a position p, $Cut(p, \varphi, G)$ can be computed from the *Cut*-value of the previous position $p-1$ (except for $p=1$ where, obviously, the *Cut*-value is always 1) as follows,

$$Cut(p, \varphi_p, G) = Cut(p-1, \varphi_p, G) + \delta^+(p) - \delta^-(p),$$

where $\delta^+(p) \in \{0, 1\}$ has the value 1 if vertex v_p has, at least, one adjacent vertex in $N_U(v_p)$, and 0, otherwise; and $\delta^-(p) = |N_L(v_p)|$.

This strategy can be extended to the incremental computation in complete solutions (i.e., solutions $\varphi = \varphi_r$ with $r = n$). Specifically, for any position p the set $N_U(v_p)$ is replaced with the set of vertices placed in a position q with $1 \le q \le p$. Symmetrically, the set $N_L(v_p)$ is replaced with the set of vertices placed in a position q with $p < q \le n$. Fig. 2 shows an example of the incremental objective function computation. Let us consider the third position in φ (i.e., vertex C). The *Cut*-value of this position can be computed using the *Cut*-value of the previous position, $Cut(2,\varphi,G)=2$. The value of $\delta^+(3)$ is 1 because vertex C has one adjacent at least (specifically, three adjacents) placed in a position greater than $\varphi(\mathrm{C})=3$ (vertices B, F and E, with $\varphi(\mathrm{B})=4$, $\varphi(\mathrm{F})=5$ and $\varphi(\mathrm{E})=6$, respectively). On the other hand, the value of $\delta^-(3)$ is 2 because C is the adjacent vertex with the largest label of vertices A and D, placed in positions $\varphi(\mathrm{A})=1$ and $\varphi(\mathrm{D})=2$, respectively. The same reasoning can be applied to compute the rest of the *Cut*-values.

4.3. Shake

In this section we propose a shake function for the VSP, called *shake*(φ,k). This procedure initially selects the vertices to be moved, based on their *Cut*-value. Specifically, *shake*(φ,k) selects the k vertices in V with the largest *Cut*-value in φ. Then, each selected vertex is exchanged with another vertex determined at random, obtaining a new ordering φ'. The rationale behind this procedure is that the k selected vertices have *Cut*-values close or equal to the maximum *Cut*-value for φ in G and, therefore, the aim is to reduce their *Cut*-value improving the objective function of φ'. Additionally, it is expected that the shaking step will contribute to the diversification of the search process. In other words, shaking will produce a solution “far away” from the current one, allowing the search to explore different regions of the solution space.

4.4. Neighborhood structures

A solution to the VSP can be represented as an ordering, where each vertex is located in the position given by its label. For example, the labeling of the graph depicted in Fig. 1 can be expressed by the ordering $\varphi=(\mathrm{D,A,F,G,E,B,C})$, where the first vertex, D, in the ordering receives label 1, the second vertex, A, receives label 2, and so on. We define neighborhood structures based on the exchange of vertices in a given ordering. Given a solution $\varphi=(v_1,\ldots,v_p,\ldots,v_q,\ldots,v_n)$, we define *Move*$(\varphi,p,q)$ as exchanging in φ the vertex in position p (i.e., v_p) with the vertex in position q (i.e., v_q) and, thus, producing a new solution $\varphi'=(v_1,\ldots,v_q,\ldots,v_p,\ldots,v_n)$.

In a direct implementation, the complexity of evaluating φ' is $O(n^2)$ in the worst case since it requires to visit, for each position p in φ, all the vertices with labels $p \le q$. However, the *Cut*-value of some vertices does not change when we perform a move. Therefore, it is not required to compute them again. Specifically, if we perform *Move*(φ,p,q), all vertices with label r for $1 \le r < p$ or $q \le r \le n$ do not change their *Cut*-values. As a consequence, we only need to update vertices whose label s is such that $p \le s < q$. For example, Fig. 3(a) shows a layout and Fig. 3(b) represents the layout resulting from performing *Move*$(\varphi,2,5)$. This move only affects the *Cut*-value of positions $2 \le s < 5$ (represented as a highlighted band in those figures).

Additionally, to compute the *Cut*-value of the vertices involved (i.e., the set of vertices {D,C,B} in Fig. 3(b)) we can use the incremental objective function computation described above, but restricted to vertices in positions from p to $q-1$. In the example shown in Fig. 3(b) we can observe that the update of the *Cut*-values is only needed for the vertices in positions $\varphi(\mathrm{D})=2$, $\varphi(\mathrm{C})=3$ and $\varphi(\mathrm{B})=4$.

Given a layout φ, its neighborhood $N_1(\varphi)$ is defined as all possible exchanges between each pair of vertices. In other words, a solution φ' belongs to $N_1(\varphi)$ if and only if φ and φ' only differ in two labels. In general, we may say that a solution φ' belongs to the kth neighborhood of solution φ (i.e., $\varphi' \in N_k(\varphi)$) if φ and φ' differ in $k+1$ labels.

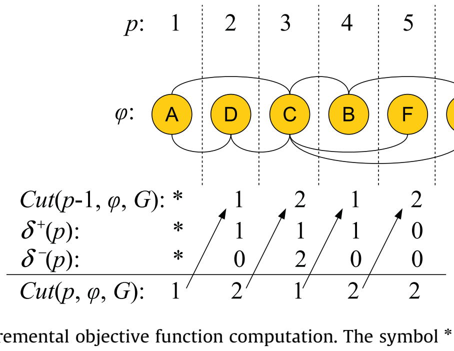


Fig. 2. Incremental objective function computation. The symbol * means that the corresponding value is not defined for the first position.

4.5. Local search

VSP is a min–max problem [8,32,35] where the value of the objective function is usually reached in several positions of the permutation φ. This kind of problems present a “flat landscape”, which turns out in a challenge for classical local search procedures as well as for 0-1 solvers to obtain the optimal layout. Typically, local search strategies do not perform well from a computational point of view, since most of the moves have associated a null value. Then, given a graph G, changing the label p of a particular vertex v_p in φ (i.e., obtaining a new solution φ') such that its *Cut*-value is decreased, does not necessarily imply that $\mathrm{VS}(G,\varphi') < \mathrm{VS}(G,\varphi)$. However, it can be considered as an interesting move if the number of vertices with a relative large *Cut*-value is reduced, regardless whether the objective function improves or not. Considering this extended definition of “improving” we overcome the lack of information provided by the objective function. Specifically, we implement a candidate list strategy classifying the vertices of the graph according to its *Cut*-value. For example, given the graph depicted in Fig. 1 and the labeling φ, we obtain three different sets: $S_3(\varphi)=\{\mathrm{G}\}$, containing vertices with *Cut*-value equal

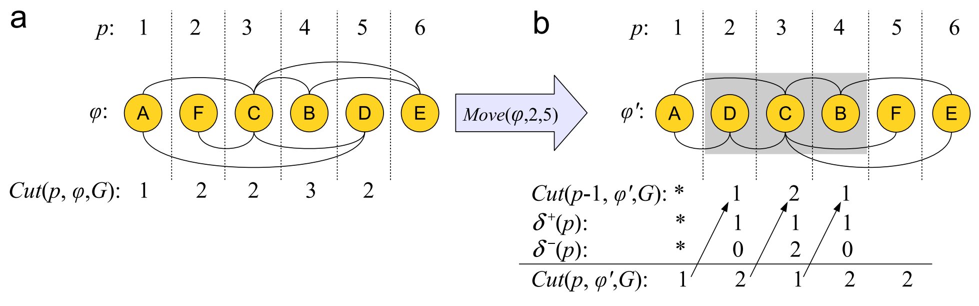


Fig. 3. Illustrative example of an interchange move: (a) layout before the move and (b) layout after the move.

to 3; $S_2(\varphi)=\{\texttt{A,F,E}\}$, with *Cut*-value equal to 2; and finally $S_1(\varphi)=\{\texttt{D,B}\}$ with *Cut*-value equal to 1.

Given the definition of $S_i(\varphi)$ that has been introduced in the previous paragraph, we consider that a move improves the current solution if any node involved in the move is removed from $S_i(\varphi)$ and included in $S_j(\varphi)$ with $j<i$ without increasing the cardinality of any set $S_l(\varphi)$ for $l>i$. According to this definition of improving, a move which removes a vertex from $S_2(\varphi)$ (for example, vertex A) including it in $S_1(\varphi)$ is considered an improving move if the cardinality of $S_3(\varphi)$ remains unaltered. We have empirically found that this criterion allows the local search procedure to explore a larger number of solutions than a typical implementation that only performs moves when the objective function is improved. Fig. 4(a) shows an improving move for the labeling of Fig. 3(a). The move consists of exchanging the labels of vertices C and A and, thus, obtaining a new solution φ'. In the new labeling, vertex F is removed from set $S_2(\varphi)$ and included in set $S_1(\varphi')$. It means that the *Cut*-value of vertex F is decreased by 1 unit. This move does not reduce the VS value of the graph, but it reduces the number of vertices with large *Cut*-value. Observe that this move does not improve the incumbent solution, but it can allow further moves that, at the end, improves it. On the other hand, if we now exchange the labels of vertices B and C obtaining a new solution φ'' (see Fig. 4(b)) then vertex C is removed from set $S_2(\varphi)$ and included in set $S_3(\varphi'')$ (i.e., its *Cut*-value is increased by one unit). Therefore, although this move does not affect the VS value, it is not accepted. Algorithm 2 shows a pseudocode of the procedure, where the acceptance criterion has been implemented. This procedure starts by comparing the objective function of the previous ordering (φ) and the related function of the ordering after the corresponding movement (φ'). If the associated move reduces the value of the objective function, then *IsImprovementMove* returns *true* (step 2). On the other hand, if the move deteriorates the objective function value then this procedure returns *false* (step 4). In case that the corresponding move does not affect the value of the objective function (steps 6–13) *IsImprovementMove* considers the extended definition of improvement move defined above.

Algorithm 2. *IsImprovementMove*(φ,φ').

```
1.  if $VS(G,\varphi') < VS(G,\varphi)$ then
2.    return true
3.  else if $VS(G,\varphi') > VS(G,\varphi)$ then
4.    return false
5.  else [$*VS(G,\varphi)=VS(G,\varphi')*$]
6.    for $i \leftarrow VS(G,\varphi)$ downto 1 do
7.      if $|S_i(\varphi)| > |S_i(\varphi')|$ then
8.        return true
9.      else if $|S_i(\varphi)| < |S_i(\varphi')|$ then
10.       return false
11.     endif
12.   end for
13.   return false
14. end if
```

The proposed local search strategy is presented in Algorithm 3. This method receives a layout φ and perform moves while an improve is produced (i.e., while-loop in step 2). Specifically, the procedure starts by arranging the vertices in descending order of the *Cut*-value, obtaining the set *posSet* which contains the position of such vertices (step 4). Then, the algorithm traverses this set (for-loop in step 5) trying to interchange each vertex in *posSet* with the remaining vertices in φ (for-loop in step 6). In each iteration, the method proves to interchange the position of the two considered vertices (step 8) accepting the move if the new layout φ' outperforms φ (steps 9–12). This procedure is repeated until no improvement in the move is found.

Algorithm 3. *LocalSearch*(φ).

```
1.  improvement ← true
2.  while improvement do
3.    improvement ← false
4.    posSet ← OrderByCutValue($\varphi$)
5.    for all $p \in$ posSet do
6.      for $q \leftarrow 1$ to $|\varphi|$ do
7.        if $p \neq q$ then
8.          $\varphi' \leftarrow$ Interchange($\varphi$,p,q)
9.          if IsImprovementMove($\varphi,\varphi'$) then
10.           $\varphi \leftarrow \varphi'$
11.           improvement ← true
12.         end if
13.       end if
14.     end for
15.   end for
16. end while
17. return $\varphi$
```

5. Computational experience

This section reports the computational experiments that we have performed for testing the efficiency of our VNS procedure, so-called BVNS, for solving the VSP. The algorithm has been implemented in Java SE 6 and all the experiments were conducted on an Intel Core i7 2600 CPU (3.4 GHz) and 2 Gb RAM. We have experimented with three sets of instances, totalizing 173 instances (all of the instances are available at http://www.optsicom.es/vsp/).

5.1. Testbed description

- HB: We derived 73 instances from the Harwell-Boeing Sparse Matrix Collection. This collection consists of a set of standard test matrices $M=M_{uv}$ arising from problems in linear systems, least squares, and eigenvalue calculations from a wide variety of scientific and engineering disciplines. The graphs are derived from these matrices by considering an edge (u,v) for every element $M_{uv}=0$. From the original set we have selected the 73 graphs with $n\leq 1000$. The number of vertices and edges range from 24 to 960 and from 34 to 3721, respectively.
- Grids: This set consists of 50 matrices constructed as the Cartesian product of two paths [34]. They are also called two

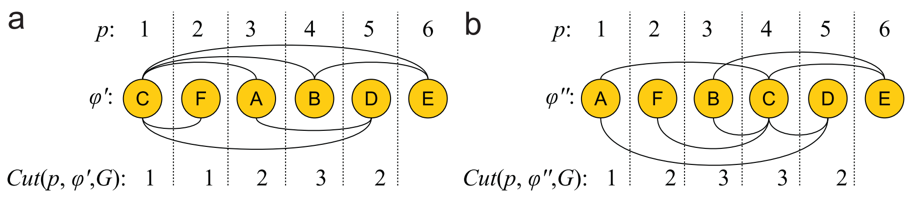


Fig. 4. (a) Improving move and (b) non-improving move over the layout depicted in Fig. 3(a).

dimensional meshes and the optimal solution of the VSP for squared grids is known by construction, see [7]. Specifically, the Vertex Separation value of a square grid of size $\lambda \times \lambda$ is λ. For this set, the vertices are arranged on a square grid with a dimension $\lambda \times \lambda$ for $5 \le \lambda \le 54$. The number of vertices and edges ranges from $5 \times 5 = 25$ to $54 \times 54 = 2916$ and from 40 to 5724, respectively.

- `Trees`: Let $T(\lambda)$ be set of trees with minimum number of nodes and Vertex Separation equal to λ. As it is stated in [11], there is just one tree in $T(1)$, namely the tree with a single edge, and another one in $T(2)$, the tree constructed with a new node acting as root of three subtrees that belong to $T(1)$. In general, to construct a tree with Vertex Separation $\lambda+1$ it is necessary to select any three members from $T(\lambda)$ and link any one node from each of these to a new node acting as the root of the new tree. The number of nodes, $n(\lambda)$, of a tree in $T(\lambda)$ can be obtained using the recurrence relation $n(\lambda) = 3n(\lambda-1)+1$ where and $n(1)=2$ (see [11] for additional details). We consider 50 different trees: 15 trees in $T(3)$, 15 trees in $T(4)$ and 20 trees in $T(5)$. The number of vertices and edges ranges from 22 to 202 and from 21 to 201, respectively.

Eight experiments are performed for assessing the validation of the proposed procedure BVNS. We have selected 32 `HB` representative instances, with different sizes and densities, to perform the experiments 1–5 oriented to establish the best configuration of the BVNS procedure. Let us name this set of instances the `HB` subset. Specifically, we consider the instances `ASH85`, `BCSPWR01`, `BCSPWR02`, `BCSPWR03`, `BCSSTK01`, `BCSSTK02`, `BCSSTK03`, `BCSSTK04`, `BCSSTK05`, `BCSSTK22`, `CAN_144`, `CAN_161`, `CAN_187`, `CAN_229`, `CAN_24`, `CAN_61`, `CAN_62`, `CAN_73`, `CAN_96`, `DWT_162`, `DWT_193`, `DWT_198`, `DWT_209`, `DWT_221`, `DWT_234`, `DWT_245`, `DWT_59`, `DWT_66`, `DWT_72`, `DWT_87`, `NOS1` and `NOS4`. Experiment 6 evaluates the performance of the optimization model (1)–(9). Experiment 7 evaluates the performance of BVNS on instances whose optimum is known and, finally, the performance of the best configuration of the procedure BVNS is analyzed using the full testbed of 173 instances (experiment 8).

5.2. Experiment 1: constructive procedures performance

In our first experiment we compare the performance of the two proposed constructive procedures for the VSP, namely C1 and C2, described in Section 4.1. We have conducted the experiment over the `HB` subset of 32 instances presented above. We generate one solution with each constructive procedure. The statistics of Table 1 are as follows: # best, number of best solutions found in the experiment; avg., average quality over all instances; dev (%), average percent deviation with respect to the best solution found in the experiment; time, average computing time in seconds required by the procedure.

Table 1 shows that C1 obtains better results than C2 in all headings and using similar computing time. Specifically, C1 reaches a smaller deviation (11.24%) than C2 (15.66%) and a larger number of best solutions (19 versus 18) in the set of instances experimented with.

5.3. Experiment 2: local search performance

We compare the two constructive procedures for the VSP (C1 and C2) coupled with the improvement procedure (LS) presented in Section 4.5. Table 2 reports the results using the same headings as above. We can observe that the method C2 coupled with LS gets more best solutions than C1+LS (22 versus 19), smaller deviation (9.48% versus 9.86%) and requires 7% smaller computing time (26.25 s versus 24.42 s) than the method C1 + LS. Despite C1 alone has better performance than C2 (see Table 1), when coupling the improvement strategy to the constructive procedures the behavior completely changes and C2 + LS clearly outperforms C1 + LS. This behavior may be partially explained by the fact that C2 is able to construct layouts with broader diversity than C1.

Table 1
Constructive procedures.

	C1	C2
# best	19	18
Avg.	17.50	17.65
Dev (%)	11.24	15.66
Time	0.010	0.018

Table 2
Constructive procedures coupled with the improvement procedure.

	C1 + LS	C2 + LS
# best	19	22
Avg.	14.88	14.88
Dev (%)	9.86	9.48
Time	26.25	24.42

5.4. Experiment 3: objective function computation

We study the computing time of C2 + LS when (1) using a direct objective function computation (DOFC) and (2) using the incremental objective function computation (IOFC) presented in Section 4.2. Fig. 5 depicts a bar diagram where the X -axis represents the ten largest instances of the `HB` subset and the Y -axis gives the computing time required to obtain a local optimum for both methods, respectively, (DOFC and IOFC) in the corresponding instance. The figure clearly shows that the saving in computing time is significant for IOFC. Specifically, for these ten instances DOFC needs 69 s on average to obtain a local optimum, while IOFC requires 3.85 s on average to obtain the same optimum value, i.e., almost 18 times faster.

5.5. Experiment 4: number of neighborhoods

This experiment consists of evaluating the impact of the $k_{\max}$ parameter on the performance of BVNS. It is expected that the larger the value of $k_{\max}$, the larger the computing time and, hopefully, the lower the objective function value. Table 3 shows the number of best solutions, the average objective function, the average deviation and the computing time for the four different values of $k_{\max} = \{0.08n, 0.1n, 0.3n, 0.5n\}$, where n indicates the number of vertices. The table shows that $k_{\max} = 0.5n$ obtains the best results in terms of quality but using the highest computing time. On the other hand, $k_{\max} = 0.08n$ is the fastest BVNS variant but it has a poor quality results. Then, we select $k_{\max} = 0.3n$ as a trade-off between quality and computing time.

5.6. Experiment 5: one-start versus multi-start strategy

In general, the shake strategy allows VNS to escape from local optima. Additionally, this procedure also diversifies the incumbent solution by incorporating/removing elements in the shaken one. In other words, the shake algorithm produces a solution which moves away from the current incumbent solution. However, it could be possible that the shake strategy is not powerful enough to be used alone (i.e., deep local optimum). Consequently,

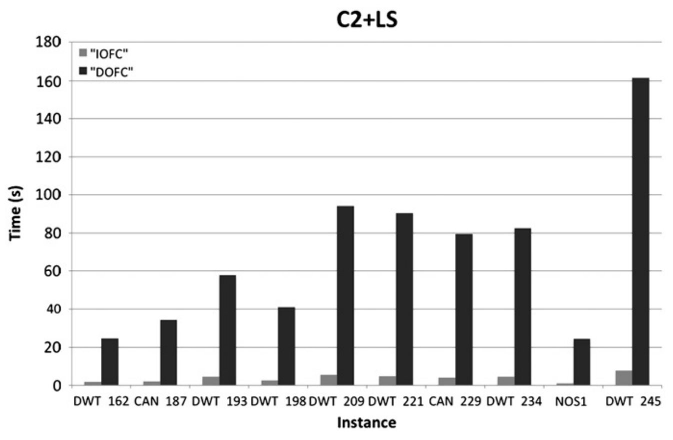


Fig. 5. Computing time comparison for direct (DOFC) and incremental (IOFC) objective function computation.

Table 3
Comparison of different $k_{\max}$ values.

	$k_{\max}$			
	$0.08n$	$0.1n$	$0.3n$	$0.5n$
# best	27	28	31	32
Avg.	13.34	13.31	12.91	12.87
Dev (%)	3.71	2.92	0.78	0.00
Time	69.70	79.54	214.32	341.74

in order to analyze the potential improvement of diversification, we consider a multi-start strategy which construct solutions in different positions of the search space and, then, execute the VNS. In this experiment we compare the performance of the best identified variant of BVNS (which includes C2, exchange-based LS and the parameter $k_{\max}$ set to $0.3n$) with a multi-start BVNS. In order to have a fair comparison, both methods use the same constructive and search strategies. The multi-start BVNS is executed starting from 10 different solutions. So, it is expected that its computing time is about 10 times slower, and the $k_{\max}$ parameter is set to $0.03n$, so it is 10 times lower than the $k_{\max}$ value in BVNS. Table 4 shows that the best variant of BVNS while using the one-start approach has better performance than the multi-start strategy in the same environment. Specifically, the former has more best solutions (30 versus 25) and it reaches smaller average deviation (0.40%) than the latter (4.46%) in the set of 32 instances of the HB subset considered in the experimentation. Furthermore, the computing time for the one-start BVNS (214.32 s) is a 93% less than the time required by the multi-start strategy (3170.09 s) for BVNS. It is important to remark that in both variants, the BVNS algorithm is exactly the same.

5.7. Experiment 6: pure 0-1 optimization model

This experiment consists of evaluating the optimization of the mathematical model introduced in Section 3, using the state-of-the-art optimization engine CPLEX v12.1 [19].

The main results of the experiment are shown in Table 5, whose headings indicate the name of the instance (*Instance*), number of vertices (n), number of constraints (n_c), number of 0-1 variables (n_{01}), number of constraint matrix non-zero elements (n_{el}), number of CPLEX branch-and-cut required for obtaining the incumbent Vertex Separation (nn), optimal Vertex Separation value for the corresponding instance G ($VS^* = VS(G)$), LP lower bound of the optimal Vertex Separation value provided by CPLEX ($\underline{VS}_{CPLEX}$),

Table 4
One-start versus multi-start BVNS.

	BVNS	Multi-start BVNS
# best	30	25
Avg.	12.91	13.41
Dev (%)	0.40	4.46
Time	214.32	3170.09

optimality gap in percentage ($GAP = (VS^* - \underline{VS}_{CPLEX})/\underline{VS}_{CPLEX}$), Vertex Separation value related to the incumbent Vertex Separation provided by CPLEX, computing time (s) required by CPLEX ($Time_{CPLEX}$), Vertex Separation value provided by our metaheuristic procedure BVNS (VS_{BVNS}), and computing time (s) required by the procedure ($Time_{BVNS}$).

Table 5 reports the main results of the following small instances: two instances from the grid set and the 21 instances from the set that have 22 nodes. In total 23 out of 173 instances in the sets that we have experimented with. We selected those instances where CPLEX could be used for its optimization. It is important to remark that CPLEX could not execute the model for the rest of instances since there was not enough memory (in our computer) to allocate the model, due to its large dimensions. Notice that the optimal Vertex Separation of the subset of instances in Table 5 is known by construction.

We can observe in Table 5 that the LP lower bound $\underline{VS}_{CPLEX}$ is so small that is useless, in fact is 1, i.e., the minimum possible value for connected graphs. Additionally, we can observe that CPLEX only obtains the optimal Vertex Separation for the smallest instance, grid_3. On the other hand, the difference between the CPLEX incumbent solution and the optimum (VS^*) is one or two units. Attending to the optimality gap, we can observe the weakness of the 0-1 model. Finally, it is worth to mention that, although CPLEX obtains the optimal value $VS^* = 3$ for instance Tree_22_3_rot4, it can not guarantee its optimality in the computing time limit, 5400 s (taking into account the value of the lower bound $\underline{VS}_{CPLEX} = 1 < VS^* = 3$).

Finally, we can observe in Table 5 that the procedure BVNS obtains the optimal Vertex Separation value in all instances that we have used in the experiment, being impressive its computing time (much less than 0.15 s for each instance). On the other hand, since our BVNS is heuristic in nature, it does not guarantee the optimality of these solutions by itself.

5.8. Experiment 7: BVNS on instances with known optimum value

We evaluate in this experiment the performance of our best identified variant of BVNS on a larger set of instances than the set used in the previous experiment. In this other set the optimal Vertex Separation is known by construction. Specifically, we use the 50 instances of the set Grids and 50 instances of the set Trees and test whether the BVNS is able to match the optimum or not. Table 6 has the same headings as the other tables plus the computing time, say, Time_to_opt, required by BVNS to reach the optimal solution on average.

Observe that BVNS is able to find the optimum solution in all the instances of the set Grids, subset Trees_22 and subset Trees_67. Therefore, BVNS is able to find the optimum in the instances where CPLEX could not due to running out of memory while solving the model presented in Section 3. Indeed, BVNS finds the optimum in subsets Trees_22 and Trees_67 in short computing time. It is worth to remark that although the BVNS method requires a relative large computing time (1422.76 s) for the whole procedure, it obtains the optimal solution in only 0.36 s on average.

Table 5
CPLEX results. Model dimensions, optimal solution, CPLEX incumbent solution, BVNS solution and computing times (s).

Instance	*n*	n_c	n_{01}	n_{el}	*nn*	*VS**	$\underline{VS}_{CPLEX}$	*GAP* (%)	VS_{CPLEX}	$Time_{CPLEX}$	VS_{BVNS}	$Time_{BVNS}$
grid_3	9	5938	2062	19,690	12,636	3	3	0.00	3	2528	3	0.054
grid_4	16	37,166	12,665	152,318	902	4	1	300.00	5	5400	4	0.126
Tree_22_3_rot0_3	22	61,532	21,044	2,922,994	533	3	1	200.00	4	5400	3	0.029
Tree_22_3_rot1_3	22	61,532	21,044	2,922,994	543	3	1	200.00	5	5400	3	0.022
Tree_22_3_rot2_3	22	61,532	21,044	2,922,994	604	3	1	200.00	4	5400	3	0.024
Tree_22_3_rot3_3	22	61,532	21,044	2,922,994	557	3	1	200.00	4	5400	3	0.037
Tree_22_3_rot4_3	22	61,532	21,044	2,922,994	553	3	1	200.00	3	5400	3	0.019
Tree_22_3_rot5_3	22	61,532	21,044	2,922,994	572	3	1	200.00	4	5400	3	0.018
Tree_22_3_rot6_3	22	61,532	21,044	2,922,994	593	3	1	200.00	4	5400	3	0.015
Tree_22_3_rot0_1	22	61,532	21,044	2,922,994	595	3	1	200.00	4	5400	3	0.04
Tree_22_3_rot1_1	22	61,532	21,044	2,922,994	597	3	1	200.00	4	5400	3	0.02
Tree_22_3_rot2_1	22	61,532	21,044	2,922,994	572	3	1	200.00	4	5400	3	0.018
Tree_22_3_rot3_1	22	61,532	21,044	2,922,994	577	3	1	200.00	4	5400	3	0.025
Tree_22_3_rot4_1	22	61,532	21,044	2,922,994	590	3	1	200.00	4	5400	3	0.016
Tree_22_3_rot5_1	22	61,532	21,044	2,922,994	566	3	1	200.00	4	5400	3	0.019
Tree_22_3_rot6_1	22	61,532	21,044	2,922,994	646	3	1	200.00	5	5400	3	0.022
Tree_22_3_rot0_2	22	61,532	21,044	2,922,994	625	3	1	200.00	5	5400	3	0.029
Tree_22_3_rot1_2	22	61,532	21,044	2,922,994	590	3	1	200.00	5	5400	3	0.022
Tree_22_3_rot2_2	22	61,532	21,044	2,922,994	547	3	1	200.00	4	5400	3	0.024
Tree_22_3_rot3_2	22	61,532	21,044	2,922,994	588	3	1	200.00	4	5400	3	0.037
Tree_22_3_rot4_2	22	61,532	21,044	2,922,994	591	3	1	200.00	4	5400	3	0.019
Tree_22_3_rot5_2	22	61,532	21,044	2,922,994	543	3	1	200.00	4	5400	3	0.018
Tree_22_3_rot6_2	22	61,532	21,044	2,922,994	636	3	1	200.00	4	5400	3	0.015

Note: CPLEX computing time limit: 5400 s.

Table 6
Grids and trees.

	Grids	Trees_22	Trees_67	Trees_202
# opt	50	15	15	1
Avg.	29.5	3.00	4.00	6.35
Dev (%)	0.00	0.00	0.00	27.00
Time_to_opt	0.36	0.002	0.58	*
Time	1422.76	0.02	3.37	314.28

Table 7
Constructive versus constructive plus local search versus BVNS.

	C2	C2+LS	BVNS
# best	16	38	73
Avg.	30.70	26.68	24.25
Dev (%)	35.97	11.73	0.00
Time	1417.05	1204.38	749.77

In sight of these results we can conclude that grids instances are easily solved by our procedure. Regarding the instances of the set Trees the algorithm is able to find the optimal solution for the small and medium instances (i.e., Trees_22 and Trees_67) in very small computing time, 0.002 s for finding the solution and 0.02 s for proving it for trees with 22 nodes, and 0.58 s and 3.37, respectively, for trees with 67 nodes. Specifically, BVNS only finds the optimal solution in one out of 20 large instances included in the set Trees_202 and the average deviation is 27.0%. But, although the computing time, 314.28 s, is large as compared to the other sets of instances, it is not too much effort considering that the instances have 202 vertices. Moreover, although it is not reported in Table 6, we can observe analyzing each instance that, besides guaranteeing optimally in one instance, BVNS obtains in 11 instances a quasi-optimal solution (the solution value is 6, i.e., one unit from the optimal one), and for the remaining 8 instances it obtains a solution value of 7 (i.e., two units from the optimal one). Notice that, since BVNS does not match the optimal value in 19 instances, the corresponding Time_to_opt value is not reported (represented by an asterisk in the table).

5.9. *Experiment 8: BVNS procedure versus C2 and C2 + LS*

The VSP has been computationally observed to be optimally solved for particular graphs (trees, grids, co-graphs, etc.). Additionally, as it was presented in Section 2, VSP has relevant practical applications. However, as far as we know, there are no previous heuristic procedures to obtain high-quality solutions on general graphs. In order to provide a final comparison, our experiment consists of evaluating the performance of BVNS versus the constructive approach C2, and the constructive C2 plus the local search procedure, C2 + LS, executed as stand-alone procedures. To provide a fair comparison, the three procedures are executed during a similar computing time. Specifically, C2 is run for 23,000 independent iterations while C2 + LS is executed for 150 independent iterations. The target of this computational comparison is to experimentally show how a systematic change of neighborhood (VNS) is able to outperform more direct approaches (say C2 and C2 + LS).

Finally, the results of the last experiment are shown in Table 7, where the performance of the procedures C2, C2 + LS and BVNS are compared by executing them over the whole set of 73 HB instances, where the optimum is not known. Taking into account that we are considering large instances (up to 1000 vertices), the computing time could not be eventually affordable. Therefore, the time limit for stopping the heuristic methods was set to 1000 s. We can observe that the inclusion of the local search procedure LS described in Section 4.5 improves the solution obtained by the constructive approach C2 alone. Furthermore, BVNS clearly outperforms C2 and C2 + LS in all headings. Specifically, BVNS is able to find the best solution in the 73 instances under consideration (in 749.77 s), while C2 + LS finds it in only 38 instances (requiring 1204.38 s) and C2 only finds it in 16 instances (requiring 1417.05 s).

6. Conclusions

In this paper we have proposed a pure 0-1 optimization model for the Vertex Separation Problem (VSP). This model has O(mn^2) variables and O(mn^3) constraints, which makes it impractical for relatively large instances, as we have reported in our experimentation. We therefore have also presented an approximation algorithm. As far as we know, it is the first heuristic procedure for general graphs for the VSP. Specifically, we have introduced two constructive procedures based on different greedy strategies, so-called C1 and C2. Experimental results show that C1 marginally improves the performance of C2. Additionally, we introduce a novel scheme for calculating the objective function which substantially reduces the computing time (by a factor of 18) as compared with the direct implementation. We also propose a local search strategy, LS, based on exchanges that incorporates a new definition of the improvement move. It allows the procedure to search in flat landscapes. Experimental results revealed that coupling constructives procedures with the local search strategy, improves the results of constructive methods by themselves in terms of quality although consuming more computing time. We can observe in our extensive experimentation that the best combination is C2 + LS which means that although as a constructive procedure C1 obtains better results than C2, when combined with the local search C2 becomes the best alternative. We also introduce a shake procedure which selects vertices according to their contribution, performing an interchange in a stochastic way (in order to favor diversification). Finally, we incorporate C2, LS and the shake procedure in a BVNS algorithm. The proposed metaheuristic has been tested on a large benchmark consisting of 173 well-known instances in the existing literature. Specifically, the BVNS has been able to find the optimal Vertex Separation in 81 out of 100 instances (whose optimal solution is known by construction). In the rest of instances, where the optimum value is unknown, the BVNS procedure clearly outperforms C2 and C2 + LS when executing all the procedures with the same running time proving experimentally the superiority of the BVNS metaheuristic.

Acknowledgments

This research has been partially supported by the Spanish Ministry of Science and Innovation, grants TIN2009-07516 and TIN2011-28151, and the Government of the Community of Madrid, grant S2009/TIC-1542.

References

[1] Bodlaender HL, Möhring RH. The pathwidth and treewidth of cographs. In: Proceedings of the second Scandinavian workshop on algorithm theory, SWAT'90; 1990. p. 301–9.

[2] Bodlaender HL, Kloks T, Kratsch D. Treewidth and pathwidth of permutation graphs. SIAM Journal of Discrete Mathematics 1995;8(4):606–616.

[3] Bodlaender HL, Gilbert JR, Hafsteinsson H, Kloks T. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. Journal of Algorithms 1995;18(2):238–255.

[4] Bodlaender HL, Gustedt J, Telle JA. Linear-time register allocation for a fixed number of registers. In: Proceedings of the symposium on discrete algorithms; 1998. p. 574–3.

[5] Bollobás B, Leader I. Edge-isoperimetric inequalities in the grid. Combinatorica 1991;11:299–314.

[6] Díaz J, Penrose MD, Petit J, Serna M. Approximating layout problems on random geometric graphs. Algorithms 2001;39(1):78–116.

[7] Díaz J, Petit J, Serna M. A survey of graph layout problems. ACM Computing Survey 2002;34(3):313–356.

[8] Duarte A, Martí R, Resende MGC, Silva RMA. GRASP with path relinking heuristics for the antibandwidth problem. Networks 2011;58(3):171–189.

[9] Dujmović V, Fellows MR, Kitching M, Liotta G, Mccartin K, Nishimura N, et al. On the parameterized complexity of layered graph drawing. Algorithmica 2008;52(2):267–292.

[10] Ellis JA, Sudborough IH, Turner JS. The vertex separation and search number of a graph. Information and Computation 1978;113:50–79.

[11] Ellis JA, Sudborough IH, Turner JS. The vertex separation and search number of a graph. Journal of Information and Computation 1994;113:50–79.

[12] Ellis JA, Markov M. Computing the vertex separation of unicyclic graphs. Information Computing 2004;192(2):123–161.

[13] Fellows MR, Langston MA. On search, decision and the efficiency of polynomial-time algorithms. Journal of Computing Systems Sciences 1994;49(3): 769–779.

[14] Fomin FV, Hie K. Pathwidth of cubic graphs and exact algorithms. Information Processing Letters 2006;97:191–196.

[15] Gustedt J. On the pathwidth of chordal graphs. Discrete Applied Mathematics 1993;45(3):233–248.

[16] Goldberg PW, Golumbic MC, Kaplan H, Shamir R. Four strikes against physical mapping of DNA. Journal of Computing Biology 1995;12(1):139–152.

[18] Hansen P, Mladenovic N, Moreno JA. Variable neighbourhood search: methods and applications. Annals of Operations Research 2010;175(1):367–407.

[19] IBM ILOG CPLEX, User manual; 2009.

[20] Kinnersley NG. The vertex separation number of a graph equals its pathwidth. Information Processing Letters 1992;42(6):345–350.

[21] Kinnersley NG, Langston MA. Obstruction set isolation for the gate matrix layout problem. Discrete Applied Mathematics 1994;54(2–3):169–213.

[22] Kirousis M, Papadimitriou CH. Interval graphs and searching. Discrete Mathematics 1985;55(2):181–184.

[23] Kirousis M, Papadimitriou CH. Searching and pebbling. Theory of Computation Sciences 1986;47(2):205–218.

[25] Leiserson CE. Area-efficient graph layouts (for VLSI). In: Proceedings of IEEE symposium on foundations of computer science; 1980. p. 270–1.

[26] Lengauer T. Black–white pebbles and graph separation. Acta Informatica 1981;16:465–475.

[27] Lewis JG. The Gibbs–Poole–Stockmeyer and Gibbs–King algorithms for reordering sparse matrices. ACM Transactions on Mathematical Software 1982;8:190–194.

[28] Lipton RJ, Tarjan RE. A separator theorem for planar graphs. SIAM Journal of Applied Mathematics 1979;36:177–189.

[29] Miller GA. The magical number seven, plus or minus two. SIAM Journal of Applied Mathematics 1956;13:81–97.

[30] Mladenović N, Hansen P. Variable neighborhood search. Computers and Operations Research 1997;24:1097–1100.

[31] Monien B, Sudborough IH. Min cut is NP-complete for edge weighted trees. Theory of Computation Sciences 1988;58:209–229.

[32] Pantrigo JJ, Martí R, Duarte A, Pardo EG. Scatter search for the cutwidth minimization problem. Annals of Operations Research, doi:10.1007/s10479-011-0907-2.

[33] Peng SL, Ho C-W, Hsu TS, Ko MT, Tang CY. A linear-time algorithm for constructing an optimal node-search strategy of a tree. In: Proceedings of the fourth annual international conference on computing and combinatorics, COCOON'98; 1998. p. 279–8.

[34] Raspaud A, Schröder H, Sýkora O, Török L, Vrt'o I. Antibandwidth and cyclic antibandwidth of meshes and hypercubes. Discrete Mathematics 2009;309: 3541–3552.

[35] Resende MGC, Martí R, Gallego M. Duarte a GRASP and path relinking for the max-min diversity problem. Computers and Operations Research 2010;37: 498–508.

[36] Skodinis K. Computing optimal strategies for trees in linear time. In: Proceedings of the eighth annual European symposium on algorithms; 2000. pp. 403–4.

Chapter 8

Scatter Search for the Profile Minimization Problem

- J. Sánchez-Oro, M. Laguna, A. Duarte, and R. Martí. Scatter search for the profile minimization problem. Networks, 65(1):10–21, 2015.
  - DOI: `https://doi.org/10.1002/net.21571`
  - Impact Factor (JCR 2015): 0.943

Subject Category	**Ranking**	**Quartile**
Computer Science, hardware & architecture	29 / 51	Q3
Operations Research & management science	54 / 82	Q3

Scatter Search for the Profile Minimization Problem

Jesús Sánchez-Oro
Departamento de Ciencias de la Computación, Universidad Rey Juan Carlos, Madrid, Spain

Manuel Laguna
Leeds School of Business, University of Colorado at Boulder, Boulder, Colorado

Abraham Duarte
Departamento de Ciencias de la Computación, Universidad Rey Juan Carlos, Madrid, Spain

Rafael Martí
Departamento de Estadística e Investigación Operativa, Universidad de Valencia, Valencia, Spain

We study the problem of minimizing the profile of a graph and develop a solution method by following the tenets of scatter search. Our procedure exploits the network structure of the problem and includes strategies that produce a computationally efficient and agile search. Among several mechanisms, our search includes path relinking as the basis for combining solutions to generate new ones. The profile minimization problem (PMP) is NP-Hard and has relevant applications in numerical analysis techniques that rely on manipulating large sparse matrices. The problem was proposed in the early 1970s but the state-of-the-art does not include a method that could be considered powerful by today's computing standards. Extensive computational experiments show that we have accomplished our goal of pushing the envelope and establishing a new standard in the solution of the PMP. © 2014 Wiley Periodicals, Inc. NETWORKS, Vol. 65(1), 10–21 2015

Keywords: profile minimization; SumCut problem; metaheuristics; scatter search; OptQuest; LocalSolver; sparse matrices

1. INTRODUCTION

Given a graph $G(V, E)$—where V is a set of n vertices and E is a set of edges—an ordering (or permutation) φ of the vertices is a one-to-one mapping between the set $\{1, 2, \ldots, n\}$ and V. An ordering in a graph can be conceptualized as locating its vertices in a line, as shown in Figure 1. For a given ordering φ, the profile $\delta_{\varphi(i)}$ of vertex $\varphi(i)$—that is, the vertex in position i—is given by $\delta_{\varphi(i)} = i - f_{\varphi(i)}$, where $f_{\varphi(i)}$ is the smallest $j < i$ such that $\varphi(j)$ is adjacent to $\varphi(i)$ (if none exists then $f_{\varphi(i)} = i$). The profile $P(G, \varphi)$ of G with the ordering φ, is the sum of the profiles of all its vertices. The profile minimization problem (PMP) consists of finding an ordering φ^* of V such that the profile is minimized. Mathematically, the PMP seeks φ^*, over the set Φ of all possible permutations, such that

$$P(G, \varphi^*) = \min_{\varphi \in \Phi} \sum_{i=1}^{n} \delta_{\varphi(i)} = \min_{\varphi \in \Phi} \sum_{i=1}^{n} (i - f_{\varphi(i)})$$

Figure 1 shows an example of a network with six vertices ordered in a line, where the first one is labeled A, the second one B, and so on. In this figure, $f_1 = 1$ because there is no vertex $\varphi(j)$ for which $j < 1$. As a result, $\delta_1 = 0$. Similarly, $f_2 = 2$ and $\delta_2 = 0$. Conversely, $f_3 = 1$ and $\delta_3 = 3 - 1 = 2$, as the smallest j is 1, corresponding to vertex A. All additional calculations are shown in Figure 1, resulting in a profile value of $P(G, \varphi) = 11$.

It is possible to reduce the profile of the graph in Figure 1 by permuting its vertices. Figure 2 shows the same graph with the ordering $\varphi^* = (B, D, F, E, C, A)$, which results in an optimal profile value of $P(G, \varphi^*) = 8$.

Another interpretation of the PMP on a graph is based on the notion of labeling. Each vertex v in the graph is assigned a label $\pi(v)$ that is nothing else than the position that the vertex would occupy if the vertices were arranged in a line as in Figures 1 and 2. A solution is fully specified when all vertices have been labeled. By definition, the relationship between φ and π is such that if $\varphi(i) = v$ then $\pi(v) = i$. For a

Received May 2012; accepted September 2014
Correspondence to: M. Laguna; e-mail: laguna@colorado.edu
Contract grant sponsor: Ministerio de Educación y Ciencia of Spain; Contract grant number: TIN2012-35632-C02
DOI 10.1002/net.21571
Published online 23 December 2014 in Wiley Online Library (wileyonlinelibrary.com).

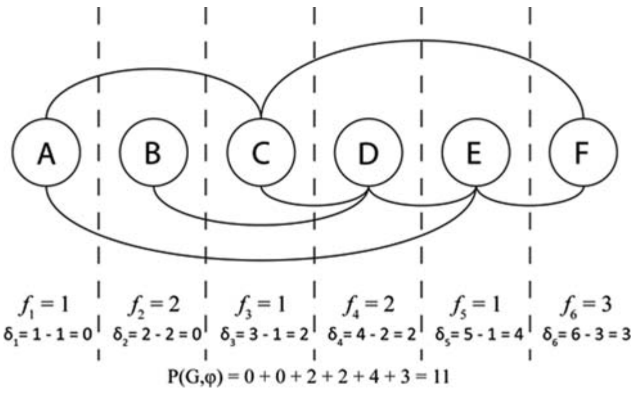

A
B
C
D
E
F
$f_1 = 1$ $\delta_1 = 1 - 1 = 0$
$f_2 = 2$ $\delta_2 = 2 - 2 = 0$
$f_3 = 1$ $\delta_3 = 3 - 1 = 2$
$f_4 = 2$ $\delta_4 = 4 - 2 = 2$
$f_5 = 1$ $\delta_5 = 5 - 1 = 4$
$f_6 = 3$ $\delta_6 = 6 - 3 = 3$
$P(G,\varphi) = 0 + 0 + 2 + 2 + 4 + 3 = 11$

FIG. 1. Illustrative graph and profile calculation.

given π, the profile of vertex v is calculated as $\delta_v = \pi(v) - f_v$, where f_v is the smallest label of a vertex adjacent to v, as long as such label is smaller than $\pi(v)$. Otherwise, $f_v = \pi(v)$ and $\delta_v = 0$. The objective of the PMP is to find a labeling π^* for which the sum of all vertex profiles is minimized. To facilitate the description of our work, we will use both interpretations.

The PMP was originally proposed as an approach to reduce the space requirements for storing sparse matrices [31]. In this context, the PMP was shown to be equivalent to the SumCut problem [1]. One of the main applications of the PMP continues to be the reduction of the space requirements to store systems of equations. Additionally, the PMP enhances the performance of operations on systems of nonlinear equations, such as the Cholesky factorization [30]. Another interesting application of the PMP is described by Karp [15] in the context of the Human Genome Project. The main goals of this project are to identify all genes in human DNA (between 20 and 25 thousand, approximately) and determine the sequences of the approximately 3 billion chemical base pairs associated with human DNA.

In archeology [16], the PMP has been used in connection with the problem of organizing items such as fossils, tools, and jewels according to a specific order. This process is known as “seriation” and consists of placing several items from the same culture in a chronological order determined by a method of establishing dates that are relative to each other. This results in a problem for which the reordering of the rows and columns of a matrix is required, which is equivalent to the reordering of a linear graph. Other applications of the PMP can be found in information retrieval [2] and fingerprinting [15].

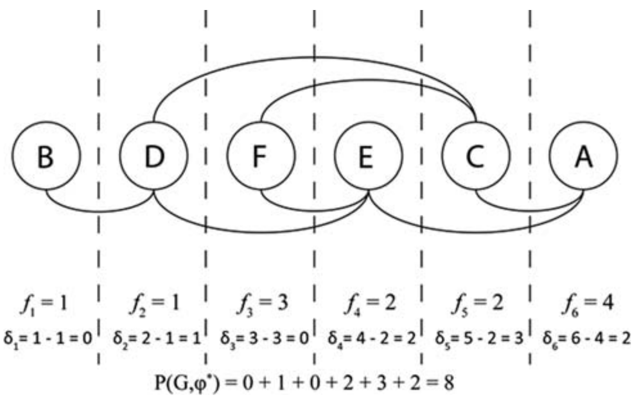

B
D
F
E
C
A
$f_1 = 1$ $\delta_1 = 1 - 1 = 0$
$f_2 = 1$ $\delta_2 = 2 - 1 = 1$
$f_3 = 3$ $\delta_3 = 3 - 3 = 0$
$f_4 = 2$ $\delta_4 = 4 - 2 = 2$
$f_5 = 2$ $\delta_5 = 5 - 2 = 3$
$f_6 = 4$ $\delta_6 = 6 - 4 = 2$
$P(G,\varphi^*) = 0 + 1 + 0 + 2 + 3 + 2 = 8$

FIG. 2. Illustrative graph reordered.

Lin and Yuan [20] proved that the PMP for an arbitrary graph is equivalent to the interval graph completion problem, which was shown to be NP-complete by Garey and Johnson [10]. However, special classes of graphs can be solved optimally in polynomial time. For example, Lin and Yuan [20] proposed several polynomial-time algorithms to find the optimal solution of the PMP for paths, wheels, complete bipartite graphs and D4-trees (trees with diameter 4). Likewise, Guan and Williams [14] developed an algorithm to find the optimal solution of the PMP for triangulated graphs.

The SumCut problem [11, 25, 26] and the PMP are equivalent in the sense that a solution to one problem provides a solution to the other problem by reversing the corresponding permutation. Consequently, the optimum of one problem corresponds to the optimum of the other [1]. The PMP is also related to the bandwidth minimization problem (BMP), which consists of finding a permutation of the rows and the columns in a matrix, such that all the nonzero elements are confined to a band that is the closest to the main diagonal. The tabu search by Campos et al. [3] is the best BMP heuristic in the literature and we adapt it to the PMP to show that a specialized PMP search is justified to find solutions of the highest quality.

Algorithmically, the PMP has been tackled as the late sixties. To the best of our knowledge, the first heuristic in the literature consists of a constructive procedure proposed by Cuthill and McKee [4] known as the Reverse Cuthill–McKee algorithm (RCM). Their procedure is based on constructing a level structure of the vertices. Poole et al. [27] improved the RCM algorithm by changing the selection of the root node within a more general level structure. The method is known in the literature as GPS. Gibbs [12] developed GK, a procedure that uses a pseudodiameter to produce a new level structure. GK outperforms GPS in solution quality but it requires more computational time. Lewis [18] introduced new implementations that improve the performance of both GK and GPS. His experimental results confirm the superiority of GK over GPS in terms of solution quality.

The best heuristic for the PMP in the literature belongs to Lewis [19]. It uses the simulated annealing (SA) methodology in combination with existing constructive procedures. The SA search starts from a solution constructed with RCM or GK, whichever is better according to the objective function value. In typical SA fashion, neighborhood exploration is performed with moves that are randomly generated. Improving moves are always executed and nonimproving moves are only executed with a probability that depends on the current temperature. A so-called cooling schedule controls the systematic reductions of the temperature and the search ends when the temperature reaches a prespecified minimum level.

In contrast to all past efforts, we develop a specialized procedure that it is built within the scatter search (SS) framework.

Our main contribution consists of designing SS methods that are effective in finding high-quality solutions to the PMP. The mechanisms that we develop exploit the network structure of the problem and may be adapted to problems with similar characteristics. For instance, our diversification generation procedure creates diverse solutions by a strategy that is based on labeling the vertices in the graph. Likewise, the strategies embedded in the improvement method use label exchanges to explore the neighborhood of the current solution. A careful analysis of the neighborhood structure is performed to design an efficient process for evaluating exchanges. Finally, the combination method creates paths between solutions that result in a series of relabeling steps from an initiating to a target graph.

Our computational experiments and statistical tests show that we have been able to establish new benchmarks for the PMP. We compare not only against the best methods in the literature but also against a state-of-the-art procedure for a closely related problem (the BMP) and against two general-purpose black-box commercial optimizers with embedded metaheuristic search engines.

2. SCATTER SEARCH

Scatter search [17] is a metaheuristic whose framework includes five methods that are designed to build, maintain, and transform a population of solutions (see Fig. 3). Three of these methods, the diversification generation, the improvement, and the combination methods, are problem dependent and therefore their strategies take advantage of information that is context specific. Conversely, the Reference Set Update and the Subset Generation Methods have standard implementations that are context independent. The SS literature is fairly rich and includes multiple examples of successful applications, such as those documented in Gallego et al. [9], Martí et al. [21], Duarte et al. [7], and Pantrigo et al. [24].

The procedure starts with the application of the diversification generation method (which we describe in section 3 in the context of the PMP) and the Improvement method (described in section 4), obtaining as a result a population P of solutions from which the initial reference set (*RefSet*) of size b is constructed. The initial *RefSet* must balance solution quality and diversity and therefore the standard update in step 3 (Fig. 3) selects the best $b/2$ solutions from P and

```
1. Diversification generation
2. Improvement
3. Reference set update
while (termination criteria not satisfied)
    4. Subset generation
    5. Combination
    6. Improvement
    7. Reference set update
```

FIG. 3. Scatter search framework.

then the $b/2$ solutions in $P \setminus RefSet$ that are most diverse with respect to those solutions already in the reference set. We point out that selecting the most diverse solutions from a set is an NP-hard problem (see for instance Duarte and Martí [6]) and hence we perform this step heuristically. In particular, after selecting the $b/2$ best solutions (i.e., the ones with the best objective function value), the remaining solutions are added one at a time. The first one to be added is the solution in $P \setminus RefSet$ that is the most diverse with respect to the solutions currently in *RefSet*. Diversity is typically measured with a function that maximizes the minimum distance between the solution under consideration and the set of solutions where the solution will be added (in this case *RefSet*). As discussed above, a solution s to the PMP is fully characterized by its ordering φ_s and equivalently by the labeling π_s of the vertices in G. The distance between s and *RefSet* is given by

$$d(s, RefSet) = \min_{r \in RefSet} \left(\sum_{v=1}^{n} |\pi_s(v) - \pi_r(v)| \right)$$

The distance is the sum of the absolute differences of the labels assigned to each vertex in the candidate solution s and all the reference solutions r. As the labels represent positions, the calculation is equivalent to the so-called positional distance [5]. Once a solution s is selected from $P \setminus RefSet$, the solution is added to *RefSet* and the process is repeated $b/2$ times, choosing at each step the solution s^* with the maximum distance to the solutions currently in the reference set, that is:

$$s^* = \arg \max_{s \in P \setminus RefSet} d(s, RefSet)$$

In our implementation, step 4 in Figure 3 consists of generating all pairs of reference solutions that have not been combined before. Details about the combination method [based on the path relinking (PR) methodology] are presented in section 5.

The reference set update in step 7 is different from the updating performed in step 3. Assume that the reference solutions r are ordered according to their objective function values in such a way that the solution $r^{(1)}$ in the first position in the *RefSet* is the best and the solution $r^{(b)}$ in the last position is the worst. To maintain the diversity among the reference solutions, a new solution s is admitted if either of the following conditions is satisfied, where $P(s)$ is the profile of solution s:

1. $P(s) < P(r^{(1)})$
2. $P(s) < P(r^{(b)})$ and $d(s, RefSet) > dthresh$

The first condition establishes that a new solution becomes a reference solution if it is better than the best solution found so far. The second condition allows a new solution s into the reference set if it is better than the worst reference solution and its distance to the current reference set is larger than the distance threshold parameter *dthresh*. If solution s is admitted to the reference set, then s replaces the closest (according to d) reference solution r from all those for which $P(s) < P(r)$.

The process terminates when no new solutions become part of the *RefSet*. That is, if at a given iteration, *RefSet* does not change after step 7, then the search ends. We point out that our step 7 update is significantly different from the one typically used in SS implementations, where a new solution s replaces $r^{(b)}$ as long as $P(s) < P(r^{(b)})$.

3. DIVERSIFICATION GENERATION METHOD

We develop four diversification generation methods (labeled C1–C4) to build a population P of solutions. These methods are based on the GRASP methodology [8, 28, 29]. To describe these methods we define U as the set of vertices that have not been labeled and $L = V \setminus U$ as the set of the vertices that have already been labeled. Initially, all vertices are in U (i.e., $U = V$). C1 starts by selecting the vertex $w \in U$ with the smallest degree (i.e., the vertex with the least number of adjacent vertices). In this step, ties are broken arbitrarily. The chosen vertex w is given the first label and therefore $\pi(w) = 1$ and $\varphi(1) = w$. Then $U = U \setminus \{w\}$ and $L = L \cup \{w\}$ and a candidate list CL consisting of the set of vertices adjacent to w is constructed:

$$\mathrm{CL} = \{u : (w,u) \in E\}$$

A greedy function value is calculated for each vertex v in CL as follows:

$$g_1(v) = |N_L(v)| - |N_U(v)|$$

where $N_L(v) = \{u \in L : (v,u) \in E\}$ and $N_U(v) = \{u \in U : (v,u) \in E\}$. The greedy function $g_1(v)$ measures the level of “urgency” of labeling vertex v next. The function considers that it is more urgent to label a vertex for which most of its adjacent vertices have already been labeled. A greedy procedure would choose, at each step, the vertex v with the largest $g_1(v)$ value. However, in the GRASP framework, constructions are semigreedy and the implementation includes a so-called restricted candidate list (RCL). The RCL includes top candidates that are equally preferred and hence equally likely to be chosen:

$$\mathrm{RCL} = \{v \in \mathrm{CL} : g(v) \leq g_1^{\min} + \alpha_1(g_1^{\max} - g_1^{\min})\}$$

where $g_1^{\min}$ ($g_1^{\max}$) is the minimum (maximum) value of $g_1(v)$ for all v in CL, and α_1 is a parameter that controls randomness/greediness of the corresponding procedure. In particular, if $\alpha_1 = 0$, the constructive method would be deterministic and completely greedy. Conversely, for $\alpha_1 = 1$ the procedure becomes totally random. A vertex v from RCL is chosen randomly, the next available label is assigned to it, and the U and L sets are updated. CL is also updated by adding the unlabeled vertices that are neighbors of the selected vertex v (i.e., $\mathrm{CL} = \mathrm{CL} \cup N_U(v)$). The construction ends when all vertices have been labeled.

The second construction procedure (C2) is similar to C1 but implements the semigreedy selection in a different way. Instead of calculating the greedy function value first and then randomly selecting from a restricted candidate list, C2 first takes from CL a random sample of vertices. Then, the vertex with the largest $g_1(v)$ value in the sample is selected. The size of the random sample is controlled by the parameter α_2, which represents a fraction of the size of the candidate list (i.e., the random sample includes $\alpha_2|\mathrm{CL}|$ vertices). As before, once the vertex is selected, the next available label is assigned to it and the U and L sets are updated.

The $g_1(v)$ greedy function does not take into consideration the values of the labels assigned to the vertices adjacent to v. For instance, if i is the next available label, $g_1(v)$ does not include in its calculation the labels that vertices in $N_L(v)$ have received (which may be any between 1 and $i-1$). Clearly, the “urgency” of vertex v is greater if its adjacent vertices have received labels that are much smaller than i. The following greedy function takes this into consideration:

$$g_2(v) = (|N_L(v)| - |N_U(v)|) \sum_{u \in N_L(v)} |\pi(v) - \pi(u)|$$

This greedy function is used to formulate two additional construction methods. C3 is the same as C1 but with g_2 as the greedy function. C4 is the same as C2 but with g_2 as the greedy function.

4. IMPROVEMENT METHOD

For our improvement method we tested both a search neighborhood defined by swap moves and one defined by insert moves. The representation of a solution as an ordering of the vertices is useful to conceptualize these search neighborhoods. A *swap*(i,j) is the exchange of positions of vertices v, u that are currently in positions i and j, respectively. That is, $\varphi(i) = v$ and $\varphi(j) = u$. An *insert*(i,j) is the movement of vertex $v = \varphi(i)$ to position j. After the move, v precedes u if $i > j$ and v follows u if $i < j$. As both of these moves produce a neighborhood of size $\mathcal{O}(n^2)$, a fast calculation of the move value is critical to search such a neighborhood efficiently and identify the best move to make.

For instance, to evaluate the move insert(i,j) for $j > i$, it is easy to see that only the profile of the vertices v for which $i \leq \pi(v) \leq n$ needs to be recalculated. Therefore, if the profile values for all vertices in the current solutions are stored, then the calculation of the move value may be done by updating only those relevant profile values and adding them to the values that the move does not affect. Additionally, the updates to the profile values may be accumulated. For instance if we first evaluate the insert (i,j), then $n-i+1$ values must be recalculated, i.e., those corresponding to the vertices with labels between i and n. If we then evaluate insert$(i,j+1)$ without storing any information from the previous evaluation, we would need to recalculate once again $n - i + 1$ values. However, observing the change of the profile values from one trial move to the other, it can be determined that the only changes occur in the vertices with labels $\pi(v) \geq j + 1$.

Figure 4 illustrates two insertion moves applied to solution $\varphi = \{A, B, C, D, E, F\}$, on the left side of the figure. The first

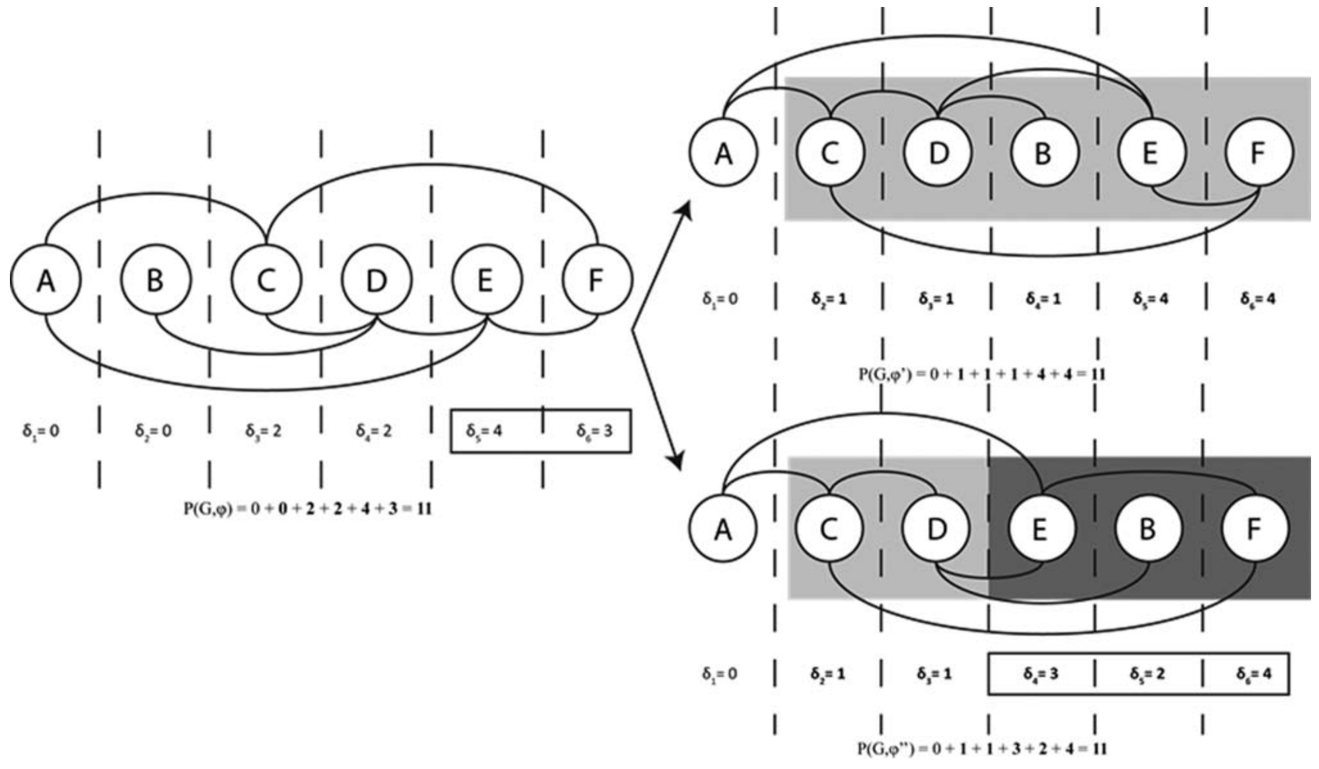


FIG. 4. Illustrative representation of a move.

move inserts vertex B in position 4, obtaining the solution $\varphi' = \{A, C, D, B, E, F\}$ at the top right of Figure 4, where vertices that changed their δ_i value are highlighted (i.e., vertices C, D, B, E, and F). The insertion of B in position 5 that yields a new solution φ'' is depicted at the bottom right of Figure 4. Considering that the improvement procedure evaluates insertions as a sequence of swap moves, evaluating the insertion of B in position 5 after evaluating the insertion of B in position 4 requires the updating of only vertices E, B, and F. Calculation savings are achieved because there is no need to update C and D again.

Therefore, an efficient way of searching the insert neighborhood is to evaluate the insert(i, j) as a sequence of swaps of the vertices in positions $(i, i + 1)$, then $(i + 1, i + 2)$ and so forth until $(j - 1, j)$. These values are stored because the sequence is the same for insert$(i, j + 1)$ with the addition of the swap$(j, j + 1)$. We have implemented this strategy and the corresponding one for the case when $j < i$.

By implementing the search as a sequence of swaps of vertices in adjacent positions, the only possible moves values are -1, 0, and +1. Consider the *swap*$(i, i + 1)$ that exchanges the labels of vertex v from $\pi(v) = i$ to $i + 1$ and vertex u from $\pi(u) = i + 1$ to i. Then, the move value is given by

$$\text{value}(i, i+1) = \begin{cases} -1 & \text{if } f_v > i \text{ and } f_u < i + 1 \\ 0 & \text{if } (f_v > i \text{ and } f_u \geq i + 1) \\ & \quad \text{or } (f_v \leq i \text{ and } f_u < i + 1) \\ +1 & \text{if } f_v \leq i \text{ and } f_u \geq i + 1 \end{cases}$$

This characteristic renders a first improving strategy impractical. A first improving refers to the strategy of stopping the neighborhood search after finding the first move that improves the current solution. As our neighborhood search is structured in such a way that complex moves [i.e., insert(i, j) for $j > i + 1$] are achieved by a sequence of simple moves [i.e., swap$(i, i + 1)$], stopping after a finding any improvement results in a process where improvements of more than one unit are not possible in a single move. Therefore, our implementation uses the best improving strategy in which the entire neighborhood is searched and the move with the best value is selected.

5. COMBINATION METHODS

We have developed one combination method (CM1) and a variant (CM2). The combination method follows the strategy known as PR [13]. The main idea behind PR is to construct a path between two elite solutions where the guide is not limited to the value of the objective function of the problem. PR was suggested in connection with tabu search because choice mechanisms in neighborhood-based searches often use the objective function as the only oracle to measure the quality of a move (just as we do in the improvement method described above). PR attempts to create search paths where the objective function is only one of the elements used to determine the direction. PR also exploits the principle that the neighborhood of an elite solution might contain other high-quality solutions that could be found if the elite solution is approached from a direction that is different from the one that was used to

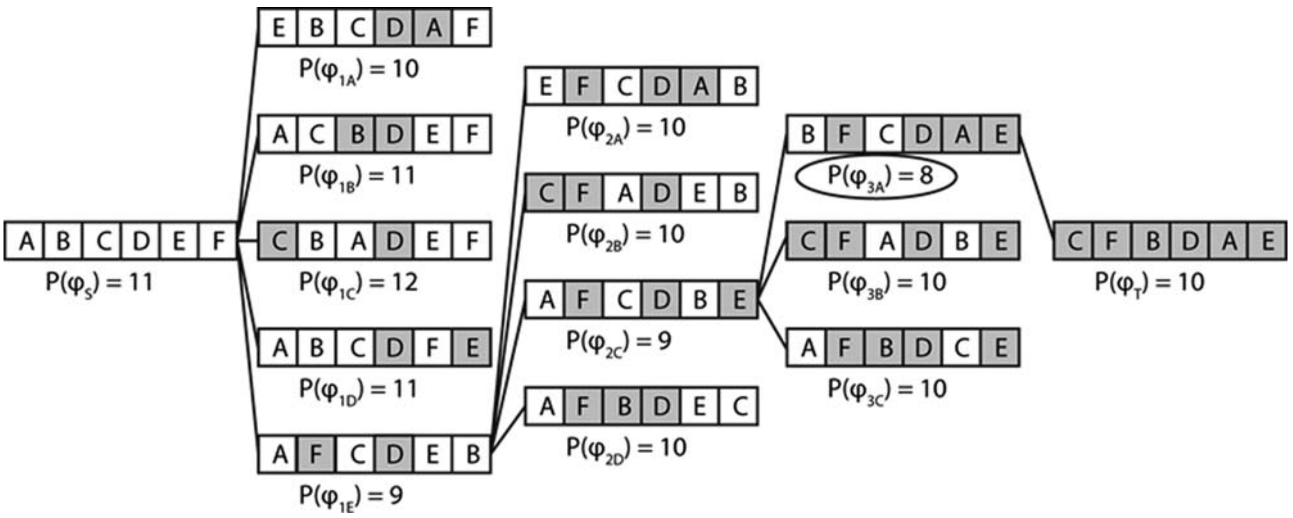


FIG. 5. Illustration of the combination method CM1.

find the elite solution in the first place. PR is implemented by choosing one or more elite solutions as the initiating solution and one or more as the guiding solutions. Strategies are then built around the notion of applying transformations to the initiating solution with the goal of moving it toward the guiding solution(s). As all solutions that have been found during a search are connected by the path that the search followed to find them, building a new path between elite solutions may be conceptualized as a relinking exercise, and hence the name of the strategy.

Let s be the initiating (or starting) solution and t the guiding (or target) solution. CM1 starts by identifying the set D of vertices that have different labels in s and t, that is, $D = \{v : \pi_s(v) \neq \pi_t(v)\}$. A set of solutions is generated by swapping the labels of vertex v and vertex $\varphi_s(\pi_t(v))$ in the initiating solution, for all $v \in D$. Note that after these swaps, $\pi_s(v)$ equals $\pi_t(v)$ and the initiating solution moves closer to the guiding solution. This step generates $|D|$ trial solutions from which we select the best according to the objective function value. The chosen solution becomes the new initiating solution and the process continues until $D = \emptyset$, that is, until the labeling in s is the same as in t. The procedure, referred to as Greedy Path Relinking in [28], returns the best trial solution found.

Figure 5 illustrates how CM1 operates on two solutions, where s is the initiating solution and t is the guiding solution. The relinking requires 4 steps and generates 12 intermediate solutions that are labeled with a digit and a letter, where the digit is the step number and the letter a solution identifier. Note that at each step, the best solution (according to the objective function value) becomes the initiating solution for the next step. Once the guiding solution is reached, the best intermediate solution (in this case solution 3A with a profile value of 8) is returned.

We created a variant of CM1 that we refer to as CM2 and that consists of replacing the selection criterion of the trial solution generated during the relinking process. In particular, instead of selecting the best solution with respect to the objective function value from the set D of trial solutions, the next initiating solution is selected randomly. In this way, CM2 favors diversification over intensification. As before, CM2 returns the best solution encountered during the PR process.

6. COMPUTATIONAL EXPERIMENTS

We now describe the computational experiments performed to test the SS approach that we developed for the PMP and then we discuss the resulting outcomes. The SS procedure was implemented in Java SE 6 and was compared against the best methods reported in the literature so far, namely, RCM [4], and SA [19]. The RCM and SA results were obtained running the original C implementations shared by the authors. We also compare against an adaptation of the best heuristic for the BMP as well as two general-purpose optimizers. All tests were performed on an Intel Core i7 2600 machine running at 3.4 GHz and with 2 GB of RAM. The test set consists of 262 instances divided into three subsets. All instances are available at www.optsicom.es/pmp/.

- HB—This set is derived from 73 instances in the Harwell-Boeing Sparse Matrix Collection [22]. The data matrices in this set correspond to problems in linear systems, least squares and eigenvalue calculations in a wide variety of scientific and engineering disciplines. We selected the 73 problems with $n \leq 1000$ and therefore the number of vertices ranges from 24 to 960 and the number of edges ranges from 34 to 3721.
- K-Graphs—This set contains 98 bipartite graphs with number of vertices ranging from 4 to 142 and number of edges ranging from 3 to 5016. A complete bipartite graph is such that the set of vertices V can be divided into two subsets V_1 and V_2 in such a way there exists an edge between every pair of vertices, one belonging to V_1 and the other belonging to V_2. At the same time, there is no edge for which its endpoint vertices are in the same subset. Optimal objective function values are known by construction [20] and are given by $n_1 n_2 + \frac{1}{2} n_1(n_1 - 1)$ for $n_1 \leq n_2$, where $n_1 = |V_1|$ and $n_2 = |V_2|$.

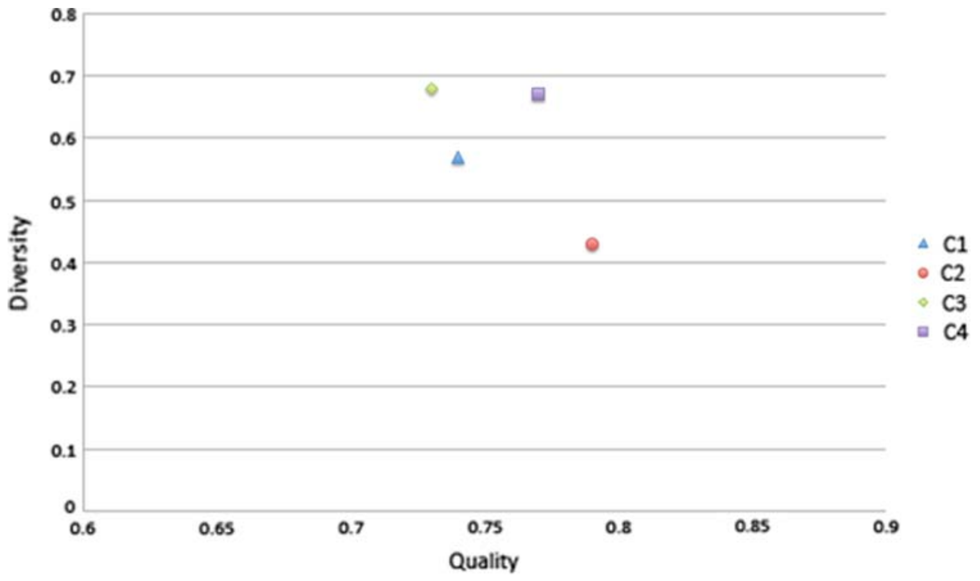


FIG. 6. Comparison of construction methods. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

- D4-Trees—This set consists of 91 instances that are based on trees with a diameter of 4 and a number of vertices ranging from 10 to 100 and a number of edges ranging from 9 to 99. These trees are built by choosing a root vertex v_0 and adjacent vertices v_1 to v_k, for $k \geq 2$. The vertices adjacent to v_0 form a star with v_0 in the center. If $\gamma(v_i)$ is the degree of vertex v_i then the v_i-branch has $\gamma(v_i) - 1$ leaves, because the other edge is the one connecting v_i to the root vertex v_0. The trees in the D4 set are built in such a way that $\gamma(v_1) \geq \gamma(v_2) \geq \cdots \geq \gamma(v_k) \geq 2$ and with a diameter of 4. This means that the tree consists of the root vertex, of all the adjacent vertices in a star configuration and of all the vertices corresponding to the leaves of the tree. The optimal objective function values are known by construction and are given by $|E|+\sum_{i=3}^{k}(\gamma(v_i)-1)$.

The experiments are divided into two main blocks. The first block has the goal of studying the behavior of the components of the solution procedure as well as determining the best values for the search parameters. The second block of experiments has the goal of comparing our procedure with the best in the literature and two commercial optimizers. To be able to test the effectiveness of our strategies in the entire PMP class, the first set of experiments is performed on a subset of all problem instances. In this way, we can test how well our choices generalize to the entire set of problems. Specifically, the tuning experiments are performed on the 30 HB instances that have less than 250 vertices. We refer to these instances as the training set and to all other instances as the testing set.

A common practice in SS implementations is to arrive to the best design by choosing the components of the solution procedure sequentially. As we are using standard implementations for the reference set update and the subset generation methods, our design process focuses on choosing a diversification generator, an improvement method and a combination method. As described in the previous sections, we have developed four diversification generation methods (C1–C4), two improvement methods (one based on swaps and one based on inserts), and two combination methods (CM1 and CM2). This results in a full-factorial design with 16 combinations. We start by contrasting the sequential process that selects components one at a time and adds them to the solution procedure with the full-factorial approach that identifies the best combination with a single experiment.

For the sequential design process, we begin by comparing the four construction methods described in section 3. As these methods are used for diversification purposes, the comparison metrics must include a measure of diversity as well as a measure of solution quality. Each method (C1–C4) is executed to generate 100 solutions. The value of α_1 for C1 and C3, and α_2 for C2 and C4 are chosen at random in each construction. For validation purposes, we also generate 100 solutions at random and refer to this set of solutions as C0. The objective function value of each solution is used to calculate an average quality of each set, which is then normalized to a value between 0 and 1. The average diversity of a set of solutions is calculated using the distance measure described in section 2. That is, for a particular set of solution, the distance between a solution and the rest in the set (i.e., the other 99 solutions) is computed and then the average of the 100 values obtained is used to represent the diversity of the set. We also normalize this diversity measure to obtain a value between 0 and 1. Figure 6 compares the diversity and quality of the construction methods.

Not surprisingly, C0 obtains the largest diversity of all the sets, as well as the lowest quality, and hence it is not depicted in the figure. However, this set of solutions is of minimum quality when compared to the constructions based on GRASP. The C2 set has the highest quality but the lowest diversity. The sets generated by C3 and C4 are very similar, with C3 slightly more diverse and C4 with slightly higher quality. To

TABLE 1. Construction procedures coupled with improvement methods

Variant	Obj	Dev (%)	#Best	Time
C2+Swap	1235.94	4.61	7	68.76
C3+Swap	1209.72	2.05	10	90.84
C2+Insert	1209.34	1.30	17	2.56
C3+Insert	1194.78	0.67	18	3.90

test the effect of adding the improvement method to these construction procedures, we select C2 and C3 to perform additional experiments. Ignoring the purely random generator C0, the set generated by C2 and C3 are at the extremes of the frontier in which C1 is dominated and C4 is the most balanced nondominated point.

The second experiment couples the construction methods C2 and C3 with the improvement methods based on swap and insert neighborhoods, resulting in 4 different variants. Once again, the α values associated with C2 and C3 are chosen randomly in each construction. Table 1 shows results obtained when applying these variants to the training set. For each variant, the table reports the average objective function value (Obj), the percent deviation of this value from the average obtained with the best known solutions (Dev), the number of best solutions found out of 30 (#Best) and the CPU time in seconds.

The improvement method based on inserts seems to be more effective than the one based on swaps according to the results in Table 1. The table also shows that local search based on swap moves is computationally more expensive than the one based on insert moves. The best combination of construction and improvement is given by C3+Insert, CPU time notwithstanding. Therefore, in this sequential process, we have determined, so far, that our final SS should have C3 as the diversification generator and Insert as the improvement method.

Finally, we must choose a combination method that works well with the C3+Insert improvement method. For this experiment, we also need to add the subset generation method and the reference set improvement method. The subset generation method is a standard implementation and has no parameters. The behavior of the reference set update method depends on the values of b and the *dthresh* parameters. Although we set b to the standard value of 10 used in the SS literature, in this experiment, we try 4 values for *dthresh* (0.01, 0.05, 0.10, and 0.30), which is given as a fraction of the maximum distance MD between the labeling of two solutions. Considering the distance function formulated in section 2, *MD* is computed as follows:

$$\text{MD} = \sum_{i=1}^{n} |i - (n - i + 1)|$$

As indicated at the end of section 2, the search terminates when no solution generated by the application of the combination and improvement methods is admitted to the reference set. Table 2 summarizes the results of this experiment, where the column labels have the same meaning as in Table 1 with the addition of the *dthresh* column containing the value of the parameter that was used to produce the results.

TABLE 2. Performance of combination methods CM1 and CM2

Variant	*dthresh*	Obj	Dev (%)	#Best	Time
	0.01* MD	1187.97	0.10	25	6.54
C3+Insert+CM1	0.05* MD	1187.38	0.08	25	6.37
	0.10* MD	1187.63	0.09	24	6.71
	0.30* MD	1188.16	0.12	24	6.73
	0.01* MD	1191.69	0.43	21	6.08
C3+Insert+CM2	0.05* MD	1191.56	0.43	21	5.74
	0.10* MD	1191.78	0.45	21	5.90
	0.30* MD	1191.56	0.43	20	5.43

According to the results shown in Table 2, the best SS configuration should include CM1 as the combination method. Within that, the best value for *dthresh* seems to be 0.05, which results in the smallest deviation from the best known solutions. The results indicate that while there is a significant difference in performance between using CM1 and using CM2, the procedure is robust with respect to the value of *dthresh*. In particular, there is no significant difference between values in the range between 0.01 and 0.10 for CM1.

The final experiment of this block consists of choosing the best SS configuration by running a single full-factorial experiment. To perform this experiment, we set $b = 10$, $dthresh = 0.05$ and α is chosen randomly in each construction. Table 3 summarizes the results obtained with the 16 variants on the training set.

The results of the full-factorial design that Table 3 summarizes indicate that there are two configurations that dominate all others when considering percent deviation and number of best solutions: C1+Insert+CM1 and C3+Insert+CM1. The C3+Insert+CM1 configuration was the one identified as the best by the sequential design process. It achieves the lowest

TABLE 3. Full-factorial experiment

Variant	Obj	Dev (%)	#Best	Time
C1+Insert+CM1	1189.44	0.80	20	6.59
C1+Insert+CM2	1195.38	0.96	18	5.30
C1+Swap+CM1	1195.94	1.35	13	92.68
C1+Swap+CM2	1200.91	1.74	12	91.24
C2+Insert+CM1	1218.31	3.30	11	3.79
C2+Insert+CM2	1219.41	3.51	12	3.48
C2+Swap+CM1	1228.91	4.42	9	76.20
C2+Swap+CM2	1232.91	4.74	7	75.36
C3+Insert+CM1	1188.06	0.75	17	6.78
C3+Insert+CM2	1191.53	1.04	14	6.58
C3+Swap+CM1	1201.91	1.79	10	101.27
C3+Swap+CM2	1202.78	1.95	12	96.13
C4+Insert+CM1	1195.28	1.27	14	6.91
C4+Insert+CM2	1200.97	1.50	15	6.14
C4+Swap+CM1	1205.44	2.25	10	94.01
C4+Swap+CM2	1205.66	2.44	10	93.00

TABLE 4. Experiments with 98 K-Graphs

Procedure	Obj	Dev	#Opt	Time
RCM	1165.64	0.00	98	1.02
SA	1167.06	0.02	97	5.87
SS	1165.64	0.00	98	0.66

deviation of 0.75% and the third largest number of best solutions of 17. The C1+Insert+CM1 configuration achieves the largest number of best solutions of 20 and an average deviation of 0.8% that is the second lowest. The configuration uses a diversification generator that is different from the one that we selected during the sequential design process. In fact, in our analysis, C1 was dominated by the other three construction procedures in terms of both quality and diversity (see Fig. 6). Hence, the full-factorial design is able to identify a configuration that performs well when all elements are put together at once but that does not seem attractive when the merit of each element is assessed separately. Nonetheless, the fact that the C3+Insert+CM1 configuration is in the nondominated set seems to indicate that in situations where running a full-factorial design is not practical, a sequential process is an approach from which it is reasonable to expect an effective combination of SS components. This analysis concludes the first block of experiments.

In the second block of experiments, we start by comparing the performance of the SS procedure configured as C1+Insert+CM1 to the best methods in the literature. In particular, the comparison includes RCM [4] and SA [19]. Tables 4 and 5 show the results associated with the K-Graph and D4-Tree data sets, respectively. The optimal solutions to these problems are known by construction and therefore the deviations are calculated using the optimal objective function values. Also, the tables report the number of optimal solutions found (#Opt) instead of the number of best solutions.

Clearly, the K-Graphs do not represent a challenge to any of the procedures in our test. Only SA fails to find the optimal solutions to one of the instances, even when using considerably more time than its counterparts. The situation changes when the methods are applied to the D4-Tree data set. The results are shown in Table 5 where the difficulty of these problems becomes evident. The best performance is achieved by SS, which is able to find the optimal solution to 97.8% (i.e., 89 out of 91) of the problems. The solution time is negligible for all procedures.

We perform nonparametric tests to provide additional support to our conclusions about the performance of the SS implementation. First, we apply the Friedman test for multiple correlated samples to the best solutions obtained by each of the three methods in Table 5. This test computes, for each instance, the rank value of each method according to solution quality (where rank 1 is assigned to the best method and rank 3 to the worst). Then, it calculates the average rank values for each method across all instances. If the averages differ greatly, the associated p-value or level of significance is small. The resulting p-value obtained in this experiment (smaller than 0.001) clearly indicates that there are statistically significant differences among the three methods. The rank values produced by this test are 1.18 (SS), 1.85 (SA), and 2.97 (RCM).

TABLE 5. Experiments with 91 D4-Trees

Procedure	Obj	Dev(%)	#Opt	Time
RCM	282.75	173.57	2	1.01
SA	126.08	30.39	31	0.68
SS	86.12	0.01	89	0.28

Next, we used the Wilcoxon test and Sign test to make a pairwise comparison of SS and SA, which consistently provide the best solutions reported in our experiments. The results of the Wilcoxon test (with a p-value smaller than 0.001) determined that the solutions obtained by the two methods indeed represent two different populations. The Sign test (with a p-value smaller than 0.001) indicated that the objective function values of the solutions obtained with SS tend to be better (i.e., smaller) than those obtained with SA.

In the final experiment of this block, we add three procedures to our comparison set: (1) an adaptation of the tabu search developed by Campos et al. [3] for the solution of the BMP (TS-BMP), (2) OptQuest, and (3) LocalSolver. The BMP is related to the PMP because they both have the goal of finding a graph labeling such that the corresponding sparse incidence matrix is transformed into one for which the nonzero elements are close to the main diagonal. The PMP achieves this goal with an additive objective function that penalizes a distance measure from the main diagonal while the BMP uses a criterion that minimizes the maximum deviation. Specifically, in the BMP, the objective is to find an ordering of the rows and columns of a matrix M in such a way that the nonzero elements are in a band that is as close as possible to the main diagonal. In terms of the graph representation, the goal is to find a labeling π of the vertices of the corresponding $G(V, E)$ graph such that the bandwidth $B(G, \pi)$ is minimized, where:

$$B(G,\pi) = max(B(v,\pi) : v \in V) \quad \text{and}$$
$$B(v,\pi) = max(|\pi(v) - \pi(u)| : (v,u) \in E).$$

That is, $B(v, \pi)$ is the bandwidth of vertex v for labeling π and it is calculated as the maximum absolute difference between the label of v and the labels of its adjacent vertices. The bandwidth of the graph is the maximum vertex bandwidth. Given these similarities, it is reasonable to believe that a solution procedure designed for the BMP might find high-quality solutions to the PMP. To test this, we applied TS-BMP without any modifications to the HB instances. TS-BMP operates with the objective of minimizing $B(G, \pi)$ and on termination the procedure returns the solution π^* with the best value according to this objective function. We then calculate $P(G, \varphi^*)$ by applying the transformation $\varphi^*(\pi^*(v)) = v$ for all vertices in the graph.

TABLE 6. Experiments with 26 HB instances with $n \leq 200$

Procedure	Obj	Dev (%)	#Best	Rank	Time
OptQuest	1466.54	46.7	2	4.3	36.9
LocalSolver	1030.04	2.4	23	1.3	60.0
TS-BMP	1596.83	42.7	1	5.1	30.8
RCM	1109.58	20.4	2	3.9	97.3
SA	1110.92	14.5	4	3.3	4.0
SS	1007.58	1.1	15	1.5	2.8

TABLE 7. Experiments with 27 HB instances with $200 < n \leq 500$

Procedure	Obj	Dev (%)	#Best	Rank	Time
OptQuest	8923.07	61.1	0	4.1	211.9
LocalSolver	7492.30	25.5	8	2.9	220.0
TS-BMP	10204.82	73.4	0	5.2	212.1
RCM	7491.85	34.0	4	3.2	335.6
SA	7542.48	33.1	1	3.6	39.5
SS	5701.48	2.2	17	1.4	78.5

OptQuest (optquest.com) is a general-purpose black-box optimizer built on a SS platform. The system includes several variable types, including permutations. We used the latest OptQuest version in C# and followed three simple steps: (1) read data, (2) define optimization model by creating *n* permutation variables, and (3) create `Evaluate()` to evaluate the objective function. Each OptQuest iteration consists of the evaluation of a trial solution that is generated by the search mechanisms implemented within the system.

LocalSolver (localsolver.com) is also a black-box optimizer for combinatorial problems. LocalSolver uses a hybrid approach to optimization that ranges from neighborhood search and metaheuristics to constraint propagation and mathematical programming. The main elements of the search are the so-called autonomous moves (complex moves, such as ejection chains, that have the goal of preserving feasibility) and incremental evaluation (a concept introduced by Michel and van Hentenryck [23]). LocalSolver admits only binary variables and therefore a transformation must be used to apply it to an ordering problem such as the PMP. In particular, we created a LocalSolver model with binary variables `x`, such that `x[i][j]=1` indicates that vertex `i` has label `j`. Appendix A contains the LocalSolver Program (LSP) that we used to perform the experiment.

The results associated with this experiment are in Tables 6–8. As the optimal solutions to the HB instances are not known, the deviations are calculated against the best-known solutions and #Best refers to the number of best-known solutions found by each procedure. These tables also include the average rank for each procedure. The tables divide the instances into three different groups according to the number of vertices in the graph: 26 instances with less than or equal to 200 vertices (Table 6), 27 instances with more than 200 and no more than 500 vertices (Table 7), and 20 instances with more than 500 vertices (Table 8). As TS-BMP is designed to tackle connected graphs and 11 HB instances correspond to graphs that are not connected, our experiments with this code are limited to 62 instances (23, 22, and 17 instances for Tables 6–7, and 8, respectively).

The procedures from the literature (TS-BMP, RCM, and SA) were executed with the parameters values suggested by their authors. As all three methods are heuristics, we impose a maximum computing time of *n* seconds, where *n*, as before, represents the number of vertices. To stop the execution of OptQuest and LocalSolver, we also use a time limit that depends on the size of the problem. However, we allow them to run considerably longer on the larger instances because these two procedures are not specialized to the PMP. The stopping times (in seconds) are 60 ($n \leq 200$), 120 ($200 < n \leq 300$), 300 ($300 < n \leq 500$), 600 ($500 < n \leq 600$), 900 ($600 < n \leq 700$), 1200 ($700 < n \leq 800$), 1800 ($800 < n \leq 900$), 3600 ($n > 900$). We configure OptQuest to report the time when the incumbent solution is last updated during the run and those are the average times that we report in Tables 6–8. The time when the procedure stops is the same as the time reported for the LocalSolver.

The results in Table 6 indicate that LocalSolver is a very good alternative for problems with up to 200 vertices. This procedure finds the largest number of best solutions and therefore has the best rank. Its average deviation is slightly larger than SS due to a 42.58% deviation in problem CAN 198 (see Appendix B for the complete list of problems in the HB dataset). In contrast, the largest deviation for SS in this subset of problems is 16.06% (in the DWT 162 instance).

The merit of SS becomes evident as the problem size increases. The results in Tables 7 and 8 support the position that SS is the best procedure for the PMP when dealing with medium to large instances.

We applied statistical tests to the results in Table 6–8. The Friedman test detects significant differences in all cases (*p*-value smaller than 0.001). We used the Wilcoxon test and Sign test to make a pairwise comparison between the best two procedures in each set of instances. In particular, both tests are inconclusive with regard of LocalSolver and SS for the instances in Table 6. Conversely, considering SS and its closest competitor, LocalSolver (Table 7) and SA (Table 8), both tests indicate that the solutions obtained with SS are consistently and significantly (*p*-value smaller than 0.001) better, confirming the superiority of the proposed method. Table 9 in Appendix B shows the details of the SS output for the HB instances.

TABLE 8. Experiments with 20 HB instances with $n > 500$

Procedure	Obj	Dev (%)	#Best	Rank	Time
OptQuest	27156.00	105.4	1	4.7	1518.4
LocalSolver	26078.30	90.0	1	4.4	1575.0
TS-BMP	25226.29	61.4	0	4.5	714.7
RCM	19256.05	28.5	1	2.7	744.1
SA	19246.65	29.2	0	3.2	1367.3
SS	15629.05	1.0	17	1.2	674.8

TABLE 9. Best individual values on HB instances obtained with SS

Instance	Size	Best	SS	Time	Instance	Size	Best	SS	Time
CAN 24	24	95	**95**	0.1	CAN 292	292	4673	4718	41.7
BCSPWR01	39	82	**82**	0.1	DWT 307	307	6676	**6676**	46.4
BCSSTK01	48	460	466	0.4	DWT 310	310	2630	**2630**	25.0
BCSPWR02	49	113	**113**	0.3	DWT 346	346	6051	**6051**	54.1
DWT 59	59	214	223	0.4	DWT 361	361	4635	**4635**	64.8
CAN 61	61	338	**338**	0.4	PLAT362	362	9150	10620	105.8
CAN 62	62	172	**172**	0.5	LSHP 406	406	5964	**5964**	83.2
BCSSTK02	66	2145	**2145**	0.4	DWT 419	419	6679	**6679**	107.3
DWT 66	66	127	**127**	0.2	BCSSTK06	420	13239	13437	123.9
DWT 72	72	147	151	0.7	BCSSTK07	420	13224	13437	123.9
CAN 73	73	520	**520**	0.9	BCSPWR05	443	3076	3354	90.3
ASH85	85	490	**490**	1.4	CAN 445	445	15494	**15494**	150.4
DWT 87	87	428	434	1.2	NOS5	468	20446	**20446**	209.0
CAN 96	96	1078	1080	2.1	BCSSTK20	485	3006	**3006**	195.8
NOS4	100	651	**651**	1.1	DWT 492	492	3361	**3361**	151.0
BCSSTK03	112	272	**272**	0.3	494 BUS	494	3499	**3499**	237.5
BCSPWR03	118	434	**434**	3.2	DWT 503	503	13152	**13152**	192.3
BCSSTK04	132	3154	3159	4.9	DWT 512	512	3975	**3975**	144.6
BCSSTK22	138	628	641	2.2	LSHP 577	577	10035	10045	222.1
CAN 144	144	969	**969**	2.3	DWT 592	592	9498	**9498**	220.5
BCSSTK05	153	2191	2192	4.3	DWT 607	607	13278	**13278**	419.3
CAN 161	161	2482	**2482**	6.6	CAN 634	634	28493	**28493**	499.9
DWT 162	162	1108	1286	5.8	662 BUS	662	8962	**8962**	318.5
CAN 187	187	2184	2195	9.7	NOS6	675	9095	**9095**	184.2
DWT 193	193	4355	4388	15.3	685 BUS	685	8528	**8528**	799.8
DWT 198	198	1092	**1092**	7.6	CAN 715	715	24414	**24414**	984.3
DWT 209	209	2494	2621	25.1	NOS7	729	34226	34675	338.7
DWT 221	221	1646	**1646**	20.1	DWT 758	758	6392	**6392**	371.3
CAN 229	229	3928	4141	27.9	LSHP 778	778	15719	**15719**	586.2
DWT 234	234	782	803	10.6	BCSSTK19	817	7638	**7638**	855.7
NOS1	237	467	**467**	2.1	DWT 869	869	13107	**13107**	1526.5
DWT 245	245	2053	**2053**	29.7	DWT 878	878	17259	**17259**	1104.6
CAN 256	256	5049	**5049**	40.8	GR 30 30	900	24311	**24311**	1190.4
LSHP 265	265	3162	**3162**	25.7	DWT 918	918	16502	**16502**	1277.3
CAN 268	268	5215	**5215**	25.0	NOS2	957	1907	**1907**	36.0
BCSPWR04	274	1808	1992	40.6	NOS3	960	38676	45631	2222.8
ASH292	292	2717	2784	61.8					

7. CONCLUSIONS

Our goal was to develop a state-of-the-art solution method for the PMP. We accomplished this goal with an implementation of a SS procedure that computational experiments show to be superior to the solution methods reported in the literature. We developed a number of mechanisms that we believe can be helpful in similar problem settings. For instance, the ideas behind our efficient move evaluation in the improvement method could be adapted to other problems with similar characteristics (i.e., labeling vertices in a graph). Also, in the process of choosing the best SS design, we discovered that a sequential approach is capable of producing a highly competitive design and thus allowing future SS implementations to avoid a full-factorial design when such a design is computationally intractable.

APPENDIX A

This is the `model()` function used within the LSP applied to the HB instances that can be found in www.optsicom.es/pmp/. Note that `vcount[i]` contains the number of vertices adjacent to vertex `i` and `vlist[i][j]` is the `j`th vertex adjacent to vertex `i`. Also, `p` is the profile of each vertex that is being added to the value of `obj`.

```
function model()
{
  // x[i][j] equal to 1 if vertex i is assigned
     label j
  x[1..nodes][1..nodes] <- bool();

  // one label per vertex i for [i in 1..nodes]
     constraint sum[j in 1..nodes](x[i][j]) == 1;
  // one vertex per label j for [j in 1..nodes]
     constraint sum[i in 1..nodes](x[i][j]) == 1;

  //label[i] is the label of vertex i
  label[i in 1..nodes] <- sum[j in 1..nodes]
     (j*x[i][j]);
  obj <- 0; for [i in 1..nodes]
  {
    p <- 0;
    for[j in 1..vcount[i]] p <- max(label[i]
      - label[vlist[i][j]], p);
    obj <- obj + p;
  }
  minimize obj;
}
```

APPENDIX B

Table 9 shows the SS results for the 73 Harwell-Boeing instances (ordered by size). The table compares the SS objective function value and the best known value (Best) and it also shows the SS computing time (in seconds). Best solutions found by the SS procedure are shown in bold.

Acknowledgment

The authors would like to thank Professor Robert R. Lewis for sharing the source code of the RCM and SA methods.

REFERENCES

[1] A. Agrawal, P. Klein, and R. Ravi, Ordering problems approximated: Single-processor scheduling and interval graph completion, Automata Languages Programming 510 (1991), 751–762.

[2] R.A. Botafogo, Cluster analysis for hypertext systems, Proceedings of the 16th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, 1993, pp. 116–125.

[3] V. Campos, E. Piñana, and R. Martí, Adaptive memory programming for matrix bandwidth minimization, Ann Oper Res 183 (2011), 7–23.

[4] E. Cuthill and J. McKee, Reducing the bandwidth of sparse symmetric matrices, Proc ACM 24th National Conf, 1969, pp. 157–172.

[5] A. Das and M. Roberts, Metric distances of permutations, 2004, Available at: www.cra.org/Activities/craw_archive/dmp/awards/2004/Das/paper.ps.

[6] A. Duarte and R. Martí, Tabu search and grasp for the maximum diversity problem, Eur J Oper Res 178 (2007), 71–84.

[7] A. Duarte, F. Glover, R. Martí, and F. Gortázar, Hybrid scatter tabu search for unconstrained global optimization, Ann Oper Res 183 (2011), 95–123.

[8] T.A. Feo and M.G.C. Resende, A probabilistic heuristic for a computationally difficult set covering problem, Oper Res Lett 8 (1989), 67–71.

[9] M. Gallego, A. Duarte, M. Laguna, and R. Martí, Hybrid heuristics for the maximum diversity problem, Comput Optim Appl 44 (2009), 411–426.

[10] M.R. Garey and D.S. Johnson, Computers and intractability: A guide to the theory of NP-completeness, W. H. Freeman and Co, 1979.

[11] A. Gibbons, M. Paterson, J. Torán, and J. Diaz, The minsum-cut problem, Algorithms Data Struct 519 (1991), 65–79.

[12] N.E. Gibbs, A hybrid profile reduction algorithm, ACM Trans Math Softw 2 (1976), 378–387.

[13] F. Glover and M. Laguna, Tabu search, Kluwer Academic Publishers, Boston, 1997.

[14] G. Guan and K.L. Williams, Profile minimization of triangulated triangles, Discrete Math 260 (2003), 69–76.

[15] R. Karp, Mapping the genome: Some combinatorial problems arising in molecular biology, STOC'93 Proc Twenty-Fifth Ann ACM Symp Theory Comput, 1993, pp. 278–285.

[16] D. Kendall, Incidence matrices, interval graphs and seriation in archaeology, Pac J Math 28 (1969), 565–570.

[17] M. Laguna and R. Martí, Scatter search: Methodology and implementations in C, Kluwer Academic Publisher, Boston, 2003.

[18] J. Lewis, The Gibbs-Poole-Stockmeyer and Gibbs-King algorithms for reordering sparse matrices, ACM Transactions on Mathematical Software 8 (1982), 190–194.

[19] R. Lewis, Simulated annealing for profile and fill reduction, Int J Numer Methods Eng 37 (1994), 905–925.

[20] Y. Lin and J. Yuan, Profile minimization problem for matrices and graphs, Acta Math Appl Sinica, English-Series 10 (1994), 107–112.

[21] R. Martí, A. Duarte, and M. Laguna, Advanced scatter search for the max-cut problem, INFORMS J Comput 21 (2009), 26–38.

[22] Matrix Market 2011, webpage. http://math.nist.gov/MatrixMarket/collections/hb.html

[23] L. Michel and P. van Hentenryck, Localizer, Constraints 5 (2000), 43–84.

[24] J.J. Pantrigo, R. Martí, A. Duarte, and E.G. Pardo, Scatter search for the cut width minimization problem, Ann Oper Res 199 (2012), 285–304.

[25] M. Penrose, J. Petit, M. Serna, and J. Diaz, Convergence theorems for some layout measures on random lattice and random geometric graphs, J Combin Probab Comput 9 (2000), 489–511.

[26] J. Petit, M. Serna, and J. Díaz, A survey of graph layout problems, ACM Comput Surv 34 (2002), 313–356.

[27] W. Poole, P. Stockmeyer, and N. Gibbs, An algorithm for reducing the bandwidth and profile of a sparse matrix, SIAM J Numer Anal 13 (1976), 236–250.

[28] M.G.C. Resende, R. Martí, M. Gallego, and A. Duarte, GRASP and path relinking for the max-min diversity problem, Comput Oper Res 37 (2010), 498–508.

[29] M.G.C. Resende, S.H. Smith, and T.A. Feo, A greedy randomized adaptive search procedure for maximum independent set, Oper Res 42 (1994), 860–878.

[30] Y. Saad, Iterative methods for sparse linear systems, Society for Industrial and Applied Mathematics, San Francisco (California), 2003.

[31] R. Tewarson, Sparse matrices, Academic Press, New York, 1973.

Chapter 9

Combining intensification and diversification in VNS

- 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, Part B:209 – 219, 2014. Recent advances in Variable neighborhood search.
  - DOI: `https://doi.org/10.1016/j.cor.2013.11.008`
  - Impact Factor (JCR 2015): 1.988

Subject Category	**Ranking**	**Quartile**
Operations Research and Management Science	19 / 82	Q1
Engineering, industrial	11 / 44	Q1
Computer Science, interdisciplinary applications	32 / 104	Q2

Computers & Operations Research 52 (2014) 209–219

Contents lists available at ScienceDirect

Computers & Operations Research

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

Combining intensification and diversification strategies in VNS. An application to the Vertex Separation problem

Jesús Sánchez-Oro, Juan José Pantrigo, Abraham Duarte*

Dpto. de Ciencias de la Computación, Universidad Rey Juan Carlos, Móstoles, Madrid, Spain

ARTICLE INFO

Available online 18 November 2013

Keywords:
Combinatorial optimization
Intensification
Diversification
Variable Neighborhood Search
Layout problems
Vertex separation problem

ABSTRACT

The Vertex Separation problem (VSP) is an NP-hard problem with practical applications in VLSI design, graph drawing and computer language compiler design. VSP belongs to a family of optimization problems in which the objective is to find the best separator of vertices or edges in a generic graph. In this paper, we propose different heuristic methods and embed them into a Variable Neighborhood Search scheme to solve this problem. More precisely, we propose (i) a constructive algorithm, (ii) four shake procedures, (iii) two neighborhood structures, (iv) efficient algorithmic strategies to explore them, (v) an extended version of the objective function to facilitate the search process and finally, (vi) we embed these strategies in a Reduced Variable Neighborhood Search (RVNS), a Variable Neighborhood Descent (VND) and a General Variable Neighborhood Search (GVNS). Additionally, we provide an extensive experimental comparison among them and with the best previous method of the literature. We consider three different benchmarks, totalizing 162 representative instances. The experimentation reveals that our best procedure (GVNS) improves the state of the art in both quality and computing time. This fact is confirmed by non-parametric statistical tests. In addition, when considering only the largest instances, the other two proposed variants (RVNS and VND) also obtain statistically significant differences with respect to the best previous method identified in the state of the art.

1. Introduction

Graph layout problems are a class of combinatorial optimization problems where the objective consists of finding a linear labeling of the vertices of a graph in such a way that a specific objective function is maximized or minimized. Given an undirected and unweighted graph $G(V,E)$ where V (with $|V|=n$) and E (with $|E|=m$) are the sets of vertices and edges, respectively, a linear labeling φ of the vertices of G is a bijection $\varphi : V \to \{1,2,\ldots,n\}$ which assigns a unique and different integer between 1 and n to each vertex $u \in V$. A labeling has also been named as linear ordering, linear arrangement layout, numbering or simply, ordering. A common way to represent a layout problem is to dispose the vertices of a graph in a straight line, where each vertex u is allocated in position $\varphi(u)$. Fig. 1a shows an example of a graph, G, with 7 vertices and 11 edges, and Fig. 1b an example of a layout φ.

The Linear Arrangement [27], Bandwidth [25], Cutwidth [23] or Antibandwidth [7] fall into this class of optimization problems. In this paper, we tackle the Vertex Separation problem (VSP). Before defining mathematically the VSP, we need to introduce some definitions. Let $L(p,\varphi,G)$ be the set of vertices in V with a position in the layout φ lower than or equal to position p. Symmetrically, let $R(p,\varphi,G)$ be the set of vertices with a position in the layout φ larger than position p. In mathematical terms,

$$L(p,\varphi,G)=\{v \in V : \varphi(v) \le p\},$$

$$R(p,\varphi,G)=\{v \in V : \varphi(v) > p\}.$$

In general, $L(p,\varphi,G)$ can be simply referred to as the set of *left vertices* with respect to position p in φ. Symmetrically, $R(p,\varphi,G)$ is the set of the *right vertices* with respect to p in φ. We define the separation value at position p of layout φ, $Sep(p,\varphi,G)$, as the number of vertices in $L(p,\varphi,G)$ with one or more adjacent vertices in $R(p,\varphi,G)$. More formally

$$Sep(p,\varphi,G)=|\{u \in L(p,\varphi,G) : \exists v \in R(p,\varphi,G) \wedge (u,v) \in E\}|.$$

The Vertex Separation value (VS) of layout φ is the maximum of the *Sep*-value among all positions in φ:

$$VS(\varphi,G)=\max_{1 \le p < n} Sep(p,\varphi,G).$$

The Vertex Separation problem (VSP) then consists of finding a layout, $\varphi^\star$, minimizing the *VS*-value of the graph G. In mathematical

* Corresponding author. Tel.: +34 914888116.
E-mail address: abraham.duarte@urjc.es (A. Duarte).

0305-0548/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cor.2013.11.008

terms

$$\varphi^\star = \arg\min_{\varphi \in \Phi} VS(\varphi, G),$$

where Φ represents the set of all possible layouts of G.

Fig. 2a–f depicts the *Sep*-value of each position p of layout φ, $Sep(p,\varphi,G)$. For instance, $Sep(1,\varphi,G)=1$ (see Fig. 2a) because $L(1,\varphi,G)=\{\mathtt{C}\}$ and $R(1,\varphi,G)=\{\mathtt{A},\mathtt{D},\mathtt{E},\mathtt{B},\mathtt{F},\mathtt{G}\}$ and there is only one vertex in L having an adjacent vertex in R. Similarly, $Sep(5,\varphi,G)= 4$ (2e) because $L(5,\varphi,G)=\{\mathtt{C},\mathtt{A},\mathtt{D},\mathtt{E},\mathtt{B}\}$ and $R(3,\varphi,G)=\{\mathtt{F},\mathtt{G}\}$ and there are four vertices $\{\mathtt{C},\mathtt{A},\mathtt{D},\mathtt{B}\}$ in L that has an adjacent vertex in R. Notice that vertex $\mathtt{E}$ is not highlighted in Fig. 2e, since it has no adjacent vertices in R. Therefore, it does not contribute to the *Sep*-value. The objective function is then computed as the maximum of these *Sep*-values. In particular, for the example depicted in Fig. 2, this value is $VS(G,\varphi)=4$, associated to position $p=5$.

The VSP is strongly related to other well-known graph problems, such as the Path-width [17], the Node Search Number [19] or the Interval Thickness [18], among others. The description of the equivalences among these problems can be found in [12,17,19]. All these optimization problems are NP-hard and have practical applications in VLSI design [20], computer language compiler design [4] or graph drawing [9].

We can find efficient exact approaches to solve the VSP on special classes of graphs. A linear algorithm to compute the optimal Vertex Separation of a tree is proposed in [10] as well as a $O(n \log n)$ algorithm for finding the corresponding optimal layout. The algorithm was improved in [28] with a linear time procedure to find the optimal layout. In [24] an alternative method to compute the Vertex Separation of trees was proposed. In [11] a $O(n \log n)$ algorithm to compute the Vertex Separation of unicyclic graphs (i.e., trees with an extra edge) is proposed. A polynomial-time algorithm to compute the Path-Width (which is identical to VSP) is proposed in [2]. However, the algorithm cannot be considered from a practical point of view, since the bound on its time complexity is $\Omega(n)$, see [11]. In [5] it is proposed a polynomial time algorithm for optimally solving the VSP for n-dimensional grids. Co-graphs and permutational graphs can also be optimally solved as it was proposed in [1,2], respectively. Approximation algorithms have also been proposed for the VSP. Specifically, [3] proposes a polynomial time $O(\log^2 n)-$approximation algorithm for general graphs and a $O(\log n)-$approximation algorithm for planar graphs. Similar results for binomial random graphs are presented in [6]. Finally, [8] proposed a Basic VNS for the VSP. In particular, the authors presented two constructive procedures and one local search based on interchange moves. As far as we know, this algorithm obtains the best results in this problem, so we use it to test the performance of our proposals.

The main objective of this paper is to provide experimental evidences that, in the context of VNS, the compromise between intensification and diversification usually obtains the best results. In order to do so, we propose three VNS algorithms (RVNS, VND, and GVNS), where RVNS mainly focuses on the diversification, VND mainly focuses on the intensification, and GVNS balances intensification and diversification. In this line, we also propose new strategies to combine intensification and diversification within shake procedures. Computational experiments reveal that our best procedure (GVNS) improves the state of the art in both quality and computing time. This fact is confirmed with non-parametric statistical tests. In addition, when considering only the largest instances, the other two proposed variants (RVNS and VND) also obtain statistically significant differences with respect to the best previous method identified in the state of the art.

The rest of the paper is organized as follows. Section 2 introduces the VNS variants studied in this paper. We propose a new constructive procedure for the VSP (see Section 3.1). We provide a formal definition of two new neighborhood structures for the VSP based on insert moves (see Section 3.2). We then introduce four different shake procedures with different balance between diversification and intensification (see Section 3.3). In Section 3.4 we propose an extended version of the objective function which provides a more convenient view about the quality of the solutions. Additionally, we include an efficient strategy to perform insert moves (see Section 3.5) and algorithmic methods to explore the neighborhood structures (Section 3.6). The paper finishes with the comparison of the proposed algorithms with the state of the art (see Section 4) and the associated conclusions (Section 5).

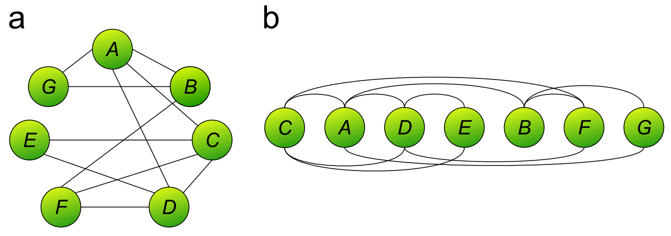


Fig. 1. (a) Graph illustrative example and (b) layout $\varphi=\{\mathtt{C},\mathtt{A},\mathtt{D},\mathtt{E},\mathtt{B},\mathtt{F},\mathtt{G}\}$.

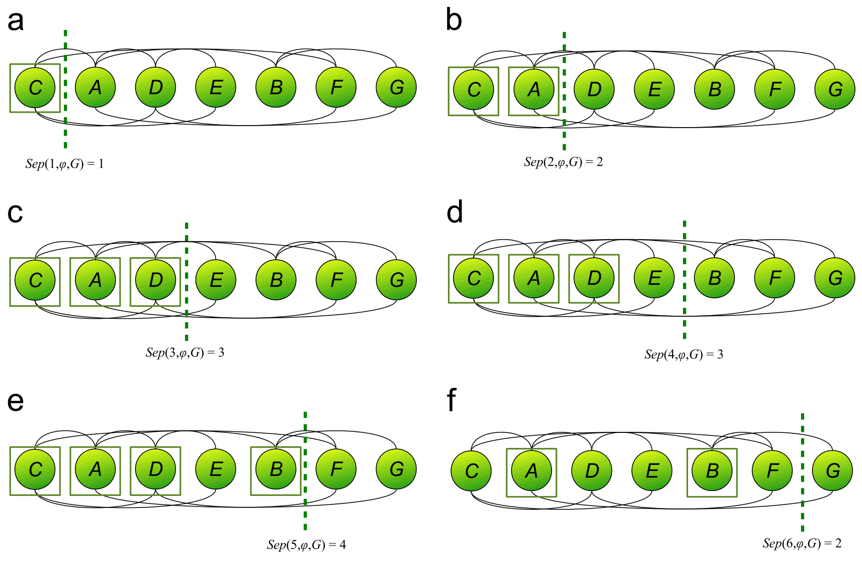


Fig. 2. (a)–(f) the computation of $Sep(p,\varphi,G)$ $(p=1,\ldots,6)$ for the layout $\varphi=\{\mathtt{C},\mathtt{A},\mathtt{D},\mathtt{E},\mathtt{B},\mathtt{F},\mathtt{G}\}$.

2. Variable Neighborhood Search

The Variable Neighborhood Search (VNS) is a metaheuristic proposed by Mladenovic and Hansen [22] as a general framework to solve hard optimization problems. It is based on the idea of performing systematic changes of neighborhood structures within the search procedure. The original metaheuristic has been widely evolved with many extensions, highlighting Variable Neighborhood Descent (VND), Reduced VNS (RVNS), Basic VNS (BVNS), Skewed VNS (SVNS), General VNS (GVNS), Variable Neighborhood Decomposition Search (VNDS) and Reactive VNS. See [14,15] for recent thorough reviews.

Let N_k with $1 \le k \le k_{max}$ be a finite set of pre-selected neighborhood structures, where $N_k(x)$ is the set of neighbor solutions of x in the k-th neighborhood. When solving an optimization problem by using different neighborhood structures (N_k), VNS methodology proposes to explore them in three different ways: (i) random, (ii) deterministic, or (iii) mixed, which hybridizes both, deterministic and random.

2.1. *Random exploration of neighborhoods*

The Reduced Variable Neighborhood Search (RVNS) consists of exploring (generating) solutions at random in each N_k neighborhood. This variant does not consider the application of a local search procedure. In fact, the values of these random solutions are directly compared with the value of the incumbent solution, updating the best solution in case of improvement. The stopping criterion in this variant is usually the CPU time allowed (t_{max}).

The pseudo-code of RVNS is shown in Algorithm 1. It has three input arguments: an initial solution (x), the largest predefined neighborhood (k_{max}) and the maximum computing time (t_{max}). The procedure starts by performing a perturbation to the current solution using the function `Shake`, in step 5, and obtaining a new solution x'. In step 6, it is decided whether the RVNS needs to explore a larger neighborhood by increasing k (if x' is worse than x) or not, which implies to set $k=1$ (if x' is better than x). Steps 5 and 6 are repeated until k_{max} is reached. This parameter determines the maximum number of different neighborhoods to be explored in the current iteration when there is no improvement in the solution. Steps 3–8 are repeated until t_{max} is reached, starting in each iteration from the incumbent solution.

Algorithm 1. Pseudocode of RVNS.

```
1:  procedure RVNS ($x, k_{max}, t_{max}$)
2:  repeat
3:    $k \leftarrow 1$
4:  repeat
5:      $x' \leftarrow$ Shake($x, k$)
6:      $(x, k) \leftarrow$ NeighborhoodChange($x, x', k$)
7:    until $k = k_{max}$
8:    $t \leftarrow$ CPUTime()
9:  until $t > t_{max}$
10: end RVNS
```

2.2. *Deterministic exploration of neighborhoods*

In Variable Neighborhood Descent (VND), several different neighborhoods are explored in order, typically from the smallest and fastest to evaluate, to the largest and slowest one. The process iterates over each neighborhood while improvements are found, performing local search until a local optimum is found at each neighborhood. Only strictly better solutions are accepted after each neighborhood search. Notice that most local search procedures are based on only one neighborhood which means that the local optimum is obtained with respect to that neighborhood. In VND, the returned solution is a local optimum in each N_k with $1 \le k \le k_{max}$. Therefore, the global optimum is more likely to be found.

The pseudo-code of VND is shown in Algorithm 2 where a nested strategy is considered. It has only two input arguments: an initial solution (x) and the number of neighborhoods (k_{max}). The procedure starts by obtaining a local optimum x' with respect to the first neighborhood. Then, instead of abandoning the search (as a local search procedure), VND resorts to the following neighborhood searching for an improvement. If so, the search starts again by considering the first neighborhood (which implies to set $k=1$). Otherwise, VND explores the next neighborhood by increasing k (until k_{max} is reached). This search strategy is condensed in steps 4 and 5 of Algorithm 2.

Algorithm 2. Pseudocode of VND.

```
1:  procedure VND($x, k_{max}$)
2:  $k \leftarrow 1$
3:  repeat
4:    $x' \leftarrow \arg\min_{y \in N_k(x)} f(y)$ {* Find the best neighbor in $N_k(x)$*}
5:    NeighborhoodChange($x, x', k$)
6:  until $k = k_{max}$
7:  end VND
```

2.3. *Mixed exploration of neighborhoods*

In general terms, the RVNS strategy favors the diversification of the search, while the VND focuses on the intensification. Basic VNS (BVNS) and its generalization, known as General VNS (GVNS) are a compromise between these two strategies. The BVNS and the GVNS methods combine deterministic and random changes of neighborhoods, where the deterministic component is given by an improvement procedure and the random component is given by the shake procedure. The main difference between BVNS and GVNS is that the improvement strategy in BVNS is typically a local search, while in GVNS it is replaced by a VND algorithm. This approach has led to some of the most successful applications reported in the literature [16]. Therefore, we consider GVNS instead of BVNS.

Algorithm 3 shows the pseudo-code of GVNS. It starts by considering an initial solution x, which is an input argument together with k_{max} and t_{max}. This solution is then perturbed using the function `Shake` (step 5), obtaining a new solution x'. Then, a local optimum, x'', is reached by using the VND procedure (step 6). In step 7, it is decided whether the GVNS needs to explore a larger neighborhood (since x'' is worse than x) or not, which implies to set $k=1$. Steps 5–7 are repeated until k_{max} is reached. This parameter determines the maximum number of different neighborhoods to be explored in the current iteration. Steps 3–9 are repeated until t_{max} is reached, starting in each iteration from the incumbent solution.

Algorithm 3. Pseudocode of GVNS.

```
1:  procedure GVNS ($x, k_{max}, t_{max}$)
2:  repeat
3:    $k \leftarrow 1$
4:    repeat
5:      $x' \leftarrow$ Shake($x, k$)
6:      $x'' \leftarrow$ VND($x'$)
7:      NeighborhoodChange($x, x'', k$)
8:    until $k = k_{max}$
9:    $t \leftarrow$ CPUTime()
10: until $t > t_{max}$
11: end GVNS
```

3. Algorithm approach

From an algorithmic perspective, VNS variants mainly differ in how they implement three basic strategies: shake (Section 3.3), neighborhood change (Section 3.4), and improvement (Section 3.6). We also consider a constructive procedure to create the initial solution needed for all VNS variants (Section 3.1). Finally, we propose different neighborhood structures (Section 3.2) and an efficient strategy to traverse them (Section 3.5).

3.1. Constructive procedure

We propose a new greedy procedure for constructing the initial solution for each VNS procedure in the context of the Vertex Separation problem. Given a graph G, this procedure is based on the creation of a tree T, where the nodes are organized in levels [21]. In this tree, two nodes belong to the same level if both have the same depth; otherwise, they belong to different levels. Let us denote $T=\{L_1, L_2, \dots, L_l\}$, where the first one, L_1, contains only one vertex (i.e., the root of the tree). L_2 contains only the adjacent vertices to the one in L_1. In general, a level L_s with $1<s\leq l$ contains all the vertices adjacent to some vertex in L_{s-1} that are not present in any previous level.

The constructed tree guarantees that the vertices in alternative levels are not adjacent. Therefore, we use a breadth-first search approach to construct the corresponding tree. The number of levels l (depth of the tree) exclusively depends on the vertex in L_1 and the graph G. In the context of the Vertex Separation problem, the larger the depth, the better the tree [8].

Algorithm 4 shows the pseudo-code of this procedure. The algorithm starts by constructing the set $\mathcal{T}$ with n different spanning trees of G. In particular, each $T\in\mathcal{T}$ is constructed by using a breadth-first search procedure (*BFS*), starting the search from a different vertex $v\in V$. Then, the algorithm finds the deepest tree $T^\star$ in the set $\mathcal{T}$ (see step 2). In the next step, we identify the set of levels in $T^\star$ and initialize the solution to the empty set (steps 3 and 4). The constructive procedure scans the levels L_i in ascending order (steps 5–10) and, for each element u in set L_i (see steps 6–9), the procedure inserts it in the best position of the partial solution under construction (steps 7 and 8). The constructive method ends when all the vertices of the tree have been inserted in the solution.

Algorithm 4. Pseudocode of the constructive procedure.

```
1:  procedure Constructive ($G$)
2:  $T^\star \leftarrow \arg\max_{T\in\mathcal{T}} Depth(BFS(G,T))$
3:  $\{L_1, L_2, \dots, L_l\} \leftarrow Levels(T^\star)$
4:  $\varphi \leftarrow \emptyset$
5:  for $L_i \in T^\star$ do
6:    for $u \in L_i$ do
7:      $j^\star \leftarrow \arg\min_{1\leq j\leq size(\varphi)} VS(Insert(\varphi,u,j))$
8:      $\varphi \leftarrow Insert(\varphi,u,j^\star)$
9:    end for
10: end for
11: return $\varphi$
12: end Constructive
```

The proposed constructive procedure has two different stages. The algorithm first constructs a spanning tree of the graph, which places the vertices belonging to a given level in consecutive positions of the layout. This strategy tries to place adjacent vertices as close as possible in the corresponding layout. In general, when a vertex has its adjacents in close positions, it is expected that the involved vertices present low *Sep*-values. However, this first stage does not provide any information about the relative ordering among the adjacent vertices. Therefore, we refine the solution under construction in the second stage. Specifically, the constructive procedure searches, for each vertex, the best position to be placed in the partial solution.

3.2. Neighborhood structures

Solutions to graph arrangement problems are typically represented as permutations, where the first vertex in the permutation receives the label 1, the second vertex receives the label 2, and so on. In this section, we define two neighborhood structures for

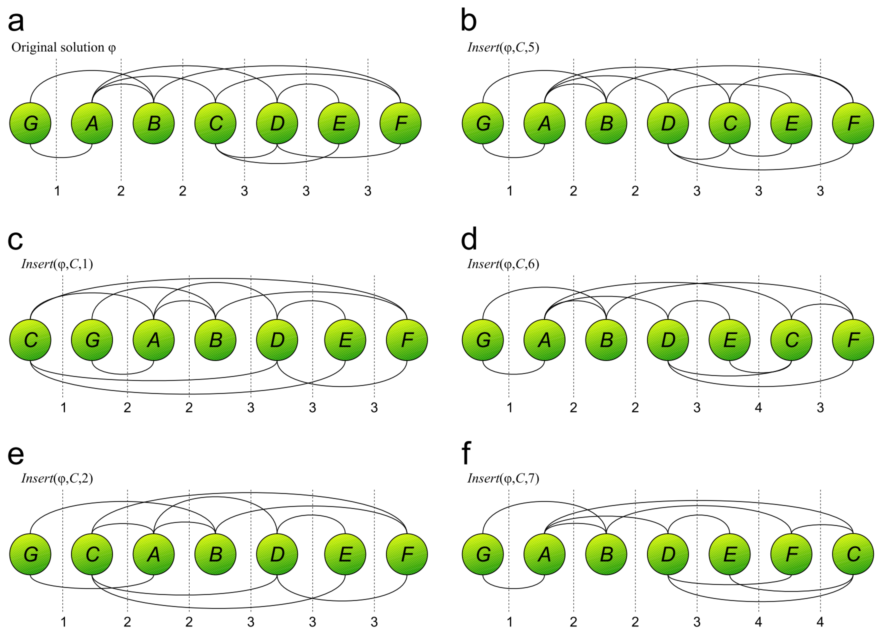


Fig. 3. (a) A given layout φ and (b)–(f) different solutions obtained inserting vertex C in positions $i=1$, 2, 5, 6 and 7.

permutation-based solutions. They are based on insert moves, which are typically defined as follows: given a solution $\varphi = (v_1, \ldots, v_{i-1}, v_i, v_{i+1}, \ldots, v_{j-1}, v_j, v_{j+1}, \ldots, v_n)$, $Insert(\varphi, v_i, j)$ consists of removing the vertex v_i from its current position $i = \varphi(v_i)$, and inserting it in position j, producing a new solution $\varphi' = (v_1 \ldots, v_{i-1}, v_{i+1}, \ldots, v_{j-1}, v_j, v_i, v_{j+1}, \ldots, v_n)$. For the sake of simplicity we denote $\varphi' = Insert(\varphi, v_i, j)$.

Considering the insert move defined above, we introduce the first neighborhood of φ as the set of solutions reachable by inserting the vertex v in each position of the permutation. This neighborhood is referred as $N_1(\varphi, v)$ and it is formally defined as

$$N_1(\varphi, v) = \{\varphi' = Insert(\varphi, v, j) : 1 \le j \le n\}.$$

The second neighborhood is based on the idea that we can identify "good" positions for a given vertex. Specifically, let $S(v)$ be the set of adjacent vertices to v. In mathematical terms, $S(v) = \{u \in V : (u, v) \in E\}$. Let $(u_1, u_2, \ldots, u_{|S(v)|})$ be the elements of $S(v)$ sorted in ascending order with respect to their label in the solution φ (i.e., $\varphi(u_1) < \varphi(u_2) < \cdots < \varphi(u_{|S(v)|})$). If we only consider moves of vertex v, we would obtain the maximum reduction of the *Sep*-value by inserting v in a position j such that $\varphi(u_1) < j < \varphi(u_2)$.

Let us consider that vertex v has a label $j = \varphi(v)$ such that $\varphi(u_1) < j < \varphi(u_2)$. We then expect that if we now insert v in a position $1 \le j \le \varphi(u_1)$ the value of the objective function can only remain equal or even get worse. Fig. 3 illustrates this behavior with the layout depicted in Fig. 3a. Consider the vertex C whose set of adjacent vertices is $S(\text{C}) = \{\text{A}, \text{D}, \text{E}, \text{F}\}$, which satisfies the property described above (i.e., $\varphi(\text{A}) < \varphi(\text{C}) < \varphi(\text{D}) < \varphi(\text{E}) < \varphi(\text{F})$) in $\varphi = \{\text{G}, \text{A}, \text{B}, \text{C}, \text{D}, \text{E}, \text{F}\}$. Fig. 3b and c shows the resulting solutions after the insertion of C in positions 1 and 2, respectively. The associated values of each solution produce a null improvement of the objective function.

If we alternatively insert the vertex v in a position j such that $\varphi(u_2) < j \le n$, we observe again the same behavior (i.e., it obtains a null improvement or deteriorates the value of the objective function). Fig. 3d–f illustrates that the obtained solutions are equal or worse than the original φ in terms of quality.

Notice that this property is a heuristic rule, which means that positions $\varphi(u_1) < j < \varphi(u_2)$ are not always the best option to insert a vertex in terms of the value of $VS(\varphi, G)$. Fig. 4 illustrates an example where the described heuristic rule does not find the best position for the insertion of a vertex. Specifically, Fig. 4a shows an example of layout obtained from the graph depicted in Fig. 1a, with a *VS*-value of 4. If we consider the heuristic rule described above, and focusing on vertex E, the best position to insert it is in j such that $\varphi(\text{C}) < j \le \varphi(\text{B})$. Then, $j=3$ since $\varphi(\text{C}) = 2$ and $\varphi(\text{B}) = 3$. Fig. 4b shows the layout obtained after performing the aforementioned insert move, whose *VS*-value is 4. Therefore, there is no improvement in the move. However, considering again the layout depicted in Fig. 4a, if we insert the vertex E in the last position of the layout (position 7), we obtain the layout depicted in Fig. 4c with a *VS*-value of 3, which improves the original one and the layout obtained when applying the heuristic rule.

This neighborhood is mathematically defined as follows:

$$N_2(\varphi, v) = \{\varphi' = Insert(\varphi, v, j) : \varphi(u_1) < j < \varphi(u_2)\}.$$

In order to further reduce the size of this neighborhood we only consider to perform the insertion of v in only one position $\varphi(u_1) < j < \varphi(u_2)$ selected at random. Therefore, considering this reduction, the neighborhood $N_2(\varphi, v)$ finally contains only one solution.

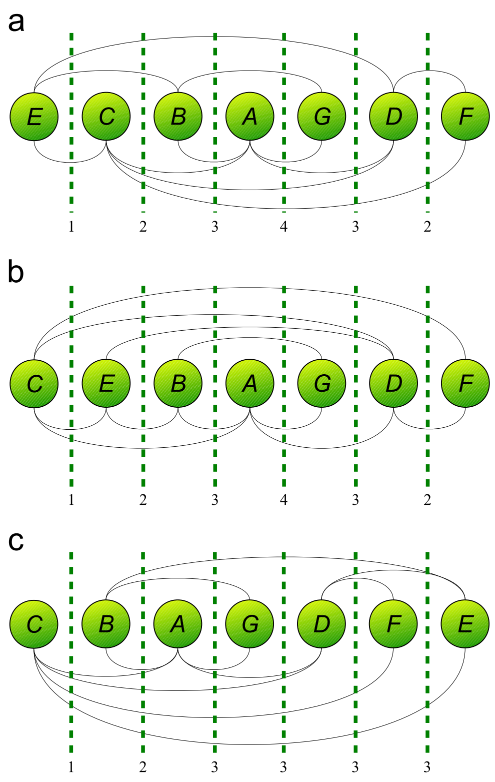


Fig. 4. (a) A given layout φ, (b) insertion of vertex E in the last position (7) starting from (a), and (c) insertion of vertex E in the most promising position (2) starting from (a).

3.3. Shake

In this section we propose 4 different shake functions for the vertex separation problem. These variants differ in how each one balances the intensification and the diversification. For each shake function we need to identify the set of vertices which will be perturbed and the associated positions for that vertices. In particular, $Shake{-}1(\varphi, k)$ selects k vertices at random and then those vertices are inserted again in positions selected at random. This method clearly favors the diversification of the search.

$Shake{-}2(\varphi, k)$ focuses on the intensification of the search. Specifically, this method selects the k vertices in V with the largest *Sep*-value in φ. Then, the procedure finds the best position to insert each vertex.

$Shake{-}3(\varphi, k)$ and $Shake{-}4(\varphi, k)$ try to find a compromise between the intensification and diversification. In particular, $Shake{-}3(\varphi, k)$ selects k vertices at random, but instead of inserting them in random positions, as $Shake{-}1(\varphi, k)$, they are inserted in the best position. Finally, $Shake{-}4(\varphi, k)$ selects the k vertices with the largest *Sep*-value in φ, and then the procedure inserts the vertices in random positions.

3.4. Neighborhood change

VSP is a min–max problem [8] where the value of the objective function is usually reached in several positions of the solution φ. This kind of problems presents a "flat landscape", which turns out in a challenge for classical local search procedures. Typically, local search strategies do not perform well from a computational point of view, since most of the moves have associated a null value. Then, given a graph G, changing the label i of a particular vertex v_i in φ (i.e., obtaining a new solution φ') such that its *Sep*-value is decreased, does not necessarily imply that $VS(G, \varphi') < VS(G, \varphi)$.

However, it can be considered as an interesting move if the number of vertices with a relative large *Sep*-value is reduced, regardless whether the objective function improves or not.

Given a solution φ, let us define $C(\varphi, i)$ as the set of vertices whose *Sep*-value is equal to i. For instance, the C sets for the layout depicted in Fig. 3a are: $C(\varphi, 1) = \{\mathtt{G}\}$, $C(\varphi, 2) = \{\mathtt{A}, \mathtt{B}\}$, and $C(\varphi, 3) = \{\mathtt{C}, \mathtt{D}, \mathtt{E}\}$.

Let c_{max} be the index associated to the set which contains the vertices with the largest *Sep*-value. Obviously, $VS(\varphi, G) = c_{max}$. We propose an alternative formulation (i.e., a new objective function) for the vertex separation problem. This extended formulation, denoted as $EVS(\varphi, G)$ overcomes the lack of information provided by the original objective function. Specifically, this new objective function is defined as follows:

$$EVS(\varphi, G) = \sum_{i=1}^{c_{max}} n^i |C(\varphi, i)|.$$

With this new formulation, we could compare any pair of solutions beyond the value of c_{max}. In particular, we consider that a move improves the current solution (using the new objective function) if any vertex involved in the move is removed from $C(\varphi, i)$ and included in $C(\varphi, j)$ with $j < i$, and without increasing the cardinality of any set $C(\varphi, l)$ for $l > i$.

It is important to remark that if we have two solutions φ and φ' such that $VS(\varphi, G) < VS(\varphi', G)$ (i.e., φ is better than φ') then $EVS(\varphi, G) < EVS(\varphi', G)$. A formal proof of this property is provided as follows.

Proof. Let $c_{max} = VS(\varphi, G)$ and $c'_{max} = VS(\varphi', G)$ be the indexes associated to the set which contains the vertices with the largest *Sep*-value in φ and φ', respectively. By definition, the extended version of the objective function for a solution φ and a graph G is

$$EVS(\varphi, G) = \sum_{i=1}^{c_{max}} n^i |C(\varphi, i)|.$$

Let us separate the last term of the summation from the rest of elements

$$EVS(\varphi, G) = n^{c_{max}} |C(\varphi, c_{max})| + \sum_{i=1}^{c_{max}-1} n^i |C(\varphi, i)|.$$

By construction, the n vertices of the graph are distributed among the $C(\varphi, i)$ sets, where $1 \le i \le c_{max}$. Obviously, if all vertices are in set $C(\varphi, c_{max})$, *EVS* presents its maximum value. In this situation $C(\varphi, c_{max}) = n$ and $C(\varphi, i) = 0$ for $1 \le i < c_{max}$. Then,

$$EVS(\varphi, G) \le n^{c_{max}} \times n = n^{c_{max}+1}.$$

Our hypotheses establish that $c_{max} < c'_{max}$. Then, considering that both c_{max} and c'_{max} are integer values, it trivially holds that $c_{max} + 1 \le c'_{max}$. Therefore

$$EVS(\varphi, G) \le n^{c'_{max}}.$$

The extended version of the objective function for a solution φ' and a graph G is

$$EVS(\varphi', G) = \sum_{i=1}^{c'_{max}} n^i |C(\varphi', i)|.$$

If we again separate the last term of the summation from the rest of elements, we have

$$EVS(\varphi', G) = n^{c'_{max}} |C(\varphi', c'_{max})| + \sum_{i=1}^{c'_{max}-1} n^i |C(\varphi', i)|.$$

We can affirm that $|C(\varphi', c'_{max})| \ge 1$ since at least one vertex must be in $C(\varphi', c'_{max})$. Additionally, $\sum_{i=1}^{c'_{max}-1} n^i |C(\varphi', i)| \ge 0$ since the summation is performed over non-negative terms. Considering that the n vertices of the graph are distributed among the $C(\varphi', i)$ sets, where $1 \le i \le c_{max}$, we analyze all the possible situations:

- There is only one vertex in $C(\varphi', c'_{max})$. Then, the remaining $n-1$ vertices are distributed among the $C(\varphi', i)$ sets where $1 \le i < c'_{max}$. Therefore,

  $$EVS(\varphi', G) = n^{c'_{max}} + \sum_{i=1}^{c'_{max}-1} n^i |C(\varphi', i)| > n^{c'_{max}} \ge EVS(\varphi, G),$$

  which implies that $EVS(\varphi', G) > EVS(\varphi, G)$
- All the vertices are in $C(\varphi', c'_{max})$. Then, $|C(\varphi', i)| = 0$, where $1 \le i < c'_{max}$. Therefore,

  $$EVS(\varphi', G) = n^{c'_{max}} \times n > n^{c'_{max}} \ge EVS(\varphi, G),$$

  which implies that $EVS(\varphi', G) > EVS(\varphi, G)$
- $1 < k < n$ vertices are in $C(\varphi', c'_{max})$. Then,

  $$EVS(\varphi', G) = n^{c'_{max}} \times k + \sum_{i=1}^{c'_{max}-1} n^i |C(\varphi', i)| > n^{c'_{max}} \ge EVS(\varphi, G),$$

  which implies that $EVS(\varphi', G) > EVS(\varphi, G)$

Then, independent of the distribution of the n vertices in the C-sets, we have proved that if $VS(\varphi, G) < VS(\varphi', G)$ then $EVS(\varphi, G) < EVS(\varphi', G)$.

We then propose a neighborhood change procedure that considers the aforementioned extended objective function. In particular, Algorithm 5 compares the incumbent solution φ with other solution φ' obtained from the k-th neighborhood using the *EVS*. If φ' is better than φ, the procedure updates the incumbent solution (step 3) and the search method resorts to the first neighborhood structure (step 4). Otherwise, the method considers the next neighborhood by increasing the value of k (step 6).

Algorithm 5. Pseudocode of the neighborhood change function.

```
1:  procedure NeighborhoodChange ($\varphi, \varphi', k$)
2:  if $EVS(\varphi', G) < EVS(\varphi, G)$ then
3:      $\varphi \leftarrow \varphi'$
4:      $k \leftarrow 1$
5:  else
6:      $k \leftarrow k+1$
7:  end if
8:  end NeighborhoodChange
```

3.5. Efficient implementation of insert moves

Given a graph G and a layout φ, the computation of $VS(\varphi, G)$ requires to scan all the edges in the graph. Therefore, in a direct and straightforward implementation, the update of the objective function after performing a move is extremely time-consuming. However, it is clear that the *Sep*-values of some vertices does not change when we perform a move and therefore we do not need to re-compute it again. This idea was originally proposed in [8]. In particular, the authors proposed a move based on the interchange of two vertices in the layout. Considering the way in which the *Sep*-value is computed (difference between the label of incumbent vertex and the label of its adjacent with the largest label), if the move interchanges two vertices with labels i and j, it is only required to update the *Sep*-values from 1 to $\max\{i,j\}$.

We use in this paper an alternative definition of the *Sep*-value (see Section 1), which allows us to reduce the number of vertices that must be update. Specifically, after performing the move $Insert(\varphi, v_i, j)$, it is only required to update the *Sep*-values in

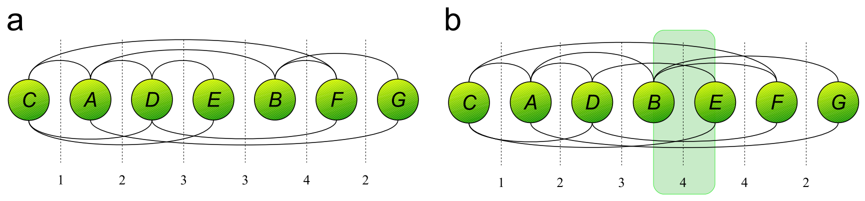


Fig. 5. Update of the objective function after a swap move.

positions k such that $i \le k < j$. As it will be shown in Section 4, this observation saves a considerable amount of computing time. We propose an additional improvement in the way that the *Insert* move is implemented. In particular, the move is decomposed as a sequence of swaps, where each swap is defined as the interchange of vertices in consecutive positions in the corresponding permutation. This move is referred to us $\varphi' = Swap(\varphi, v_i, v_{i+1})$ in such a way that $\varphi'(v_i) = \varphi(v_{i+1})$ and $\varphi'(v_{i+1}) = \varphi(v_i)$.

Let us consider, without loss of generality, that $(i<j)$. Then, the move $Insert(\varphi, v_i, j)$ can be computed as the sequence of swaps of vertices in positions $(i, i+1)$, $(i+1, i+2)$ and so forth until $(j-1, j)$. If we store these values, the computation of the move $Insert(\varphi, v_i, j+1)$ is equivalent to compute the swap move between positions $(j, j+1)$ after performing $Insert(\varphi, v_i, j)$. The situation when $i>j$ is completely equivalent.

The computation of the objective function after a swap move between the vertex v_i in position $\varphi(v_i) = i$ and the vertex v_{i+1} in position $\varphi(v_{i+1}) = i+1$ only involves the update of the *Sep*-value in position i. Fig. 5 shows an example of a swap move between vertices B and E, in positions 4 and 5, respectively. It is easy to see that the only *Sep*-value that needs to be updated is the one in position 4 (highlighted in Fig. 5b). In particular, $Sep(4, \varphi, G) = 3$ and, after performing the swap, $Sep(4, \varphi', G) = 4$, while the other *Sep*-values remain unaltered.

In addition, the value of $Sep(i, \varphi', G)$ can be incrementally computed by considering the value of $Sep(i-1, \varphi, G)$. In particular, $Sep(i, \varphi', G) = Sep(i-1, \varphi, G) + \delta^+ - \delta^-$, where δ^+ determines if the vertex in position i (previously in position $i+1$) has, at least, one adjacent in the set $R(i, \varphi, G)$. If so, the *Sep*-value is incremented by one unit. Otherwise, it holds the same value. On the other hand, δ^- refers to those vertices in $L(i, \varphi, G)$ whose adjacent vertex with the largest label is the one in position i. In mathematical terms

$$\delta^+ = \begin{cases} 1 & \text{if } |R(i, \varphi, G) \cap S(v)| > 0, \\ 0 & \text{otherwise} \end{cases}$$

$$\delta^- = \left| \left\{ v \in L(i, \varphi, G) \cap S(v) : \max_{u \in S(v)} \varphi(u) = i \right\} \right|,$$

where $S(v)$ represents the set of adjacent vertices to v. This strategy saves a considerable CPU time since it only updates one position and the computation of the *Sep*-value in that position is incrementally performed.

Considering again the example shown in Fig. 5, $\delta^+ = 1$ because the vertex B in position 4 has one or more adjacent vertices (F and G) placed in a position larger than 4. On the other hand, $\delta^- = 0$ since there are not vertices in the solution whose adjacent vertex with the largest label is B.

3.6. Local search methods

In this paper we propose two local search strategies, LS_1 and LS_2, that scan the neighborhoods N_1 and N_2, respectively. Algorithm 6 represents the pseudocode for both methods, where $LS_1 = LocalSearch(\varphi, 1)$ and $LS_2 = LocalSearch(\varphi, 2)$. The pseudocode starts by sorting the vertices in descending order of their *Sep*-value, constructing the ordered set A (step 7). Then, *LocalSearch* finds the best solution for each node of A in their corresponding neighborhood (step 9). Next, the new solution φ' is compared with the best solution so far $\varphi^\star$ considering the alternative objective function (i.e., *EVS*) in step 10, updating it if needed (step 11). The *LocalSearch* procedure performs moves while an improvement is produced (steps 4–15). Finally, the method returns the best solution found in the corresponding neighborhood.

Algorithm 6. Pseudocode of the local search procedure.

```
1:  procedure LocalSearch($\varphi, k$)
2:  $improvement \leftarrow true$
3:  $\varphi^\star \leftarrow \varphi$
4:  while $improvement$ do
5:      $improvement \leftarrow false$
6:      $\varphi \leftarrow \varphi^\star$
7:      $A \leftarrow OrderBySepValue(\varphi)$
8:      for all $v \in A$ do
9:          $\varphi' \leftarrow \arg\min_{\varphi \in N_k(v)} EVS(\varphi, G)$
10:         if $EVS(\varphi', G) < EVS(\varphi^\star, G)$ then
11:             $\varphi^\star \leftarrow \varphi'$
12:             $improvement \leftarrow true$
13:         end if
14:     end for
15: end while
16: return $\varphi^\star$
17: end LocalSearch
```

4. Computational experience

This section reports the computational experiments that we have performed for testing the efficiency of the proposed three VNS variants (RVNS, VND, and GVNS) for solving the VSP. All the algorithms were implemented in Java SE 6 and the experiments were conducted on an Intel Core i7 2600 CPU (3.4 GHz) and 4 GB RAM. We have considered three sets of instances previously used in this problem. All instances are available at http://www.optsicom.es/vsp/. A detailed description of each type of instances follows:

- HB: We derived 62 instances from the Harwell–Boeing Sparse Matrix Collection. This collection consists of a set of standard test matrices $M = M_{uv}$ arising from problems in linear systems, least squares, and eigenvalue calculations from a wide variety of scientific and engineering disciplines. The graphs are derived from these matrices by considering an edge (u, v) for every element $M_{uv} \neq 0$. From the original set we have selected the 62 graphs with $n \le 1000$. The number of vertices and edges ranges from 24 to 960 and from 34 to 3721, respectively.
- Grids: This set consists of 50 matrices constructed as the Cartesian product of two paths [26]. They are also called two-dimensional meshes and the optimal solution of the VSP for squared grids is known by construction, see [6]. Specifically, the vertex separation value of a square grid of size $\lambda \times \lambda$ is λ. For this set, the vertices are arranged on a square grid with a

dimension $\lambda \times \lambda$ for $5 \le \lambda \le 54$. The number of vertices and edges range from $5 \times 5 = 25$ to $54 \times 54 = 2916$ and from 40 to 5724, respectively.

- `Trees`: Let $T(\lambda)$ be set of trees with minimum number of nodes and vertex separation equal to λ. As it is stated in [10], there is just one tree in $T(1)$, namely the tree with a single edge, and another one in $T(2)$, the tree constructed with a new node acting as root of three subtrees that belong to $T(1)$. In general, to construct a tree with vertex separation $\lambda+1$ it is necessary to select any three members from $T(\lambda)$ and link any one node from each of these to a new node acting as the root of the new tree. The number of nodes, $n(\lambda)$, of a tree in $T(\lambda)$ can be obtained using the recurrence relation $n(\lambda) = 3n(\lambda-1)+1$ where and $n(1) = 2$ (see [10] for additional details). We consider 50 different trees: 15 trees in $T(3)$, 15 trees in $T(4)$ and 20 trees in $T(5)$. The number of vertices and edges range from 22 to 202 and from 21 to 201, respectively.

We have divided our experimentation into two different parts: preliminary experimentation and final experimentation. The preliminary experiments were performed to set the values of the key search parameters of each VNS variant as well as to show the merit of the proposed search strategies. We consider a representative subset of 23 `HB` instances, with different sizes and densities. Specifically, we consider: `494_BUS`, `662_BUS`, `685_BUS`, `BCSPWR04`, `CAN_292`, `CAN_445`, `CAN_634`, `CAN_715`, `DWT_245`, `DWT_307`, `DWT_310`, `DWT_361`, `DWT_419`, `DWT_503`, `DWT_592`, `DWT_758`, `LHSP_778`, `NOS3`, `NOS4`, `NOS5`, `NOS6`, `NOS7`, and `PLAT362`.

In our first experiment we compare the performance of the proposed constructive procedure (named as C_{greedy} and described in Section 3.1) with the best previous constructive approach (named as C2 in [8]). We generate one solution for each instance with each constructive procedure. Table 1 reports #Best, number of best solutions found in the experiment; Avg., average quality over all instances; Dev. (%), average percent deviation with respect to the best solution found in the experiment; and Time, average computing time in seconds required by the procedure. We report these statistics in the remaining experiments.

Table 1 shows that C_{greedy} clearly outperforms C2 in terms of both, number of best (23 versus 8) and average percentage deviation (0.00% versus 11.88%). Our constructive procedure needs more CPU time (0.53 versus 0.18). However, this time is negligible when considering the computing time of the whole algorithm (constructive procedure plus the corresponding VNS variant). Therefore, we consider C_{greedy} as the best constructive procedure and it will be used for the remaining experiments.

In the next experiment, we study the performance of the incremental computation of the objective function described in Section 3.5 over the instances used in the preliminary experimentation. In particular, we test whether the use of swap moves (as a way of implementing insert moves) reduces the CPU time or not. We construct a solution with C_{greedy} and then we improve it with the local search based on insert moves. In order to have a clear idea about the saving in the computing time, we compare our proposal with the one proposed in [8], and with a direct computation of the objective function. Fig. 6 depicts a diagram where the X-axis represents the set of instances considered for this experiment (ordered according to the number of vertices) and the Y-axis gives the computing time required to obtain a local optimum for the three considered methods (Insert Moves, Interchange Moves, and Direct) in the corresponding instance. The Y-axis uses a logarithmic scale to reduce the ranges to a more manageable size. The figure clearly shows that the saving in computing time is significant for the introduced method. Specifically, for these instances the proposed method is about 40 times faster than the incremental computation described in [8] and it is about 1000 times faster than the direct computation of the objective function. Notice that the

Table 1
Comparison of different constructive procedures.

	C_{greedy}	C2
#Best	23	8
Avg.	42.26	46.43
Dev. (%)	0.00	11.88
Time	0.53	0.18

Table 2
Comparison of different shake procedures.

	RVNS-1	RVNS-2	RVNS-3	RVNS-4
#Best	15	7	19	7
Avg.	37.74	42.09	37.48	42.04
Dev. (%)	2.33	16.24	1.27	16.06
Time	101.96	102.00	101.99	101.96

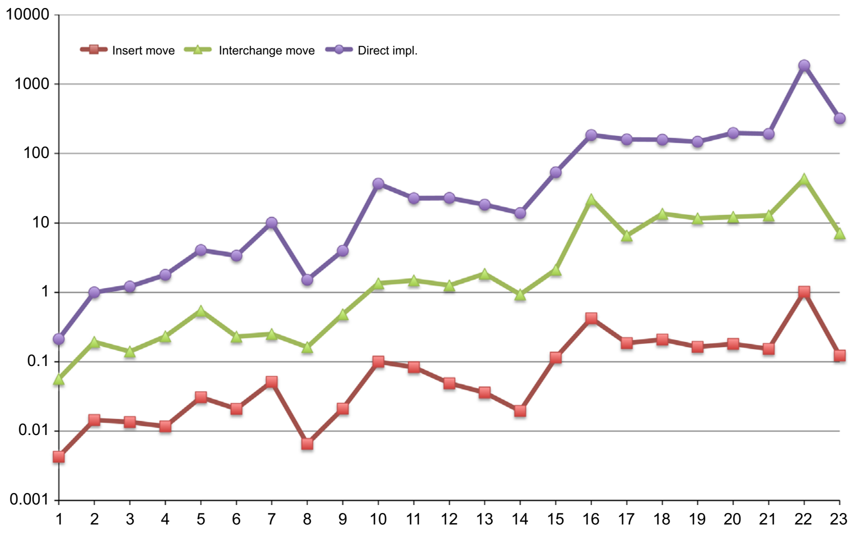


Fig. 6. Incremental computation of the objective function.

Table 3
Comparison of different neighborhoods.

	LS_1	LS_2	VND
#Best	17	6	23
Avg.	37.35	41.13	36.65
Dev. (%)	2.59	15.01	0.00
Time	3.93	0.78	6.94

incremental computation described in [8] is about 25 times faster than the direct computation.

In the next experiment, we compare the performance of 4 RVNS variants (RVNS-1, RVNS-2, RVNS-3, and RVNS-4). The only difference among them is the shake procedure (see Section 3.3) used in each one. In particular, RVNS-x considers the $Shake-x$ function, where $x \in \{1,2,3,4\}$. As it is pointed out in [15], the best outcomes in RVNS are obtained when the value of k_{max} is 2 or 3. Then, we select $k_{max}=3$ since we obtained slightly better results. Table 2 shows the results of the 4 RVNS variants when t_{max} is set to 100 s. In this time horizon, RVNS-3 emerges as the best procedure, where the shake procedure selects vertices at random that are then placed in their best position (maximum decreasing of the objective function). This strategy seems to be the most promising one since it balances diversification (random selection of vertices) with intensification (greedy selection of the best position). In fact, selecting vertices at random (RVNS-1 and RVNS-3) produces better outcomes than selecting them in a greedy fashion (RVNS-2 and RVNS-4).

In the fourth preliminary experiment, we compare the performance of the VND with the local search procedures in isolation (i.e., LS_1 and LS_2). Typically, VND explores the neighborhoods from the smallest and fastest to evaluate, to the slowest and largest one. Consequently, our VND first considers N_2 and then N_1. Notice that the neighborhoods are explored in a nested strategy. Results in Table 3 confirm that the VND procedure compares favorably with simple local search methods. Specifically, VND achieves the lowest deviation (0.00%) compared with the two local search methods tested (2.59% and 15.01% for LS1 and LS2, respectively). It is worth to mention that the results of LS2 seem to be quite bad. However, when coupling LS1 and LS2 in a VND strategy, the resulting algorithm obtains the best outcomes.

The next experiment consists of selecting the best GVNS variant by running a single full-factorial experiment. In particular, we consider the 4 shake functions described in Section 3.3 and $k_{max}=\{0.05n, 0.10n, 0.15n, 0.20n\}$, n being the number of vertices of the instance. For the sake of simplicity we do not show this experiment in a table and we only report that the best results are obtained using the $Shake-1$ procedure. This result is in line with the ones shown in Table 2. Specifically, the best strategy consists of selecting the vertices at random. However, in the GVNS context, the positions (where those vertices are placed) are selected at random. This result can be partially explained when considering that the shake procedure in GVNS mainly diversify the search and the intensification is performed by the improvement strategy (i.e., VND algorithm). As expected, the higher the value of k_{max}, the better the results are. Unfortunately, the computing time also increases with the value of k, so we set $k=0.15n$ as a compromise between computing time and quality.

We compare in the next experiment the best of each VNS variant (RVNS, VND, and GVNS) with the best previous method identified in the state of the art [8], which is also a VNS algorithm. Consequently, in order to avoid a misunderstanding with the VNS variants that we are proposing in this paper, we refer to this method as BestPrev. To provide a fair comparison, GVNS has the same stopping criterion than BestPrev (either k reaches k_{max} or the CPU time exceeds $t_{max}=1000$ s). RVNS has t_{max} as input parameter, which is set to the same CPU time than BestPrev. Finally, in the case of VND, t_{max} is not an input parameter. Therefore, to be able to execute VND for the same CPU time than the other variants, we consider a multi-start scheme, where each iteration consists of a construction (with the procedure described in Section 4) and then a VND execution. Table 4 reports the results of this experiment. This table is structured into three main rows. Each row represents the results associated to each set of instances. In particular, the first row reports the results of the four algorithms over the set `Grids`. In sight of these results we can conclude that grids instances are easily solved by the four procedures. Although the computing time seems to be relatively large, it is important to note that all the procedures find the optimum value in less than a second. Therefore, we do believe that this set of instances does not worth to be included in future comparisons. Regarding the instances of the set `Trees` (second main row), the four algorithms are able to find the optimal solution for the small and medium instances (i.e., Trees with 22 and 67 vertices). Considering the largest tree instances (with 202 vertices), the BestPrev only obtains one optimum solution, while the three proposed methods obtain, 4 (RVNS), 5 (MS_VND) and 10 (GVNS) out of 20, respectively. Although the CPU time of the four procedures is relatively small, our three methods need much less time to produce better results.

Table 4
Final comparison over the sets `Grids` (50 instances), `Trees` (50 instances), and `HB` (62 instances).

		GVNS	RVNS	MS_VND	BestPrev
`Grids`	#Opt.	50	50	50	50
	Avg.	29.5	29.5	29.5	29.5
	Dev. (%)	0.00	0.00	0.00	0.00
	Time	93.32	215.30	101.08	1422.76
`Trees`	#Opt	40	34	35	31
	Avg.	4.30	4.46	4.40	4.64
	Dev. (%)	4.00	7.00	6.00	11.00
	Time	9.07	21.10	2.98	126.73
`HB`	#Best	53	28	27	34
	Avg.	24.60	27.00	26.77	26.11
	Dev. (%)	2.07	10.85	11.91	6.90
	Time	482.08	700.03	702.89	705.60

The third main row reports the results over the most challenging set of instances (i.e., `HB`). As it was aforementioned, the optimum of these instances is not known. Moreover, the properties of each instance (density, max and min degree, etc.) enormously vary from one to another. Table 4 shows that GVNS clearly outperforms the other three methods (including the best method found in the state of the art) in average percentage deviation and number of best solutions found. Moreover, it only needs a half of the computing time used by the other three methods to produce much better results. Therefore, results reported in this table suggest the superiority of GVNS over the other three methods. Regarding RVNS and MS_VND, BestPrev apparently obtains better results. In order to confirm these hypothesis, we applied the non-parametric Friedman test [13] for multiple correlated samples to the best solutions obtained by GVNS, RVNS, MS_VND, and BestPrev. This test computes, for each instance, the rank value of each method according to solution quality (where rank 1 is assigned to the best method and rank 4 to the worst one). Then, it calculates the average rank values of each method across all the instances solved. If the averages differ greatly, the associated p-value or significance will be small. We only consider in this experiment the instances in the `HB` set, and the largest instances of the `Tree` set (trees with 202 vertices), since grids and small and medium trees are optimally solved by all the algorithms.

The resulting p-value of 0.00001 (considering a level of significance of 0.05) obtained in this experiment clearly indicates that there are statistically significant differences among the four methods tested. Specifically, the rank values produced by this test are 1.88 (GVNS), 2.61 (BestPrev), 2.75 (MS_VND), and 2.76 (RVNS). To detect the differences among BestPrev and the three proposed methods in this paper and considering the proximity in the rank between BestPrev, RVNS and MS_VND, we conduct a Wilcoxon's test. Let $R+$ be the sum of ranks for the functions on which the first algorithm outperforms the second one, and $R-$ be the sum of ranks for the opposite. Ranks corresponding to zero differences are split evenly among the sums. If $\min\{R+, R-\}$ is less than or equal to the critical value, the Wilcoxon's test detects significant differences between the algorithms, which means that an algorithm outperforms its opponent.

Table 5 summarizes the results of this statistical test with a level of significance 0.05, where the values of $R+$ (associated to one of our methods) and $R-$ (associated to BestPrev) of the test are specified. The third column reports the p-value associate to each experiment and the last column indicates whether Wilcoxon's test found statistical differences between these algorithms or not. In particular, if $\min\{R+, R-\}$ is less than or equal to the critical value [29] in this experiment, this test detects that an algorithm outperforms its opponent. In particular, if this fact occurs and, simultaneously, $R- = \min\{R+, R-\}$, then our method is better than BestPrev. However, the confidence of the test is always determined by the p-value.

These results complement the ones reported in Table 4. Specifically, the Wilcoxon's test again confirm the superiority of GVNS over BestPrev. Additionally, this test shows that there is no significant statistical differences between RVNS versus BestPrev and MS_VND versus BestPrev. Therefore, even considering the results reported in Tables 4 and 5, it is not possible to say that BestPrev is statistically better than either RVNS or MS_VND.

In the final experiment we explore the behavior of these methods over a long-term time horizon, we run GVNS, RVNS, MS_VND and BestPrev for 30 min, reporting every 30 s the average deviation of the best solution found. We select the ten largest instances in HB to illustrate the performance of the four procedures over the hardest instances. Fig. 7 shows the corresponding average time profile, where it is represented with a dash line the best results found in the state of the art. We can observe that GVNS consistently produces the best results, from the very beginning to the end of the experiment. It is important to remark that GVNS produces high quality solutions in the short-term horizon (outperforming the state of the art in the first 30 s) and in the long-term horizon (producing considerable improvements about 1500 s). This fact shows that the intensification stage (VND procedure) produces good solutions in short CPU time, but also the diversification stage (shake procedure) is able to find promising regions in the search space.

MS_VND presents competitive results only in the short-time horizon. This fact can be partially explained by considering that the constructive procedure is not designed to perform multiple constructions. Consequently, it produces high quality but not diverse solutions. The improvement of the RVNS algorithm is constant during the 30 min. As it was pointed out above, the shake procedure embedded in this algorithm presents a balance between diversification (selecting vertices at random) and intensification (inserting them in the best possible position). This strategy seems to be successful over the whole experiment. In fact, it is expected that the RVNS would outperform MS_VND in larger computing times. It is worth to mention that the superiority of GVNS over the remaining methods comes from the fact that GVNS behaves like MS_VND in short CPU times, while it behaves like RVNS in larger computing times.

Table 5
Wilcoxon's test results.

	$R-$	$R+$	p-value	Significant
GVNS versus BestPrev	134.5	900.5	0.00	Yes
RVNS versus BestPrev	520.5	299.5	0.14	No
MS_VND versus BestPrev	684.5	491.5	0.32	No

Table 6
Friedman's test results over the ten largest instances.

	GVNS	RVNS	MS_VND	BestPrev	p-value
1 min	1.90	3.15	1.90	3.05	0.034
10 min	1.80	2.90	2.10	3.20	0.049
20 min	1.75	2.80	2.20	3.25	0.048
30 min	1.50	2.60	2.45	3.45	0.009

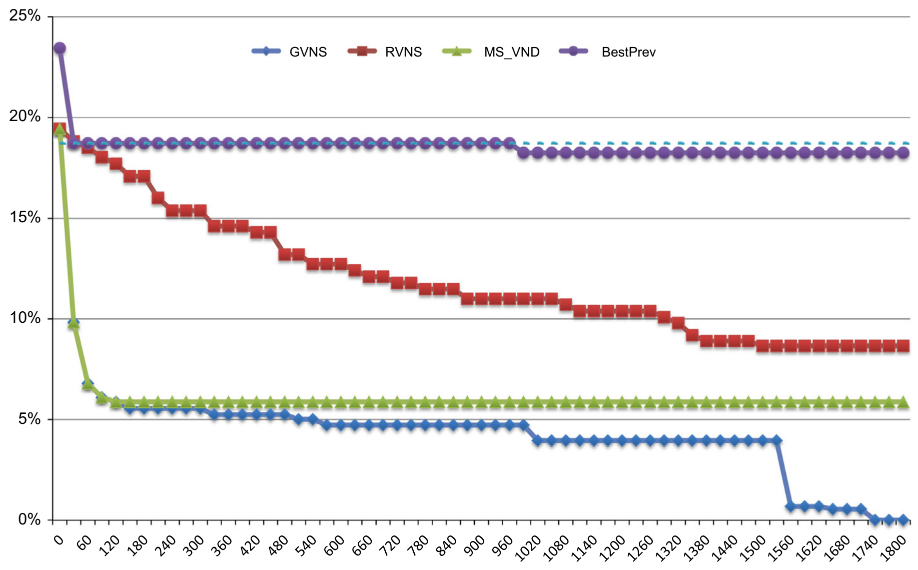


Fig. 7. Search profile for a 30 min run on the largest instances.

Analyzing the results obtained by BestPrev, we can determine that this procedure is not suitable when facing large instances. The algorithm starts improving at the very beginning (60 s). After that the method gets stuck until reaching 1000 s, mainly because the local search method is slower than the ones proposed in this paper. As a consequence, this method presents the worst performance (at least in these instances).

Finally, we analyze the results of this experiment by considering the Friedman's test. Specifically, we apply a Friedman's test in four different time stamps: 1 min, 10 min, 20 min, and 30 min. Table 6 shows the rankings when considering the 10 largest (and hardest) instances of the HB set. The resulting p -value in all the considered time stamps indicates that the differences are statistically significant. Specifically, for 1 min GVNS and MS_VND emerge as the best methods (with a ranking 1.90), followed by BestPrev (3.05) and RVNS (3.15). When considering the results after 10 and 20 min, the GVNS improves its ranking as well as RVNS. It is interesting to mention that MS_VND and BestPrev seem to be stuck so their ranking get worse. Finally, after 30 min, the GVNS even increases more the difference with respect to the other algorithms (1.50). MS_VND holds the second position according to the ranking value (2.45), but closely followed by the RVNS (third method with ranking 2.60). Finally, BestPrev systematically produces the worst results, which led it to the last position of the ranking with a value of 3.45.

5. Conclusions

In this paper, we propose different methods based on the Variable Neighborhood Search methodology to deal with the Vertex Separation problem. We provide an extensive experimental comparison among them and with best previous method in the state of the art. In more detail, we propose a constructive procedure, four shake methods (with different balance between intensification and diversification), two new neighborhood structures (and efficient strategies to explore them), and an extended version of the objective function for the considered problem which allows the comparison of solutions beyond the original objective function. We embed these strategies in a Reduced Variable Neighborhood Search (RVNS), a Variable Neighborhood Descent (VND) and a General Variable Neighborhood Search (GVNS). We performed an extensive computational testing over a set of 162 instances, considering different time horizons. Experimental results show that the proposed algorithms outperform the best method identified in the state of the art. We also proved statistical tests to confirm the significance of the obtained results, and GVNS emerges as the best algorithm in terms of quality in any set of instances and time horizon.

According to our experimentation, the Grids and Trees instances can be considered “easy to solve”. In our opinion, these instances should no longer be considered for future studies since they do not allow us to evaluate the actual performance of the compared methods. On the contrary, the Harwell–Boeing instances are actually a real challenge for modern heuristic methods. Therefore, we recommend this set of instances to be used as the benchmark for further experimental studies.

Of particular interest in our work has been testing the combination of diversification and intensification strategies in the context of VNS. Through extensive experimentation, we have been able to determine the benefits of this combination. We purposefully added these mechanisms in order to measure their effects and studied the combinations that resulted in effective solution procedures with improved outcomes. We believe that our findings can be translated to other combinatorial problems and it will help in the development of more elaborated VNS methods.

Acknowledgments

This research has been partially supported by the Spanish Ministry of “Economía y Competitividad”, Grants ref. TIN2011-28151, TIN2012-35632-C02, and the Government of the Community of Madrid, Grant ref. S2009/TIC-1542.

References

[1] Bodlaender HL, Möhring RH. The pathwidth and treewidth of cographs. In: Proceedings of the second Scandinavian workshop on algorithm theory. SWAT'90; 1990. p. 301–9.
[2] Bodlaender HL, Kloks T, Kratsch D. Treewidth and pathwidth of permutation graphs. SIAM J Discrete Math 1995;8(4):606–16.
[3] Bodlaender HL, Gilbert JR, Hafsteinsson H, Kloks T. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. J Algorithms 1995;18(2):238–55.
[4] Bodlaender HL, Gustedt J, Telle JA. Linear-time register allocation for a fixed number of registers. In: Proceedings of the symposium on discrete algorithms; 1998. p. 574–83.
[5] Bollobás B, Leader I. Edge-isoperimetric inequalities in the grid. Combinatorica 1991;11:299–314.
[6] Díaz J, Petit J, Serna M. A survey of graph layout problems. ACM Comput Surv 2002;34(3):313–56.
[7] Duarte A, Martí R, Resende MGC, Silva RMA. GRASP with path relinking heuristics for the antibandwidth problem. Networks 2011;58(3):171–89.
[8] Duarte A, Escudero LF, Mart R, Mladenovic N, Pantrigo JJ, Snchez-Oro J. Variable neighborhood search for the vertex separation problem. Comput Oper Res 2012;39:3247–55.
[9] Dujmović V, Fellows MR, Kitching M, Liotta G, Mccartin K, Nishimura N, et al. On the parameterized complexity of layered graph drawing. Algorithmica 2008;52(2):267–92.
[10] Ellis JA, Sudborough IH, Turner JS. The vertex separation and search number of a graph. J Inf Comput 1994;113:50–79.
[11] Ellis JA, Markov M. Computing the vertex separation of unicyclic graphs. Inf Comput 2004;192(2):123–61.
[12] Fellows MR, Langston MA. On search, decision and the efficiency of polynomial-time algorithms. J Comput Syst Sci 1994;49(3):769–79.
[13] Friedman M. A comparison of alternative tests of significance for the problem of m rankings. Ann Math Stat 1940;11:86–92.
[14] Hansen P, Mladenovic N, Moreno JA. Variable neighbourhood search: methods and applications. Ann Oper Res 2010;175(1):367–407.
[15] Hansen P, Mladenovic N, Brimberg J, Moreno-Pérez JA. Variable neighbourhood search. Handbook of Metaheuristics 2010;146:61–86.
[16] Aleksandar Ilić, Dragan Urosšvic`, Jack Brimberg, Nenad Mladenovic. A general variable neighborhood search for solving the uncapacitated single allocation p-hub median problem. Eur J Oper Res 2010;206(2):289–300.
[17] Kinnersley NG. The vertex separation number of a graph equals its path-width. Inf Process Lett 1992;42(6):345–50.
[18] Kirousis M, Papadimitriou CH. Interval graphs and searching. Discrete Math 1985;55(2):181–4.
[19] Kirousis M, Papadimitriou CH. Searching and pebbling. Theory Comput Sci 1986;47(2):205–18.
[20] Leiserson CE. Area-efficient graph layouts (for VLSI). In: Proceedings of the IEEE symposium on foundations of computer science; 1980. p. 270–81.
[21] Lewis JG. The Gibbs–Poole–Stockmeyer and Gibbs–King algorithms for reordering sparse matrices. ACM Trans Math Software 1982;8:190–4.
[22] Mladenović N, Hansen P. Variable neighborhood search. Comput Oper Res 1997;24:1097–100.
[23] Pantrigo JJ, Martí R, Duarte A, Pardo EG. Scatter search for the cutwidth minimization problem. Ann Oper Res 2012;199:285–304.
[24] Peng SL, Ho C-W, Hsu TS, Ko MT, Tang CY. A linear-time algorithm for constructing an optimal node-search strategy of a tree. In: Proceedings of the 4th annual international conference on computing and combinatorics, COCOON '98; 1998. p. 279–88.
[25] Pi nana E, Plana I, Campos V, Martí R. GRASP and Path relinking for the matrix bandwidth minimization. J Oper Res 2004;153:200–10.
[26] Raspaud A, Schröder H, Sýkora O, Török L, Vrt'o I. Antibandwidth and cyclic antibandwidth of meshes and hypercubes. Discrete Math 2009;309:3541–52.
[27] Rodriguez-Tello E, Jin-Kao H, Torres-Jimenez J. An effective two-stage simulated annealing algorithm for the minimum linear arrangement problem. Comput Oper Res 2008;35(10):3331–46.
[28] Skodinis K. Computing optimal strategies for trees in linear time. In: Proceedings of the 8th annual European symposium on algorithms; 2000. p. 403–14.
[29] Wilcoxon F. Individual comparisons by ranking methods. Biometrics 1945;1:571–95.

Chapter 10

Parallel VNS strategies for the Cutwidth Minimization Problem

- 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):55–73, 2016.
  - DOI: `https://doi.org/10.1093/imaman/dpt026`
  - Impact Factor (JCR 2015): 0.878

Subject Category	**Ranking**	**Quartile**
Operations Research and Management Science	59 / 82	Q3
Mathematics, interdisciplinary applications	65 / 101	Q3

IMA Journal of Management Mathematics Advance Access published December 11, 2013

IMA Journal of Management Mathematics (2013) Page 1 of 19
doi:10.1093/imaman/dpt026

Parallel variable neighbourhood search strategies for the cutwidth minimization problem

Abraham Duarte*, Juan J. Pantrigo, Eduardo G. Pardo and Jesús Sánchez-Oro
Dept. Ciencias de la Computación, Universidad Rey Juan Carlos, Móstoles, Spain
*Corresponding author: abraham.duarte@urjc.es juanjose.pantrigo@urjc.es
eduardo.pardo@urjc.es jesus.sanchezoro@urjc.es

[Received on 31 January 2013; accepted on 17 November 2013]

Variable neighbourhood search (VNS) and all its variants have been successfully proved in hard combinatorial optimization problems. However, there are only few works concerning parallel VNS algorithms, compared with the amount of works devoted to sequential VNS design. In this paper, we propose different parallel designs for the VNS schema. We illustrate the performance of these general strategies by parallelizing a new VNS variant called variable formulation search (VFS). Specifically, we propose six different variants which differ in the VNS stages to be parallelized as well as in the communication mechanisms among processes. We group these variants into three different templates. The first one is oriented to parallelize the whole VNS method. The second one parallelizes the shake and the local search procedures. Finally, the third one explores in parallel the set of predefined neighbourhoods. We test the resulting designs on the cutwidth minimization problem (CMP). Experimental results show that the parallel implementation of the VFS outperforms previous methods in the state of the art for the CMP. This fact is also confirmed by non-parametric statistical tests.

Keywords: variable neighbourhood search; variable formulation search; parallel designs; cutwidth minimization problem.

1. Introduction

Metaheuristics (MHs) are among the most prominent and successful techniques to solve a large amount of complex and computationally hard combinatorial and numerical optimization problems arising in human activities, such as economics (e.g. portfolio selection), industry (e.g. scheduling or logistics) or engineering (e.g. routing), among many others (Gendreau & Potvin, 2010). MHs can be seen as general algorithmic frameworks that require relatively few modifications to be adapted to tackle a specific problem. They constitute a very diverse family of optimization algorithms including methods such as simulated annealing, tabu search, multi-start methods, iterated local search, variable neighbourhood search (VNS), greedy randomized adaptive search procedure (GRASP), memetic algorithms, scatter search, evolutionary algorithms or ant colony optimization.

The VNS and all its variants have proved to be very successful in hard optimization problems. This MH was originally proposed by Mladenović & Hansen (1997). It is based on the exploration of a dynamic neighbourhood model. Contrary to other trajectory-based MHs, VNS allows changes of the neighbourhood structure along the search. In particular, VNS explores increasingly neighbourhoods of the current best found solution. The basic idea of the VNS is to change the neighbourhood structure when the local search is trapped on a local optimum.

Considering that VNS is a relatively new MH, it has not yet been investigated much from a parallelization point of view. As per the authors' knowledge, there are only two relevant papers reported in the literature on the parallelization of VNS. Specifically, García-López *et al.* (2002) proposed three new parallel VNS procedures to deal with the p-median problem: synchronous parallel VNS (SP-VNS), replicated parallel VNS (RP-VNS) and replicated shaking VNS (RS-VNS). The main goal of the first method, SP-VNS, is to parallelize the local search strategy since, in general, it is the most time-consuming part. In SP-VNS, the neighbourhood is divided into n subsets, where n indicates the number of processors. Each subset is assigned to a different processor. Then, each processor returns an improved neighbour solution in its partition. Finally, the best neighbour is selected as the current solution to continue the search. The second approach, called RP-VNS, is directly a parallel independent multi-start method, which executes several independent VNS procedures. Specifically, RP-VNS is a multi-start procedure where each local search is replaced by a VNS. Each available processor, then, performs a sequential VNS. The parallel algorithm returns the best solution obtained by the processors. Finally, in the third parallel algorithm, RS-VNS, the shake and local search procedures are replicated as many times as the number of available processors. If the best local optimum found by the processors improves the previous solution (before executing the shake and improve methods), the RS-VNS procedure resorts to the first neighbourhood and continues the search. Otherwise, RS-VNS explores a larger neighbourhood. These three methods were tested on large instances derived from the travelling salesman problem library (TSPLIB) (Reinelt, 1991). In particular, the authors considered the instance RL1400 (with 1400 points) and derived nine different instances for the p-median problem where p ranges from 20 to 100. Reported results showed that the multi-start scheme (RP-VNS) obtained the best results.

Crainic *et al.* (2004) presented a different variant of parallel VNS, called cooperative neighbourhood VNS (CN-VNS). In this strategy, each individual VNS process (executed in a different processor) communicates exclusively with a central process called central memory or master. Therefore, there are no communications among individual VNS processes. The master is responsible for maintaining, updating and communicating the current overall best solution. The master also initiates and terminates the algorithm. Crainic *et al.* (2004) applied the CN-VNS method to the p-median problem, where each process (processor) implements the same VNS variant. The local search follows the first improvement strategy, implements fast interchange and considers $k_{\max} = p$, where $k_{\max}$ represents the maximum number of neighbourhoods that will be explored. Each processor, then, executes a 'normal' VNS exploration (shake, improve and neighbourhood change) while improving the current solution. When that solution is not improved, the corresponding processor communicates it to the master, requesting the best solution so far. The search is continued starting from the best overall solution in the current neighbourhood. The authors tested the CN-VNS method over a benchmark of problem instances from TSPLIB, where the number of customers ranges from 1400 to 11948 and the number of locations ranges from 10 to 1000. Reported results showed that the CN-VNS method reduces the CPU time without deteriorating the quality (when compared with the sequential VNS version). In addition, when both methods (parallel and sequential) are executed for the same CPU time, the parallel version finds better solutions.

Moreno *et al.* (2004) surveyed the aforementioned parallel VNS strategies. The authors analysed and tested them by considering large instances of the p-median problem. The paper concluded that cooperative mechanisms obtain, in general, better outcomes.

We propose in this paper six different variants of VNS which differ in how each one parallelizes the VNS stages. We group them into three different templates. The first one is oriented to parallelize the whole VNS method. The second one parallelizes the shake and the local search procedures. Finally, the third one explores in parallel the set of predefined neighbourhoods. We illustrate the performance of these strategies by considering a specific VNS variant, called variable formulation search (VFS) (Pardo *et al.*, 2013), applied to the cutwidth minimization problem (CMP). It is important to remark that the proposed methods are general parallelization strategies in the context of VNS, since they do not consider either the optimization problem or the VNS variant.

The rest of the paper is organized as follows. Section 2 describes the CMP and presents a brief survey of the main contributions to this problem. The sequential VNS method is briefly described in Section 3. Section 4 recalls the main parallel technologies, mainly focusing on Java threads. Section 5 presents the adaptation of the previous parallel VNS strategies as well as the new proposed methods. Section 6 presents and analyses the results of the computational experiments. Conclusions and perspectives are the subject of Section 7.

2. Cutwidth minimization problem

The CMP is an NP-Hard (Gavril, 1977) min–max layout problem. It consists of finding an ordering of the vertices of a graph on a line, in such a way that the maximum number of edges between each pair of consecutive vertices is minimized. The CMP has been formulated in both, combinatorial (Petit, 2003) and mathematical programming (Luttamaguzi *et al.*, 1977) ways. In this work, we consider the first of them. Given a graph $G = (V, E)$ with $|V| = n$, the CMP can be defined as follows. Let s be a labelling $s : V \to \{1, 2, \dots, n\}$ of G that assigns the integers $\{1, 2, \dots, n\}$ to the vertices in V, in such a way that each vertex receives a different label. The cutwidth of a vertex v with respect to s, denoted as $\mathrm{CW}_s(v)$, is the number of edges $(u, w) \in E$ satisfying $s(u) \leqslant s(v) < s(w)$, i.e.

$$\mathrm{CW}_s(v) = |\{(u, w) \in E : s(u) \leqslant s(v) < s(w)\}|.$$

The cutwidth of G with respect to s is defined as the maximum value of $\mathrm{CW}_s(v)$ for all $v \in V$. In mathematical terms, the objective function of the CMP is defined as

$$f(s) = \max_{v \in V} \mathrm{CW}_s(v).$$

The optimum cutwidth of G is then defined as the minimum $f(s)$ value over all possible labellings of G.

The CMP has also been referred to in the literature using alternative names such as Minimum Cut Linear Arrangement (Diaz *et al.*, 1997; Takagi & Takagi, 1999) or Network Migration Scheduling (Andrade & Resende, 2007a,b). There can be also found a generalization of the CMP for hypergraphs named Board Permutation Problem (Cohoon & Sahni, 1983, 1987). Several practical applications of this problem in different areas of Engineering and Computer Science, such as: Circuit Design (Adolphson & Hu, 1973; Cohoon & Sahni, 1987; Makedon & Sudborough, 1989), Network Reliability (Karger, 1999), Information Retrieval (Botafogo, 1993), Automatic Graph Drawing (Mutzel, 1995; Shahrokhi *et al.*, 2001), Protein Engineering or Networks Migration (Resende & Andrade, 2009) have also been reported. This last application refers to the problem where inter-nodal traffic from an obsolete telecommunications network needs to be migrated to a new network. This problem is now happening in phone traffic, where a migration between 4ESS switch-based networks to IP router-based networks (Resende & Andrade, 2009) is performed. Nodes are migrated, one at each time period, from the old to the new network. All traffic originating or terminating at a given node in the old network is moved to a specific node in the new network.

Let us consider this application in more detail as well as how it is related to the CMP. Suppose that the traffic from/to a node v_{old} in the old network is migrated to a node v_{new} in the new network. Let c_{old} be the capacity of the edge $(v_{\text{old}}, v_{\text{new}})$. The traffic between node v_{old} and node v_{new} must use a temporary link $(v_{\text{old}}, v_{\text{new}})$ connecting the two nodes with capacity c_{old}. When a node is migrated, one or more temporary links may need to be added, since v_{old} may be adjacent to more than one node still

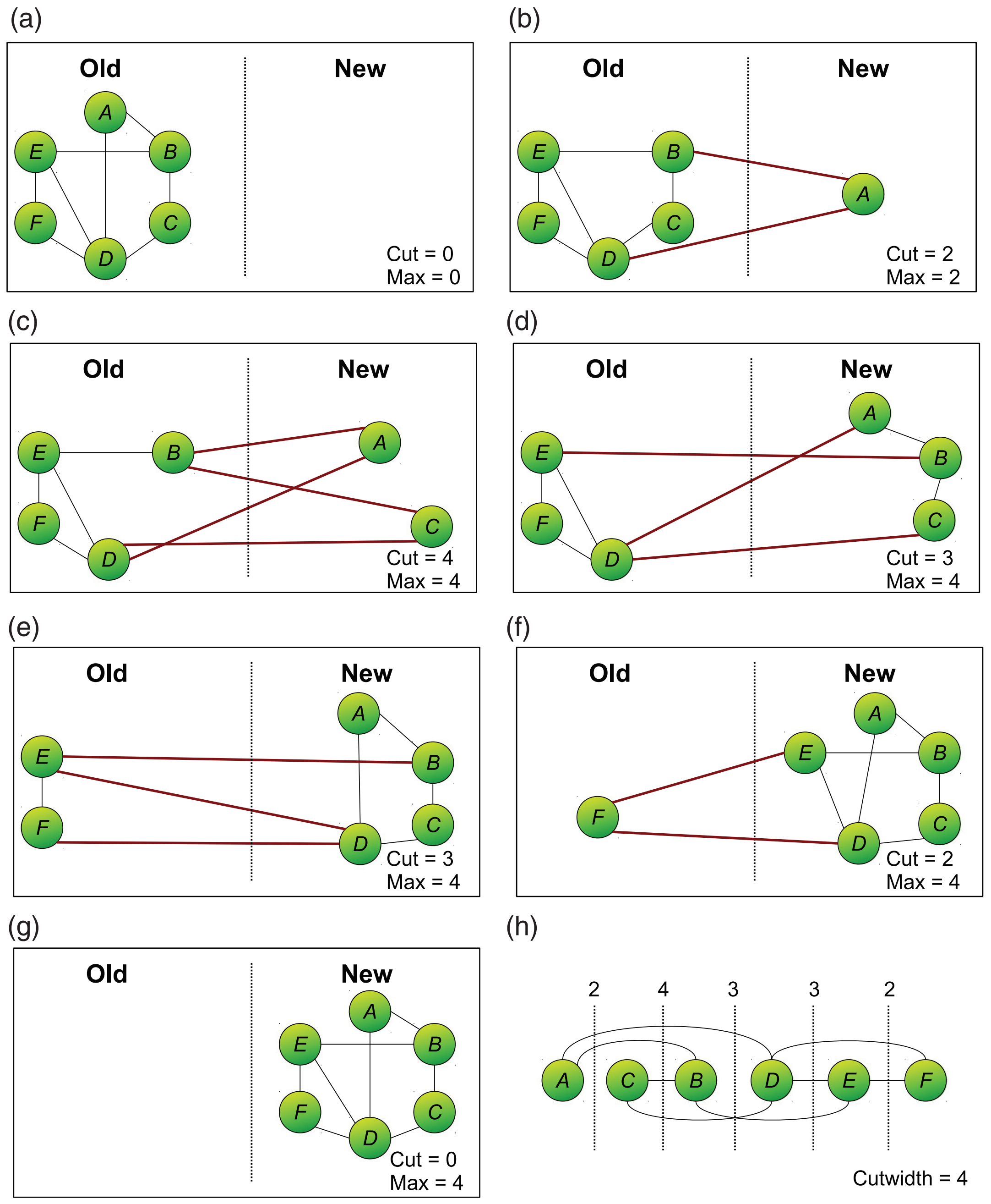


FIG. 1. (a)–(g) Node decommissioning process in a Network Migration Scheduling example and (h) the representation of the whole process as a linear layout.

active in the old network. A temporary link remains active until both nodes connected by the links are migrated to the new network.

The network is usually modelled as a graph $G = (V, E)$, where V is the vertex set, with $|V| = n$, and E is the edge set, with $|E| = m$. A solution to the Network Migration Scheduling Problem is an ordering (labelling, permutation, layout, etc.) of the nodes that need to be migrated. The CMP tries to find the ordering that minimizes the maximum sum of capacities of the temporary links (edges). Figure 1(a–g) iteratively show the migration of a network. In particular, in Fig. 1(a) all the vertices are in the old network. Figure 1(b) shows the migration of the first vertex. In this case, we need to consider the inclusion of two temporary edges. Figure 1(c) shows the migration of the second vertex, which increases the maximum number of temporary edges to 4. Figure 1(g) shows the situation where all vertices have been migrated. Finally, Fig. 1(h) shows a solution of the CMP where the cut between each pair of edges represents the situation illustrated in Fig. 1(a–h). Note that the solution depicted in Fig. 1(h) is just a possible solution, but not the optimum one, since it is possible to find a different ordering (e.g. A, B, C, D, E, F) with a cutwidth value equal to 3.

In the scientific literature, the CMP has been addressed from both exact and heuristic approaches. Most of the exact approaches present polynomial-time algorithms for particular graphs, such as hypercubes (Harper, 1966), trees (Chung *et al.*, 1982; Yannakakis, 1985; Thilikos *et al.*, 2005) or grids (Rolim *et al.*, 1995). However, little work has been devoted to devise exact methods for general graphs. As far as the authors know, there are an Integer Linear Programming formulation (Luttamaguzi *et al.*, 1977) and two Branch and Bound algorithms (Palubeckis & Rubliauskas, 2012; Martí *et al.*, 2013) proposed in the literature for the CMP. These approaches can only solve relatively small instances. Therefore, different heuristic procedures have been proposed to address this problem. The work by Cohoon & Sahni (1987) was the first one in proposing heuristic algorithms for the generalized version of the problem. These authors proposed several constructive and local search procedures and embedded them in a Simulated Annealing MH. Twenty years later, Andrade & Resende (2007a,b) proposed a GRASP procedure hybridized with a Path Relinking method. Pantrigo *et al.* (2012) introduced a Scatter Search algorithm for the problem which outperformed previous results. The most recent approach (Pardo *et al.*, 2013) presented a new variant of the VNS schema, called VFS. This new algorithm is specially useful to deal with min–max (or max–min) problems, where it is usual that many solutions of the problem have associated the same value of the objective function. VFS makes use of alternative formulations of the problem to determine which solution is more promising when they have the same value of the objective function in the original formulation. The obtained results in the CMP show that the latter proposal outperforms the state-of-the-art algorithms in terms of quality and computing time. We propose in this paper six different strategies (three of them are adaptations of previous strategies to the CMP) to parallelize the VFS. The next section details the strategies to be parallelized in this work.

3. Variable formulation search

VFS is a new variant of the VNS MH proposed in Pardo *et al.* (2013). We refer the reader to this reference for a comprehensive description of the method. VFS tries to avoid getting trapped in a local optimum by considering different formulations of the optimization problem. This idea was originally introduced in Mladenović *et al.* (2005) in the context of the circle packing problem. The method changes formulations sequentially within the shaking, local search and neighbourhood change steps of the basic VNS. The aim is to determine whether a given solution is more promising than the other to continue the search, beyond the value of the original objective function. VFS is especially helpful in tackling problems which present a flat landscape where many solutions have associated the same value of the objective function. See Resende *et al.* (2010) and Duarte *et al.* (2011) for examples of other problems which present a flat landscape. The rest of this section is devoted to describing the method to construct the initial solution, the shake, local search and neighbourhood change procedures.

3.1 *Initial solution*

The initial solution is constructed with a GRASP procedure. In particular, this method labels the vertices sequentially (i.e. from 1 to n). To select the next vertex to be labelled, a candidate list (CL) is formed with all the unlabelled vertices that are adjacent to one or more vertices already labelled. Note that, in the first iteration, all vertices are candidates. For each vertex in the CL, an evaluation of its cutwidth is performed, considering that it is labelled with the next available label. Then, a restricted CL (RCL) is constructed with the vertices in the CL with a cutwidth value lower than a given threshold. Finally, a vertex is selected at random from the RCL and assigned the corresponding label. We used this procedure to construct a predefined number of solutions, selecting the best one as the initial solution for the VFS.

3.2 *Shake*

The shake strategy consists of performing perturbations in the current solution to diversify the search. Therefore, the local search (within VFS) starts the search from a new point with a different neighbourhood. The shake procedure receives a solution s and a parameter k representing the size of the perturbation. In particular, this procedure performs k interchange moves in s. Given a solution s and two different vertices $u, v \in V$, an interchange move produces a new solution s' where $s'(u) = s(v)$, $s'(v) = s(u)$ and $s'(w) = s(w)$ for all $w \in V$ not equal to u or v. Finally, the procedure finishes when k moves are performed, returning the corresponding perturbed solution.

3.3 *Local search*

The local search procedure allows one to reach a local optimum from the current (perturbed) solution at each iteration of the VFS. We designed a local search procedure based on insertion moves. Given a solution s, a vertex v placed in the position $s(v)$ and a position j such that $s(v) \neq j$, an insertion move consists of removing v from its current position $s(v)$ and inserting it in position j. This operation results in the ordering s', as follows:

- Let v_j be the vertex in position j in the ordering s. Then, if $s(v) = i$ is larger than j, then v is inserted just before v_j in position j. For example, considering the solution s

  $$s = (\ldots, v_{(j-1)}, v_j, v_{(j+1)}, \ldots, v_{(i-1)}, v, v_{(i+1)}, \ldots),$$

  we would obtain the solution s':

  $$s' = (\ldots, v_{(j-1)}, v, v_j, v_{(j+1)}, \ldots, v_{(i-1)}, v_{(i+1)}, \ldots).$$

- If $s(v) = i$ is smaller than j, v is inserted just after v_j in position j. Therefore, from the solution s:

  $$s = (\ldots, v_{(i-1)}, v, v_{(i+1)}, \ldots, v_{(j-1)}, v_j, v_{(j+1)}, \ldots),$$

  we would obtain the solution s':

  $$s' = (\ldots, v_{(i-1)}, v_{(i+1)}, \ldots, v_{(j-1)}, v_j, v, v_{(j+1)}, \ldots).$$

A naive idea using this kind of moves would be to try the insertion of any vertex in every position of the ordering. Therefore, the associated neighbourhood of this kind of moves has a relatively large size. However, taking into account the characteristics of the considered problem, it is possible to identify a set of preferred positions for each vertex in the permutation. We refer the reader to Pardo *et al.* (2013) for an exhaustive description and formal derivation of these properties. As a consequence, the complexity of the local search is reduced from $O(n^2)$ to $O(n)$.

The resulting local search receives an initial solution, s, as an input parameter. Then, the procedure identifies the list C of vertices able to produce improvements if they are inserted in a different position of the ordering. The list C is then sorted according to the cutwidth of each vertex, in such a way that those vertices with a higher cutwidth value are evaluated earlier. The evaluation for each vertex v in C is then performed taking into account the aforementioned properties. If the move is accepted, the best solution found will be updated. Finally, the local search procedure ends when none of the moves performed with the vertices in the list C is able to produce an improvement, returning the best solution found.

3.4 *Neighbourhood change*

This procedure examines whether the solution obtained after the local search is better than any other solution previously found, or not. If so, the best solution found is updated and k (the value that determines which neighbourhood is explored) is set again to its initial value. Otherwise, k is increased with a constant value until the parameter $k_{\max}$ is reached, when a whole iteration of the VFS ends. As previously mentioned, the CMP presents a flat landscape and so, when two solutions have the same objective function value, it is hard to determine which one is better to carry on the search. For that reason, the VFS schema uses alternative formulations to compare two solutions, when it is not possible to determine which is the most promising one. In particular, the method first considers the original formulation of the problem. When two solutions present the same value of the objective function, the method uses an alternative formulation. Note that two formulations are equivalent when an optimum solution in one of them is also optimum in the other, although the value of the alternative objective function may have a different value. This schema can be repeated as far as new formulations of the problem are available. We refer the reader to Pardo *et al.* (2013) for a detailed description of the definition and use of alternative formulations.

4. Parallel technologies

The traditional Flynn classification of parallel architectures is based on two criteria: the number of instruction streams and the number of data streams that define four different classes: single instruction stream, single data stream (SISD); single instruction stream, multiple data streams (SIMD); multiple instruction streams, single data stream (MISD) and multiple instruction streams, multiple data streams (MIMD). Among these four models, MIMD is considered the most general model of parallel architectures (Alba & Nebro, 2005). Therefore, we consider MIMD architecture. This model has two kinds of memory models: shared memory (the whole memory can be accessed by each process) and distributed memory (each process has its own memory). Shared memory is considered the natural extension of the sequential programming. In fact, its foundations were established in the late 1960s to early 1970s (Alba & Nebro, 2005), so they are well known. In addition, common computers with several cores (processors) use a shared-memory model. Consequently, the easiest way of dealing with parallelism is the shared-memory model under the MIMD architecture. In this context, there are several technologies that can be used to implement parallel algorithms: OpenMP and threads (Pthreads and Java threads).

4.1 *OpenMP*

OpenMP is a set of compiler directives and library routines that can be used to implement parallel algorithms. The programmer then adds these compiler directives to a sequential program in order to inform the compiler which part of the code must be concurrently executed. It is also possible to establish (if needed) synchronization points. The main advantage of this technology is the simplicity of its implementation. In other words, transforming a sequential algorithm to the parallel version can be performed by including only one compiler directive, without modifying the original sequential code. Therefore, OpenMP is adequate if the algorithm follows a data parallel model. However, it can be difficult to use in task parallel applications (i.e. not all the processes execute the same code).

4.2 *Threads*

In programming languages, a thread is an independent flow of control inside a process. Although it has always been possible to write parallel programs using processes and other resources provided by the operating system, multi-threaded processes are themselves concurrent programs that bring a number of advantages over multiple processes. In particular, they provide faster context switching among threads and lower resource usage.

Inside this paradigm, Pthreads and Java threads are considered the most representative tools. Pthreads (POSIX threads) was defined in the mid 1990s as an effort to provide a unified set of C library routines in order to make multi-threaded programs portable. Java threads are a version of Pthreads for Java programming language.

Java threads offer the advantages of portability inherent in Java programs. It also provides a multi-threaded programming model adapted to the object-oriented features of Java. In addition, Java threads can be easily used to tackle task parallel applications. Therefore, we select this technology to implement our parallel algorithms.

5. Parallel VNS

The application of the parallelism to an MH can and must allow reducing the computational time (obtaining similar results to the sequential version) or increasing the exploration in the search space (obtaining better results than the sequential version). Notwithstanding, designing parallel MHs involves a considerable complexity since doing it appropriately implies that the researches must have a solid background in both fields. In general, these two fields are generally populated by distinct and very specialized groups of people. However, the rapid development of technology in designing processors (multi-core processors or dedicated architectures) has made use of parallel computing more and more popular.

According to Crainic *et al.* (2004), parallel MH strategies may be classified into one of the three following categories:

- *Low-level parallelism*: This strategy aims mainly at speeding up computations by executing in parallel one or several computing-intensive tasks (for instance, the local search) within one iteration of the method. It is usually implemented following the master–slave computing model. In particular, the master process dispatches work to the other processors (the ‘slaves’), recuperates and fuses the results, and then continues the sequential algorithm. Variants of this approach may be defined according to the quantity of work assigned to slave processors.
- *Domain decomposition*: The partitioning reduces the size of the solution space, but the procedure need to be repeated with different partitions to allow the exploration of the complete solution space. A master–slave scheme is often chosen as the implementation mechanism. A master process performs the decomposition and slave processors concurrently execute the MH on the resulting

sub-problems. Then, the master collects the partial solutions and builds a complete one. Finally, it decides on a new partition and the search is restarted.

- *Multiple search*: Parallelism is obtained from multiple concurrent explorations of the solution space. Concurrent searches may or may not execute the same heuristic method, and may start from the same or from different initial solutions. They may communicate during the search or only at the end to identify the best overall solution. The latter strategies are known as independent search methods, while the former are often called cooperative multi-search methods. Communications may be performed synchronously or asynchronously and may be event-driven or executed at predetermined or dynamically decided moments.

In this paper, we adapt three previously proposed VNS parallel strategies for the CMP. We additionally propose three new parallel strategies for this optimization problem. In particular, the six parallel VNS strategies fall in the 'Multiple Search' category. We do not propose low-level parallelism strategies since the shake, neighbourhood change and local search strategies are extremely fast (Pardo *et al.*, 2013). In the same line, we do not propose a domain decomposition parallel strategy since the shake, local search and neighbourhood change procedures needs to consider the whole solution (Pardo *et al.*, 2013). Therefore, the proposed methods are general parallelization strategies in the context of VNS, since they do not consider either the optimization problem or the VNS variant.

5.1 *Replicated parallel VNS*

The RP-VNS strategy consists of executing several parallel VNS procedures. We present two different strategies, RP-VNS1 and RP-VNS2. In particular, considering RP-VNS1, Algorithms 1 and 2 show the master and slave pseudo-code, respectively. The master process (Algorithm 1) starts by creating a pool of different threads (step 2). Then, for each thread in the pool, the master process sends the start signal to each thread (steps 3–5). When a thread finishes its execution, the master process compares the returned solution with the best overall solution, updating it if necessary (step 6).

Algorithm 1 RP-VNS1: master process

```
1: function RP-VNS1-Master($k_{max}, t_{max}$)
2: $P = CreateThreadPool(NTHREADS)$
3: for all $p \in P$ do
4:     $s_p \leftarrow ExecuteThread(p, k_{max}, t_{max})$
5: end for
6: $s_{best} = \arg\min_{p \in P}\{f(s_p)\}$
7: return $s_{best}$
8: end
```

Each slave process (Algorithm 2) executes the same VNS variant. Parameters $k_{\max}$ and $t_{\max}$ are set by the master process. The algorithm constructs an initial solution s in step 3 and sets k to the smallest neighbourhood. The current solution, s, is then perturbed with the shake procedure (step 6) obtaining a new solution s' in the kth neighbourhood. This solution is improved (step 7), resulting in a local optimum s''. The neighbourhood change procedure (step 8) determines whether s'' improves upon s (updating k to 1 and s to s'') or not (increasing the value of k). These steps are repeated until k reaches $k_{\max}$. Steps 2–11 are repeated until t reaches $t_{\max}$. Finally, the procedure returns the best solution found during the search.

Algorithm 2 RP-VNS1: slave process

```
1: function RP-VNS1-Slave($p, k_{max}, t_{max}$)
2: repeat
3:     $s \leftarrow CreateInitialSolution()$
4:     $k \leftarrow 1$
5:     repeat
6:         $s' \leftarrow Shake(s, k)$
7:         $s'' \leftarrow LocalSearch(s')$
8:         $NeighborhoodChange(s, s'', k)$
9:     until $k = k_{max}$
10:    $t \leftarrow CpuTime()$
11: until $t > t_{max}$
12: return $s$
13: end
```

Algorithm 3 RP-VNS2: master process

```
1: function RP-VNS2-Master($k_{max}, t_{max}$)
2: $P = CreateThreadPool(NTHREADS)$
3: $s_{best} \leftarrow CreateInitialSolution()$
4: repeat
5:     for all $p \in P$ do
6:         $s_p \leftarrow ExecuteThread(p, s_{best}, k_{max})$
7:     end for
8:     $s_{best} = \arg\min_{p \in P}\{f(s_p)\}$
9: until $t = t_{max}$
10: return $s_{best}$
11: end
```

The RP-VNS2 considers the cooperation among threads. In particular, each thread receives the best overall solution to start the corresponding VNS procedure. When a thread finishes, it notifies the master process the best solution found. Then, the master sends the best overall solution.

Algorithm 3 shows the master process pseudo-code. In this case, the master process constructs the initial solution (step 3) and controls the allowed execution time (steps 4–9). Additionally, it sends the best solution to each slave process (step 6). Algorithm 4 shows the slave pseudo-code. Now, this process does not construct a solution (which is received as an input argument), and it finishes when k reaches $k_{\max}$.

5.2 *Replicated shake VNS*

The RS-VNS strategy consists of parallelizing the shake and local search procedures. We consider two different strategies, RS-VNS1 and RS-VNS2. Algorithm 5 reports the pseudo-code of the RS-VNS1 master process. The algorithm begins by creating the pool of threads and an initial solution (steps 2 and 3). As usual, the VNS search is started in the smallest neighbourhood (step 5). Then, the master process delegates the execution of the shake and local search procedures to each thread. When a thread

Algorithm 4 RP-VNS2: slave process

```
1: function RP-VNS1-Slave($(p, s, k_{max})$)
2: $s \leftarrow CreateInitialSolution()$
3: $k \leftarrow 1$
4: repeat
5:     $s' \leftarrow Shake(s, k)$
6:     $s'' \leftarrow LocalSearch(s')$
7:     $NeighborhoodChange(s, s'', k)$
8: until $k = k_{max}$
9: return $s$
10: end
```

Algorithm 5 RS-VNS1: master process

```
1: function RS-VNS1-Master($(k_{max}, t_{max})$)
2: $P = CreateThreadPool(NTHREADS)$
3: $s_{best} \leftarrow CreateInitialSolution()$
4: repeat
5:     $k \leftarrow 1$
6:     repeat
7:         for all $p \in P$ do
8:             $s_p \leftarrow ExecuteThread(p, s_{best}, k_{max})$
9:         end for
10:        $s_{local} = \arg\min_{p \in P}\{f(s_p)\}$
11:        $NeighborhoodChange(s_{best}, s_{local}, k)$
12:    until $k = k_{max}$
13: until $t = t_{max}$
14: return $s_{best}$
15: end
```

finishes, it returns the best found solution (step 8). Once all threads have finished, the master process gathers the best solution found by the threads (step 10). The neighbourhood change procedure (step 11) determines whether s'' improves upon s (updating k to 1 and s to s'') or not (increasing the value of k). These steps are repeated until k reaches $k_{\max}$. Steps 4–13 are repeated until t reaches $t_{\max}$. Finally, the master process returns the best solution found during the search. For the sake of brevity, we do not include a pseudo-code of the slave process since it only consists of two instructions (shake and local search functions).

This strategy is equivalent to a best improvement method in the sense that the search waits until all threads finish, performing the best available move. RS-VNS2 considers an alternative strategy. In particular, it performs the first move which improves the current best found solution (first improvement), instead of waiting for all threads. Specifically, the pseudo-code of RS-VNS2 is similar to RS-VNS1. For that reason, we do not include the associated pseudo-code. The only difference between them is which solution is used in the neighbourhood change procedure. In RS-VNS2, we compare the solution returned by each thread with the best found solution so far. In case of improvement, we update the corresponding best solution and stop the execution of the remaining threads. Note that in RS-VNS2, step 10 must also be removed, since it is not required to find the best solution among all the treads.

5.3 *Cooperative neighbourhood VNS*

The CN-VNS strategy considers the cooperative exploration of different neighbourhoods by different threads. We present two different strategies: CN-VNS1 and CN-VNS2. In both, the master process is responsible for maintaining, updating and communicating the current overall best solution. It also initiates and terminates the algorithm executed in each thread. The logic of the master process is similar to the one presented in Algorithm 5. The main difference resides in the management of the neighbourhood. In particular, both strategies (CN-VNS1 and CN-VNS2) delegate this task to each thread. Therefore, the pseudo-code of these two strategies are equivalent to RS-VNS, but removing the instruction in step 11 (neighbourhood change procedure) and the input argument k in step 8 (since this value is set in the slave process).

In CN-VNS1, the slave process explores the $k_{\max}$ available neighbourhoods at random, while in CN-VNS2, the exploration is performed systematically. Pseudo-codes of CN-VNS1 and CN-VNS2 are shown in Algorithms 6 and 7, respectively. In particular, the random exploration of neighbourhoods starts by selecting at random a value of k, between 1 and $k_{\max}$ (step 3). Considering the solution s, the procedure performs the corresponding shake (step 5), obtaining a perturbed solution s' in the kth neighbourhood. This solution is then improved with the local search method (step 6), obtaining s''. Then, it is decided whether the slave process performs another iteration or not. Specifically, the shake and local search procedures are repeated while s'' is better than s (see steps 7–11); otherwise, the thread communicates the best solution found to the master process. Therefore, the corresponding thread waits until the master process sends again the best overall solution.

In CN-VNS2, slave processes explore systematically the $k_{\max}$ available neighbourhoods. In order to do so, the thread p performs the search in the $[k^p_{\text{first}}, k^p_{\text{last}}]$ sub-ranges. For example, the first thread explores neighbourhoods from 1 to $\lfloor k_{\max}/|P|\rfloor$, the second thread explores the neighbourhoods from $\lfloor k_{\max}/|P|\rfloor + 1$ to $\lfloor 2 * k_{\max}/|P|\rfloor$ and so on. Then, starting from solution s, the algorithm perturbs it (step 7), obtaining a new solution s' in the kth neighbourhood, with $k^p_{\text{first}} \leqslant k \leqslant k^p_{\text{last}}$. This solution is then

Algorithm 6 CN-VNS1: slave process

```
1:  function CN-VNS1-Slave($p, s, k_{max}$)
2:  $improved \leftarrow TRUE$
3:  $k \leftarrow Random(1, k_{max})$
4:  while $improved = TRUE$ do
5:      $s' \leftarrow Shake(s, k)$
6:      $s'' \leftarrow LocalSearch(s')$
7:      if $f(s'') > f(s)$ then
8:          $improved \leftarrow FALSE$
9:      else
10:         $s \leftarrow s''$
11:     end if
12: end while
13: return $s$
14: end
```

Algorithm 7 CN-VNS2: slave process

```
1: function CN-VNS2-Slave($p, s, k_{max}$)
2: $improved \leftarrow TRUE$
3: $k_{first} = \lfloor (p-1) * k_{max}/|P| \rfloor + 1$
4: $k_{last} = \lfloor p * k_{max}/|P| \rfloor$
5: $k \leftarrow k_{first}$
6: repeat
7:     $s' \leftarrow Shake(s, k)$
8:     $s'' \leftarrow LocalSearch(s')$
9:     if $f(s'') < f(s)$ then
10:        $s \leftarrow s''$
11:        $k \leftarrow k_{first}$
12:    else
13:        $k \leftarrow k + 1$
14:    end if
15: until $k = k_{last}$
16: return $s$
17: end
```

improved with the local search method (step 8), obtaining s''. Then, it is decided whether s'' improves upon s (resetting k to k_{first} and assigning s'' to s) or not (increasing the value of k). The thread finishes the search when k reaches k_{last}, communicating the best solution found to the master process. Then, it waits until the master process again sends it the best overall solution.

6. Computational experiments

This section reports and analyses the computational experiments that we have performed for testing the efficiency of the six proposed parallel VNS variants for solving the CMP. All the algorithms were implemented in Java SE 7 and the experiments were conducted on an Intel Core i7 2600 CPU (3.4 GHz) and 4 GB RAM. We derived 101 instances from the Harwell-Boeing Sparse Matrix Collection. This collection consists of a set of standard test matrices $M = M_{uv}$ arising from problems in linear systems, least squares and eigenvalue calculations from a wide variety of scientific and engineering disciplines. The graphs are derived from these matrices by considering an edge (u, v) for every element $M_{uv} = 0$. From the original set, we have selected the 101 graphs with $n \leqslant 3025$. The number of vertices and edges ranges from 30 to 3025 and from 103 to 8904, respectively.

We have divided our experimentation in two different parts: preliminary experimentation and final experimentation. In the preliminary experimentation, we study the effect of the number of threads in each algorithm. We consider a representative subset of 14 instances to conduct the preliminary experimentation. Then, in the final experimentation, we compare the best variants with the state-of-the-art algorithm.

According to Crainic & Toulouse (2003), the classical performance measure (i.e. speedup described by Barr & Hickman, 1993) is not adequate to evaluate the performance of parallel MHs since asynchronous interactions between threads generally induce significant differences in search behaviour, not only for the global parallel method, but also for each search process participating in the cooperation. Therefore, the sequential and parallel methods may then be viewed as different MHs, requiring a redefinition of speedup and other performance measures. This situation is further aggravated by the

Table 1 *Performance of RP-VNS*

	RP-VNS1				RP-VNS2			
Threads	2	4	8	16	2	4	8	16
Avg.	105.29	105.29	106.14	106.50	107.64	107.07	106.14	106.79
Dev (%)	2.03	2.03	3.48	4.41	7.77	6.93	7.11	7.44
#Best	9	9	8	8	7	7	8	7

randomness embedded in the VNS methods considered in this paper. However, it is important to remark that a parallelization strategy should speed up the search or produce better results than the sequential method. Consequently, we compare the quality of the solutions obtained by the sequential and parallel VNS methods to evaluate the quality of the parallel design strategy.

To have an illustrative comparison, each parallel variant is executed for the same CPU time than the best method identified in the literature, i.e. the VFS introduced by Pardo *et al.* (2013). In particular, we execute VFS (considering the parameters suggested by the authors) over the set of 14 instances used in the preliminary experimentation. The average CPU time of VFS is 840 s. Therefore, we execute all parallel VNS variants for this computing time.

In our first experiment, we compare the performance of RP-VNS1 and RP-VNS2 (described in Section 5.1) considering different number of threads: 2, 4, 8 and 16. Table 1 reports the average quality over all instances (Avg.), the average percent deviation with respect to the best known solutions (Dev (%)) and number of times that each method matches the best known solutions (#Best). Let s be a heuristic solution whose objective function value is $f(s)$ and let $s^\star$ be the best-known solution (with objective function value $f(s^\star)$). The deviation is then computed as $\text{Dev} = (f(s) - f(s^\star))/f(s^\star)$. We consider these three performance metrics for the rest of the experiments since they complement each other.

Table 1 shows that RP-VNS1 clearly outperforms RP-VNS2 in all the considered statistics. This results can be partially explained by the fact that the diversification of RP-VNS1 is larger than PR-VNS2 (i.e. it explores a larger number of different regions of the search space), since the last method starts the search from the best-known solution found so far. As it is well documented in the literature, the CMP presents a flat landscape. As a consequence, most of the moves performed by the search procedure have associated a null value. Therefore, it is more interesting to start the search from other point of the solution space (RP-VNS1) rather than continuing the search from the same point (RP-VNS2). We observe that better outcomes are obtained with a lower number of threads. As a result, RP-VNS1 with four threads is selected as the best variant of RP-VNS. We will use this algorithm in the final experimentation.

In the second experiment, we compare the performance of RS-VNS1 and RS-VNS2 (described in Section 5.2). As in the previous experiment, we consider that the number of threads ranges from 2 to 16. Table 2 reports that differences between both variants are smaller than in the first experiment. In particular, RS-VNS1 with 4 threads presents an average percentage deviation of 1.38%, and it matches 11 times the best-known solutions (out of 14). The best RS-VNS2 method uses 16 threads and finds the same number of best values. However, it presents a slightly larger deviation (1.45%). Therefore, we select RS-VNS1 with four threads as RS-VNS variant for the final experimentation.

TABLE 2 *Performance of RS-VNS*

	RS-VNS1				RS-VNS2			
Threads	2	4	8	16	2	4	8	16
Avg.	106.50	105.14	105.43	105.36	105.79	105.36	105.07	105.07
Dev (%)	3.29	1.38	2.37	2.15	3.50	2.35	1.45	1.45
#Best	7	11	8	10	8	8	10	11

TABLE 3 *Performance of CN-VNS*

	CN-VNS1				CN-VNS2			
Threads	2	4	8	16	2	4	8	16
Avg.	110.29	108.64	108.64	108.50	107.64	108.43	107.57	107.50
Dev (%)	10.25	9.59	9.56	8.82	7.53	7.83	7.51	7.24
#Best	8	7	7	7	7	7	7	7

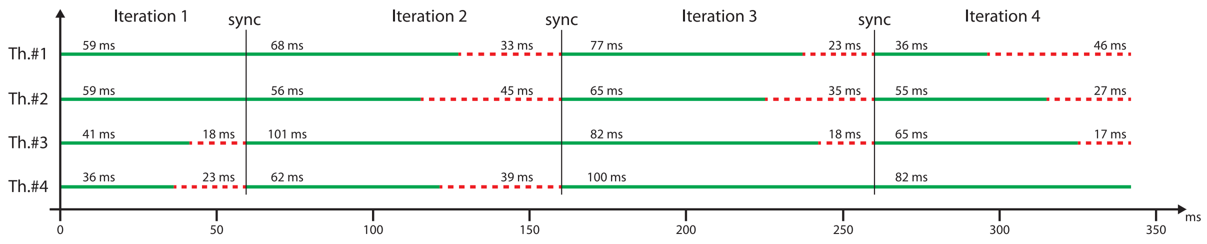


FIG. 2. Performance of each thread in RS_VNS1 on *lshp3025* instance.

In the next preliminary experiment, we compare the performance of two CN-VNS (i.e. CN-VNS1 and CN-VNS2, both described in Section 5.3). Table 3 reports the results obtained by both parallel algorithms, considering the same number of threads than in the previous experiments. The performance of all these variants seems to be really poor compared with the previous methods. In particular, CN-VNS2 with 16 threads is the best variant with a deviation of 7.24% and matches the best known solution 7 times (out of 14). It seems that the larger the number of threads, the better is the performance of the method. Then this strategy could be more competitive if we would consider a larger number of threads. However, due to our hardware limitation, we are not able to test this hypothesis. Specifically, if we increase the number of threads, the required time to manage them deteriorates the performance of the method. It is interesting to compare the best CN-VNS variant with the state-of-the-art method.

In the last two preliminary experiments, we study the time spent in the parallelization management of the algorithms. First, in Fig. 2, we illustrate the behaviour of the different threads within an algorithm. In order to do that, we have selected RS-VNS1 with four threads and we have depicted its behaviour over a representative instance (*lshp3025*) during four iterations. For each iteration and thread, we depict the time that the thread is working (straight lines) and the time that the thread is waiting for synchronization (dashed lines). As shown in the figure, an iteration ends when all threads have finished working. After that, the parallel algorithm synchronizes the information among the threads (represented in the figure as

TABLE 4 *Comparison of the synchronization time spent by each algorithm*

	RP-VNS1 (4) (%)	RS-VNS1 (4) (%)	CN-VNS2 (16) (%)
Avg. waiting time	9.43	1.44	0.04
Max. waiting time	11.25	4.13	0.07

vertical black lines at the end of each iteration). Obviously, there is at least one thread that is working until the end of the iteration.

The total computing time of the previous example is 342 ms, which includes the working time, the waiting time and the synchronization time of each thread. In all our experiments, the synchronization time is negligible; therefore, we do not consider further analysis of this time in the performance of the parallel strategies.

In the example shown in Fig. 2, we can observe that the thread that waits longer (Th.#2) spends approximately 31% of the time waiting, while Th.#3 is the one with the minimum waiting time (approximately 15% of the time). On average, the threads are waiting 23% of the time.

Once the behaviour of the threads has been described, we conduct an additional experiment over the set of instances used in the preliminary experimentation. In Table 4, we present the best variants of the three proposed parallel algorithms (where the number of threads are depicted in parenthesis). For each method, we report the average CPU time spent by all threads waiting for synchronization (Avg. waiting time) and also the maximum CPU time that a single thread within the method needs to wait (Max. waiting time). Both results are presented in percentage over the total CPU time spent. As expected, RP-VNS1 presents the largest average and maximum waiting time, since each thread in this strategy performs a completely independent iteration. In other words, if the constructed solution has a relatively high quality, the VNS strategy will reach a local optimum faster. Therefore, the waiting time of that thread will be longer, since it will need the other threads (starting from a worse solution) to finish. On the other hand, the CN-VNS2 strategy presents the shortest average and maximum waiting time. This behaviour can be explained by considering that, in this strategy, all the threads start the search from the same initial solution (the best one found in the iteration). Therefore, the iterations performed by the parallel VNS are quite similar. Finally, the RS-VNS1 strategy meets a balance among the strategies of the other two methods. Particularly, at each iteration it starts from the best solution found (similar to CN-VNS2) but it applies an independent shake and local search procedures to that solution in each thread (similar to RP-VNS1).

Finally, we compare the previous best variants with the state-of-the-art method (VFS). In this experiment, we consider the whole set of 101 instances. Table 5 clearly shows the superiority of the RS-VNS strategy over the other competitors. In particular, it presents a deviation of 0.28% which compares favourably to the RP-VNS (0.64%), the state-of-the-art method (0.89%) and the CN-VNS (1.52%). Considering the number of best solutions found, the analysis is similar. It is important to remark that, although CN-VNS2 (16) obtains the worst deviation, it only finds one less better solution than the state-of-the-art method. Even more, when considering the average quality, the CN-VNS2 (16) ranks second (284.87), improving the RP-VNS1 (4) with an average quality of 285.01 and VFS (285.46).

To confirm these conclusions, we conduct the non-parametric Friedman test (Friedman, 1940) for multiple correlated samples to the best solutions obtained by VFS, RP-VNS1, RS-VNS1 and CN-VNS2. This test computes, for each instance, the rank value of each method according to the solution quality (where rank 1 is assigned to the best method and rank 4 to the worst one). Then it calculates the average rank values of each method across all the instances solved. If the averages differ greatly, the associated

Table 5 *Comparison of the parallel VNS variants with the state of the art*

	VFS	RP-VNS1(4)	RS-VNS1(4)	CN-VNS2(16)
Avg.	285.46	285.01	284.64	284.87
Dev (%)	0.89	0.64	0.28	1.52
#Best	84	86	92	83

p-value or significance will be small. The resulting *p*-value of 0.028 (considering a level of significance of 0.05) obtained in this experiment clearly indicates that there are statistically significant differences among the four methods tested. Specifically, the rank values produced by this test are 2.20 (RS-VNS1), 2.45 (RP-VNS1), 2.47 (VFS) and 2.88 (CN-VNS2). This experiment confirms again the superiority of two parallel VNS procedures over the state of the art.

We conduct a test of Wilcoxon to further analyse the differences between our best method RS-VNS1 and the state-of-the-art algorithm (VFS). In particular, this experiment has a *p*-value of 0.024. Then, considering a level of significance of 0.05, this experiment indicates that there are statistically significant differences between both methods, confirming again that the parallel version improves the sequential one.

7. Conclusion

We proposed in this paper six different general parallel designs for the VNS schema. We group these variants into three different templates. The first one is oriented to parallelize the whole VNS method (RP-VNS1 and RP-VNS2). The second one parallelizes the shake and the local search procedures (RS-VNS1 and RS-VNS2). Finally, the third one explores in parallel the set of predefined neighbourhoods (CN-VNS1 and CN-VNS2). These general strategies are then applied to parallelize a recently introduced VNS algorithm, called VFS. We conducted an experimental comparison among these variants on the CMP. A preliminary experimentation allows us to identify the best variant from each template, resulting in the selection of RP-VNS1, RS-VNS1 and CN-VNS2, respectively. Then we compared the selected three methods with the sequential VFS, which is the best algorithm in the state of the art for the CMP. Experimental results showed that two of the three proposed parallel methods (RS-VNS1 and RP-VNS1) outperform previous best methods in the state of the art of the CMP in terms of quality. We also conducted statistical tests to confirm the significance of the obtained results with RS-VNS1 emerging as the best algorithm.

Funding

This research was partially supported by the Spanish Ministry of 'Economía y Competitividad', grants ref. TIN2009-07516, TIN2011-28151 and TIN2012-35632, and the Government of the Community of Madrid, grant ref. S2009/TIC-1542.

References

Adolphson, D. & Hu, T. C. (1973) Optimal linear ordering. *SIAM J. Appl. Math.*, **25**, 403–423.

Alba, E. & Nebro, A. J. (2005) New technologies in parallelism. *Parallel Metaheuristics. A New Class of Algorithms*, vol. 2. Wiley-Interscience.

Andrade, D. V. & Resende, M. G. C. (2007a) GRASP with path-relinking for Network Migration Scheduling. *Proceedings of International Network Optimization Conference (INOC).*

Andrade, D. V. & Resende, M. G. C. (2007b) GRASP with evolutionary path-relinking. *Proceedings of Seventh Metaheuristics International Conference (MIC).*

Barr, R. S. & Hickman, B. L. (1993) Reporting computational experiments with parallel algorithms: issues, measures, and experts opinions. *ORSA J. Comput.*, **5**, 2–18.

Botafogo, R. A. (1993) Cluster analysis for hypertext systems. *16th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*, pp. 116–125.

Chung, M. J., Makedon, F., Sudborough, I. H. & Turner, J. (1982) Polynomial time algorithms for the MIN CUT problem on degree restricted trees. *SFCS '82: Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, pp. 262–271.

Crainic, T. G., Gendreau, M., Hansen, P. & Mladenović, N. (2004) Cooperative parallel variable neighborhood search for the p-median. *J. Heuristics*, **10**, 293–314.

Crainic, T. G. & Toulouse, M. (2003) Parallel strategies for meta-heuristics. *State-of-the-Art Handbook in Metaheuristics*, pp. 475–513.

Cohoon, J. & Sahni, S. (1983) Exact algorithms for special cases of the Board Permutation Problem. *Proceedings of the Allerton Conference on Communication, Control and Computing*, pp. 246–255.

Cohoon, J. & Sahni, S. (1987) Heuristics for backplane ordering. *J. VLSI Comput. Syst.*, **2**, 37–61.

Díaz, J., Gibbons, A., Pantziou, G. E., Serna, M. J., Spirakis, P. G. & Toran, J. (1997) Parallel algorithms for the minimum cut and the minimum length tree layout problems. *Theor. Comput. Sci.*, **181**, 267–287.

Duarte, A., Martí, R., Resende, M. G. C. & Silva, R. M. A. (2011) GRASP with path relinking heuristics for the antibandwidth problem. *Networks*, **58**, 171–189.

Friedman, M. (1940) A comparison of alternative tests of significance for the problem of m rankings. *Ann. Math. Statist.*, **11**, 86–92.

García-López, F., Melián-Batista, B., Moreno-Pérez, J. A. & Moreno-Vega, J. M. (2002) The parallel variable neighborhood search for the p-median problem. *J. Heuristics*, **8**, 375–388.

Gavril, F. (1977) Some NP-complete problems on graphs. *Proceedings of the Eleventh Conference on Information Sciences and Systems*, pp. 91–95.

Gendreau, M. & Potvin, J. Y. (1977) *Handbook of Metaheuristics*. Berlin: Springer Publishing Company, Incorporated.

Harper, L. H. (1966) Optimal numberings and isoperimetric problems on graphs. *J. Combin. Theory*, **1**, 385–393.

Karger, D. R. (1999) A randomized fully Polynomial Time Approximation Scheme for the all-terminal network reliability problem. *SIAM J. Comput.*, **29**, 492–514.

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 Appl. Math.*, **23**, 243–265.

Martí, R., Pantrigo, J. J., Duarte A. & Pardo, E. G. (2013) Branch and bound for the cutwidth minimization problem. *Comput. Oper. Res.*, **40**, 137–149.

Mladenović, N. & Hansen, P. (1997) Variable neighborhood search. *Comput. Oper. Res.*, **24**, 1097–1100.

Mladenović, N., Plastria, F. & Urosevic, D. (2005) Reformulation descent applied to circle packing problems. *Comput. Oper. Res.*, **32**, 2419–2434.

Moreno, J. A., Moreno, J. M. & Verdegay, J. L. (2004) Parallel Variable neighborhood search. The *p*-median problem: a survey of metaheuristic approaches. *Les Cahiers du GERAD*, **92**, 1–22.

Mutzel, P. (1995) A polyhedral approach to planar augmentation and related problems. *ESA'95: Proceedings of the Third Annual European Symposium on Algorithms*, pp. 494–507.

Palubeckis, G. & Rubliauskas, D. (2012) A branch-and-bound algorithm for the minimum cut linear arrangement problem. *J. Combin. Optim.*, **24**, 540–563.

Pantrigo, J. J., Martí, R., Duarte, A. & Pardo, E. G. (2012) Scatter search for the cutwidth minimization problem. *Ann. Oper. Res.*, **199**, 285–304.

Pardo, E. G., Mladenovic, N., Pantrigo, J. J. & Duarte, A. (2013) Variable formulation search for the cutwidth minimization problem. *Appl. Soft Comput.*, **13**, 2242–2252.

Petit, J. (2003) Experiments on the minimum linear arrangement problem. *ACM J. Exp. Algorithmics*, **8**, 1084–6654.

Reinelt, G. (1991) TSPLIB—a traveling salesman problem library. *INFORMS J. Comput.*, **3**, 376–384.

Resende, M. G. C. & Andrade, D. V. (2009) *Method and System for Network Migration Scheduling*. Atlanta, Georgia: 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. *Comput. Oper. Res.*, **37**, 498–508.

Rolim, J., Skora, O. & Vrt'o, I. (1995) Optimal cutwidths and bisection widths of 2- and 3-dimensional meshes. *Graph-Theoretic Concepts in Computer Science*, vol. 1017, pp. 252–264.

Shahrokhi, F., Skora, O., Szkely, L. A. & Vrt'o, I. (2001) On bipartite drawings and the linear arrangement problem. *SIAM J. Comput.*, **30**, 1773–1789.

Takagi, K. & Takagi, N. (1999) Minimum Cut Linear Arrangement of p-q dags for VLSI layout of adder trees. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, **E82-A**, 767–774.

Thilikos, D. M., Serna, M. J. & Bodlaender, H. L. (2005) Cutwidth II: algorithms for partial w-trees of bounded degree. *J. Algorithms*, **56**, 25–49.

Yannakakis, M. (1985) A polynomial algorithm for the Min-Cut linear arrangement of trees. *J. ACM*, **32**, 950–988.

Chapter 11

Scatter Search for the Bandpass Problem

- J. Sánchez-Oro, M. Laguna, R. Martí, and A. Duarte. Scatter search for the bandpass problem. Journal of Global Optimization, pages 1–22, 2016.
    - DOI: `https://doi.org/10.1007/s10898-016-0446-0`
    - Impact Factor (JCR 2015): 1.219

Subject Category	**Ranking**	**Quartile**
Mathematics, applied	63 / 254	Q1
Operations Research and Management Science	42 / 82	Q3

J Glob Optim
DOI 10.1007/s10898-016-0446-0

Scatter search for the bandpass problem

Jesús Sánchez-Oro[1] · Manuel Laguna[2] · Rafael Martí[3] · Abraham Duarte[1]

Received: 11 November 2015 / Accepted: 23 May 2016

Abstract We tackle a combinatorial problem that consists of finding the optimal configuration of a binary matrix. The configuration is determined by the ordering of the rows in the matrix and the objective function value is associated with a value B, the so-called bandpass number. In the basic version of the problem, the objective is to maximize the number of non-overlapping blocks containing B consecutive cells with a value of one in each column of the matrix. We explore variants of this basic problem and use them to test heuristic strategies within the scatter search framework. An existing library of problem instances is used to perform scientific testing of the proposed search procedures to gain insights that may be valuable in other combinational optimization settings. We also conduct competitive testing to compare outcomes with methods published in the literature and to improve upon previous results.

Keywords Bandpass problem · Scatter search · Path relinking · Telecommunications · Metaheuristics

✉ Abraham Duarte
abraham.duarte@urjc.es

Jesús Sánchez-Oro
rafael.marti@urjc.es

Manuel Laguna
laguna@colorado.edu

Rafael Martí
Rafael.Marti@uv.es

[1] Departamento de Ciencias de la Computación, Universidad Rey Juan Carlos, Madrid, Spain

[2] Leeds School of Business, University of Colorado at Boulder, Boulder, CO, USA

[3] Departamento de Estadística e Investigación Operativa, Universidad de Valencia, Valencia, Spain

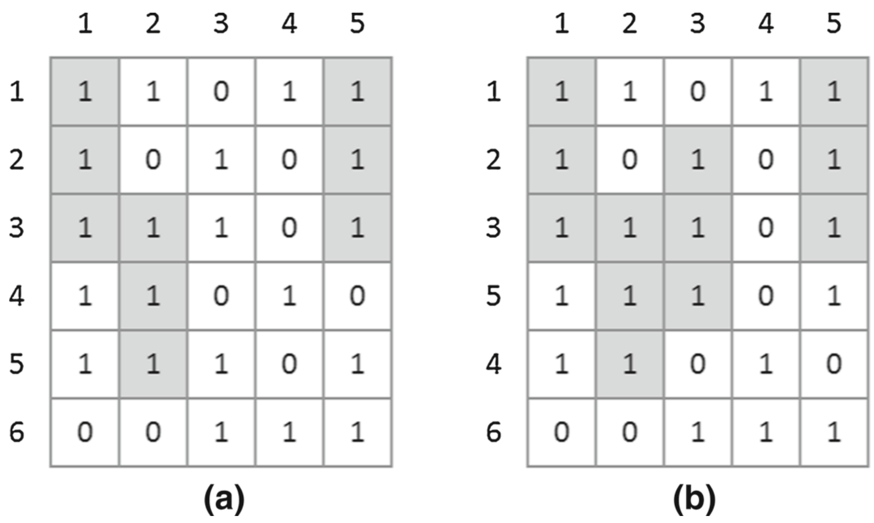
1 Introduction

The bandpass problem is associated with early-generation DWDM (Dense Wavelength Division Multiplexing) systems in fiber optic networks [8,16,27]. This technology used the *band* concept to allow the transmission of several wavelengths in a single fiber-optic cable. Wavelengths enter or leave the fiber optic cables via a set of devices known as OADMs (Optical Add/Drop Multiplexers), which are installed along the fiber optic cable at origin and destination points [16,18]. OADMs contain special cards that control the wavelengths and identify which wavelengths should pass through the device and which should be dropped. Typically, several wavelengths pass through the same OADM. The wavelengths may be handled as a block when they are consecutive. A block of wavelengths is known as a bandpass. Without this assembling of wavelengths, a card is needed for each wavelength. However, when a bandpass is created, a single card can handle the entire block of wavelengths. The number of wavelengths in the block is referred to as the bandpass number (B). Therefore, the process of creating a wavelength block results in a reduction of cards from B to 1 and hence a cost reduction in the design and maintenance of the fiber optic network. The optimization opportunity is to assign wavelengths in order to maximize the number of bandpasses with a bandpass number of B.

This bandpass problem was first described by Bell and Babayev in the Annual INFORMS Meeting held in Denver Colorado in October 2004 and the first publication appeared in 2009 [1]. Since then, newer generations of DWDM systems have been developed to add more flexibility. This newer equipment employs reconfigurable optical ADMs (ROADMs) for which the band concept does not apply. The ROADM is a dynamic wavelength arrangement scheme enabled by wavelength selective switching. These new systems, however, are much more expensive and therefore the older systems still have a market in small DWDM deployments.

The bandpass problem consists of creating blocks of wavelengths that need to be sent from a set of origins to a set of destinations. The goal of the blocks is to reduce the number of devices needed for the transmission of the wavelengths, resulting in a decrease of both installation and maintenance costs. The telecommunication network is modeled as a $m \times n$ binary matrix A, where a_{ij} is equal to 1 if wavelength i must be delivered to destination j. In the basic version of the problem, the value of B (i.e., the bandpass number) is given. A bandpass is formed by B consecutive 1s in the same column. Each value of 1 can only be used for one bandpass and therefore two distinct bandpasses in the same column cannot have a common element. Figure 1a shows a network with 6 wavelengths and 5 destinations. The network shows that, for instance, wavelength 1 must be delivered to all but the third destination. Assuming that $B = 3$, the matrix in Fig. 1a contains 3 bandpasses, as indicated by the gray blocks, where O refers to the ordering of the wavelengths, that is, $O(i)$ is the index of the wavelength assigned to row i.

In Fig. 1a, the bandpass for column 1 is shown as rows 1, 2, and 3. However, the bandpass for that column could also be formed by rows 2–4 or 3–5. Regardless of which one is chosen, there can only be one bandpass in column 1. In order to increase the number of bandpasses, the assignment of wavelengths to rows could be changed. This assignment could also be thought of as a reordering of the rows in the matrix. For instance, Fig. 1b shows the matrix that results by assigning wavelength 4 to row 5 and wavelength 5 to row 4. In other words, wavelengths 4 and 5 have exchanged positions from the original lexicographical order shown in Fig. 1a. The reassignment results in an increase of bandpasses from 3 to 4.



(a)

	1	2	3	4	5
1	1	1	0	1	1
2	1	0	1	0	1
3	1	1	1	0	1
4	1	1	0	1	0
5	1	1	1	0	1
6	0	0	1	1	1

(b)

	1	2	3	4	5
1	1	1	0	1	1
2	1	0	1	0	1
3	1	1	1	0	1
5	1	1	1	0	1
4	1	1	0	1	0
6	0	0	1	1	1

Fig. 1 Bandpasses for two possible assignments of wavelengths to rows

For a given bandpass number B, the bandpass problem consists of finding an ordering O^* of the rows that maximizes the number of bandpasses. The upper bound on the number of bandpasses in column j is given by:

$$\bar{b}_j = \left\lfloor \frac{\sum_i a_{ij}}{B} \right\rfloor$$

Therefore, if $f(O)$ is the number of bandpasses in solution O, then the objective function value of an optimal solution O^* is bounded as follows:

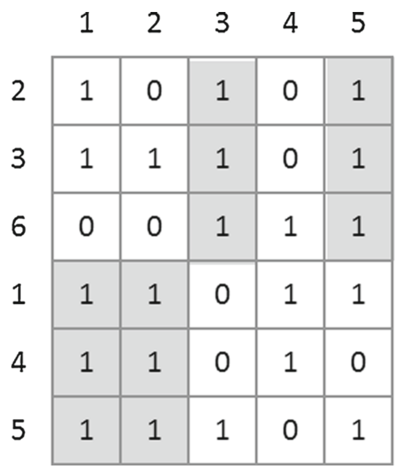
$$f\left(O^*\right) \leq \sum_j \bar{b}_j$$

For $B = 3$, the upper bound for the example in Fig. 1 is 5 bandpasses. It can be easily verified that this upper bound is achieved by the optimal solution $O = (5, 4, 1, 6, 3, 2)$.

Babayev, Bell, and Nuriyev [1] refer to the problem described above as BP1. They also suggest an alternative version of the problem, denoted by BP2, which reflects some specific network technologies. In this second version, a total of up to $G+1$ groups of rows are created, where the first G groups have B rows. That is, the first group consists of rows 1 to B, the second group consist of rows $B + 1$ to $2B$, and so on. If the remainder of the *m/B* quotient is greater than zero, then an additional group with the last $m - GB$ rows is created. The value of G is determined by:

$$G = \left\lfloor \frac{m}{B} \right\rfloor$$

A bandpass in BP2 is defined as a column within a group for which all its elements are nonzero. If the group is one of the first G, then the bandpass contains B nonzero elements. For group $G + 1$, if it exists, the bandpass contains $m - GB$ elements with nonzero values. As mentioned in [1], this model reflects some specific equipment constraints. In contrast with BP1, the group of rows that can form bandpasses are all the same across columns. Figure 2 shows an optimal solution to BP2 for the example matrix shown in Fig. 1. This solution corresponds to the row ordering $O = (2, 3, 6, 1, 4, 5)$. Note that because $m = 6$ and $B = 3$ then $G = 2$ and the remainder of *m/B* is zero. Therefore, there are only 2 groups, one consisting of the first three rows in the matrix and the second one consisting of rows 4 to 6.



	1	2	3	4	5
2	1	0	1	0	1
3	1	1	1	0	1
6	0	0	1	1	1
1	1	1	0	1	1
4	1	1	0	1	0
5	1	1	1	0	1

Fig. 2 Optimal solution to BP2 for matrix in Fig. 1

For BP2, the order of the rows within each group is not relevant and thus, for instance, $O = (2, 3, 6, 1, 4, 5)$ is the same solution as $O = (6, 3, 2, 5, 4, 1)$ or as any ordering where the first three rows contain the wavelengths 2, 3, and 6. A permutation vector clearly is not an effective solution representation for BP2 given that it creates a search space with $m!$ solutions while the number of unique solutions for a BP2 instance with G groups of size B is given by:

$$\frac{m!}{(B!)^G \, G!}$$

In the example of Fig. 2, there are 720 permutations of the rows but only 10 unique ways of creating two groups of 3 rows. A more direct representation is given by S for which $S(i)$ is the index of the group to which wavelength i is assigned.

Both BP1 and BP2 assume that the value of B is given. Kurt et al. [19] argue that fixing the value of B may not be optimal and therefore they suggest an optimization model in which B becomes a decision variable. However, since the range of relevant values for B is expected to be somewhat limited, the optimization process to determine the best value for B can be achieved by simply solving BP1 multiple times. We do not address this variant as a separate version of the problem.

The third version that we include in our study is the one introduced by Nuriyev et al. [25] and for which Gürsoy and Nnuriyev [17] developed a genetic algorithm. This is the so-called multi bandpass problem (MBP). In this version, instead of a single B value across the entire network, each destination j has its own B_j value. The assumption is that ADMs may be programmed independently from each other with different bandpass numbers. In the MBP, the input data include the B_j values for $j = 1, \ldots, n$. An extension of the MBP, suggested by Kurt et al. [19], turns the B_j values into variables. This extension is referred to as the "Optimization of Lengths in the Multi-Bandpass Problem". The optimization problem involves not only finding the best ordering of the rows in the matrix but also the best bandpass number for each destination. The literature does not include any experimentation on this problem and it is not clear whether the extension corresponds to an actual technology for which the B_j values can be optimized within a given range. We do not address this variant here.

The literature on the bandpass problem is limited to seven articles [1,2,17,19,21,22,25]. Mathematical models for BP1 and BP2 are presented in [1] along with a NP-hard proof of BP1. Babayev, Bell, and Nuriyev [1] developed two solution procedures: an exact polynomial algorithm for BP1 with $n = 2$ and a heuristic for the general case. The authors also report optimal solutions for BP1 problem instances with $m \leq 32$ and $n \leq 8$, found by solving a mathematical programming model. A description of a generator of problem instances is provided and used to produce 90 instances with $m = 64$ and 96 and $n = 8, 12, 16$, and 25. These instances are divided into two groups of 45 instances each, one for which the optimal solutions are known (OS instances) and one for which the optimal solutions are not known (BKS instances). Collectively, these problems are referred to as the Library of Bandpass Problems (BPLIB).[1] No experiments are performed for BP2 in [1]. Additional mathematical proofs for special cases of BP1 are provided in [21,22].

Berberler and Gürsoy [2] developed four variants of a genetic algorithm and applied them to the 45 OS instances in BPLIB. Optimality gaps are reported for all variants in all problem instances. Several GA variants are also developed and tested in [19]. This article introduces a new set of problems for the MBP. The data include both the B_j values and the binary A matrix.[2] Finally, some of the same authors in [1,19] are co-authors of [25] where mathematical models for BP1, BP2, and MBP are presented. Some of these models already appeared in [1,19] and some are new. No additional experimentation or results are included in [25].

Our current investigation into this problem class includes BP1, BP2, and MBP. We use these problems as a platform to test heuristic strategies within the scatter search (SS) framework. An existing library of problem instances is used to perform scientific testing to identify the contributions of several search strategies. In particular, we develop and test:

- Two diversification generation methods.
- Four improvement methods.
- Two combination methods.
- A scatter search method, SS1, for which solutions are represented by an ordering O of the wavelengths and therefore equipped to tackle BP1 and MBP instances.
- A scatter search method, SS2, for which solutions are represented as an assignment S of wavelengths to groups and that it is configured to solve BP2 instances.

We also conduct competitive testing to compare the solutions obtained with SS1 and SS2 with the best methods published in the literature for these problems (DSR [1], GAs [2, 17]). In addition, we compare performance against commercial optimizers, one based on metaheuristic technology (LocalSolver) and the other built on exact procedures (Gurobi).

2 Standard scatter search methods

Scatter search (SS) [13,20] is a population-based metaheuristic framework that consists of methods that generate, maintain, and transform a reference set of solutions ($RefSet$). The methods are organized as shown in Algorithm 1. Basic and advanced SS implementations have been developed for a variety of problems (see, e.g., [5,10,23,26,29]). In this section, we describe the implementation of the standard methods within the SS framework that we developed for BP1, BP2, and MBP. Two solution representations are used: 1) O, the ordering of the wavelengths (for BP1 and MBP); and 2) S, the group assignment (for BP2). Since the search strategies are linked to the solution representation, the same SS procedure is used to search for solution to BP1 and MBP.

[1] The Library of Bandpass Problems can be found here http://sci.ege.edu.tr/~math/BandssProblemsLibrary/.

[2] Problem instances introduced in [19] are found here http://fen.ege.edu.tr/arifgursoy/mopt/.

```
Diversification generator
Improvement
Initial $RefSet$
**do**
    Subset generator
    Combination
    Improvement
    $RefSet$ update
**until** termination is criteria satisfied
```

Algorithm 1 Scatter search template.

Following the template in Algorithm 1, we implemented standard methods for creating the initial $RefSet$, for generating solution subsets, and for updating the $RefSet$. The diversity generator and the improvement method (which will be described in Sections 3 and 4, respectively) are applied first to produce a population P of feasible solutions to the problem. Then, the initial $RefSet$ (of size β) is built by first selecting the $\beta/2$ best (in terms of the objective function value) solutions from P. The selected solutions are removed from P. Then, the $\beta/2$ most diverse solutions with respect to those currently in the $RefSet$ are selected from it p. Because the selection of the most diverse group of elements out of a set is a hard problem (see, e.g., [7]), the solutions are heuristically selected one at a time.

Any diversification strategy requires a measure of distance between any two solutions. The distance for each solution representation is measured as follows.

$$p\left(O, O'\right) = \sum_{i=1}^{m} |O\left(i\right) - O'\left(i\right)|$$
$$c\left(S, S'\right) = m - \text{number of common elements}$$

$p\left(O, O'\right)$ represents the so-called positional distance in permutation vectors (see Das and Roberts [4]), where $O\left(i\right)$ is the wavelength in row i. To calculate $c\left(S, S'\right)$, we count the number of common assignments between both solutions by matching the groups in S with the groups in S'. For each group and its match, we identify the wavelengths that are assigned to both (i.e., the common elements). The matching problem has the following form:

$$\begin{aligned} &\text{Maximize} && \sum\nolimits_k \sum\nolimits_l c_{kl} x_{kl} \\ &\text{Subject to} && \sum\nolimits_l x_{kl} = 1 \quad \forall k \\ &&& x_{kl} \in \{0, 1\} \quad \forall k, l \in [1, \ldots m] \end{aligned}$$

In this problem, $x_{kl} = 1$ when group k in solution S is matched with group l in solution S'. The c_{kl} value indicates the number of common elements in groups k and l. The total number of common elements is given by cx^*, where x^* is an optimal solution to the matching problem. Then, $c\left(S, S'\right) = m - cx^*$.

For the sequential selection of the $\beta/2$ diverse solutions to be included in the reference set, a maxmin criterion is used. The goal at each step is to find the solution that has the maximum distance between itself and the solutions currently in the reference set. The distance between a solution in P and the solutions in the reference set depends on the solution representation, as follows:

$$d\left(O, RefSet\right) = \min_{O' \in Refset} p\left(O, O'\right)$$
$$d\left(S, RefSet\right) = \min_{S' \in Refset} c\left(S, S'\right)$$

Then, at each step of the process, the solution R from P that has the maximum distance between itself and the solutions currently in $RefSet$ is selected to be added to the reference set. Mathematically, R is identified as follows:

$$R = \arg\max_{X \in P} d\left(X, RefSet\right)$$

where X is either O or S, depending on the solution representation. Solution R is added to $RefSet$ and deleted from P. This is repeated $\beta/2$ times in order to complete the $RefSet$ with β solutions. As shown in Algorithm 1, the iterative process begins after the initial $RefSet$ has been created. This process consists of selecting subsets of reference solutions to perform combinations that will result in new trial solutions. These solutions are then considered for inclusion in the reference set once they have been subjected to the improvement method. As recommended in [20], we employ a standard subset generation method that consists of selecting all pairs of reference solutions that contain at least one new solution. A new solution is one that has been added in the previous iteration of the do-loop in Algorithm 1. In the first iterations, all the solutions are new and therefore the number of subsets generated the first time around equals $\binom{\beta}{2}$.

The updating of $RefSet$ occurs at the end of each iteration. This is where the new solutions that were generated by the combination method and potentially improved by the improvement method are considered for membership in the reference set. The goal of the updating procedures is to maintain both quality and diversity in the reference set. Therefore, a solution is admitted to the reference set if its objective function value is better than the objective function value of the worst solution in the current reference set. For the sake of diversity, a solution that passes the above quality test replaces the reference solution that is closest to it according to the distance measures described above. This means that a solution that is included in the reference set does not necessarily replace the worst reference solution, unless the worst solution in the current $RefSet$ happens to be the closest to this new reference solution.

3 Diversification generation method

The diversification generator is one of three scatter search methods that are problem-specific. That is, this method along with the combination and improvement methods take advantage of the problem context to create a customized solution procedure within the scatter search framework. Like it is often done to induce diversification, our method relies on a controlled randomization process. Controlled randomization refers to a strategy that makes reference to the objective function value while constructing solutions. This is in contrast to full randomization where solutions are constructed without any support or direction from the objective function.

Algorithm 2 summarizes the steps performed by the diversification generation method for solutions represented as an ordering of the wavelengths. The set U consists of all the wavelengths that have not been assigned to a row. Initially, U contains all m wavelengths in the problem and the solution O being constructed is empty. The iterative process assigns one wavelength at a time and ends when no more unassigned wavelengths are left (i.e., when U is empty). In each step, a wavelength k is randomly selected from U by the $Random$ () function. The $BestRow$ () function then searches for the best row i^* where to assign k in the partial solution O. In the first iteration, the selected wavelength is assigned to the first row, that is $i^* = 1$. In the ith iteration, there are $i - 1$ wavelengths in the partial solution and the search for the best row i^* consists of inserting wavelength k in the $i - 1$ available positions, starting in row 1. The row that produces the largest increase in the objective function value (i.e., that increases the number of bandpasses the most) is chosen as i^*. If no row assignment from 1 to $i - 1$ is able to improve upon the current objective function value then $i^* = i$, meaning that wavelength k is assigned to the last row.

```
$U \leftarrow \{1, \dots, m\}$
$O \leftarrow \emptyset$
**while** $U \neq \emptyset$ **do**
    $k \leftarrow Random(U)$
    $i^* \leftarrow BestRow(O, k)$
    $O(i^*) = k$
    $U \leftarrow U \backslash \{k\}$
**end while**
```

Algorithm 2 Diversification generation method for solutions represented by O.

The controlled randomization aspect of the method is evident in Algorithm 2 by a random selection of a wavelength that is followed by a purposeful search for the best row. The random nature of the procedure enables it to generate the population of solutions needed to initiate the scatter search.

Algorithm 3 shows the procedure for solutions represented by S that like Algorithm 2 is also semi-greedy [8] but that follows the tenets of GRASP [9]. For each candidate wavelength, the method calculates a greedy function $h_1(i, g)$ that consists of the number of the potential bandpasses that result from adding wavelength i to group g. A potential bandpass is a column within the group that contains all 1s after wavelength i has been added to the group. The function counts potential bandpasses because the final existence of a bandpass is not confirmed until B wavelengths are assigned to a group.

For instance, consider a problem with $n = 5$ and $B = 4$ and the two partial groups with two wavelengths each shown in Fig. 3. Wavelengths 4 and 8 are already assigned to group 1. Wavelengths 3 and 6 are assigned to group 2. The construction procedure is considering wavelength number 7 to be added to either group 1 or group 2. Each group in the example has two potential bandpasses, destinations 1 and 4 for group 1 and destinations 2 and 4 for group 2. Adding row 7 to group 1 preserves the potential of two bandpasses for that group (see blocks with bold lines in Fig. 3). Adding wavelength 7 to group 2 reduces the potential bandpasses to 1. Therefore, the greedy function evaluations for adding wavelength 7 to each group are:

$$h_1(7, 1) = 2$$
$$h_1(7, 2) = 1$$

To initiate the diversification generation procedure for solutions represented by S, the first wavelength is randomly chosen and is arbitrarily assigned to group 1 (see the first three lines in Algorithm 3). In the iterative loop, the greedy function is calculated for each pair of wavelength-group for all wavelengths in U and all groups with fewer than B wavelengths. The wavelength-group combination (i^*, g^*) is chosen among those pairs for which $h_1(i, g) \geq$

Group	Wavelength	Destination				
		1	2	3	4	5
1	4	1	1	0	1	0
	8	1	0	1	1	0
	7	**1**	**0**	**0**	**1**	**1**
	?					
2	3	1	1	1	1	0
	6	0	1	0	1	0
	7	**1**	**0**	**0**	**1**	**1**
	?					

Fig. 3 Example of greedy group assignment

$h_{min} + \alpha \cdot (h_{max} - h_{min})$, where h_{min} and h_{max} are the minimum and maximum values of h_1 among all the candidate wavelength-group combinations. As is customary in GRASP, α is a tunable parameter that varies between zero (random selection) and one (deterministic selection).

```
$U \leftarrow \{1, \ldots, m\}$
$i \leftarrow Random(U)$
$S(i) \leftarrow 1$
$U \leftarrow U \backslash i$
while $U \neq \emptyset$ do
    $(i^*, g^*) \leftarrow SemiGreedy(U, S, \alpha)$
    $S(i^*) \leftarrow g^*$
    $U \leftarrow U \backslash \{i^*\}$
end while
```

Algorithm 3 Diversification generation method for solutions represented by S

A slight variant of this method is also tested in which the greedy function, $h_2(i, g)$, includes the number of bandpasses "eliminated" by adding a wavelength i to group g. That is, $h_2(i, g)$ is the difference between potential bandpasses and eliminated bandpasses. A wavelength eliminates a bandpass in a group if it adds a zero to a column that previously contained all 1s within the group. Under this variant, the h_2 values for the example in Fig. 3 are $h_2(7, 1) = 2$ and $h_2(7, 2) = 0$ because adding wavelength 7 to group 2 eliminates the potential bandpass at destination 2.

4 Improvement method

In scatter search, solutions that are either constructed by the diversification generator or created by the combination method are subjected to an improvement process that is typically based on neighborhood searches. The search in an improvement method is local, meaning that it stops as soon as no improved solution is found in the neighborhood of the current solution. Four improvement methods were developed and tested for solutions represented by an ordering O of the wavelengths:

IM1	Best insertion
IM2	Best swap
IM3	Block merging
IM4	Variable neighborhood descent

All these improvement methods treat the bandpass problem as a search for the best ordering of a set of elements. The search neighborhood in IM1 is built via insertions of each wavelength in each row. Let O be the solution before the insertion of wavelength $O(i)$ in row k, and O' the solution after the insertion:

$$\begin{aligned} O &= (O(1), \ldots, O(i-1), O(i), O(i+1), \ldots, O(k-1), O(k), O(k+1), \ldots, O(m)) \\ O' &= (O(1), \ldots, O(i-1), O(i+1), \ldots, O(k-1), O(k), O(i), O(k+1), \ldots, O(m)) \end{aligned}$$

All insertions are attempted and the best is chosen. The procedure stops when no insertion results in an increase of the number of bandpasses in the current solution. The same best-move strategy is used by IM2 with the difference that swaps define the neighborhood instead of insertions. A swap between the wavelengths in rows i and k transforms O into O' as follows:

$$\begin{aligned} O &= (O(1), \ldots, O(i-1), O(i), O(i+1), \ldots, O(k-1), O(k), O(k+1), \ldots, O(m)) \\ O' &= (O(1), \ldots, O(i-1), O(k), O(i+1), \ldots, O(k-1), O(i), O(k+1), \ldots, O(m)) \end{aligned}$$

IM3 is significantly different from IM1 and IM2. This improvement method analyzes the binary matrix associated with the current solution to identify higher-order exchanges. In particular, for each column in the matrix of the current solution it searches for two blocks of non-zero elements, one primary block with $B-2$ wavelengths and one secondary block with 2 wavelengths, where a block has non-zero elements in consecutive rows. Once these two blocks have been identified, the secondary block is merged "at the end" of the primary block by rearranging the row assignments of the wavelengths in the secondary block. This merger forms a new bandpass, however, the row reassignment of the two wavelengths in the secondary block could have eliminated one or more bandpasses in other columns. In an attempt to reestablish those bandpasses or finding new ones, an exhaustive search is performed among the rows corresponding to the new bandpass. This means that all permutations of the rows are explored in order to identify the best. The process is repeated for primary blocks with $B-3$ wavelengths, then $B-4$ wavelengths, and so on until the number of wavelengths in the primary block reaches 2. This local search procedure terminates when no improvement is possible after exploring all destinations.

Figure 4 shows a simple example of one move in IM3. As in previous examples, the gray blocks indicate the bandpasses. Figure 4a shows the current solution with an objective function value of 3. In this figure, the primary block identified by IM3 is the one with the black background, which corresponds to wavelengths (rows) 5 and 6 in destination (column) 3. The secondary block is shown as a solid bold outline, corresponding to wavelengths (rows) 2 and 3 in destination (column) 3. As prescribed by the IM3 logic, the secondary block is inserted at the end of the primary block. This means that rows in the secondary block are removed and inserted right below the rows in the primary block, and then all rows (except the first one) are shifted up. The result of this exchange is an increase in the number of bandpasses, as shown in Fig. 4b. It can be shown that no permutation of the last four rows (i.e., the ones that participated in the exchange) results in a better solution.

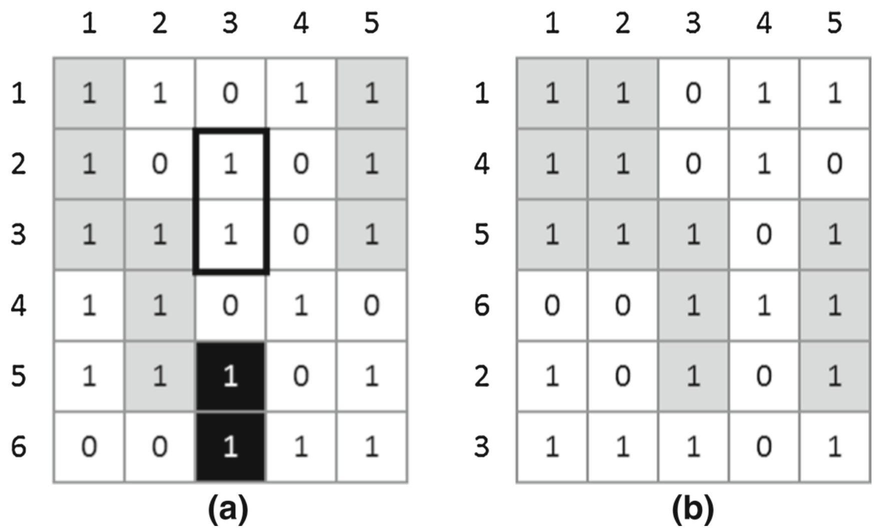


Fig. 4 Example of a move in IM3

IM4 combines IM2 and IM3 in the framework of a variable neighborhood descent (VND). VND employs multiple neighborhoods that are typically ordered in increasing level of complexity [24]. The first neighborhood is applied until no improvement is possible. At that time, the second neighborhood is explored until an improved solution is found at which point the search reverts back to the first neighborhood and the process starts again. If no improved solution is identified, then the search moves to the next neighborhood. The entire VND ends when the last (and often the most complex) neighborhood is not able to identify an improved solution. In our case, we only use two neighborhoods: IM2 and IM3. The VND starts with IM2 and it identifies the first local optimum. Then, IM3 is applied and as soon as an improved solution is found it goes back to IM2. The search keeps alternating between IM2 and IM3 until IM3 fails to find an improved solution. The selection of IM2 followed by IM3 was supported by a full-factorial experimental design where all orderings of two neighborhoods and also all orderings of the three neighborhoods were considered. The experiment yielded IM2 followed by IM3 as the best combination in terms of solution quality and exploration time.

A single improvement method was developed for solutions represented by S. This method, like IM2, is based on swaps, which preserve the feasibility of a solution. The neighborhood search is organized in such a way that the groups are ordered by increasing number of bandpasses, and hence the first group is the one with the least number of bandpasses and the last group is the one with the largest number of bandpasses. The exploration is such that it starts by attempting swaps of wavelengths from the first group with wavelengths in group 2 to G. Then, wavelengths in group 2 are exchanged with wavelengths in groups 3 to G, and so on until wavelengths from group $G - 1$ are exchanged with wavelengths in group G. If a trial swap results in an increase in the number of bandpasses, the swap is executed. In other words, a first-improving strategy is used in this neighborhood exploration. The process terminates when no more improving swaps can be identified.

5 Combination method

The combination method is a procedure that generates new solutions from a given subset of solutions. As mentioned above, our implementation considers only subsets of size 2 (i.e., pairs of reference solutions). Path relinking (PR) is used as the method to create new solutions from the subsets created by the subset generation method. PR was originally proposed in [12] and formalized in [13]. The main idea behind PR consists of creating a path (i.e., a sequence of solutions) between two or more solutions. It is called relinking because the process was conceived in the context of a solution method that performs neighborhood searches to move from one solution to the next. In such a search, solutions are connected by the paths that the procedure created to reach them. The relinking in that context refers to creating a new path to connect solutions that are typically selected due to their elite status. In scatter search, solutions in the reference set may not have historical paths that connect them. Hence, in most cases, the application of PR to two reference solutions builds the first path between them. The PR principles have been successfully applied to multiple problem classes and in a variety of methodological frameworks, including tabu search [15], scatter search [29], and GRASP [3].

Our first combination method based on PR (referred to as CM1) operates on solutions represented by an ordering O. Given a pair of reference solutions, one is denominated the initiating solution O^i and the other the guiding solution O^g. PR consists of transforming the initiating solution into the guiding solution by a sequence of moves. Consider for example the initiating solution $O^i = (5, 2, 3, 4, 6, 1)$ and the guiding solution $O^g = (5, 3, 2, 1, 4, 6)$ shown in Fig. 5. The gray cells represent wavelengths in positions that do not coincide with the positions that they occupy in the guiding solution. Swaps are used to make the transformation from the initiating solution to the guiding solution, which is achieved in three moves. The most-improving swap is chosen at each step as prescribed by a greedy path relinking process [28]. The moves are represented by arrows that are labeled with the pair of wavelengths that are swapping positions.

Initially, there are four swaps that transform the initiating solution into intermediate solutions that move toward to the guiding solutions. The (6,1) swap results in an intermediate solution with an improved objective function value. For this particular example, the relinking process could stop at this point because the objective function value of the intermediate solution matches the upper bound value of 5. However, for completeness, the figure shows that, at each step, the relinking process continues from the best intermediate solution. In this case, the solution resulting from the (4,1) swap is chosen and then the (2,3) swap completes the process.

We developed a second combination method (referred to as CM2) based on a PR strategy that has been recently suggested and that has been labeled *Exterior Path Relinking* (EPR) [6, 14]. EPR is a diversification strategy because, instead of moving the initiating solution toward the guiding solution, it searches for intermediate solutions that move away from

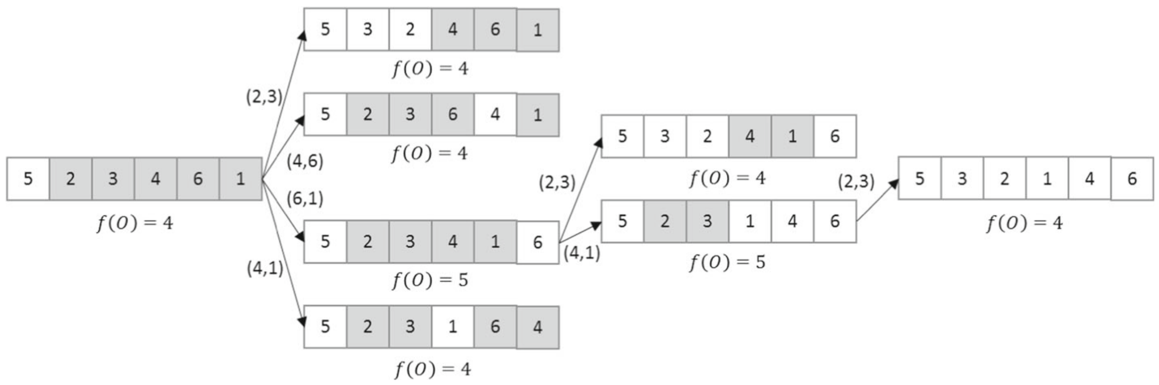


Fig. 5 Path relinking example

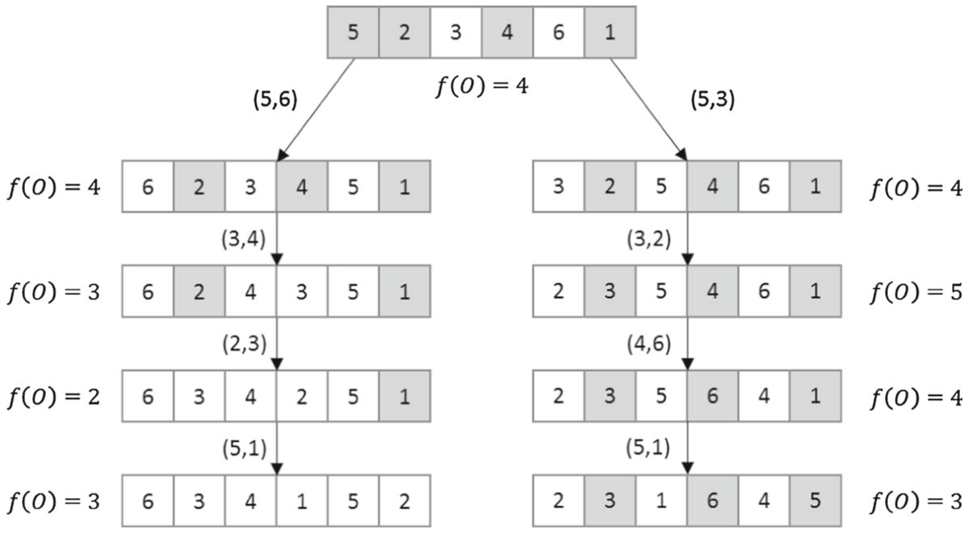


Fig. 6 Exterior path relinking example

the guiding solution. Figure 6 shows an EPR example applied to a $O^i = (5, 2, 3, 4, 6, 1)$ and the guiding solution $O^g = (5, 2, 6, 4, 3, 1)$. The process starts by randomly selecting one of the elements in O^i that occupies the same position in O^g, these are the cells with a gray background (i.e., 5, 2, 4, and 1 in the initiating solution). Assume that the random selection is 5. Then another random selection is made among those elements that, if swapped with 5, do not produce an increase in the number of wavelengths that the resulting intermediate solution will have in the same position as in the guiding solution.

In Fig. 6, two possible paths are shows, where the left path is created by first swapping 5 and 6, while the right path is initiated by the swap of 5 and 3. Clearly, this process relies heavily on randomization to create a diverse set of solutions that move away from the guiding solution. A large number of paths could be created from any O^i and O^g pair. As done in [14], we generate additional solutions by reversing the roles of the initiating and guiding solutions.

The path relinking for solutions represented by a group assignment S starts with the solution of the matching problem described in Section 2. Recall that, given two solutions, the problem matches the groups in one solution with the groups in the other solution in order to maximize the total number of wavelengths assigned to the same group (also referred to as the common elements). The optimal matching identifies the common elements between an initiating solution S^i and a guiding solution S^g. Then, swaps are performed to move S^i toward S^g. As done in CM1, the swap with the largest improvement of the objective function is chosen. The process terminates when the guiding solution is reached.

Table 1 shows the group assignments for two reference solutions for a BP2 instance with $m = 18$ and $B = 6$, and therefore $G = 3$.

The matching problem associated with the solutions in Table 1 is:

$$\begin{aligned}
\text{Maximize}\quad & x_{11} + 3x_{12} + 3x_{13} + 4x_{21} + x_{22} + x_{23} + x_{31} + 2x_{32} + 3x_{33} \\
\text{Subject to}\quad & x_{11} + x_{12} + x_{13} = 1 \\
& x_{21} + x_{22} + x_{23} = 1 \\
& x_{31} + x_{32} + x_{33} = 1
\end{aligned}$$

Table 1 Two reference solutions for a BP2 instances with $m = 18$ and $B = 6$

Solution	Group 1	Group 2	Group 3
1	1, 3, 6, 9, 12, 17	2, 5, 8, 10, 11, 18	4, 7, 13, 14, 15, 16
2	1, 4, 5, 10, 11, 18	2, 3, 6, 12, 13, 15	7, 8, 9, 14, 16, 17

An optimal solution to this problem is $x^*_{12} = x^*_{21} = x^*_{33} = 1$, resulting in the following common elements:

$$\begin{aligned}&\text{Group1} : 3, 6, 12\\&\text{Group2} : 5, 10, 11, 18\\&\text{Group3} : 7, 14, 16\end{aligned}$$

The PR process then starts by switching the labels of the first two groups in solution 2 (i.e., the original group 1 becomes group 2 and the original group 2 becomes group 1). Since $x^*_{33} = 1$, the original label for group 3 remains unchanged. Then, a sequence of swaps must be performed to move the wavelengths 1, 2, 4, 8, 9, 13, 15, and 17 from their current group in solution 1 to their group in solution 2. For instance wavelength 1 must be moved to group 2 and wavelength 2 must be moved to group 1. Therefore, a swap between wavelength 1 and 2 is considered since it moves solution 1 closer to solution 2.

6 Computational experiments

Before engaging in competitive testing, we performed a series of scientific tests to determine the contribution of the various elements that we have designed for the two problem classes defined by the solution representations. These experiments are also used to determine the best SS configuration for each problem class. Henceforth, SS1 refers to the implementation for which solutions are presented by an ordering O of the wavelengths and therefore equipped to tackle BP1 and MBP instances. The SS2 procedure refers to the scatter search for which solutions are represented as an assignment S of wavelengths to groups and that it is configured to solve BP2 instances. Table 2 summarizes the SS components for each SS implementation.

Our scientific testing focuses on determining the best combination of components for each SS. All components were programmed in Java 8 and the experiments were executed on an Intel Core i7 2600 (3.4 GHz) processor with 4 GB of RAM. We use the Library of Bandpass Problems (BPLIB)[3] for BP1 and BP2 and the MBP library (MBPLIB)[4] for the MBP. The BPLIB contains 90 instances, 45 referred to as OS and 45 referred to as BKS. The OS instances were constructed in such a way that the optimal solutions are known for BP1 but not for BP2. The BKS problems were randomly generated and optimal solutions are not known for either BP1 or BP2. There are 45 instances with known optimal solutions in the MBP library. We use the BKS instances for the scientific experimentation and reserve the OS and the MBP instances for the competitive testing. This means that the experiments used to configure SS1 include only BP1 instances and that the configuration is expected to generalize to the MBP instances.

[3] http://sci.ege.edu.tr/~math/BandpassProblemsLibrary/.

[4] http://fen.ege.edu.tr/arifgursoy/mbpopt/.

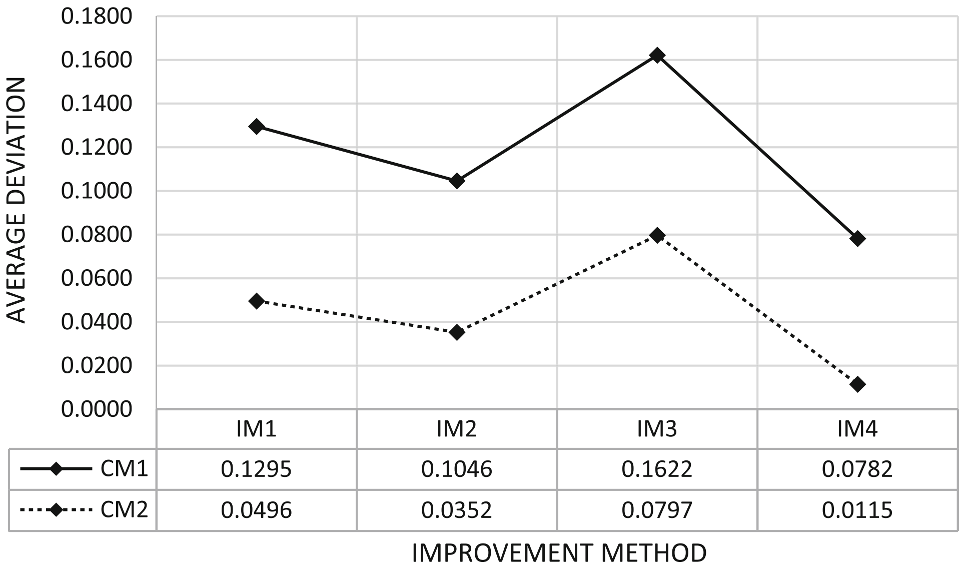
Table 2 Summary of SS components for the two bandpass problem classes

Component	SS1 (BP1 and MBP)	SS2 (BP2)
Solution representation	Ordering of wavelengths (O)	Assignment of wavelengths to groups (S)
Distance measure	Positional distance	Number of wavelengths assigned to different groups
Subset generation method	All pairs of reference solutions	All pairs of reference solutions
Initial reference set	Half of the reference solutions chosen by quality and half by diversity	Half of the reference solutions chosen by quality and half by diversity
Reference set update	Eligible new trial solution replaces the reference solution closest to it	Eligible new trial solution replaces the reference solution closest to it
Diversification generation	Semi-greedy (random selection of a wavelength followed by best insertion)	GRASP construction with two greedy functional forms (h_1 and h_2)
Improvement method	Four variants (IM1 to IM4) based on exchanges of positions	First-improvement swap of group assignments
Combination method	PR (CM1) and EPR (CM2) based on swapping positions	PR based on swapping group assignments

A two-factor factorial design is employed to configure SS1 by fixing the population size $|P|$ to 100 and the reference set size β to 10, the default values suggested in the literature. Also, the stopping criterion (see last line of Algorithm 1) is set as the iteration when no trial solution is admitted to the reference set. Since there are four improvement methods and two combination methods, the factorial design consists of 8 treatment combinations with 45 replications each. For this experiment, the deviation from the best-known solution is used as the response variable. The deviation is calculated as $\left(\frac{f^*-f}{f^*}\right)$, where f^* is the objective function value of the best-known solution and f is the objective function value of the solution obtained by the treatment combination. The ANOVA associated with the experiment revealed that both main effects are significant, with p values smaller than 10^{-9}. However, the experiment did not detect a significant interaction effect between CM and IM, as indicated by a p value of 0.881. Figure 7, a plot of the average deviation values (y-axis) for each improvement method (x-axis), confirms this finding, where it can be observed that the CM1 line (solid) does not cross the CM2 line (dotted).

Figure 7 shows that CM2 is more effective than CM1 regardless of the IM. It also shows that the response obtained by the IM alternatives does not change with the choice of CM. This means that the selection of the CM and the IM could have been done independently. The experiment indicates that the best treatment combination and therefore the SS1 configuration that is expected to yield the best results is CM2 with IM4.

In the process of conducting this experiment, 39 solutions were found that improved upon the best-known solution currently published in the BPLIB website. We point out that the website does not indicate how those solutions were found given that neither [1] nor [2] reports results on these instances. The objective function values of the new best-known solutions, which improve upon the current best-known solutions by an average of 16.2%, are shown in “Appendix 1”.



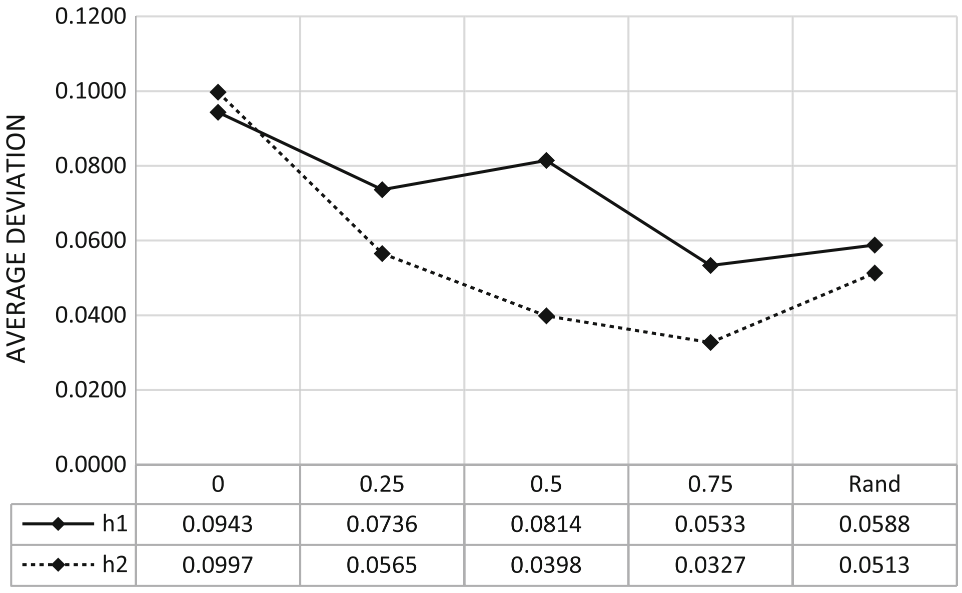
	IM1	IM2	IM3	IM4
CM1	0.1295	0.1046	0.1622	0.0782
CM2	0.0496	0.0352	0.0797	0.0115

Fig. 7 Plot of average response for all (CM, IM) treatment combinations

To configure SS2 we also fix $|P|$ and β to their default values and execute the procedure (Algorithm 1) a total of 10 times on each instance. Recall that the standard SS stopping criterion (which we also use for SS1) terminates the search when the reference set does not change, that is, when in a given iteration, no trial solution is admitted to the reference set. In SS2, the procedure is allowed to rebuild the reference set by seeding it with the incumbent solution. That is, the incumbent solution is placed in the new reference set first and the remaining $b - 1$ solutions are added following the construction processes within SS2. A two-factor factorial design is employed to study the effects of various α values and the greedy functions h_1 and h_2. The α values included in the experiment are 0, 0.25, 0.5, and 0.75. We also include a variant where the value of α is selected at random during the search. This factorial design has a total of 10 treatment combinations and 45 replications. We again use the average deviation—calculated as described above—as the response variable. The ANOVA yielded p values of 0.03 and 0.00015 for the main effects associated with the greedy functions and the α values, respectively. Hence, we consider both main effect significant. The interaction effect between the two factors has a p value of 0.366. Therefore, we conclude that there is no significant interaction between the choice of h and α. Figure 8 is a graphical representation of the average deviations obtained by each treatment combination.

It can be concluded from Fig. 8 that h_2 is generally more effective than h_1, except when $\alpha = 0$. Also in general, increasing the value of α—thus making the construction less random—improves performance (except for h_1 and $\alpha = 0.5$). Allowing α to change randomly during the search seems to be better than using small α values. Given these results, the combination h_2 and $\alpha = 0.75$ is selected.

We now turn our attention to the competitive testing for which we use the OS and the MBP instances. The competitors for BP1 are the DSR (Different Start Rows) method of Babayev, Bell, and Nuriyev [1] and the four variants of a genetic algorithm (GA1 to GA4) proposed by Berbeler and Gürsoy [2]. Results of the OS instances associated with DSR and the GA variants were obtained from [1,2] respectively. The articles do not report computational time. For future reference, we report that the SS1 results shown in Table 3 were obtained in an average of 875 seconds. In Table 3, OF Value refers to the average objective function value and #Opt indicates the number of optimal solutions that each competitor is able to find out of the 45 in the set. The column labeled GAP contains the average relative optimality gap



	0	0.25	0.5	0.75	Rand
h1	0.0943	0.0736	0.0814	0.0533	0.0588
h2	0.0997	0.0565	0.0398	0.0327	0.0513

Fig. 8 Plot of average response for all (h, α) treatment combinations

Table 3 Summary of competitive testing for BP1

Procedure	OF value	GAP	#Opt
DSR	37.00	0.3407	0
GA1	40.53	0.2516	3
GA2	40.53	0.2550	3
GA3	40.24	0.2613	2
GA4	40.29	0.2575	3
SS1	46.07	0.1027	16

Table 4 Friedman test ranking

Procedure	Ranking
SS1	1.12
GA2	3.33
GA1	3.34
GA4	3.61
GA3	3.73
DSR	5.86

(i.e., the absolute gap divided by the optimal objective function value and averaged over all instances).

The results in Table 3 indicate that there is little difference in performance among the GA variants. DSR produces results that are inferior to those produced by all other methods while SS1 dominate all competitors. In order to support this conclusion, we performed a Friedman test. This test assigns, for each instance in the test set, the value of 1 to the procedure that obtains the best result, the value 2 to the second best, and so forth. The average ranking is then calculated for each procedure and a p value is calculated to assess the strength of the ranking. The rankings shown in Table 4 result in a p value of less than 0.001.

The next competitive test compares the performance of SS1 with four variants of a genetic algorithm (GA1 to GA4) proposed by Gürsoy and Nuriyev [17] for MBP. The 45 MBPLIB

Table 5 Summary of competitive testing for MBP instances

Procedure	OF value	GAP
GA1	20.38	0.5153
GA2	19.49	0.5326
GA3	19.69	0.5324
GA4	18.76	0.5458
SS1	30.09	0.2349

instances are employed for this comparison and the summary of the results is presented in Table 5. Gürsoy and Nuriyev [17] do not report solution times but for future reference we note that the SS1 solutions summarized in Table 5 were obtained in an average of 267 seconds. Table 5 does not include the #Opt column because none of the procedures is capable of finding a single optimal solution. This is in agreement with the large optimality gap shown in Table 5. Nonetheless, the average objective function value of the SS1 solutions is at least 47.6% better than the average objective function values of all the GA variants, resulting in optimality gap that is less than half of those corresponding to the GAs. Although this is a significant improvement over the existing GA procedures, it is clear that the strategies designed for BP1 and embedded in SS1 lose some of their effectiveness when applied to MBP instances. Additional strategies that directly exploit the structure of the MBP instances are necessary to produce improved outcomes.

Our third and last competitive testing compares the performance of SS2 to LocalSolver, a commercial metaheuristic optimizer, and Gurobi, a commercial MIP solver. To the best of our knowledge, there is no specialized procedure for BP2 in the literature. The Gurobi solutions were found with the IP formulation presented in [1], and the solver was run with its default optimization settings. This means that the search took advantage of the 8 processors on the machine that was used for testing. The LocalSolver model that we used is included in "Appendix 2". The model is a simplified version of the IP formulation of BP2 presented in [1]. The LocalSolver model is simpler because it requires of only one set of binary variables, namely, the `x[i][g]` variables that indicate whether or not wavelength i is assigned to group g. We perform two experiments, one with a relatively short search time limit of 600 seconds per instance and one with a longer search horizon of 3000 seconds. As described above, SS2 is set up to rebuild the reference set every time the stopping criterion is satisfied and therefore is capable of continuing the search until any specified time limit. The results of this experiments are summarized in Table 6.

The GAP column in Table 6 is calculated against the best-known solution for each instance, 19 of which are optimal, as confirmed by the 3000-s Gurobi runs. The #Opt is the number of optimal solutions, out of the 19 that are known, that each procedure is able to match. The #Best column contains the number of best-known solutions matched by each procedure. The #Best values are divided into two groups, the number of outright best solutions (#Outright) and the number of ties with at least one of the competing procedures (#Ties). The #Worst column indicates the number of times that the solution procedure yielded the worst solution

The following observations can be made from the analysis of the results in Table 6:

- SS2 exhibits the most robust behavior of the three competing procedures. Its #Best values are at the top in both the short and the long runs. At the same time, its #Worst values are the lowest in both runs.

Table 6 Summary of results for the competitive testing on BP2 instances

Procedure	OF value	GAP	#Opt	#Best	#Outright	#Ties	#Worst
Time limit of 600 s							
Gurobi	31.044	0.0434	19	25	1	24	18
LocalSolver	32.577	0.0644	11	31	7	24	12
SS2	32.711	0.0119	17	33	1	32	2
Time limit of 3000 s							
Gurobi	31.733	0.0291	19	27	0	27	17
LocalSolver	32.778	0.0516	12	37	8	29	8
SS2	32.956	0.0050	19	37	1	36	2

- The GAP values of 0.0119 and 0.005 for SS2 supports the robustness argument. That is, SS2 produces high-quality solutions in most runs and avoids arbitrarily inferior solutions. SS2's worst deviation from the best-known solutions is 0.067 in both, short and long runs.
- LocalSolver is highly competitive and a viable option for finding high-quality solutions to BP2 instances. This solution alternative is particularly attractive when one takes into consideration the development effort of creating a customized solution methods such as SS2. The only caveat is the variability of the solution quality produced by Local Solver. Note that for the long runs LocalSolver has the same number of outright best solutions (8) as the number of worst solutions. This variability causes LocalSolver to have the worst average deviation.
- Gurobi's GAP values are reasonably low because, even though it has the highest #Worst counts in both runs, its deviations from the best-known solutions are never more than 0.2

Additional search strategies could widen the gap between the performance of a specialized procedure like SS2 for the BP2 and a commercial software LocalSolver. For instance, strategic oscillation proved critical in boosting performance in a tabu search for a grouping problem [11].

7 Conclusions

The bandpass problem is somewhat new to the OR literature in the sense that, although it was originally introduced at a conference more than 10 years ago, the first publication did not appear until 2009. Several versions of the problem have been introduced in subsequent publications along with additional test data. One of our goals in this project was to gain understanding of the current state of knowledge associated with the bandpass problem and to identify avenues for advancing this area. Through our investigation, we identified two problem classes and a total of three problem variants within those classes. We then designed a common SS solution framework and developed the individual strategies for each problem class.

We performed meticulous scientific testing that provided some interesting insights on the behavior and interaction of the search components. These experiments produced new benchmarks that will help future researchers test solution methods that could prove even more effective than those described here. We were also able to show that although BP1 and MBP belong to the same problem class, the effectiveness of the search strategies designed for

BP1 diminish when applied to MBP instances. Nonetheless, competitive testing showed that the proposed SS1 method applied (without customization) to both BP1 and MBP instances produces results that are significantly better than those produced by the existing procedures. Finally, our experiments show that the adaptation of the SS to BP2 exhibits a robust behavior when compared to commercial software.

Acknowledgments This research was partially supported by the Ministerio de Economía y Competitividad of Spain (Project Number TIN2015-65460-C2-P) and the Comunidad de Madrid (Project Number S2013/ICE-2894).

Appendix 1

Table 7 shows the objective function values of the new best-known solutions for the BKS instances. A dash indicates that no solution better than the current best known was found in our experimentation.

Table 7 New best solutions found during the scientific experimentation for SS1

Instance	Previous	New	%Improve	Instance	Previous	New	%Improve
P01-A1B8	10	11	10.00	P24-A10B16	10	12	20.00
P02-A1B16	3	–	–	P25-A11B5	54	59	9.26
P03-A2B8	14	16	14.29	P26-A11B8	23	29	26.09
P04-A2B16	5	6	20.00	P27-A11B16	6	9	50.00
P05-A3B8	20	23	15.00	P28-A12B5	78	83	6.41
P06-A3B16	8	–	–	P29-A12B8	37	40	8.11
P07-A4B5	27	30	11.11	P30-A12B16	9	13	44.44
P08-A4B8	12	15	25.00	P31-A13B5	71	–	–
P09-A4B16	4	–	–	P32-A14B5	83	87	4.82
P10-A5B5	40	41	2.50	P33-A15B8	53	56	5.66
P11-A5B8	18	21	16.67	P34-A16B16	29	35	20.69
P12-A5B16	5	7	40.00	P35-A17B5	104	106	1.92
P13-A6B5	62	67	8.06	P36-A18B5	215	223	3.72
P14-A6B8	31	34	9.68	P37-A19B16	71	–	–
P15-A6B16	9	11	22.22	P38-A20B5	132	134	1.52
P16-A7B8	17	20	17.65	P39-A22B8	74	78	5.41
P17-A8B8	25	27	8.00	P40-A22B25	17	22	29.41
P18-A8B16	9	10	11.11	P41-A23B5	161	–	–
P19-A9B5	44	47	6.82	P42-A24B8	93	95	2.15
P20-A9B8	18	23	27.78	P43-A25B16	51	53	3.92
P21-A9B16	5	8	60.00	P44-A26B5	245	248	1.22
P22-A10B5	62	68	9.68	P45-A27B5	28	41	46.43
P23-A10B8	33	35	6.06				

Appendix 2

LocalSolver input and model functions for the experiments with BP2.

```
function input()
{
  usage = "\nUsage: localsolver BP2.lsp "
        + "B=BandpassNumber fileName=dataFile "
        + "[lsTimeLimit=timeLimit] [lsVerbosity=0 or 1]\n";

  if (fileName == nil) error(usage);
  dataFile = openRead(fileName + ".txt");
  m = readInt(dataFile);
  n = readInt(dataFile);
  a[i in 1..m][j in 1..n] = readInt(dataFile);
  close(dataFile);
  G = floor(m/B);
  b = m-G*B;
  GP = (b > 0) ? G+1 : G;
}

function model()
{
  // x[i][g] equal to 1 if wavelength i is assigned to group g
  x[1..m][1..GP] <- bool();

  // each group from 1 to G has exactly B rows
  for [g in 1..G] constraint sum[i in 1..m](x[i][g]) == B;

  // group G+1 if it exists has b rows
  if (GP > G) constraint sum[i in 1..m](x[i][GP]) == b;

  // each row is included in exactly one group
  for [i in 1..m] constraint sum[g in 1..GP](x[i][g]) == 1;

  // y[j][g] is equal to 1 if there is a bandpass in destination j of group g
  y[j in 1..n][g in 1..G] <- sum[i in 1..m] (a[i][j]*x[i][g]) == B;
  if (GP > G) y[j in 1..n][GP] <- sum[i in 1..m] (a[i][j]*x[i][GP]) == b;

  // objective function is the sum of bandpasses
  obj <- sum[g in 1..GP][j in 1..n] (y[j][g]);
  maximize obj;
}
```

References

1. Babayev, V., Bell, G.I., Nuriyev, U.G.: The bandpass problem: combinatorial optimization and library of problems. J. Combin. Optim. **18**(2), 151–172 (2009)
2. Berberler, M.E., Gürsoy, A.: Genetic algorithm approach for bandpass problem. In: Kasimbeyli R., Dinçer C., Özpeynirci S., Sakalauskas L., (eds.) 24th Mini EURO Conference on Continuous Optimization and Information-Based Technologies in the Financial Sector (MEC EurOPT 2010), pp. 201–206. Vilnius Gediminas Technical University Publishing House Technika, İzmir (2010)
3. Campos, V., Martí, R., Sánchez-Oro, J., Duarte, A.: GRASP with path relinking for the orienteering problem. J. Oper Res. Soc. **65**(2), 1800–1813 (2014)
4. Das, A., Roberts, M.: Metric distances of permutations. www.cra.org/Activities/craw_archive/dmp/awards/2004/Das/paper.ps. (2004)
5. Duarte, A., Martí, R., Glover, F., Gortázar, F.: Hybrid scatter tabu search for unconstrained global optimization. Ann. Oper. Res. **183**(1), 95–123 (2009)
6. Duarte, A., Sánchez-Oro, J., Resende, M.G.C., Glover, F., Martí, R.: GRASP with exterior path relinking for differential dispersion minimization. Inf. Sci. **296**(1), 46–60 (2015)

7. Duarte, A., Martí, R.: Tabu search and grasp for the maximum diversity problem. Eur. J. Oper. Res. **178**(1), 71–84 (2007)
8. Feo, T.A., Resende, M.G.C.: A probabilistic heuristic for a computationally difficult set covering problem. Oper. Res. Lett. **8**, 67–71 (1989)
9. Feo, T.A., Resende, M.G.C.: Greedy randomized adaptive search procedures. J. Glob. Optim. **6**, 109–133 (1995)
10. Gallego, M., Duarte, A., Laguna, M., Martí, R.: Hybrid heuristics for the maximum diversity problem. Comput. Optim. Appl. **44**(3), 411–426 (2009)
11. Gallego, M., Laguna, M., Martí, R., Duarte, A.: Tabu search with strategic oscillation for the maximally diverse grouping problem. J. Oper. Res. Soc. **64**(5), 724–734 (2013)
12. Glover, F.: Heuristics for integer programming using surrogate constraints. Decis. Sci. **8**(1), 156–166 (1977)
13. Glover, F.: A template for scatter search and path relinking. Lecture Notes in Computer Science, **1363**, 1–51 (1998)
14. Glover, F.: Exterior path relinking for zero-one optimization. Int. J. Appl. Metaheur. Comput. **5**(3), 1–8
15. Glover, F., Laguna, M.: Tabu Search. Springer, New York (1997)
16. Goralski, W.J.: SONET, a Guide to Synchronous Optical Networks. McGraw-Hill, New York (1997)
17. Gürsoy, A., Nuriyev, U.G.: Genetic algorithm for multi bandpass problem and library of problems. In: Proceedings of the IV International Conference on Problems of Cybernetics and Informatics (PCI), Section I, Information and Communication Technologies, Baku, Azerbaijan, Sept 12–14, 134–138 (2012)
18. Kaminow, I.P., Saleh, A.A.M., Thomas, R.E., Chan, V.W.S., Stevens, M.L.: A wideband all-optical WDM network. IEEE J. Sel. Areas Commun. **14**(5), 780–799 (1996)
19. Kurt, M., Kutucu, H., Gürsoy, A., Nuriyev, U.: The optimization of the bandpass lengths in the multi-bandpass problem. In: Proceedings of the Seventh International Conference on Management Science and Engineering Management: Focused on Electrical and Information Technology, Xu, J., Fry, J. A., Lev, B., Hajiyev, A. (eds.) (1): 115–123. Springer, Berlin (2014)
20. Laguna, M., Martí, R.: Scatter Search: Methodology and Implementations in C. Kluwer, Boston (2003)
21. Li, Z., Lin, G.: The three column bandpass problem is solvable in linear time. Theor. Comput. Sci. **412**(4–5), 281–299 (2011)
22. Lin, G.: On the bandpass problem. J. Combin. Optim. **22**(1), 71–77 (2011)
23. Martí, R., Duarte, A., Laguna, M.: Advanced scatter search for the max-cut problem. INFORMS J. Comput. **21**(1), 26–38 (2009)
24. Mladenović, N., Hansen, P.: Variable neighborhood search. Comput. Oper. Res. **24**(11), 1097–1100 (1997)
25. Nuriyev, U.G., Kutucu, H., Kurt, M.: Mathematical models of the bandpass problem and OrderMatic computer game. Math. Comput. Model. **53**, 1282–1288 (2011)
26. Pantrigo, J.J., Martí, R., Duarte, A., Pardo, E.G.: Scatter search for the cutwidth minimization problem. Ann. Oper. Res. **199**(1), 285–304 (2011)
27. Ramaswami, R., Sivarajan, K., Sasaki, G.: Optical Networks: A Practical Perspective. Morgan Kaufmann, San Francisco (1998)
28. Resende, M., Martí, R., Gallego, M., Duarte, A.: GRASP and path relinking for the max–min diversity problem. Comput. Oper. Res. **37**, 498–508 (2010)
29. Sánchez-Oro, J., Laguna, M., Duarte, A., Martí, R.: Scatter search for the profile minimization problem. Networks. **65**(1), 10–21 (2015)

Additional publications derived from this research

The research in the problems and metaheuristics described in this dissertation has resulted in additional publications related to them, with respect to the methodology used, the problem considered or the adaptation of previously proposed ideas. For the sake of completeness, this chapter is intended to present the additional articles that have been published as a results of the research. The articles are sorted in ascending order with respect to the publication date.

Journal articles indexed in the Journal of Citation Reports

1. A. Duarte, L. F. Escudero, R. Martí, N. Mladenovic, J. J. Pantrigo, and J. Sánchez-Oro, *Variable neighborhood search for the Vertex Separation Problem*, Computers & Operations Research, vol. 39, pp. 3247–3255, 2012 (Chapter 7).

   - DOI: `https://doi.org/10.1016/j.cor.2012.04.017`
   - Impact Factor: 1.909
   - Categories:
     - Operations Research and Management Science, 10 / 79 (Q1)
     - Engineering, industrial, 6 / 44 (Q1)
     - Computer Science, interdisciplinary applications, 33 / 102 (Q2)

2. J. Sánchez-Oro, D. Fernández-López, R. Cabido, A. S. Montemayor, and J. J. Pantrigo, *Radar-based road-traffic monitoring in urban environments*, Digital Signal Processing, vol. 23, issue 1, pp. 364 - 374, 2013.
    - DOI: `https://doi.org/10.1016/j.dsp.2012.09.012`
    - Impact Factor: 1.495
    - Categories:
        - Engineering, electrical and electronic, 111 / 255 (Q2)
3. A. Duarte, M. Laguna, R. Martí, and J. Sánchez-Oro, *Optimization Procedures for the Bipartite Unconstrained 0-1 Quadratic Programming Problem*, Computers & Operations Research, vol. 51, pp. 123-129, 2014.
    - DOI: `https://doi.org/10.1016/j.cor.2014.05.019`
    - Impact Factor: 1.861
    - Categories:
        - Operations Research and Management Science, 19 / 81 (Q1)
        - Engineering, industrial, 9 / 43 (Q1)
        - Computer Science, interdisciplinary applications, 29 / 102 (Q2)
4. V. Campos, R. Martí, J. Sánchez-Oro, and A. Duarte, *GRASP with Path Relinking for the Orienteering Problem*, Journal of the Operational Research Society, vol. 65, issue 12, pp. 1800-1813, 2014.
    - DOI: `https://doi.org/10.1057/jors.2013.156`
    - Impact Factor: 0.953
    - Categories:
        - Operations Research and Management Science, 50 / 81 (Q3)
5. 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, vol. 52, issue B, pp. 209-219, 2014 (Chapter 9).
    - DOI: `https://doi.org/10.1016/j.cor.2013.11.008`
    - Impact Factor: 1.861
    - Categories:
        - Operations Research and Management Science, 19 / 81 (Q1)
        - Engineering, industrial, 9 / 43 (Q1)

    - Computer Science, interdisciplinary applications, 29 / 102 (Q2)

6. J.Sánchez-Oro, M. Laguna, A. Duarte, and R. Martí, *Scatter search for the profile minimization problem*, Networks, vol. 65, issue 1, pp. 21, 2015 (Chapter 8).

    - DOI: `https://doi.org/10.1002/net.21571`
    - Impact Factor: 0.943
    - Categories:
        - Computer Science, hardware & architecture, 29 / 51 (Q3)
        - Operations Research & management science, 54 / 82 (Q3)

7. A. Duarte, J. Sánchez-Oro, M. G. C. Resende, F. Glover, and R. Martí, *GRASP with Exterior path relinking for differential dispersion minimization*, Information Sciences, vol. 296, pp. 46–60, 2015.

    - DOI: `https://doi.org/10.1016/j.ins.2014.10.010`
    - Impact Factor: 3.364
    - Categories:
        - Computer Science, Information Systems, 8 / 143 (Q1)

8. A. Duarte, J. J. Pantrigo, J. Sánchez-Oro, and E. G. Pardo, *Parallel VNS strategies for the Cutwidth Minimization Problem*, IMA Journal of Management Mathematics, vol. 27, issue 1, pp. 55-73, 2016 (Chapter 10).

    - DOI: `https://doi.org/10.1093/imaman/dpt026`
    - Impact Factor: 0.878
    - Categories:
        - Operations Research & management science, 59 / 82 (Q3)
        - Mathematics, interdisciplinary applications, 65 / 101 (Q3)

9. J. Sánchez-Oro, A. Duarte, S. Salcedo-Sanz, *Robust total energy demand estimation with a hybrid Variable Neighborhood Search – Extreme Learning Machine algorithm*, Energy Conversion and Management, vol. 123, pp. 445-452, 2016.

    - DOI: `https://doi.org/10.1016/j.enconman.2016.06.050`
    - Impact Factor: 4.801
    - Categories:

  - Thermodynamics, 2 / 58 (Q1)
  - Energy & Fuels, 12 / 88 (Q1)
  - Mechanics, 3 / 135 (Q1)
  - Physics, nuclear, 3 / 21 (Q1)

10. J. Sánchez-Oro, N. Mladenović, and A. Duarte, *General Variable Neighborhood Search for computing graph separators*, Optimization Letters, In press, 2016.

    - DOI: `https://doi.org/10.1007/s11590-014-0793-z`
    - Impact Factor: 1.019
    - Categories:
      - Operations Research & management science, 51 / 82 (Q3)
      - Mathematics, applied, 87 / 254 (Q2)

11. J. Sánchez-Oro, M. Laguna, R. Martí, A. Duarte, *Scatter search for the bandpass problem*, Journal of Global Optimization, In press, 2016 (Chapter 11).

    - DOI: `https://doi.org/10.1007/s10898-016-0446-0`
    - Impact Factor: 1.219
    - Categories:
      - Operations Research & management science, 42 / 82 (Q3)
      - Mathematics, applied, 63 / 254 (Q1)

12. B. Menéndez, E.G. Pardo, J. Sánchez-Oro, A. Duarte, *Parallel variable neighborhood search for the min–max order batching problem*, International Transactions in Operational Research, In press, 2016.

    - DOI: `https://doi.org/10.1111/itor.12309`
    - Impact Factor: 1.255
    - Categories:
      - Operations Research & management science, 39 / 82 (Q2)

13. J.D. Quintana, J. Sánchez-Oro, and A. Duarte, *Efficient Greedy Randomized Adaptive Search Procedure for the Generalized Regenerator Location Problem*, International Journal of Computational Intelligence Systems, In press, 2016.

    - DOI: Not available

- Impact factor: 0.391
- Categories:
  - Computer Science, artificial intelligence, 125 / 130 (Q4)
  - Computer Science, interdisciplinary applications, 101 / 104 (Q4)

Journal articles not indexed in the Journal of Citation Reports and book chapters

1. J. Sánchez-Oro, M. Sevaux, A. Rossi, R. Martí, and A. Duarte, *Solving dynamic memory allocation problems in embedded systems with parallel variable neighborhood search strategies*, Electronic Notes in Discrete Mathematics, vol 47, pp. 85-92, 2015.
2. J. Sánchez-Oro and A. Duarte, *Beyond Unfeasibility: Strategic Oscillation for the Maximum Leaf Spanning Tree Problem*, Lecture Notes in Computer Science, vol. 9422, pp. 322-331, 2015.
3. J. Sánchez-Oro, D. Fernández-López, R. Cabido, A. S. Montemayor, and J. J. Pantrigo, *Urban Traffic Surveillance in Smart Cities Using Radar Images*, Lecture Notes in Computer Science, vol. 7931, pp. 296-305, 2015.
4. J. Sánchez-Oro and A. Duarte, *An experimental comparison of Variable Neighborhood Search variants for the minimization of the vertex-cut in layout problems*, Electronic Notes in Discrete Mathematics, vol. 39, pp. 59-66, 2012.
5. J. Sánchez-Oro and A. Duarte, *GRASP with Path Relinking for the SumCut Problem*, International Journal of Combinatorial Optimization Problems and Informatics, vol. 3, issue 1, pp. 3-11, 2012.

Works presented in national and international conferences

1. J. Sánchez-Oro and A. Duarte, *Beyond Unfeasibility: Strategic Oscillation for the Maximum Leaf Spanning Tree Problem*, Conferencia de la Asociación

Española para la Inteligencia Artificial (CAEPIA), Albacete (Spain), 2015.

2. J. Sánchez-Oro, M. Sevaux, A. Rossi, R. Martí, and A. Duarte, *Efficient memory use in embedded systems: a Variable Neighborhood Search approach*, 15th EU/ME Workshop, Madrid (Spain), 2015.

3. J. Sánchez-Oro, M. Sevaux, A. Rossi, R. Martí, and A. Duarte, *Solving dynamic memory allocation problems in embedded systems with parallel variable neighborhood search strategies*, 3rd International Conference on Variable Neighborhood Search, Djerba (Tunisia), 2014.

4. D. Concha, D. Fernández-López, J. Sánchez-Oro, A. S. Montemayor, J. J. Pantrigo, and A. Duarte, *Búsqueda de Vecindad Variable secuencial y paralela: una aplicación al problema de la maximización del corte*, IX Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB), Madrid (Spain), 2013.

5. D. Fernández-López, J. Sánchez-Oro, and A. Duarte, *Cálculo de separadores de grafos utilizando búsqueda de vecindad variable reducida*, IX Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB), Madrid (Spain), 2013.

6. J. Sánchez-Oro, A. Duarte, M. Laguna, and R. Martí *Scatter Search aplicado al problema de la minimización del profile de matrices y grafos*, IX Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB), Madrid (Spain), 2013.

7. J. Sánchez-Oro and A. Duarte, *Comparison of different Variable Neighborhood Search strategies for solving the Bipartite unconstrained 0-1 Quadratic programming Problem*, XI Balkan Conference On Operational Research - BALCOR, Belgrade (Serbia), 2013.

8. D. Fernández-López, J. Sánchez-Oro and A. Duarte, *Computing Graph Separators with Variable Neighborhood Search*, XI Balkan Conference On Operational Research - BALCOR, Belgrade (Serbia), 2013.

9. J. Sánchez-Oro, D. Fernández-López, R. Cabido, A. S. Montemayor, and J. J. Pantrigo, *Urban Traffic Surveillance in Smart Cities Using Radar Images*, 5th. International work-conference on the interplay between natural and artificial computation - IWINAC, Mallorca (Spain), 2013.

10. J. Sánchez-Oro, M. Laguna, A. Duarte, R. Martí, *Scatter Search for the Profile Minimization Problem*, INFORMS Computing Society Conference, Santa Fe (USA), 2013.

11. J. Sánchez-Oro, S. Montalvo, A. S. Montemayor, J. J. Pantrigo, A. Duarte, V. Fresno, and R. Martínez, *URJC & UNED at ImageCLEF 2013 Photo Annotation task*, Valencia (Spain), 2013.

12. J. Sánchez-Oro and A. Duarte, *An experimental comparison of Variable Neighborhood Search variants for the minimization of the vertex-cut in layout problems*, EURO Mini Conference XXVIII on Variable Neighbourhood Search, Herceg Novi (Montenegro), 2012.

13. J. Sánchez-Oro and A. Duarte, *GRASP con Path Relinking para el problema del SumCut*, VIII Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados - MAEB, Albacete (Spain), 2012.

14. J. Sánchez-Oro, S. Montalvo, A. S. Montemayor, J. J. Pantrigo, A. Duarte, V. Fresno, and R. Martínez, *URJC & UNED at ImageCLEF 2012 Photo Annotation task*, Rome (Italy), 2012.

15. J. Sánchez-Oro and A. Duarte, *GRASP for the SumCut Problem*, XVII International Congress on Computer Science Research, Morelia (Mexico), 2011.

Part III

Resumen y conclusiones

La optimización es una disciplina que trata de encontrar solución a problemas de la vida cotidiana. Numerosos problemas de interés en diferentes áreas científicas y tecnológicas pueden ser enunciados como problemas de optimización. Aquellos problemas en los que, existiendo un gran número de soluciones posibles y una clara forma de comparación entre ellas, se pretende encontrar la solución óptima (la mejor de todas) se consideran problemas de optimización.

Esta tesis se centra en un conjunto de problemas de optimización combinatoria, los cuales son un tipo de problemas de optimización cuyas soluciones están formadas por números enteros. Para la mayoría de los problemas con interés práctico no se conoce ningún algoritmo capaz de resolverlo en tiempo polinómico. Sin embargo, debido a su interés, es necesario disponer de técnicas eficientes que sean capaces de abordarlos. Existen técnicas exactas que encuentran la solución óptima dado un problema, pero necesitan un tiempo de cómputo demasiado elevado para ser viable cuando el tamaño del problema aumenta. Por otra parte, las técnicas aproximadas no garantizan la obtención de la solución óptima, pero proporcionan soluciones de presumible alta calidad en un tiempo de cómputo razonable. Entre las técnicas aproximadas destacan los algoritmos heurísticos y los metaheurísticos, los cuales son el centro de atención de la presente tesis.

El conjunto de problemas tratado en esta tesis se compone de problemas de ordenaciones lineales, los cuales han cobrado gran importancia en los últimos años debido a sus numerosas aplicaciones prácticas. Estos problemas, originalmente propuestos para el diseño de circuitos integrados, consisten en la proyección de un grafo original sobre un grafo destino, de forma que los vértices del grafo original se asocia los del grafo destino, creando un camino en el grafo destino por cada arista del original. La calidad de la proyección reside en la expansión, la congestión y la carga del grafo destino.

Los problemas considerados se centran en el caso en el cual un grafo de n vértices se debe proyectar sobre un grafo camino que presente una carga igual a uno. Desde un punto de vista práctico, este grafo camino se representa alineando sus vértices sobre una línea horizontal, donde cada vértice v se localiza en la posición $\varphi(v)$, siendo φ una posible ordenación lineal del grafo original.

Gran parte de los problemas sobre ordenaciones lineales con aplicaciones prácticas en diferentes áreas son difíciles de resolver. En esta tesis se han considerado cuatro problemas de ordenaciones lineales, en concreto: *Vertex Separation Problem*, *Profile Minimization Problem*, *Cutwidth Minimization Problem*, y *Bandpass Problem.*

Los algoritmos propuestos para cada uno de los problemas han obtenido los mejores resultados encontrados en el estado del arte para cada uno de ellos, convirtiéndose en referencia para el estudio de los mismos.

El problema de *Vertex Separation* se ha afrontado siguiendo la metodología de búsqueda de vecindad variable, la cual ha obtenido buenos resultados en numerosos trabajos recientes. En concreto, se ha publicado un primer trabajo en el que se propone un algoritmo basado en la búsqueda de vecindad variable básica, además de dos métodos constructivos y un procedimiento de búsqueda local. Además, se propone un conjunto de instancias para realizar una comparativa entre diferentes métodos obtenido de la colección de grafos *Harwell-Boeing*. Los resultados obtenidos prueban la eficiencia y eficacia del método propuesto, convirtiéndose en el primer heurístico para el problema en cuestión.

Posteriormente se publica un segundo trabajo cuyo objetivo es estudiar cómo afecta la diversificación y la intensificación al procedimiento de búsqueda. Para ello se proponen tres métodos diferentes: uno de ellos centrado en la diversificación (*Reduced VNS*), otro centrado en la intensificación (*VND*) y un último método que combina ambos aspectos (*General VNS*). Además, se proponen cuatro nuevos métodos de perturbación de soluciones que varían la importancia de la diversificación y la intensificación, así como dos nuevas estructuras de vecindad. Los resultados obtenidos demuestran que el balance entre diversificación e intensificación proporciona mejores resultados, emergiendo *GVNS* como el mejor algoritmo para el *Vertex Separation Problem.*

El *Profile Minimization Problem* es un problema con aplicaciones en diversos campos como la arqueología o la reducción del espacio requerido para almacenar sistemas de ecuaciones. Para resolver este problema se propone un algoritmo basado en la metodología de búsqueda dispersa.

Los resultados obtenidos por dicho algoritmo lo convierten en el estado del arte para el problema, como lo confirma la extensa experimentación llevada a cabo. Además, las conclusiones derivadas pueden ser aplicadas a problemas relacionados. Por ejemplo, la evaluación eficiente de los movimientos propuesta puede ser adaptada a un gran número de problemas de ordenaciones lineales.

Por último, el diseño factorial llevado a cabo muestra que un diseño secuencial puede obtener resultados similares, lo que resulta interesante para aquellos casos en los que no es posible diseñar un experimento factorial por problemas de requisitos o tiempo de cómputo.

La paralelización de metaheurísticas se ha estudiado utilizando el *Cutwidth Minimization Problem.* Se han diseñado seis estrategias paralelas basadas en la búsqueda de vecindad variable que pueden ser clasificadas en tres grandes clases. La primera de ellas se centra en la paralelización del método completo, mientras la segunda está orientada a simultanear los procedimientos de perturbación y búsqueda local. Una última clase tiene como objetivo la exploración paralela de diferentes estructuras de vecindad.

Las estrategias propuestas se han utilizado para paralelizar una variante de búsqueda de vecindad variable recientemente introducida, denominada *Variable Formulation Search*, que hasta el momento era considerada el estado del arte para el problema. La experimentación preliminar selecciona la mejor variante para cada clase, mientras que en la experimentación final se compara cada una de las variantes con la versión secuencial del algoritmo.

Los resultados obtenidos muestran la superioridad de dos de las versiones paralelas. En concreto, *Replicated Shaking VNS* y *Replicated Parallel VNS* superan al método previo. Estos resultados son respaldados con test estadísticos, estableciendo a *Replicated Shaking VNS* como el método de referencia para el *Cutwidth Minimization Problem*.

Por último, se ha propuesto un algoritmo basado en la metodología de búsqueda dispersa combinado con el reencadenamiento de trayectorias para solucionar el problema del *Bandpass*. Dicho problema surge en una conferencia hace más de diez años, pero la primera publicación relacionada no aparece hasta 2009. Además, ha derivado en diferentes variantes, cada una de ellas con sus propias instancias de test para evaluar la calidad de los algoritmos propuestos.

Uno de los objetivos del estudio del problema del *Bandpass* es establecer una revisión del estado actual del problema y de sus variantes, habiendo identificado dos tipos de problemas y un total de tres variantes. Además, se ha diseñado un algoritmo basado en búsqueda dispersa que puede ser aplicado a cualquiera de las variantes sin llevar a cabo modificaciones.

Los resultados obtenidos revelan interesantes conclusiones acerca de la interacción de los elementos que conforman el algoritmo de búsqueda dispersa. Además, se ha comprobado que aunque dos de las variantes pertenecen a la misma clase, un algoritmo específicamente diseñado para la primera variante disminuye su calidad cuando se aplica a la segunda, aunque sigue siendo claramente superior a los algoritmos previamente publicados. Finalmente, los experimentos llevados a cabo prueban que el algoritmo propuesto muestra una mayor robustez que los programas comerciales.

Los resultados obtenidos muestran como se ha cumplido el objetivo inicial de la tesis. Además, la investigación realizada en esta tesis ha resultado en la publicación de 13 artículos indexados en el *Journal of Citation Reports* (JCR), de los cuales 6 están publicados en revistas del primer cuartil, 3 en el segundo cuartil, 3 en el tercer cuartil y, por último, 1 en el cuarto cuartil. Finalmente, se han publicado 5 artículos en revistas no indexadas y capítulos de libro y los resultados obtenidos en la investigación se han presentado en un total de 15 conferencias nacionales e internacionales.

Bibliography

[1] D. Adolphson and T.C. Hu. Optimal Linear Ordering. *SIAM Journal on Applied Mathematics*, 25(3):403–423, 1973.

[2] E. Alba. *Parallel metaheuristics: a new class of algorithms.* Wiley-Interscience, 2005.

[3] E. Alba and A.J. Nebro. *New Technologies in Parallelism*, pages 63–78. John Wiley & Sons, Inc., 2005.

[4] S. Amaran, N. V. Sahinidis, B. Sharda, and S.J. Bury. Simulation optimization: a review of algorithms and applications. *Annals of Operations Research*, 240(1):351–380, 2016.

[5] D.V. Andrade and Resende. GRASP with evolutionary path-relinking. In *Seventh Metaheuristics International Conference (MIC)*, 2007.

[6] D.V. Andrade and M. G. C Resende. GRASP with path-relinking for network migration scheduling. In *Proceedings of International Network Optimization Conference (INOC)*, 2007.

[7] G. Ausiello, M. Protasi, A. Marchetti Spaccamela, G. Gambosi, P. Crescenzi, and V. Kann. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edition, 1999.

[8] D.A. Babayev, G.I. Bell, and U.G. Nuriyev. The bandpass problem: combinatorial optimization and library of problems. *Journal of Combinatorial Optimization*, 18(2):151–172, 2009.

[9] R. S. Barr and B. L. Hickman. Feature Article—Reporting Computational Experiments with Parallel Algorithms: Issues, Measures, and Experts' Opinions. *ORSA Journal on Computing*, 5(1):2–18, 1993.

[10] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

[11] T. Benoist, B. Estellon, F. Gardi, R. Megel, and K. Nouioua. LocalSolver 1.x: a black-box local-search solver for 0-1 programming. *4OR*, 9(3):299–316, 2011. www.localsolver.com.

[12] M.E. Berberler and A. Gursoy. Genetic algorithm approach for bandpass problem. In *24th Mini EURO Conference on Continuous Optimization and Information-Based Technologies in the Financial Sector (MEC EurOPT 2010)*, pages 201–206, Izmir, 2010.

[13] G. Blin, G. Fertin, D. Hermelin, and Vialette. Fixed-parameter algorithms for protein similarity search under mRNA structure constraints. *Journal of Discrete Algorithms*, 6(4):618 – 626, 2008. Selected papers from the 1st Algorithms and Complexity in Durham Workshop (ACiD 2005)1st Algorithms and Complexity in Durham Workshop (ACiD 2005).

[14] H. L. Bodlaender, T. Kloks, and D. Kratsch. Treewidth and Pathwidth of Permutation Graphs. *SIAM Journal on Discrete Mathematics*, 8(4):606–616, 1995.

[15] Hans Bodlaender, Jens Gustedt, and Jan Arne Telle. Linear-time Register Allocation for a Fixed Number of Registers. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '98, pages 574–583, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

[16] H.L. Bodlaender, J.R. Gilbert, H. Hafsteinsson, and Kloks. Approximating Treewidth, Pathwidth, Frontsize, and Shortest Elimination Tree. *Journal of Algorithms*, 18(2):238 – 255, 1995.

[17] H.L. Bodlaender and R.H. Möhring. *The pathwidth and treewidth of cographs*, pages 301–309. Springer Berlin Heidelberg, Berlin, Heidelberg, 1990.

[18] B. Bollobás and I. Leader. Edge-isoperimetric inequalities in the grid. *Combinatorica*, 11(4):299–314, 1991.

[19] R. A. Botafogo. Cluster Analysis for Hypertext Systems. In *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '93, pages 116–125, New York, NY, USA, 1993. ACM.

[20] J. Brimberg, P. Hansen, N. Mladenović, and Taillard. Improvements and Comparison of Heuristics for Solving the Uncapacitated Multisource Weber Problem. *Operations Research*, 48(3):444–460, 2000.

[21] V. Campos, R. Martí, J. Sánchez-Oro, and A. Duarte. GRASP with path relinking for the orienteering problem. *Journal of the Operational Research Society*, 65(12):1800–1813, 2014.

[22] V. Campos, E. Piñana, and R. Martí. Adaptive memory programming for matrix bandwidth minimization. *Annals of Operations Research*, 183(1):7–23, 2011.

[23] 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.

[24] M.J. Chung, F. Makedon, I.H. Sudborough, and J. Turner. Polynomial Time Algorithms for the Min Cut Problem on Degree Restricted Trees. *SIAM J. Comput.*, 14(1):158–177, 1985.

[25] J. Cohoon and S. Sahni. Exact algorithms for special cases of the Board Permutation Problem. In *Proc. 21st Ann. Allerton Conf. on Communication, Control, and Computing*, pages 246–255, 1983.

[26] J.P. Cohoon and S. Sahni. Heuristics for Backplane Ordering. *Adv. VLSI Comput. Syst.*, 2(1-2):37–60, 1987.

[27] T. G. Crainic and M. Toulouse. *Parallel Strategies for Meta-Heuristics*, pages 475–513. Springer US, Boston, MA, 2003.

[28] T.G. Crainic, M. Gendreau, P. Hansen, and N. Mladenović. Cooperative Parallel Variable Neighborhood Search for the p-Median. *Journal of Heuristics*, 10(3):293–314, 2004.

[29] E. Cuthill and J. McKee. Reducing the Bandwidth of Sparse Symmetric Matrices. In *Proceedings of the 1969 24th National Conference*, ACM '69, pages 157–172, New York, NY, USA, 1969. ACM.

[30] J. Díaz, A. Gibbons, G. E. Pantziou, M. J. Serna, P. G. Spirakis, and J. Toran. Computing and Combinatorics Parallel algorithms for the minimum cut and the minimum length tree layout problems. *Theoretical Computer Science*, 181(2):267 – 287, 1997.

[31] J. Díaz, A. M. Gibbons, M. S. Paterson, and J. Torán. *The MINSUMCUT problem*, pages 65–79. Springer Berlin Heidelberg, Berlin, Heidelberg, 1991.

[32] J. Díaz, M. D. Penrose, J: Petit, and M. Serna. Convergence Theorems for Some Layout Measures on Random Lattice and Random Geometric Graphs. *Comb. Probab. Comput.*, 9(6):489–511, 2000.

[33] J. Díaz, J. Petit, and M. Serna. A Survey of Graph Layout Problems. *ACM Comput. Surv.*, 34(3):313–356, 2002.

[34] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[35] Duarte, J. Sánchez Oro, M.G.C. Resende, F. Glover, and R. Martí. Greedy randomized adaptive search procedure with exterior path relinking for differential dispersion minimization. *Information Sciences*, 296:46 – 60, 2015.

[36] A. Duarte, L. F. Escudero, R. Martí, N. Mladenović, J. J. Pantrigo, and J. Sánchez-Oro. Variable neighborhood search for the Vertex Separation Problem. *Computers & Operations Research*, 39(12):3247 – 3255, 2012.

[37] A. Duarte, M. Laguna, R. Martí, and J. Sánchez-Oro. Optimization procedures for the bipartite unconstrained 0-1 quadratic programming problem. *Computers & Operations Research*, 51:123 – 129, 2014.

[38] A. Duarte, J.J. Pantrigo, and M. Gallego. *Metaheurísticas*. Dykinson, 2007.

[39] 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):55–73, 2016.

[40] A. Duarte, J. Sánchez Oro, M.G.C. Resende, F. Glover, and R. Martí. Greedy randomized adaptive search procedure with exterior path relinking for differential dispersion minimization. *Information Sciences*, 296:46 – 60, 2015.

[41] V. Dujmović, M. R. Fellows, M. Kitching, G. Liotta, C. McCartin, N. Nishimura, P. Ragde, F. Rosamond, S. Whitesides, and D. R. Wood. On the Parameterized Complexity of Layered Graph Drawing. *Algorithmica*, 52(2):267–292, 2008.

[42] J. Ellis and M. Markov. Computing the vertex separation of unicyclic graphs. *Information and Computation*, 192(2):123 – 161, 2004.

[43] J.A. Ellis, I.H. Sudborough, and J.S. Turner. The Vertex Separation and Search Number of a Graph. *Information and Computation*, 113(1):50 – 79, 1994.

[44] Faria, S. Binato, M.G.C. Resende, and D. M. Falcao. Power transmission network design by greedy randomized adaptive path relinking. *IEEE Transactions on Power Systems*, 20(1):43–49, 2005.

[45] T. A Feo and M. G. C Resende. A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem. *Oper. Res. Lett.*, 8(2):67–71, 1989.

[46] T.A. Feo and M. G. C. Resende. Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, 6(2):109–133, 1995.

[47] M. Flynn. *Flynn's Taxonomy*, pages 689–697. Springer US, Boston, MA, 2011.

[48] F. V. Fomin and K. Høie. Pathwidth of cubic graphs and exact algorithms. *Information Processing Letters*, 97(5):191 – 196, 2006.

[49] L.R Ford and D.R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[50] F. García López, B. Melián Batista, J.A. Moreno Pérez, and J. M. Moreno Vega. The Parallel Variable Neighborhood Search for the p-Median Problem. *Journal of Heuristics*, 8(3):375–388, 2002.

[51] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[52] F. Gavril. Some simplified NP-complete graph problems. In *Proceedings of the Eleventh Conference on Information Sciences and Systems*, pages 91 – 95, Baltimore, MD, 1977.

[53] N.E. Gibbs. Algorithm 509: A Hybrid Profile Reduction Algorithm [F1]. *ACM Trans. Math. Softw.*, 2(4):378–387, 1976.

[54] N.E. Gibbs, W.G. Poole, 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.

[55] F. Glover. *ONR Research Memorandum*, chapter Probabilistic and Parametric Learning Combinations of Local Job Shop Scheduling Rules. Carnegie Mellon University, 1963.

[56] F. Glover. Applications of Integer Programming Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533 – 549, 1986.

[57] F. Glover. *Tabu Search and Adaptive Memory Programming — Advances, Applications and Challenges*, pages 1–75. Springer US, Boston, MA, 1997.

[58] F. Glover. *A template for scatter search and path relinking*, pages 1–51. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[59] F. Glover. *Multi-Start and Strategic Oscillation Methods — Principles to Exploit Adaptive Memory*, pages 1–23. Springer US, Boston, MA, 2000.

[60] F. Glover. Exterior path relinking for zero-one optimization. *Int. J. Appl. Metaheur. Comput.*, 5(3), 2014.

[61] F. Glover and M. Laguna. *Tabu Search.* Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[62] F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. *Control and cybernetics*, 29(3):653–684, 2000.

[63] F. Glover, M. Laguna, and R. Martí. *Scatter Search and Path Relinking: Advances and Applications*, pages 1–35. Springer US, Boston, MA, 2003.

[64] F. Glover, M. Laguna, and R. Martí. *New Ideas and Applications of Scatter Search and Path Relinking*, pages 367–383. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[65] F. Glover, M. Laguna, and R. Martí. *Scatter Search and Path Relinking: Foundations and Advanced Designs*, pages 87–99. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[66] Fred Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8(1):156–166, 1977.

[67] Fred Glover and Gary A. Kochenberger. *An Introduction to Tabu Search*, pages 37–54. Springer US, Boston, MA, 2003.

[68] W.J. Goralski. *SONET, a guide to synchronous optical networks.* McGraw-Hill Higher Education, 1997.

[69] Y. Guan and K. L. Williams. Profile minimization on triangulated triangles. *Discrete Mathematics*, 260(1):69 – 76, 2003.

[70] A. Gursoy and Nuriyev. Genetic Algorithm for Multi Bandpass Problem and library of problems. In *Problems of Cybernetics and Informatics (PCI), 2012 IV International Conference*, pages 1–5, 2012.

[71] P. Hansen and N. Mladenović. J-Means: a new local search heuristic for minimum sum of squares clustering. *Pattern Recognition*, 34(2):405 – 413, 2001.

[72] P. Hansen, N. Mladenović, J. Brimberg, and J.A. Moreno Pérez. *Handbook of Metaheuristics*, volume 146, chapter Variable Neighborhood Search, pages 61 – 86. Springer US, 2010.

[73] P. Hansen, N. Mladenović, and J.A. Moreno Pérez. Variable Neighborhood Search: methods and applications. *Annals of Operations Research*, 175(1):367–407, 2010.

[74] P. Hansen, N. Mladenović, and D. Perez Britos. Variable Neighborhood Decomposition Search. *Journal of Heuristics*, 7(4):335–350, 2001.

[75] L.H. Harper. Optimal numberings and isoperimetric problems on graphs. *Journal of Combinatorial Theory*, 1(3):385 – 393, 1966.

[76] L. M. Hvattum, A. Duarte, F. Glover, and R. Martí. Designing effective improvement methods for scatter search: an experimental study on global optimization. *Soft Computing*, 17(1):49–62, 2013.

[77] T. Jones. One operator, one landscape. Technical report, Santa Fe Institute, Santa Fe, USA, 1995.

[78] M. Juvan and B. Mohar. Optimal linear labelings and eigenvalues of graphs. *Discrete Applied Mathematics*, 36(2):153 – 168, 1992.

[79] I. P. Kaminow, C. R. Doerr, C. Dragone, T. Koch, U. Koren, A. A. M. Saleh, A. J. Kirby, C. M. Ozveren, B. Schofield, R. E. Thomas, R. A. Barry, D. M. Castagnozzi, V. W. S. Chan, B. R. Hemenway, D. Marquis, S. A. Parikh, M. L. Stevens, E. A. Swanson, S. G. Finn, and R. G. Gallager. A wideband all-optical WDM network. *IEEE Journal on Selected Areas in Communications*, 14(5):780–799, Jun 1996.

[80] D.R. Karger. A Randomized Fully Polynomial Time Approximation Scheme for the All Terminal Network Reliability Problem. In *Proceedings of the Twenty-seventh Annual ACM Symposium on Theory of Computing*, STOC '95, pages 11–17, New York, NY, USA, 1995. ACM.

[81] R. M. Karp. Mapping the Genome: Some Combinatorial Problems Arising in Molecular Biology. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 278–285, New York, NY, USA, 1993. ACM.

[82] D. G. Kendall. Incidence matrices, interval graphs and seriation in archeology. *Pacific J. Math.*, 28(3):565–570, 1969.

[83] J. B. Kruskal. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956.

[84] M. Kurt, H. Kutucu, A. Gursoy, and U. Nuriyev. *The Optimization of the Bandpass Lengths in the Multi-Bandpass Problem*, pages 115–123. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[85] M. Laguna. Optimization of complex systems with OptQuest. *A White Paper from OptTek Systems, Inc*, 1997.

[86] M. Laguna, F. Gortázar, M. Gallego, A. Duarte, and R. Martí. A black-box scatter search for optimization problems with integer variables. *Journal of Global Optimization*, 58(3):497–516, 2014.

[87] M. Laguna and R. Martí. GRASP and Path Relinking for 2-Layer Straight Line Crossing Minimization. *INFORMS Journal on Computing*, 11(1):44–52, 1999.

[88] M. Laguna and R. Martí. *Scatter Search: Methodology and Implementations in C.* Kluwer, Boston, 2003.

[89] Leiserson. Area-efficient graph layouts. In *Foundations of Computer Science, 1980., 21st Annual Symposium on*, pages 270–281, 1980.

[90] J. G. Lewis. Algorithm 582: The Gibbs-Poole-Stockmeyer and Gibbs-King Algorithms for Reordering Sparse Matrices. *ACM Trans. Math. Softw.*, 8(2):190–194, 1982.

[91] R.R. Lewis. Simulated annealing for profile and fill reduction of sparse matrices. *International Journal for Numerical Methods in Engineering*, 37(6):905–925, 1994.

[92] Z. Li and G. Lin. The three column Bandpass problem is solvable in linear time. *Theoretical Computer Science*, 412(4):281 – 299, 2011.

[93] G. Lin. On the Bandpass problem. *Journal of Combinatorial Optimization*, 22(1):71–77, 2011.

[94] Y. Lin and J. Yuan. Profile minimization problem for matrices and graphs. *Acta Mathematicae Applicatae Sinica*, 10(1):107–112, 1994.

[95] R. J. Lipton and R. E. Tarjan. A Separator Theorem for Planar Graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[96] H. R. Lourenço, O. C. Martin, and T. Stützle. *Iterated Local Search*, pages 320–353. Springer US, Boston, MA, 2003.

[97] J. Luttamaguzi, M. Pelsmajer, Z. Shen, and B. Yang. Integer programming solutions for several optimization problems in graph theory,. Technical report, DIMACS, 2005.

[98] F. Makedon and I.H. Sudborough. On minimizing width in linear layouts. *Discrete Applied Mathematics*, 23(3):243 – 265, 1989.

[99] R. Martí and M. Laguna. Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial. *19*, 7:123–130, 2003.

[100] 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.

[101] B. Menéndez, E. G. Pardo, J. Sánchez-Oro, and A. Duarte. Parallel variable neighborhood search for the min–max order batching problem. *International Transactions in Operational Research*, In press, 2016.

[102] N. Mladenović. A variable neighborhood algorithm-a new metaheuristic for combinatorial optimization. In *papers presented at Optimization Days*, page 112, 1995.

[103] N. Mladenović and P. Hansen. Variable Neighborhood Search. *Comput. Oper. Res.*, 24(11):1097–1100, 1997.

[104] N. Mladenović, J. Petrović, V. Kovačević-Vujčić, and M. Čangalović. Solving spread spectrum radar polyphase code design problem by tabu search and variable neighbourhood search. *European Journal of Operational Research*, 151(2):389 – 399, 2003. Meta-heuristics in combinatorial optimization.

[105] P. Mutzel. *A polyhedral approach to planar augmentation and related problems*, pages 494–507. Springer Berlin Heidelberg, Berlin, Heidelberg, 1995.

[106] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* Wiley-Interscience, New York, NY, USA, 1988.

[107] U.G. Nuriyev, H. Kutucu, and M. Kurt. Mathematical models of the Bandpass problem and OrderMatic computer game. *Mathematical and Computer Modelling*, 53(5–6):1282 – 1288, 2011.

[108] 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.

[109] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1982.

[110] 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.

[111] E. G. Pardo, M. Soto, and C. Thraves. Embedding signed graphs in the line. *Journal of Combinatorial Optimization*, 29(2):451–471, 2015.

[112] S. Peng, C. Ho, T. Hsu, M. Ko, and C.Y. Tang. *A Linear-Time Algorithm for Constructing an Optimal Node-Search Strategy of a Tree*, pages 279–289. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[113] J. Petit. Experiments on the Minimum Linear Arrangement Problem. *J. Exp. Algorithmics*, 8, 2003.

[114] D. C. Porumbel, J.-K. Hao, and F. Glover. A simple and effective algorithm for the MaxMin diversity problem. *Annals of Operations Research*, 186(1):275, 2011.

[115] R. C. Prim. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401, 1957.

[116] J.D. Quintana, J. Sánchez Oro, and A. Duarte. Efficient Greedy Randomized Adaptive Search Procedure for the Generalized Regenerator Location Problem. *International Journal of Computational Intelligence*, In press, 2016.

[117] R. Ramaswami, K. N. Sivarajan, and G. H. Sasaki. *Optical Networs: a practical perspective.* Morgan Kaufmann, 2010.

[118] R. Ravi, A. Agrawal, and P. Klein. *Ordering problems approximated: single-processor scheduling and interval graph completion*, pages 751–762. Springer Berlin Heidelberg, Berlin, Heidelberg, 1991.

[119] M. G. C Resende and D.V. Andrade. Method and system for Network Migration Scheduling, 2009.

[120] M.G.C. Resende, R. Martí, M. Gallego, and A. Duarte. {GRASP} and path relinking for the max–min diversity problem. *Computers & Operations Research*, 37(3):498 – 508, 2010. Hybrid Metaheuristics.

[121] C. C. Ribeiro and M. C. Souza. Variable neighborhood search for the degree-constrained minimum spanning tree problem. *Discrete Applied Mathematics*, 118(1–2):43 – 54, 2002. Special Issue devoted to the ALIO-EURO Workshop on Applied Combinatorial Optimization.

[122] C. C. Ribeiro, E. Uchoa, and R. F. Werneck. A Hybrid GRASP with Perturbations for the Steiner Problem in Graphs. *INFORMS Journal on Computing*, 14(3):228–246, 2002.

[123] C.C. Ribeiro and I. Rosseti. *A Parallel GRASP Heuristic for the 2-Path Network Design Problem*, pages 922–926. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[124] J. Rolim, O. Sýkora, and I. Vrt'o. *Optimal cutwidths and bisection widths of 2- and 3-dimensional meshes*, pages 252–264. Springer Berlin Heidelberg, Berlin, Heidelberg, 1995.

[125] Y. Saad. *Iterative Methods for Sparse Linear Systems.* Society for Industrial and Applied Mathematics, second edition, 2003.

[126] J. Sánchez-Oro, A. Duarte, and S. Salcedo-Sanz. Robust total energy demand estimation with a hybrid Variable Neighborhood Search – Extreme Learning Machine algorithm. *Energy Conversion and Management*, 123:445 – 452, 2016.

[127] J. Sánchez-Oro, D. Fernández López, R. Cabido, A. S. Montemayor, and J. J. Pantrigo. Radar-based road-traffic monitoring in urban environments. *Digital Signal Processing*, 23(1):364–374, 2013.

[128] J. Sánchez-Oro, M. Laguna, A. Duarte, and R. Martí. Scatter search for the profile minimization problem. *Networks*, 65(1):10–21, 2015.

[129] J. Sánchez-Oro, M. Laguna, R. Martí, and A. Duarte. Scatter search for the bandpass problem. *Journal of Global Optimization*, page In press, 2016.

[130] J. Sánchez-Oro, N. Mladenović, and A. Duarte. General Variable Neighborhood Search for computing graph separators. *Optimization Letters*, pages 1–21, 2014.

[131] 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, Part B:209 – 219, 2014. Recent advances in Variable neighborhood search.

[132] J. Sánchez-Oro, M. Sevaux, A. Rossi, R. Martí, and A. Duarte. The 3rd International Conference on Variable Neighborhood Search (VNS'14) Solving dynamic memory allocation problems in embedded systems with parallel variable neighborhood search strategies. *Electronic Notes in Discrete Mathematics*, 47:85 – 92, 2015.

[133] F. Shahrokhi, O. Sýkora, L.A. Székely, and I. Vrto. On Bipartite Drawings and the Linear Arrangement Problem. *SIAM J. Comput.*, 30(6):1773–1789, 2001.

[134] Y. Shiloach. A Minimum Linear Arrangement Algorithm for Undirected Trees. *SIAM Journal on Computing*, 8(1):15–32, 1979.

[135] K. Skodinis. *Computing Optimal Linear Layouts of Trees in Linear Time*, pages 403–414. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[136] K. Takagi and N. Takagi. Minimum Cut Linear Arrangement of p-q dags for VLSI layout of adder trees. In *IEICE Transactions on Fundamentals of Electronics, Com- munications and Computer Sciences E82-A*, pages 767–774, 1999.

[137] R. Tewarson. *Sparse Matrices.* Academic Press, NY (USA), 1973.

[138] D.M. Thilikos, M. J. Serna, and H. L. Bodlaender. Cutwidth II: Algorithms for Partial W-trees of Bounded Degree. *J. Algorithms*, 56(1):25–49, 2005.

[139] P.J. van Laarhoven and E.H. Aarts. *Simulated Annealing: Theory and Applications.* Kluwer Academic Publishers, Norwell, MA, USA, 1987.

[140] M. Yannakakis. A Polynomial Algorithm for the Min-cut Linear Arrangement of Trees. *J. ACM*, 32(4):950–988, 1985.