Applied Soft Computing 13 (2013) 2242–2252

Contents lists available at SciVerse ScienceDirect

Applied Soft Computing

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

Variable Formulation Search for the Cutwidth Minimization Problem

Eduardo G. Pardo[a], Nenad Mladenović[b], Juan J. Pantrigo[a], Abraham Duarte[a]

[a] Universidad Rey Juan Carlos, C/Tulipán s/n, 28933 Móstoles, Spain
[b] Brunel University, Kingston Lane, Uxbridge UB8 3PH, UK

ARTICLE INFO

Article history:
Received 22 June 2012
Received in revised form 7 December 2012
Accepted 15 January 2013
Available online 31 January 2013

Keywords:
Variable Neighbourhood Search
Variable Formulation Search
Discrete optimization
Cutwidth Minimization Problem
Formulation Space Search

ABSTRACT

Many optimization problems are formulated as min–max problems where the objective function consist of minimizing a maximum value. In this case, it is usual that many solutions of the problem has associated the same value of the objective function. When this happens it is difficult to determine which solution is more promising to continue the search. In this paper we propose a new variant of the Variable Neighbourhood Search methodology to tackle this kind of problems. The new variant, named Variable Formulation Search, 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. We do that in shaking, local search and neighbourhood change steps of the basic Variable Neighbourhood Search. We apply the new methodology to the Cutwidth Minimization Problem. Computational results show that our proposal outperforms previous algorithms in the state of the art in terms of quality and computing time.

1. Introduction

Soft computing is a discipline which includes Fuzzy Logic, Neural Networks, Evolutionary Computing or Rough Sets among others with the aim of solving real-life problems (see for instance [4,5,28]). These techniques involve a collection of methodologies to exploit tolerance for imprecision, uncertainty and partial truth to achieve tractability, robustness and low solution cost. Many real-life problems can be enunciated as optimization problems. Let us consider an optimization problem in its general form:

$$\min \quad \{\mathcal{F}(x) \mid x \in X \subset \mathcal{S}\},$$

where x is any solution, X is a set of feasible solutions, $\mathcal{S}$ is a solution space and $\mathcal{F}(.)$ is a real valued (objective) function that maps $\mathcal{S}$ to $\mathbb{R}$. Most of real-world optimization problems are hard to solve because they have many local optima and neighbourhood type methods cannot escape easily from the so-called local optima traps. Besides exact solution methods, that can be only used for moderate size problems, there are many approximate solution techniques.

Metaheuristics emerge in soft computing as a set of methodologies which provide near optimal solutions (see [21,20] for a compilation of the best known) and they have also been combined with other techniques for optimization [7]. One well-know metaheuristic is Variable Neighbourhood Search (VNS) [34]. Its basic idea is the change of neighbourhood structures during the search, since a local minimum with respect to one structure is not necessary so with another one [34,23,22].

In this paper we propose a new variant of VNS metaheuristic that we call Variable Formulation Search (VFS). It takes into account different formulations of the considered problem. Formulations are used in cases where there are many solutions in the neighbourhood with the same best value with respect to previous formulations used in the sequence. We do change formulations sequentially within the local search, shaking and neighbourhood change steps of the basic VNS.

The idea of changing the formulation within the search of the best solution is not new. As far as we know, the first attempt to systematically explore this idea was suggested in [35,36]. The approach was called Formulation Space Search (FSS) and was applied for solving Circle Packing problem. The basic idea of FSS approach is quite general: the distance between any 2 formulations is defined in order to introduce the formulation space of each particular problem considered. In that way local optima traps are avoided by searching not only through the solution space but also through the formulation space. Improved version of FSS for solving the Circle Packing problem is proposed in [29]. In [27] FSS is adapted for solving Teacher/Class Timetabling problem. In [25] the idea of changing formulation and therefore, change of the search space, is called Variable Space Search (VSS). In this case, it is used for solving the Graph Colouring problem. More recently, without mentioning FSS or VSS references from above, in [9] several 0–1 quadratic formulations for solving Maximum Independent Set (or Maximum Clique) problem are used. The method was called

E-mail addresses: eduardo.pardo@urjc.es (E.G. Pardo), nenad.mladenovic@brunel.ac.uk (N. Mladenović), juanjose.pantrigo@urjc.es (J.J. Pantrigo), abraham.duarte@urjc.es (A. Duarte).

1568-4946/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.asoc.2013.01.016

Variable Objective Search (VOS). Basically, FSS, VSS and VOS follow the same ideas and may be considered as the same approach or methodology.

In this paper we propose a new variant of the Variable Neighbourhood Search (VNS) methodology. This variant, named Variable Formulation Search (VFS), considers the characteristics of the VNS framework and introduces new criteria based on the use of alternative formulations of the problem to compare solutions. The aim of VFS is to determine whether a given solution is more promising than other to continue the search, beyond the value of the objective function. VFS is specially helpful to tackle problems which present a flat landscape where many solutions have associated the same value of the objective function. Particularly, we apply VFS to the Cutwidth Minimization Problem (CMP) and we compare experimentally the proposed algorithm with other state-of-the-art methods. Additionally, we introduce some properties of the solutions of the CMP. Those properties are useful to identify, for each vertex, the positions in the ordering where they can improve the value of the objective function. As a result, the number of positions which are necessary to explore within the local search procedure are drastically reduced.

The rest of the paper is structured as follows. In Section 2 we first describe the classical schema of the Basic Variable Neighbourhood Search (BVNS) followed by the new VFS and the methodological variation of the BVNS that we propose in this paper. In Section 3 we introduce the Cutwidth Minimization Problem (CMP) and we derive some properties of the solutions of the problem. Additionally, we propose two alternative formulations for the CMP. In Section 4 we illustrate the use of the VFS by solving the CMP using the new formulations proposed. The paper ends with the experimental comparison and the associated conclusions.

2. Variable Formulation Search

In this section we present the Variable Formulation Search (VFS), a new variant of the VNS methodology. First, we introduce the main characteristics of the VNS methodology. Particularly, we center our attention in the BVNS and, later, we outline the basic rules of VFS.

2.1. VNS

The Variable Neighbourhood Search (VNS) is a metaheuristic proposed in [34] as a general framework to solve hard optimization problems. It is based on a simple idea: systematical changes of neighbourhood structures within the search procedure. Let N_k with $1 \leq k \leq k_{max}$ be a finite set of pre-selected neighbourhood structures, where $N_k(x)$ is the set of neighbour solutions of x in the k-th neighbourhood. When solving an optimization problem by using different neighbourhood structures (N_k), VNS methodology proposes to explore them in three different ways: (i) random, (ii) deterministic, or (iii) mixed (both, deterministic and random).

From an algorithmic perspective, VNS variants mainly differ in how they implement three basic strategies: shaking (Section 4.2), improvement (Section 4.3), and neighbourhood change (Section 4.4). The original metaheuristic has been widely evolved with many extensions, depending on the aforementioned strategies. For instance: Variable Neighbourhood Descent (VND) explores the neighbourhood in a deterministic way by using several local search procedures (in this case there is not shaking procedure); Reduced VNS (RVNS) explores solutions at random in each neighbourhood by perturbing the solutions with the shaking procedure (in this case there is not local search procedure); Basic VNS (BVNS) combines deterministic and random exploration of the neighbourhoods (by considering both, shaking and local search procedures). Other example of more complex well-known variants are Skewed VNS (SVNS), General VNS (GVNS), Variable Neighbourhood Decomposition Search (VNSD), and Reactive VNS. See [23,22] for a recent thorough review.

In this paper we focus our attention in the Basic Variable Neighbourhood Search (BVNS) schema. The pseudocode of the BVNS schema is presented in Algorithm 1.

Algorithm 1. Pseudocode of BVNS

```
1:  Procedure BVNS (x, k_max, t_max)
2:  repeat
3:    k ← 1
4:    repeat
5:      x' ← Shake(x, k)
6:      x'' ← LocalSearch(x')
7:      NeighbourhoodChange(x, x'', k)
8:    until k = k_max
9:    t ← CPUTime()
10: until t > t_max
11: end BVNS
```

This procedure receives three input arguments: an initial solution (denoted as x), the largest predefined neighbourhood (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'. Then, in step 6, a local optimum, x'', is reached by using an improvement strategy. In step 7, it is decided whether the BVNS needs to explore a larger neighbourhood by increasing k (x'' is worse than x) or not, which implies to set $k = 1$ (x'' is better than x). Steps 5–7 are repeated until k_{max} is reached. This parameter determines the maximum number of different neighbourhoods to be tried in the current iteration when there is no improvement in the solution. Steps 3–9 are repeated until t_{max} is reached, starting in each iteration from the best found solution.

2.2. Variable Formulation Search

Many optimization problems in the literature present a flat landscape. This means that, given a formulation of the problem, there are many solutions with the same value of the objective function $\mathcal{F}(x)$ [38,40,43]. This fact can make these problems harder than usual. For many of them, typical ways of searching are driven by the moves within the set of feasible solutions which usually results in a null value. The change of neighbourhood is a strategy (used by the VNS methodology) which can be helpful to avoid some of the aforementioned difficulties. Nevertheless, there are problems where the change of neighbourhood is not enough to produce improvements in the objective function after a move. To address this drawback we propose to use alternative formulations of the problem. Specifically, in this paper we propose a new variant of the VNS methodology (named Variable Formulation Search, VFS) which combine the change of neighbourhood within the VNS framework, with the use of alternative formulations. In particular, the alternative formulations will be used to compare different solutions with the same value of the objective function, when considering the original formulation.

Let us assume that, beside the original formulation and the corresponding objective function $\mathcal{F}_0(x) = \mathcal{F}(x)$ there are several (p) other formulations that can be denoted with $\mathcal{F}_1(x), \mathcal{F}_2(x), \ldots, \mathcal{F}_p(x), \forall x \in X$. Note that two formulations are equivalent if the optimal solution of one is the optimal solution of the other, and vice versa. Thus, two equivalent but different formulations could have the same objective function. However, without loss of clarity, we will denote different formulations as different objectives $\mathcal{F}_i(x), i = 1, \ldots, p$. The idea of VFS is to add the procedure `Accept`(x, x', p), given in Algorithm 2 in all three basic steps of BVNS: `Shaking`, `LocalSearch` and `NeighbourhoodChange`. Clearly, if a better solution is not obtained by any formulation among the p preselected, the move is rejected.

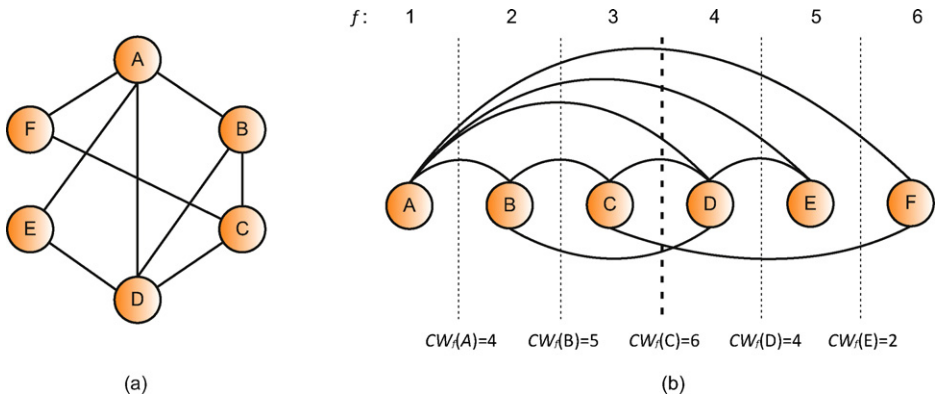

Fig. 1. (a) Graph G with six vertices and nine edges. (b) Ordering f of the vertices of the graph in (a) with the corresponding cutwidth of each vertex.

Algorithm 2. Pseudocode of the `Accept` procedure

```
Procedure Accept (x, x', p)
for i = 0 to p do
  condition1 = Fi(x') < Fi(x)
  condition2 = Fi(x') > Fi(x)
  if(condition1)then
    return True
  else
    if (condition2) then
      return False
    else
      continue /*with the next alternative formulation*/
    endif
  endif
endfor
return False
end Accept
```

If `Accept` (x, x', p) is satisfied into `LocalSearch` subroutine of BVNS, then it will not stop the first time a non improved solution is found. In order to stop `LocalSearch` and thus claim that x' is local minimum, x' should not be improved by any among the p different formulations. Thus, for any particular problem, one needs to design different formulations of the problem considered and decide the order they will be used in the `Accept` subroutine. Answers to those two questions are problem specific and sometimes not easy. The `Accept` (x, x', p) subroutine can obviously be added to `NeighbourhoodChange` and `Shaking` steps of BVNS from Algorithm 1 as well.

3. Cutwidth Minimization Problem

Layout problems consist of finding a labelling of the vertices of a graph in such a way that an objective function is minimized (or maximized). These problems have been increasingly studied in recent years. In this category fall some well known problems such as the Minimum Linear Arrangement (MinLA) Problem [39], or the Profile Minimization Problem [45]. We refer the reader to [14] for a survey. Among layout problems, we can identify a subset of them where the objective function is to minimize a maximum value (similarly maximize a minimum). Examples of the latest subset are the Bandwidth [28], Antibandwidth [16], or Vertex Separation [15] problems. These problems become a challenge for heuristic methods because there may be many different solutions with the same objective function value. It is said that those problems present a "flat landscape" [33,40,43] which implies that most of the moves performed in the search strategies result in a null value.

In this paper we tackle the Cutwidth Minimization Problem (CMP) which is a NP-Hard classical min–max layout problem. The CMP consist 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. This problem can be formulated in both, combinatorial and mathematical programming ways. In the literature, it is possible to find an Integer Linear Programming formulation of the CMP [30]. In this paper we center our attention in the combinatorial formulation.

The classical combinatorial formulation can be described as follows. Given a graph $G=(V, E)$ with $|V|=n$, a labelling $f: V \rightarrow \{1, 2, \ldots, n\}$ of G assigns the integers $\{1, 2, \ldots, n\}$ 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 f, denoted as $CW_f(v)$, is the number of edges $(u, w) \in E$ satisfying $f(u) \leq f(v) < f(w)$. Therefore,

$$CW_f(v) = |\{(u, w) \in E : f(u) \leq f(v) < f(w)\}|$$

On the basis of the previous definition, it is possible to define the cutwidth of G with respect to f, denoted as $CW_f(G)$, as the maximum of all cutwidth values among its vertices. In mathematical terms:

$$CW_f(G) = \max_{v \in V} CW_f(v)$$

The optimum cutwidth of G is defined as the minimum $CW_f(G)$ value over all possible labellings. In other words, the Cutwidth Minimization Problem (CMP) consist of finding an ordering f that minimizes $CW_f(G)$ over the set of all possible labellings with size n, Π_n:

$$CW(G) = \min_{f \in \Pi_n} CW_f(G)$$

Example. In Fig. 1a it is shown a graph G with six vertices and nine edges. Additionally, in Fig. 1b it is depicted an ordering f of the vertices of G and the cutwidth associated to each vertex. In this example, the cutwidth of G with respect to f is $CW_f(G)=6$ as it is the maximum cutwidth of any vertex in the ordering. This is represented as a bold dashed line in the figure.

One of the first studies about the complexity of the CMP is attributed to Stockmeyer who, in a private communication to Garey and Johnson, performed a reduction of the problem to the "*Simple Max-Cut*" [18]. However, it was not until 1977 when Gavril published the previous result stating that the decision version of the CMP is NP-Complete [19]. In a more recent approach it was proved that the CMP remains NP-Hard even for graphs with maximum degree of three [32].

3.1. Cutwidth applications and literature

The CMP has also been referred in the literature using other names such as Minimum Cut Linear Arrangement [13,48] or Network Migration Scheduling [2,3]. There exist also a generalization of the CMP for hypergraphs named Board Permutation Problem [11,12]. Practical applications of the problem can be found in very different areas such as: Circuit Design [1,12,31], Network Reliability

[26], Information Retrieval [8], Automatic Graph Drawing [37,47] or Protein Engineering [6]. There is also a patent where the CMP is applied to Networks Migration [42].

Previous attempts to produce approximate and exact solutions to the CMP can be traced back to the eighties. The work by [12] 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 metaheuristic. Twenty years later [2,3] proposed a GRASP procedure hybridized with a Path Relinking method. The most recent approach was carried out by [38] who proposed an Scatter Search algorithm for the problem 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. For example [24] studied the CMP for hypercubes, [10,50,49] proposed algorithms for special cases of trees and [44] worked in the problem for Grids. All these algorithms present polynomial-time complexity.

Additionally, there are some exact methods for general graphs. Specifically, in [30] the authors proposed an Integer Linear Programming formulation for the CMP and, in [33], they proposed a Branch & Bound algorithm to find exact solutions to general graphs. However, these approaches can only solve relatively small instances.

3.2. Properties of CMP solutions

The solutions of layout problems are typically expressed as labellings, where each vertex occupies the position given by its label. In this subsection, we define a neighbourhood structure based on insertion movements for this kind of problems. Additionally, we provide a formal proof of two properties which drastically reduce the size of the corresponding neighbourhood.

3.2.1. Reduction of insertion neighbourhood

Generally speaking, insertions consist of selecting a vertex in the solution, removing it from the current position and placing it in a different one. In a more precise way, insertion moves are defined as follows: given an ordering f, a vertex v placed in the position $f(v)$, and a position j such as $f(v) \neq j$, we denote by $\texttt{ins}(f, j, v)$ an insertion movement, consisting of removing v from its current position $f(v)$ and inserting it in position j. This operation results in the ordering f', as follows:

- If $f(v) = i > j$, then v is inserted just before v_j in position j. From $f = (\ldots, v_{(j-1)}, v_j, v_{(j+1)}, \ldots, v_{(i-1)}, v, v_{(i+1)}, \ldots)$, we would obtain $f' = (\ldots, v_{(j-1)}, v, v_j, v_{(j+1)}, \ldots, v_{(i-1)}, v_{(i+1)}, \ldots)$.
- If $f(v) = i < j$, v is inserted just after v_j in position j. Therefore, from $f = (\ldots, v_{(i-1)}, v, v_{(i+1)}, \ldots, v_{(j-1)}, v_j, v_{(j+1)}, \ldots)$, we would obtain $f' = (\ldots, v_{(i-1)}, v_{(i+1)}, \ldots, v_{(j-1)}, v_j, v, v_{(j+1)}, \ldots)$.

Considering insertion moves previously defined, a solution f and a vertex v, we define the neighbourhood of f, $N_f(v)$, as the set of solutions reachable by inserting v in each position of the ordering. More precisely,

$$N_f(v) = \{f_i = ins(f, i, v) : 1 \leq i \leq n, i \neq f(v)\}$$

Before enunciating the properties, it is necessary to introduce the following notation. Given a solution f for the CMP, let $\deg(v)$ be the degree of a vertex $v \in V$. In mathematical terms:

$$\deg(v) = |\{u \in V : (u, v) \in E\}|$$

Let $\deg_f^R(v) = |\{u \in V : (u, v) \in E \wedge f(v) > f(u)\}|$ and symmetrically let $\deg_f^L(v) = |\{u \in V : (u, v) \in E \wedge f(v) \leq f(u)\}|$. Trivially, it holds that $\deg(v) = \deg_f^R(v) + \deg_f^L(v)$.

Property 1. *Given a graph $G=(V, E)$ let f be a solution for the CMP and $v \in V$ a vertex such that $deg_f^R(v) = \deg_f^L(v)$. Then, for all $f_i \in N_f(v)$, it holds that $CW_{f_i}(G) \geq CW_f(G)$.*

Proof. As it was stated in [46], any insertion move of a vertex in an ordering can be performed with a sequence of consecutive swaps. For the sake of simplicity, let us consider a swap move in f between a vertex v with $\deg_f^R(v) = \deg_f^L(v)$ and other vertex u such that $f(u) = f(v) - 1$ (symmetrically, $f(u) = f(v) + 1$) obtaining a new solution f' where $f'(u) = f(v)$ and $f'(v) = f(u)$. The aim is to compute $CW_{f'}(G)$ in terms of $CW_f(G)$. From the definition of the CMP we have that:

$$CW_{f'}(G) = \max\{CW_{f'}(v) \forall v \in V\}$$

It holds that $CW_{f'}(w) = CW_f(w)$ for all the vertices w which are not involved in the swap. Then,

$$\begin{aligned} CW_{f'}(G) &= \max\{CW_{f'}(w) \forall w \in V \setminus \{u, v\}, CW_{f'}(u), CW_{f'}(v)\} \\ &= \max\{CW_f(w) \forall w \in V \setminus \{u, v\}, CW_{f'}(u), CW_{f'}(v)\} \end{aligned}$$

The cutwidth value of u and v in f' can be computed as follows:

$$CW_{f'}(u) = CW_f(u) - \deg_f^L(v) + \deg_f^R(v) = CW_f(u) \quad (1)$$

$$CW_{f'}(v) = CW_f(v) + \deg_{f'}^L(u) - \deg_{f'}^R(u) \quad (2)$$

By (1), we get $CW_{f'}(u) = CW_f(u)$. Therefore, $CW_{f'}(G)$ can be computed as:

$$CW_{f'}(G) = \max\{CW_f(w) \forall w \in V \setminus \{v\}, CW_{f'}(v)\} \quad (3)$$

Let us now express (2) in the terms of the solution f. We need to distinguish between two situations: (i) vertices involved in the swap (u and v) are adjacent vertices and (ii) they are not adjacent vertices. If u and v are adjacent vertices, then,

$$\deg_{f'}^R(u) = \deg_f^R(u) - 1$$

$$\deg_{f'}^L(u) = \deg_f^L(u) + 1$$

replacing it in (2):

$$CW_{f'}(v) = CW_f(v) + \deg_f^L(u) - \deg_f^R(u) + 2 \quad (4)$$

If u and v are not adjacent vertices:

$$\deg_{f'}^R(u) = \deg_f^R(u)$$

$$\deg_{f'}^L(u) = \deg_f^L(u)$$

replacing it in (2):

$$CW_{f'}(v) = CW_f(v) + \deg_f^L(u) - \deg_f^R(u) \quad (5)$$

Considering both expressions (4) and (5) we can rewrite (3) as follows:

$$\begin{aligned} CW_{f'}(G) \geq \max\{&CW_f(w) \forall w \in V \setminus \{v\}, \\ &CW_f(v) + \deg_f^L(u) - \deg_f^R(u)\} \end{aligned} \quad (6)$$

There are again two cases to be considered: (i) $CW_f(G) = CW_f(w)$ for any w such that $w \neq v$ or (ii) $CW_f(G) = CW_f(v)$. If we are in the first case, (6) can be rewritten as follows:

$$CW_{f'}(G) \geq \max\{CW_f(G), CW_f(v) + \deg_f^L(u) - \deg_f^R(u)\} \geq CW_f(G)$$

Table 1
Solutions obtained when A is inserted in any position of f (see Fig. 1b) different from $f(A)$.

Solution	Cutwidth
$f_1 = \{B, A, C, D, E, F\}$	$CW_{f_1}(B) = 3, CW_{f_1}(A) = 5, CW_{f_1}(C) = 6, CW_{f_1}(D) = 4, CW_{f_1}(E) = 2$
$f_2 = \{B, C, A, D, E, F\}$	$CW_{f_2}(B) = 3, CW_{f_2}(C) = 4, CW_{f_2}(A) = 6, CW_{f_2}(D) = 4, CW_{f_2}(E) = 2$
$f_3 = \{B, C, D, A, E, F\}$	$CW_{f_3}(B) = 3, CW_{f_3}(C) = 4, CW_{f_3}(D) = 4, CW_{f_3}(A) = 4, CW_{f_3}(E) = 2$
$f_4 = \{B, C, D, E, A, F\}$	$CW_{f_4}(B) = 3, CW_{f_4}(C) = 4, CW_{f_4}(D) = 4, CW_{f_4}(E) = 4, CW_{f_4}(A) = 2$
$f_5 = \{B, C, D, E, F, A\}$	$CW_{f_5}(B) = 3, CW_{f_5}(C) = 4, CW_{f_5}(D) = 4, CW_{f_5}(E) = 4, CW_{f_5}(F) = 4$

which demonstrates Property 1 for this case. Let us now consider the second one. For any pair of vertices u, v in a solution f such that $f(u) = f(v) - 1$ it is possible to compute the cutwidth value of v from the corresponding value of u as follows:

$$CW_f(v) = CW_f(u) - \deg_f^L(v) + \deg_f^R(v)$$

If v holds the property $\deg_f^L(v) = \deg_f^R(v)$ then, $CW_f(v) = CW_f(u)$, and in particular, if $CW_f(v) = CW_f(G)$ then $CW_f(u) = CW_f(G)$. Therefore, for this case, Eq. (6) can be rewritten as follows:

$$CW_{f'}(G) \geq \max\{CW_f(G), CW_f(G) + \deg_f^L(u) - \deg_f^R(u)\} \geq CW_f(G)$$

which demonstrates the Property 1 for the second case.

The same reasoning can be easily extended considering any number of consecutive swaps and therefore, reaching any solution in $f_i \in N_f(v)$.

□

From Property 1 we can derive the following consequence. Let $v \in V$ be a vertex such that $\deg(v)$ is even and $\deg_f^R(v) \neq \deg_f^L(v)$. Then, if we consider only movements of vertex v, producing a new ordering f', we would obtain the maximum reduction of the cutwidth value by placing v in positions such that $deg_{f'}^R(v) = deg_{f'}^L(v)$.

Let us illustrate the previous property with the example in Table 1. Given $f = \{A, B, C, D, E, F\}$ (see Fig. 1b) it is clear that vertex A does not verify the condition by Property 1 ($\deg_f^L(A) = 0$ and $\deg_f^R(A) = 4$). Each row of the Table 1 shows a solution obtained by the insertion of vertex A in every position in the ordering f. Additionally, solution f_3 satisfies the condition by Property 1. Notice that no any other solution in $N_f(A)$ is better than f_3. This does not mean that other solutions in $N_f(A)$ could not have the same cutwidth value.

Property 2. *Let f be a solution for the CMP and $v \in V$ a vertex such that $deg_f^R(v) > deg_f^L(v)$. Then, if v is placed in any position i such that $1 \leq i < f(v)$, the objective function can not be improved. Similarly, when $deg_f^R(v) < deg_f^L(v)$ if v is placed in any position i such that $f(v) < i \leq n$, the objective function can not be improved.*

Proof. Given an ordering f of the vertices of a graph, let us suppose that the vertex v in the ordering satisfies that $\deg_f^L(v) < \deg_f^R(v)$. As in Property 1 we consider a swap move between vertices v and u such that $f(u) = f(v) - 1$, producing a new ordering f'. The cutwidth value of u and v in f' can be now computed as:

$$CW_{f'}(u) = CW_f(u) - \deg_f^L(v) + \deg_f^R(v) > CW_f(u) \quad (7)$$

$$CW_{f'}(v) = CW_f(v) + \deg_{f'}^L(u) - \deg_{f'}^R(u) \quad (8)$$

Table 2
Solutions obtained when C is inserted in f (see Fig. 1b) in any position p such that $p < f(C)$.

Solution	Cutwidth
$f_6 = \{A, C, B, D, E, F\}$	$CW_{f_6}(A) = 4, CW_{f_6}(C) = 7, CW_{f_6}(B) = 6, CW_{f_6}(D) = 4, CW_{f_6}(E) = 2$
$f_7 = \{C, A, B, D, E, F\}$	$CW_{f_7}(C) = 3, CW_{f_7}(A) = 7, CW_{f_7}(B) = 6, CW_{f_7}(D) = 4, CW_{f_7}(E) = 2$

Taking into account the relation presented in (7), Eq. (1) can be now rewritten as follows:

$$CW_{f'}(G) \geq \max\{CW_f(w) \forall w \in V \setminus v, CW_{f'}(v)\}$$

As the expression in (8) is equivalent to the expression in (2), the same reasoning as the one presented in Property 1 can be applied here to find that $CW_{f'}(G) \geq CW_f(G)$.

Similarly, when $\deg_f^R(v) < \deg_f^L(v)$ and we perform a swap between u and v such that $f(u) = f(v) + 1$ an equivalent demonstration can be performed. □

From Property 2 we can derive the following consequence. Let $v \in V$ be a vertex such that $\deg(v)$ is odd and therefore it holds $\deg_f^R(v) > \deg_f^L(v)$ or $\deg_f^R(v) < \deg_f^L(v)$. Then, if we consider only movements with vertex v, producing a new ordering f', we would obtain the maximum reduction of the cutwidth value in one of the following two situations: (i) when v is placed in any position such that $\deg_{f'}^R(v) = \deg_{f'}^L(v) + 1$ or (ii) when v is placed in any position such that $\deg_{f'}^R(v) + 1 = \deg_{f'}^L(v)$.

Let us illustrate the Property 2 with the example in Table 2. Considering again the solution depicted in Fig. 1b, it is possible to see that vertex C, with $f(C) = 3$, holds that $\deg_f^L(C) = 1$ and $\deg_f^R(C) = 2$. Therefore, attending to Property 2, any insertion of C in a position $p < 3$ cannot improve the objective function. In Table 2 we show the possible orderings computed by inserting C in positions 2 and 1 obtaining new solutions f_6 and f_7, respectively. It is easy to see that the objective function has not been improved.

3.3. Alternative formulation of the CMP based on the definition of subsets

This alternative formulation for the CMP is based on the ideas presented in [38]. Given a labelling f, it is possible to define subsets of vertices according to their cutwidth value. Let us define S_f^i as the set of vertices with cutwidth value of i in the ordering f. In mathematical terms:

$$S_f^i = \{v \in V : CW_f(v) = i\}$$

Considering the example presented in Fig. 1b, we have the following subsets: $S_f^0 = \{F\}$, $S_f^1 = \emptyset$, $S_f^2 = \{E\}$, $S_f^3 = \emptyset$, $S_f^4 = \{A, D\}$, $S_f^5 = \{B\}$ and $S_f^6 = \{C\}$. Let i_{max} be the index which identifies the set that contains the vertices with the largest cutwidth value. For instance, in the example shown in Fig. 1b, the value of i_{max} is 6. Obviously, $i_{max} = CW_f(G)$, so the CMP is equivalent to reduce i_{max}.

However, in this alternative formulation we could compare two solutions beyond the value of i_{max} by taking into account the cardinality of the subsets S_f^i. In Fig. 2 it is shown a different solution, f', over the graph depicted in Fig. 1a where i_{max} is again 6. Despite of the fact that $CW_f(G) = CW_{f'}(G)$, these solutions are not identical since $S_f^{i_{max}} = \{C\}$ and $S_{f'}^{i_{max}} = \{C, E\}$. In fact, we could say that f is better than f' because $|S_f^{i_{max}}| < |S_{f'}^{i_{max}}|$.

In general, given two solutions f and f' and an index j ($1 \leq j \leq i_{max}$) we say that f is better than f' if and only if $|S_f^j| < |S_{f'}^j|$ and $|S_f^i| = |S_{f'}^i|$

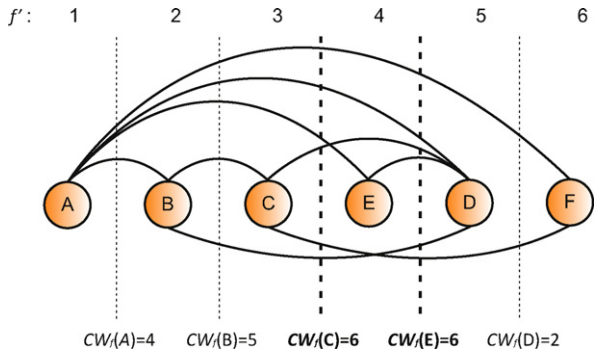

Fig. 2. Ordering f of the vertices of the graph in Fig. 1a with the corresponding cutwidth of each vertex.

for all i, such that $j < i \leq i_{max}$. To compress these conditions in a mathematical expression, we propose the definition of a new objective function, $CW_f^2(G)$, as follows:

$$CW_f^2(G) = \sum_{i=1}^{i_{max}} n^i |S_f^i|$$

Therefore, the CMP can be now reformulated as follows:

$$CW^2(G) = \min_{f \in \Pi_n} CW_f^2(G)$$

where $CW^2(G)$ is the minimum value of CW_f^2 over all possible labellings Π_n. Notice that an optimal solution of $CW^2(G)$ is also optimal in $CW(G)$, but the reverse statement is not necessarily true.

3.4. Alternative formulation of the CMP based on the distance to an ideal solution

Let f_1 and f_2 be two different solutions for the CMP. We can define $d(f_1, f_2)$ as a distance function between those solutions. Particularly, if one of the previous solutions (f^*) is optimal, then the CMP can be alternatively formulated as follows:

$$CW^3(G) = \min_{f \in \Pi_n} d(f, f^*)$$

where $CW^3(G)$ is the minimum value of $d(f, f^*)$ over all possible labellings Π_n. Unfortunately, considering the optimal solution as a point of reference is not possible as, in general, it is unknown and in fact, it is the original goal of the CMP. However, instead of using the optimal solution we could consider an "ideal" (maybe infeasible) solution f_-, as a reference point. The "ideal" solution should have a cutwidth value at least as good as the optimal one. It means that $CW_{f_-}(G)$ is a lower bound for the CMP. Consequently, minimizing the distance between any solution and f_- does not necessarily implies approaching to the optimal solution. Therefore, minimizing $d(f, f_-)$ is not equivalent to solve the CMP. However, this alternative formulation could help us to find good solutions for the problem. Consequently, any solution (probably infeasible) whose objective function is a lower bound for the CMP can be used as an "ideal" solution. In this paper we use the procedure proposed by [33] (see Section 3.4 in that paper) to construct an ideal solution.

We propose now two different distance functions. The first one is inspired by the formulation introduced in Section 3.3. Specifically, it is based on the comparison of the subsets defined in the previous formulation. Given a solution f, we define $d_1(f, f_-)$ as follows:

$$d_1(f, f_-) = \sum_{i=1}^{i_{max}} \text{abs}\left(|S_f^i| - |S_{f_-}^i|\right)$$

The distance d_1 only considers the absolute difference between subsets S_f^i and $S_{f_-}^i$, but not where these differences appear.

The second distance that we propose, d_2, is based on the definition of labelling. Each vertex v in the graph has assigned a different label $f(v)$ that is nothing but the position i that the vertex occupies in the arrangement (see Figs. 1b and 2). Let us now define $f^{-1}(i)$ as the function that, given a position i, determines the corresponding vertex associated to that position. By definition, the relation between f and f^{-1} is such that if $f(v) = i$ then $f^{-1}(i) = v$. Through the minimization of d_2 we are trying to find solutions closer to the "ideal" one. Specifically, we minimize the difference between the cutwidth of the vertices that are placed in the same position in f_- and f. Therefore, the distance is defined as follows:

$$d_2(f, f_-) = \min\left(\sum_{1 \leq i < n} \text{abs}(CW_f(f^{-1}(i)) - CW_{f_-}(f_-^{-1}(i))), \sum_{1 \leq i < n} \text{abs}(CW_f(f^{-1}(i)) - CW_{f_-}(f_-^{-1}(n-i)))\right)$$

Distance d_2 considers that reverse labellings are equivalent for the CMP. Let $f = (v_1, v_2, \ldots, v_n)$ and $f' = (v'_1, v'_2, \ldots, v'_n)$ be two labellings of a set of n vertices where $f^{-1}(i) = f'^{-1}(n - i + 1)$, $1 \leq i \leq n$. Then, we say that f and f' are reverse labellings and therefore equivalent solutions for the CMP. Specifically, it holds that $CW_f(f^{-1}(i))$ is equal to $CW_{f'}(f'^{-1}(n-i))$, for all $1 \leq i < n$.

4. VFS for the CMP

In this section we present the adaptation of the VFS methodology to tackle the CMP. In particular, we describe in detail all the steps of the VFS algorithm, which include: initialization (see Section 4.1), shaking (see Section 4.2) and improvement (see Section 4.3). Finally, in Section 4.5 we describe how the new formulations for the CMP, presented in Sections 3.3 and 3.4, are combined in the `Accept` procedure.

4.1. Initial solution

The VFS algorithm, as well as the BVNS, starts from an initial solution which is represented by the parameter f (see step 1 in Algorithm 1). Despite of the fact that it is possible to start from a random solution, a constructive procedure will allow the VFS to perform faster. However, it is not our intention to propose a new constructive procedure for the CMP in this paper, since it is possible to find many in the related literature. Specifically, in [12] the authors proposed three greedy algorithms to construct solutions for the problem. In [3] it was proposed a new one where they started with a randomized depth-first search ordering and then, the vertices were placed in the position where the increase in the objective function is minimized. Recently, in [38] the authors proposed two new constructive procedures (named C1 and C2) based on the GRASP methodology [17] each of them using a different greedy criterion. They also proposed two additional algorithms (named C3 and C4) where the random and greedy stages of GRASP were permuted. We have selected one of the newer constructive procedures by [38] as the procedure to construct the initial solution for our VFS. The authors of that paper compared their different constructive

procedures concluding that C1 is the best alternative when considering only quality and C2 is the best option when considering both, quality and diversity. Therefore, we select C1 for the VFS, since it requires only one solution with high quality.

C1 performs a GRASP construction by labelling the vertices sequentially with labels 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 (in the first iteration all vertices are candidates). For each vertex $u \in CL$ it is performed an evaluation of its cutwidth $CW_f(u)$ considering that u is labelled with the next available label k, so $f(u) = k$. Minimum and maximum $CW_f(u)$ are computed from any vertex $u \in CL$ and denoted as CW_{min} and CW_{max}, respectively. These values, together with the search parameter α (selected at random in $[0, 1]$) are used to compute a threshold ($th = CW_{min} + \alpha(CW_{max} - CW_{min})$). Then, a Restricted Candidate List (RCL) is constructed with the vertices in CL with a cutwidth value lower than th. Finally, a vertex u^* is selected at random from the RCL and assigned the label k. We used this procedure to construct a predefined number of solutions (1000) selecting the best one as the initial solution for the VFS.

4.2. Shaking

One of the main strategies within the BVNS schema and, therefore in VFS, is the shaking procedure (denoted as `Shake` in the step 4 of Algorithm 1). This strategy consists of performing perturbations in the current solution to diversify the search, giving the local search in the VFS schema a new starting point with a different neighbourhood. The diversification of the shaking procedure is carried out in a controlled way. In other words, the number of perturbation moves performed in the current solution is increased gradually until it reaches a maximum value. The pseudocode of the `Shake` procedure is shown in Algorithm 3. This procedure receives two input parameters: a solution, f, and the number of perturbations to perform, k. It is well documented in the VNS literature that the neighbourhood defined by an insertion move is different than the one defined by an interchange move. As we already use insertion moves in our local search procedure, for the sake of diversification, we use interchange moves in the shaking procedure. An interchange move can be defined as follows: given a solution f and two different vertices $u, v \in V$, an interchange move, $\mathrm{exc}(f, u, v)$, produces a new solution f' where $f'(u) = f(v)$, $f'(v) = f(u)$ and $f'(w) = f(w)$ for all $w \in V$ not equal to u or v (step 5 in Algorithm 3). The vertices that interchange their positions are previously selected at random (step 4). The procedure finishes when the maximum number of movements k are performed, returning the corresponding perturbed solution f.

Algorithm 3. Pseudocode of the shaking procedure

```
1: Procedure Shake (f, k)
2: i ← 1
3: repeat
4:   Select two vertices u and v at random such that u ≠ v
5:   f ← exc(f, u, v)
6:   i ← i + 1
7: until i = k
8: return f
9: end Shake
```

4.3. Local search

The local search procedure (denoted as `LocalSearch` in step 6 of Algorithm 1) will allow us to reach a local minimum from the current solution at each iteration of the VFS.

In this paper we propose a local search procedure based on insertion moves. 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, N, of this kind of moves has a size relatively large. Therefore, an exhaustive exploration of N could deteriorate the performance of the method, since it requires to insert each vertex in any position of the ordering ($|N| = n^*(n-1)$). Consequently, the complexity of the local search method would be $O(n^2)$. However, considering the properties presented in Section 3.2 we can drastically reduce the size of the associated neighbourhood. Specifically, from Property 1 we can conclude that given a vertex v with even degree, it is only required to perform one insertion. Additionally, from Property 2 we can also conclude that two insertions are only required when v has an odd degree. As a consequence, It is important to remark that this fact reduces the complexity of the local search from $O(n^2)$ to $O(n)$. To determine the positions where a vertex v will be inserted, we define the set of labels of the vertices adjacent to v, denoted as $P(v)$. In mathematical terms $P(v) = \{f(u) : (u, v) \in E\}$. Let us consider that the elements in $P(v)$ are sorted in ascending order. Then, the median label of a vertex v, denoted as $med(v)$ is computed as the numerical label which separates the higher half of $P(v)$ from the lower half.

The value of $med(v)$ can be used to determine the best position to insert the vertex v. Specifically, if v has an even degree, then we can insert v in $med(v)$, obtaining a new solution f' where half of the neighbours of v have a label smaller than $med(v)$ and the other half larger. Notice that after this move, v in f' satisfy the conditions required by Property 1. Consequently, we do not need to perform any additional insertion with v. When $\deg(v)$ is odd it is not possible to find a label that separates the set of adjacent vertices into equal subsets. However, considering Property 2, it is only needed to insert vertex v in two positions, obtaining two solutions f' and f''. Let u be the vertex such that $f(u) = med(v)$. Then, one insertion is performed right before u, obtaining f' ($f'(v) - 1 = f'(u)$). Symmetrically, the other insertion is performed right after u, obtaining f'' ($f''(v) + 1 = f''(u)$).

As a side effect of the properties described in Section 3.2, we can conclude that some vertices in the ordering does not need to be moved. Specifically, those vertices satisfying Property 1 cannot improve the objective function when they are inserted in other position in the ordering. This fact would also help the procedure to save computing time.

In Algorithm 4 we present the pseudocode of the proposed local search. This procedure receives an initial solution, f, as an input parameter. In step 2, f is considered as the best solution found (*bestSol*) and it will be updated in future iterations. In step 6, we construct the list of vertices able to produce improvements in *bestSol*, C, when they are inserted in other positions according to the Property 1 (see Section 3.2). The list C is then sorted on the basis of 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 performed in steps 7–16. Specifically, in step 8 it is calculated, for the considered vertex v at this iteration, the list of positions, P, where it will be inserted according to Property 2 (see Section 3.2). Particularly, if $d(v)$ is even, v will be inserted only in one position and if $d(v)$ is odd it will be inserted in two positions, obtaining a new solution f' (step 10). If the move is accepted (step 11–14) the best solution found will be updated. The CMP presents a flat landscape, therefore, many moves results in a null value with respect to the objective function. To overcome this lack of information, we will extend the meaning of improving, in the procedure `Accept` (step 11), as presented in Section 2. The specific `Accept` procedure for the CMP which make use of the new formulations of the problem is described in Section 4.5. Finally, the local search procedure ends when no one of the movements performed with the vertices in the list C is able to produce an improvement, returning *bestSol* (step 18).

Algorithm 4. Local search procedure

```
1:  Procedure LocalSearch (f)
2:  bestSol ← f
3:  improved ← True
4:  while improved do
5:    improved ← False
6:    C← InitializeC (f) /*Vertices able to produce an
      improvement in f*/
7:    for all v ∈ C do
8:      P← InitializeP (f, v) /*Candidate positions for v*/
9:      for all p ∈ P do
10:       f' ← ins(bestSol, p, v)
11:       if Accept(f', bestSol) then
12:         bestSol ← f'
13:         improved ← True
14:       end if
15:     end for
16:   end for
17: end while
18: return bestSol
19: end LocalSearch
```

4.4. Neighbourhood change

The change of neighbourhood is also a key factor of the VNS methodology. This is performed in a controlled way on the basis of the value of k. The control of the value of k is performed by the procedure `NeighbourhoodChange` introduced in step 7 of Algorithm 1. This procedure examines if the solution obtained after the local search is better than any other solution previously found. If so, the best solution found is updated and k 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. The pseudocode of the `NeighbourhoodChange` procedure can be found in Algorithm 5 where f and f' represent respectively the best solution found and the new solution obtained after the local search procedure in the VFS schema. As it was mentioned earlier, the CMP presents a flat landscape so, when two solutions have the same objective function value, it is hard to determine which one is better. In step 2, the method `Accept` uses the extended meaning of improving (see Section 4.5) to compare two solutions beyond the value of the objective function. If f' is considered as the best solution found, then f is updated (step 3) and the value k is set to 1. Otherwise, the value of k is increased in one unit (step 6).

Algorithm 5. Pseudocode of the `NeighbourhoodChange` procedure

```
1: Procedure NeighbourhoodChange (f, f', k)
2: if Accept(f, f') then
3:   f← f'
4:   k ← 1
5: else
6:   k ← k+1
7: endif
8: end NeighbourhoodChange
```

4.5. VFS for the CMP

The new version of the VNS methodology is based on the idea of using different formulations of a problem when more than one solution has the same value of the objective function. In this way, any solution found in a formulation space (e.g. the search space defined by a given formulation) should be easily translatable to another equivalent one in any different formulation space.

The previous idea can be exploited in the VNS framework within the `Shake`, `NeighbourhoodChange` and `LocalSearch` procedures beyond the comparison performed with $CW(G)$. Specifically, in every case it is used in the `Accept` procedure to determine which solution is more promising to continue the search. The pseudocode of the `Accept` procedure of VFS within the CMP context is shown in Algorithm 6. In particular, we use the original formulation ($CW(G)$) and the proposed alternative formulations ($CW^2(G)$ and $CW^3(G)$) to compare two solutions f_1 and f_2. If $CW_{f_1}(G) < CW_{f_2}(G)$, we can conclude that f_1 is better than f_2 (step 2). On the other hand, if both solutions have the same value of the objective function it is not possible to assure which solution is better. However, f_1 and f_2 can be compared using the alternative formulations proposed above. In particular, if $CW_{f_1}(G) = CW_{f_2}(G)$, we consider that f_1 is more promising than f_2 when $CW^2_{f_1}(G) < CW^2_{f_2}(G)$ (step 4). Similarly, we also consider that f_1 is more promising than f_2 when $CW_{f_1}(G) = CW_{f_2}(G)$ and $CW^3_{f_1}(G) < CW^3_{f_2}(G)$ (step 6). Finally, if none of the previous assumptions are fullfilled the procedure returns False which means that f' is not as promising as f.

Algorithm 6. `Accept` procedure for the CMP

```
1:  Procedure Accept (f, f')
2:  if(CW_f'(G) < CW_f(G))then
3:    return True
4:  else if(CW_f'(G) = CW_f(G)) and (CW^2_f'(G) < CW^2_f(G)) then
5:    return True
6:  elseif(CW_f'(G) = CW_f(G)) and (CW^3_f'(G) < CW^3_f(G)) then
7:    return True
8:  else
9:  return False
10: end if
11: end Accept
```

We have empirically found that the strategy presented above allows the search procedure to explore a larger number of solutions than the implementation where only one formulation is considered. Notice that, in general, this strategy is more time consuming since it requires to evaluate more than one objective function over the same solution. However, it is worth using it coupled with the properties described in Section 3.2, as we illustrate in the experiments.

5. Computational results

In this section we present the computational experiments that we performed to test the efficiency of our Variable Formulation Search algorithm. We first present the set of instances used for the evaluation and some preliminary experiments. These experiments were performed to test, separately, the effect of some of the key factors of the proposed procedure such as: the properties of the solutions of the CMP, the effect of the size of k_{max} and the performance of the use of different formulations. Finally, we perform a comparison with previous state-of-the-art algorithms.

The algorithms were implemented in Java 6 SE and all the experiments were carried out on an Intel Core 2 Quad CPU with 6 GB of RAM.

5.1. Instances

To evaluate the proposed algorithm we have used two sets of instances previously reported by other authors in the evaluation of the CMP. Particularly, we used the sets of instances proposed by [38] who, as far as we know, holds the best previous heuristic algorithm for the CMP. Next, we describe each set of instances, named "*Grid*" and "*Harwell-Boeing*":

- "*Grid*": This data set consists of 81 matrices constructed as the Cartesian product of two paths [41]. They were originally introduced by [44] and the optimal solution of the CMP for these type of instances is known by construction [41]. For this set of instances, the vertices are arranged on a grid with a dimension width × height where width and height are selected from the set {3, 6, 9, 12, 15, 18, 21, 24, 27}.

Table 3
Effectiveness of the properties presented in Section 3.2.

	Constr.	Constr. + Naive LS	Constr. + LS
Avg.	554.38	330.94	237.75
Dev. (%)	1132.13	692.43	489.98
CPUt (s)	30.07	31.58	30.07

- "*Harwell-Boeing*" (HB): This data set is a subset of the public-domain Matrix Market library.[1] This collection consists of a set of standard test matrices $M = (M_{ij})$ arising from problems in linear systems, least squares, and eigenvalue calculations from a wide variety of scientific and engineering disciplines. Graphs were derived from these matrices by [38] by considering an edge (i, j) for every element $M_{ij} \neq 0$. The data set is formed by the selection of the 87 instances were $n \leq 700$. Their number of vertices ranges from 30 to 700 and the number of edges from 46 to 41686.

Notice that [38] also included in their comparison another set of instances (named "*Small*"). However, this set does not help to discriminate among the algorithms, since all of them were able to find the optimum solution in a few seconds. All the data sets are available at http://www.optsicom.es/cutwidth.

5.2. Preliminary experiments

In this section we perform three different preliminary experiments to illustrate some of the strategies proposed in this paper. The first experiment is devoted to measure the effectiveness of the properties presented in Section 3.2 within the local search procedure. The second one deals with the impact of the value of the search parameter k_{max}. Finally, the third preliminary experiment is devoted to compare the use of different formulations, instead of using only one.

To perform the preliminary experiments we selected a representative 10% of the instances of each data set presented in Section 5.1. For short, let us refer to these set of 16 instances as *Test* data set. Particularly, the instances in the *Test* data set are: *grid12x27, grid15x12, grid15x15, grid18x18, grid21x15, grid21x21, grid24x9, grid27x27, 662_bus, bcsstk06, can_445, dwt_503, impcol_a, plat362, saylr1, shl_200*.

5.2.1. Effectiveness of the properties of the CMP solutions

In the first preliminary experiment we test the efficiency of the properties presented in Section 3.2 when applied within the local search procedure. In Table 3 it is shown the average of the objective function, Avg., the deviation to the best known solution, Dev. (%), and the computing time, CPUt(s), in seconds, of three procedures. The procedures were executed for 30 s over each instance of the *Test* data set. In the first column of Table 3 (Constr.) we present the results obtained by a randomized constructive procedure. The second column (Constr. + Naive LS) contains the results obtained by the same randomized constructive procedure paired with a naive implementation of the local search (i.e. any vertex is inserted in all positions performing the first improvement move). Finally, the third column (Constr. + LS) of Table 3 contains the results of the randomized constructive procedure paired with the new version of the local search presented in Section 4.3 which makes use of the properties in Section 3.2. Notice that the algorithms which include a local search procedure construct and improve as many solutions as they can in 30 s while the other only constructs solutions.

As it is shown in Table 3 the new local search is able to reach higher quality solutions in both, average and deviation, in the same CPU time than the naive implementation. This is due to the effectiveness of the properties mentioned above which help to reach a local optimum faster. This fact allows the procedure to scan a higher number of solutions than the naive implementation. Therefore, this local search will be used for future experiments.

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

Table 4
Preliminary experiment: adjust of the search parameter k_{max}.

k_{max}		Time limit (s) 0.4n	0.5n	0.6n
0.1n	Dev. (%)	1.71	1.67	1.60
	CPUt (s)	156.37	195.46	234.59
0.2n	Dev. (%)	1.70	1.69	1.68
	CPUt (s)	156.37	195.49	234.53
0.3n	Dev. (%)	1.77	1.75	1.70
	CPUt (s)	156.37	195.45	234.53

5.2.2. Adjust of the value of the search parameter k_{max}

The next experiment is devoted to adjust the value of the search parameter k_{max}. The higher k_{max}, the more exploration performed and the more CPU time. The aim is to find a compromise between the quality of the results obtained and the CPU time needed. In Table 4 we present the results of different executions of a BVNS algorithm which makes use of the best local search of the previous experiment. We have tested this algorithm with three different values of k_{max}, calculated as a percentage of the number of vertices of each instance. Particularly, we have considered a 10%, 20% and 30% of n (being n the number of vertices of the graph). The algorithm was stopped after a determined amount of time for each instance, also considered on the basis of the number of vertices of the instance. Specifically, we have considered a 40%, 50% and 60% of n in seconds as the time limit. BVNS starts from a random solution.

In Table 4 it is shown that, for the same k_{max} value, if the time limit is higher, the deviation, Dev. (%), is smaller. Additionally, if we compare the results obtained with different values of k_{max} for the same CPU time, we identify that when the time limit is 0.4n seconds, $k_{max} = 0.2n$ is the best setup. However, for higher time limits, $k_{max} = 0.1n$ is the best configuration.

5.2.3. Effectiveness of the use of alternative formulations

In the last preliminary experiment we illustrate the effectiveness of using more than one formulation to determine which solution is more promising to continue the search. In Table 5, we present the results obtained with four different algorithms when executing them for 30 s over each instance of the *Test* data set. The column BVNS represents an algorithm based on the BVNS methodology which make use only of the original formulation of the CMP to determine which solution is more promising to carry on the search. VFS_1 is the first version of our new Variable Formulation Search algorithm. It is based in the previous BVNS but combines the original formulation of the CMP with the formulation CW^2 presented in Section 3.3. VFS_2 is equivalent to the previous one with the difference that now CW^3 (see Section 3.4) is considered instead of CW^2. Finally, the fourth column of the table, denoted as VFS_3 combines the original formulation of the CMP with the two alternative ones, in the way presented in Section 4.5. All the algorithms were configured with $k_{max} = 0.1n$ and they start from the same random solution.

Table 5
Impact of the use of alternative formulations in the search process.

	BVNS	VFS_1	VFS_2	VFS_3
Avg.	137.31	93.56	91.56	90.75
Dev. (%)	192.44	60.40	49.23	48.22
CPUt (s)	30.17	30.47	30.50	30.96

Table 6
Comparison with the state-of-the-art algorithms over the Grid data set.

Grid (81)	GPR [3]	SA [12]	SS [38]	VFS
Avg.	38.44	16.14	13.00	12.23
Dev. (%)	201.81	25.42	7.76	3.25
#Opt.	2	37	44	59
CPU t (s)	235.16	216.14	210.07	90.34

Table 7
Comparison with the state-of-the-art algorithms over the Harwell–Boeing data set.

HB (87)	GPR [3]	SA [12]	SS [38]	VFS
Avg.	364.83	346.21	315.22	314.39
Dev. (%)	95.13	53.30	3.40	1.77
#Best	2	8	47	61
CPU t (s)	557.49	435.40	430.57	128.12

The results presented in Table 5 clearly confirm that the use of more than one formulation to determine the best solution to carry on the search is better than using only one (the original one). In this sense, the combination $CW + CW^3$ (column VFS_2) performed better than the combination $CW + CW^2$ (column VFS_1). Finally, combining both strategies at the same time (column VFS_3) was the best variant over the considered amount of time. Therefore we will use this combination for the final experiments.

5.3. Comparison with other state-of-the-art algorithms

Finally, we compare the best version of our VFS algorithm with the algorithms in the state of the art, over the whole sets of instances presented in Section 5.1. Specifically, the compared procedures were: Simulated Annealing (SA) [12]; Greedy Randomized Adaptive Search Procedure with Path Relinking (GPR) [2]; Scatter Search (SS), [38] and the new VFS (VFS). The VFS algorithm was configured as follows: we constructed 1000 solutions with the procedure presented in Section 4.1 and we selected the best one to start the search. We used the local search presented in Section 4.3 making use of the properties of the solutions of the CMP presented in Section 3.2. The parameter k_{max} was set to $0.2n$ and the maximum CPU time per instance was set to $0.4n$ being n the number of vertices of each instance. Notice that the CPU time also includes the time spent by the constructive procedure. Finally, we used the variant VFS_3 where the two alternative formulations of the CMP presented in Section 3 were used to determine which solution is more promising to carry on the search.

In Table 6 we present the comparison with the previous mentioned algorithms over the *Grid* data set while, in Table 7, we present the comparison over the *HB* data set. In both cases it is possible to see that the VFS variant outperforms previous algorithms in the state of the art. Specifically, in the *Grid* data set VFS obtained a 3.25% of deviation while the closer algorithm (SS) obtained a 7.76% of deviation in more than double of computing time than VFS. Similarly, in the *HB* data set, VFS is again the best algorithm obtaining a 1.77% of deviation while the closer algorithm (SS) obtained a 3.40% of deviation, again in more than double of computing time than VFS. The detailed results per instance can be found at http://www.optsicom.es/cutwidth.

To complement this information, we have applied the Friedman test to the best solutions obtained by each of the four algorithms. This experiment was performed separately for the *Grid* data set and for the *HB* data set. In both cases the resulting p-value of 0.000 obtained clearly indicates that there are statistically significant differences among the four methods tested. A typical post-test analysis consists of ranking the methods under comparison according to the average rank values computed with this test. According to this, the best method in the *HB* data set is VFS (with a rank value of 1.44), followed by the SS (1.70), SA (3.24) and finally GPR (with 3.63 rank value). With respect to the *Grid* data set, VFS is again the best method (rank value 1.52) followed by SS (1.98), SA (2.54) and GPR (3.96). Additionally, we compared the two best procedures (VFS and SS) with two well known nonparametric tests for pairwise comparisons: the Wilcoxon test and the Sign test. The former, measures whether the solutions come or not from different methods (i.e. they are significantly different) while, the latter, computes the number of instances on which an algorithm improves upon the other. The obtained p-value of 0.000 over the *Grid* data set and the p-value of 0.009 over the HB data set confirm that the there are differences between both methods, resulting VFS as the best one of this experiment.

6. Conclusions

In this paper we propose a new variant of the VNS methodology. The new variant, named Variable Formulation Search (VFS), makes use of the main characteristics of the VNS framework and adds new criteria to compare solutions. The aim of VFS is to determine whether a given solution is more promising than other to continue the search, beyond the value of the objective function. This fact is specially helpful when dealing with min–max problems, where there are many solutions with the same value of the objective function. In this case, when two solutions have the same value of the objective function, VFS performs a new comparison based on the use of alternative formulations of the problem. We have applied this new VNS variant to the Cutwidth Minimization Problem by proposing two alternative formulations. Additionally, we have devised formal properties of the solutions to the CMP which can drastically reduce the size of the associated neighbourhood. Specifically, these properties determine the positions where each vertex can improve the objective function. We performed extensive computational experiments over a set of 171 instances and we compared the efficiency of our proposal with previous solution procedures. Experimental results show that VFS approach clearly outperforms previous attempts in the state of the art for the CMP in terms of quality and computing time. In particular, considering the best method in the state of the art, VFS obtained much better solutions in half of the computing time.

Acknowledgment

This research has been partially supported by the Spanish Government grants, Refs. TIN2008-06890-C02-02, TIN2009-07516, TIN2011-28151 and TIN2012-35632.

References

[1] D. Adolphson, T.C. Hu, Optimal linear ordering, SIAM Journal on Applied Mathematics 25 (1973) 403–423.
[2] D.V. Andrade, M.G.C. Resende, GRASP with evolutionary path-relinking, in: Seventh Metaheuristics International Conference (MIC)., 2007.
[3] D.V. Andrade, M.G.C. Resende, GRASP with path-relinking for network migration scheduling, in: Proceedings of International Network Optimization Conference (INOC)., 2007.
[4] H.M. Azamathulla, A. Ab.Ghani, S.Y. Fei, Short communication: anfis-based approach for predicting sediment transport in clean sewer, Applied Soft Computing 12 (2012) 1227–1230.
[5] H.M. Azamathulla, F.C. Wu, Support vector machine approach for longitudinal dispersion coefficients in natural streams, Applied Soft Computing 11 (2011) 2902–2905.
[6] G. Blin, G. Fertin, D. Hermelin, S. Vialette, Fixed-parameter algorithms for protein similarity search under mRNA structure constraints, Journal of Discrete Algorithms 6 (2008) 618–626.
[7] C. Blum, J. Puchinger, G.R. Raidl, A. Roli, Hybrid metaheuristics in combinatorial optimization: a survey, Applied Soft Computing 11 (2011) 4135–4151.
[8] R.A. Botafogo, Cluster analysis for hypertext systems, in: 16th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, 1993, pp. 116–125.

[9] S. Butenko, O. Yezerska, B. Balasundaram, Variable objective search, Journal of Heuristics, http://dx.doi.org/10.1007/s10732-011-9174-2, in press.
[10] M.J. Chung, F. Makedon, I.H. Sudborough, J. Turner, Polynomial time algorithms for the M in Cut problem on degree restricted trees, in: SFCS'82: Proceedings of the 23rd Annual Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, USA, 1982, pp. 262–271.
[11] J. Cohoon, S. Sahni, Exact algorithms for special cases of the board permutation problem, in: Proceedings of the Allerton Conference on Communication, Control and Computing, 1983, pp. 246–255.
[12] J. Cohoon, S. Sahni, Heuristics for backplane ordering, Journal of VLSI and Computer Systems 2 (1987) 37–61.
[13] J. Díaz, A. Gibbons, G.E. Pantziou, M.J. Serna, P.G. Spirakis, J. Toran, Parallel algorithms for the minimum cut and the minimum length tree layout problems, Theoretical Computer Science 181 (1997) 267–287.
[14] J. Díaz, J. Petit, M.J. Serna, A survey of graph layout problems, ACM Computing Surveys (CSUR) 34 (2002) 313–356.
[15] A. Duarte, L.F. Escudero, R. Martí, N. Mladenović, J.J. Pantrigo, J. Sánchez-Oro, Variable Neighborhood Search for the Vertex Separation Problem, Technical Report, Universidad Rey Juan Carlos, 2012.
[16] A. Duarte, R. Martí, M. Resende, R. Silva, Grasp with path relinking heuristics for the antibandwidth problem, Networks (2011) 171–189.
[17] T. Feo, M.G.C. Resende, Greedy randomized adaptive search procedures, Journal of Global Optimization (1995) 109–133.
[18] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman, USA, 1979.
[19] F. Gavril, Some NP-complete problems on graphs, in: Proceedings of the Eleventh Conference on Information Sciences and Systems, Baltimore, MD, 1977, pp. 91–95.
[20] M. Gendreau, J.Y. Potvin, Handbook of Metaheuristics (International Series in Operations Research & Management Science), vol. 146, 2nd ed., Springer, New York, USA, 2010.
[21] F. Glover, G.A. Kochenberger, Handbook of Metaheuristics (International Series in Operations Research & Management Science), Springer, New York, USA, 2003.
[22] P. Hansen, N. Mladenović, J. Brimberg, J.A. Moreno Pérez, Variable neighbourhood search, in: Handbook of Metaheuristics, 2nd ed., Springer, New York, USA, 2010, pp. 61–86.
[23] P. Hansen, N. Mladenović, J.A. Moreno Pérez, Variable neighbourhood search: algorithms and applications, Annals of Operations Research 175 (2008) 367–407.
[24] L.H. Harper, Optimal numberings and isoperimetric problems on graphs, Journal of Combinatorial Theory (1966) 385–393.
[25] A. Hertz, M. Plumettaz, N. Zufferey, Variable space search for graph coloring, Discrete Applied Mathematics 156 (2008) 2551–2560.
[26] D. Karger, A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem, SIAM Journal on Computing 29 (1999) 492–514.
[27] Y. Kochetov, P. Kononova, M. Paschenko, Formulation space search approach for the teachers/class timetabling problem, Yugoslav journal of Operations Research (2008) 1–11.
[28] A. Lim, J.B.R. Lin, F. Xiao, Ant colony optimization with hill climbing for the bandwidth minimization problem, Applied Soft Computing 6 (2006) 180–188.
[29] C. LA3pez, J. Beasley, A heuristic for the circle packing problem with a variety of containers, European Journal of Operational Research 214 (2011) 512–525.
[30] J. Luttamaguzi, M. Pelsmajer, Z. Shen, B. Yang, Integer programming solutions for several optimization problems in graph theory, Technical Report, Center for Discrete Mathematics and Theoretical Computer Science, DIMACS, 2005.
[31] F. Makedon, I.H. Sudborough, On minimizing width in linear layouts, Discrete Applied Mathematics 23 (1989) 243–265.
[32] F.S. Makedon, C.H. Papadimitriou, I.H. Sudborough, Topological bandwidth, SIAM Journal on Algebraic and Discrete Methods 6 (1985) 418–444.
[33] R. Martí, J.J. Pantrigo, A. Duarte, E.G. Pardo, Branch and bound for the cutwidth minimization problem, Computers & Operations Research 40 (2013) 137–149.
[34] N. Mladenović, P. Hansen, Variable Neighborhood Search, Computers & Operations Research 24 (1997) 1097–1100.
[35] N. Mladenović, F. Plastria, D. Urošević, Reformulation descent applied to circle packing problems, Computers & Operations Research 32 (2005) 2419–2434.
[36] N. Mladenović, F. Plastria, D. Urošević, Formulation space search for circle packing problems, in: T. Stützle, M. Birattari, H. Hoos (Eds.), Engineering Stochastic Local Search Algorithms, Designing, Implementing and Analyzing Effective Heuristics, vol. 4638 of Lecture Notes in Computer Science, Springer, Berlin/Heidelberg, 2007, pp. 212–216.
[37] P. Mutzel, A polyhedral approach to planar augmentation and related problems, in: ESA'95: Proceedings of the Third Annual European Symposium on Algorithms, Springer-Verlag, London, UK, 1995, pp. 494–507.
[38] J.J. Pantrigo, R. Martí, A. Duarte, E.G. Pardo, Scatter search for the cutwidth minimization problem, Annals of Operations Research 199 (2012), http://dx.doi.org/10.1007/s10479-011-0907-2.
[39] J. Petit, Experiments on the minimum linear arrangement problem, ACM Journal of Experimental Algorithmics (2003) 8.
[40] E. Piñana, I. Plana, V. Campos, R. Martí, GRASP and path relinking for the matrix bandwidth m inimization, European Journal of Operational Research 153 (2004) 200–210.
[41] A. Raspaud, H. Schröder, O. Sýkora, L. Torok, I. Vrt'o, Antibandwidth and cyclic antibandwidth of meshes and hypercubes, in: 7th International Colloquium on Graph Theory – ICGT 05, Discrete Mathematics 309 (2009) 3541–3552.
[42] M.G.C. Resende, D.V. Andrade, Method and system for Network Migration Scheduling, United States Patent Application Publication US2009/0168665, 2009.
[43] M.G.C. Resende, R. Martí, M. Gallego, A. Duarte, Grasp and path relinking for the max–min diversity problem, Computers and Operations Research 37 (2010) 498–508.
[44] J. Rolim, O. Sýkora, I. Vrt'o, Optimal cutwidths and bisection widths of 2- and 3-dimensional meshes, in: Graph-Theoretic Concepts in Computer Science, vol. 1017 of Lecture Notes in Computer Science, Springer, Berlin/Heidelberg, 1995, pp. 252–264.
[45] J. Sanchez-Oro, M. Laguna, A. Duarte, R. Martí, Scatter Search for the Profile minimization problem, Technical Report, Universidad Rey Juan Carlos, 2012.
[46] T. Schiavinotto, T. Stützle, The linear ordering problem: instances, search space analysis and algorithms, Journal of Mathematical Modelling and Algorithms 3 (2004) 367–402.
[47] F. Shahrokhi, O. Sýkora, L.A. Székely, I. Vrt'o, On bipartite drawings and the linear arrangement problem, SIAM Journal on Computing 30 (2001) 1773–1789.
[48] K. Takagi, N. Takagi, Minimum Cut Linear Arrangement of p-q dags for VLSI layout of adder trees., IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences E82-A (1999) 767–774.
[49] D.M. Thilikos, M.J. Serna, H.L. Bodlaender, Cutwidth II: algorithms for partial w-trees of bounded degree, Journal of Algorithms 56 (2005) 25–49.
[50] M. Yannakakis, A polynomial algorithm for the M in-Cut linear arrangement of trees, Journal of the ACM (JACM) 32 (1985) 950–988.