Branch and Bound for the Cutwidth Minimization Problem

RAFAEL MARTÍ

Departamento de Estadística e Investigación Operativa,
Universidad de Valencia, Spain
rafael.marti@uv.es

JUAN J. PANTRIGO

Departamento de Ciencias de la Computación,
Universidad Rey Juan Carlos, Spain.
juanjose.pantrigo@urjc.es

ABRAHAM DUARTE

Departamento de Ciencias de la Computación,
Universidad Rey Juan Carlos, Spain.
abraham.duarte@urjc.es

EDUARDO G. PARDO

Departamento de Ciencias de la Computación,
Universidad Rey Juan Carlos, Spain.
eduardo.pardo@urjc.es

Submitted to *Computers and Operations Research*

Original version:	June 8, 2010
First revision:	May 4, 2011
Second revision:	October 17, 2011
Third revision:	February 22, 2012

Abstract

The cutwidth minimization problem consists of finding a linear arrangement of the vertices of a graph where the maximum number of cuts between the edges of the graph and a line separating consecutive vertices is minimized. We first review previous approaches for special classes of graphs, followed by lower bounds and then a linear integer formulation for the general problem. We then propose a branch-and-bound algorithm based on different lower bounds on the cutwidth of partial solutions. Additionally, we introduce a Greedy Randomized Adaptive Search Procedure (GRASP) heuristic to obtain good initial solutions. The combination of the branch-and-bound and GRASP methods results in optimal solutions or a reduced relative gap (difference between upper and lower bounds) on the instances tested. Empirical results with a collection of previously reported instances indicate that the proposed algorithm is able to solve all the small instances (up to 32 vertices) as well as some of the large instances tested (up to 158 vertices) using less than 30 minutes of CPU time. We compare the results of our method with previous lower bounds, and with the best previous linear integer formulation solved using Cplex. Both comparisons favor the proposed procedure.

KeyWords: Cutwidth, Branch-and-Bound algorithm, Integer Programming.

1. Introduction

Let $G(V, E)$ be a graph with vertex set V ($|V| = n$) and edge set E ($|E| = m$). A labeling or linear arrangement f, assigns the integers $\{1, 2, \ldots, n\}$ to the vertices of G in such a way that each vertex $v \in V$ has a different label $f(v)$ (i.e., $f(v) \neq f(u)$ for all $u, v \in V$). The cutwidth of v, with respect to f, $CW_f(v)$, is the number of edges $(u, w) \in E$ satisfying $f(u) \leq f(v) < f(w)$. Note that $f(u) = f(v)$ if and only if $u = v$. Then, the cutwidth of v is computed as:

$$CW_f(v) = |\{(u, w) \in E : f(u) \leq f(v) < f(w)\}| \qquad (1)$$

Therefore, the vertex with label n has an associated cutwidth of 0. Given f, the cutwidth of G is defined as:

$$CW_f(G) = \max_{v \in V} CW_f(v) \qquad (2)$$

The optimum cutwidth of G, $CW(G)$, is defined as the minimum $CW_f(G)$ value over all possible labelings. In other words, the cutwidth minimization problem consists of finding an f that minimizes $CW_f(G)$ over the set $\prod_n$ of all possible labelings.

$$CW(G) = \min_{f \in \Pi_n} CW_f(G) \qquad (3)$$

Finding the optimum cutwidth is usually referred to as the Cutwidth Minimization Problem (CMP). This is an *NP-hard* problem as stated in Gavril (1977) even for graphs with a maximum degree of three (Makedon et al., 1985). Practical applications of the CMP can be traced back to the early seventies. Adolphson and Hu (1973) used it as the theoretical model to establish the number of channels in an optimal layout of a circuit (see also Adolphson and Hu, 1973; Makedon and Sudborough, 1989). More recent applications of this problem include network

reliability (Karger 1999), automatic graph drawing (Mutzel, 1995) and information retrieval (Botafogo, 1993). Despite of the practical applicability of the CMP, researchers on heuristic optimization have paid little attention to it. We have only found three references concerning heuristic methods for this problem. Specifically, a Simulated Annealing method (Cohoon and Sahni, 1987), an Evolutionary Path Relinking (Resende and Andrade, 2009) and, more recently, a Scatter Search procedure (Pantrigo et al., 2012), which as far as we know, obtains the best results so far.

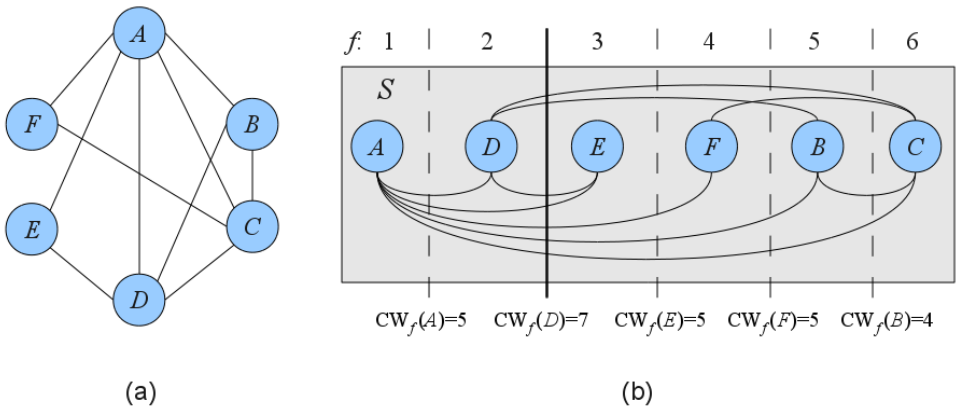


Figure 1: (a) Graph example, (b) Cutwidth of $\vec{G}$ for f.

Figure 1.a is an example of an undirected graph with six vertices and ten edges. A labeling of this graph is depicted in Figure 1.b, setting the vertices in a line in the labeling order as commonly represented in the cutwidth problem. In this way, since $f(A) = 1$, vertex A comes first, followed by vertex D $(f(D) = 2)$ and so on. We represent f with the ordering (A, D, E, F, B, C), meaning that vertex A is located in the first position (label 1), vertex D is located in the second position (label 2) and so on. In Figure 1.b, the cutwidth of each vertex is represented as a dashed line with its corresponding value at the bottom. For example, the cutwidth of vertex A is $CW_f(A) = 5$, because the edges $(A, D), (A, E), (A, F), (A, B)$ and (A, C) have an endpoint in A labeled with 1, and the other endpoint in a vertex labeled with a value larger than 1. Similarly, we can compute the cutwidth of vertex B, $CW_f(B) = 4$, by counting the appropriate number of edges $((A, C), (D, C), (F, C)$ and $(D, C))$. Then, since the cutwidth of

$\vec{G}$, $CW_f(\vec{G})$ is the maximum of the cutwidth of all vertices in $\vec{V}$, in this particular example we obtain $CW_f(\vec{G}) = CW_f(D) = 7$, represented in the figure as a bold line with the corresponding value at the bottom.

In this paper we propose a branch-and-bound algorithm for the Cutwidth Minimization Problem. It basically consists of a systematic enumeration of all its solutions (labelings) based on the definition of *partial solutions*. We review the related literature on the CMP in Section 2 and propose four new lower bounds in Section 3 that will enable us to discard a large number of solutions in the enumeration process. This latter section ends with a study of the dominance among the lower bounds. In Section 4 we study the relative dominance among nodes in the search tree. In Section 5 we introduce a heuristic based on the Greedy Randomized Search Procedure (GRASP) methodology to obtain an initial upper bound for the CMP. The reader is referred to Resende and Ribeiro (2010); Festa and Resende (2009a) and Festa and Resende (2009b) for further details concerning the GRASP methodology. In Section 6, we describe the search tree and its associated strategies for an efficient enumeration of the problem solutions, and the paper concludes with the computational experiments and the associated conclusions.

2. Previous methods, bounds and formulations

The CMP has been optimally solved for some special classes of graphs. For example, Harper (1966) solved the cutwidth for hypercubes, Chung et al. (1982) presented an $O(\log^{d-2} n)$ time algorithm for the cutwidth of trees with n vertices and with maximum degree d. Yannakakis (1985) improved these results by giving an $O(n \log n)$ time algorithm for the same kind of graphs. In particular, for *k*-level *t*-ary trees, $T_{t,k}$, it holds that:

$$CW_f(T_{t,k}) = \left\lceil \frac{1}{2}(k-1)(t-1) \right\rceil, \forall k \leq 3 \qquad (4)$$

Exact methods to obtain the optimal cutwidth of grids have been proposed in Rolim et al.

(1995). Specifically, for a grid $L_{w,h}$ with width $w \geq 2$ and height $h \geq 2$, these authors proved that:

$$CW(L_{w,h}) = \begin{cases} 2, & if\ w = h = 2 \\ \min\{w+1, h+1\}, & otherwise \end{cases} \quad (5)$$

Recently, Thilikos et al. (2005) presented an algorithm to compute the cutwidth of bounded degree graphs with small tree-width in polynomial time. As far as we know, there is no previous exact method for the CMP on general graphs, and all the previous methods, as shown above, target special classes of graphs. However, we have identified four previous lower bounds and a linear integer formulation that we describe in the following subsections.

2.1 Lower bounds for the CMP

Díaz et al., (2002) proposed two lower bounds for the CMP. The first one is based on fundamental cuts and the second one in spectral properties of graphs. The computation of the former is based on the well-known max-flow min-cut theorem (Ford and Fulkerson, 1962), which states that the maximal flow value from an origin o to a destination d in a given graph is equal to the minimal edge cut separating o and d (called a *fundamental cut*). If we compute the value of the fundamental cut for all the possible pairs (o, d) in a given graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the maximum of these values is a lower bound of the CMP (Díaz et al., 2002) that we denote as LB_{FF}. In mathematical terms:

$$CW(\mathcal{G}) \geq LB_{FF} = \max_{o,d \in \mathcal{V}} \{cut(o, d)\} \quad (6)$$

where $cut(o, d)$ represents the value of the fundamental cut from o to d.

Considering the Laplacian matrix associated to a graph, it is possible to derive a lower bound for the CMP using its second smallest eigenvalue (Juvan and Mohar, 1992). Given a connected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with $|\mathcal{V}| = n$, let λ_2 be the second smallest eigenvalue. The LB_{LM} lower bound

can be computed as:

$$CW(\mathcal{G}) \geq LB_{LM} = \lambda_2 \frac{\left\lfloor \frac{n}{2} \right\rfloor \left\lceil \frac{n}{2} \right\rceil}{n} \quad (7)$$

Additionally, we can derive new lower bounds by studying the relations of the CMP with other layout optimization problems. Specifically, Díaz et al., (2002) presented an inequality between the CMP and the Minimum Linear Arrangement problem, MinLA (Garey et al., 1976; Petit, 2003), and another one between the CMP and the Edge Bisection problem, EB (Garey et al. 1976). Given a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with $|\mathcal{V}| = n$, and a labeling f, then:

$$LA_f(\mathcal{G}) \leq n \cdot CW_f(\mathcal{G}) \text{ with } f \in \Pi_n \quad (8)$$

$$EB_f(\mathcal{G}) \leq CW_f(\mathcal{G}) \text{ with } f \in \Pi_n \quad (9)$$

where $LA_f(\mathcal{G})$ and $EB_f(\mathcal{G})$ are the values of the MinLA and EB objective functions, respectively. Consequently, two additional lower bounds, LB_{MinLA} and LB_{EB} can be derived:

$$CW(G) \geq LB_{MinLA} = \frac{LA(G)}{n} \quad (10)$$

$$CW(G) \geq LB_{EB} = EB(G) \quad (11)$$

2.2 Integer programming model

Luttamaguzi et al. (2005) proposed the following CMP linear integer formulation based on the binary decision variables x_i^k, with indices $i, k \in \{1, 2, \ldots, n\}$, specifying whether i is placed in position k in the ordering. This binary variable takes on value 1 if and only if i occupies the position k in the ordering; otherwise x_i^k takes on value 0.

$\min b$

s.t.

$$\sum_{k\in\{1,\dots,n\}} x_i^k = 1 \tag{12}$$

$$\sum_{i\in\{1,\dots,n\}} x_i^k = 1 \tag{13}$$

$$y_{i,j}^{k,l} \leq x_i^k \tag{14}$$

$$y_{i,j}^{k,l} \leq x_j^l \tag{15}$$

$$x_i^k + x_j^l \leq y_{i,j}^{k,l} + 1 \tag{16}$$

$$\sum_{k \leq c<l} y_{i,j}^{k,l} + \sum_{l \leq c<k} y_{i,j}^{k,l} \leq b, \forall c \in \{1,\dots,n-1\} \tag{17}$$

$$x_i^k \in \{0,1\} \tag{18}$$

where $i,j \in \{1,2,\dots,n\}$, $(v_i,v_j)\in\mathcal{E}$, $k,l \in \{1,2,\dots,n\}$. Constraints (12) and (13) ensure that each vertex is only assigned to one position, and one position is only assigned to one vertex respectively. Consequently, constraints (12), (13), and (18) together imply that a solution of the problem is an ordering. Let $y_{i,j}^{k,l}$ indicate whether both $x_i^k = 1$ and $x_j^l = 1$. Thus, $y_{i,j}^{k,l} \geq x_i^k x_j^l$, which is expressed by the traditional linear constraints (16).

Constraint (17) computes for each position c in the ordering, the number of edges whose origin is placed in any position k ($1 \leq k \leq c$) and destination in any position l ($c < l \leq n$). The cutwidth problem consists of minimizing the maximum number of cutting edges in any position,

$c \in \{1, \ldots, n-1\}$ of the labeling. Therefore, the objective function b must be larger than or equal to this quantity.

3. Lower bounds

Given a subset S of $\mathcal{V}$ with $k < n$ vertices and an ordering $g \in \Pi_k$ assigning the integers $\{1, 2, \ldots, k\}$ to the vertices in S, we define a partial solution as the pair (S, g). A complete solution of the cutwidth problem in $\mathcal{G}$ can be obtained by adding $n - k$ elements from $\mathcal{V} \setminus S$ to S, assigning them the integers $\{k+1, k+2, \ldots, n\}$. Therefore, the elements in S ordered according to g can be viewed as an incomplete or partial solution of the cutwidth problem in $\mathcal{G}$. We define U as the set of unlabeled vertices ($U = \mathcal{V} \setminus S$) and S_g as the set of all complete solutions of the problem in $\mathcal{G}$ obtained by adding ordered elements to S. In Figure 2, the partial solution (S, g) of the example introduced in Figure 1.a is shown with the vertices in $S = \{A, D, E\}$ labeled with g ($g(A) = 1, g(D) = 2$ and $g(E) = 3$). Vertices B, C and F remain unlabeled and therefore belong to set U.

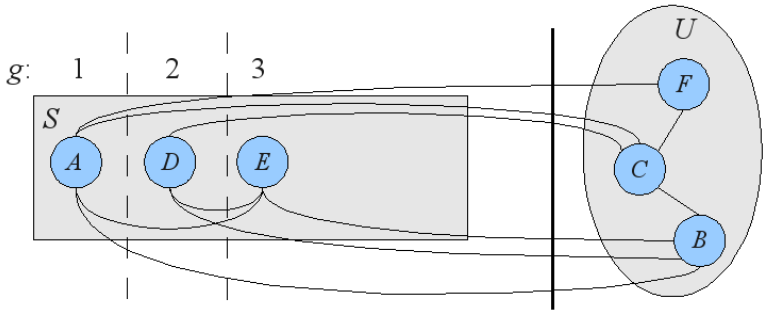


Figure 2: Partial solution.

Given a partial solution (S, g) with $S \subset \mathcal{V}$ and $g \in \Pi_k$, we consider the graph $\mathcal{G}_S(S, E_S)$ where S is the set of labeled vertices and $E_S \subset \mathcal{E}$ is the set of edges among them. In the example depicted in Figure 2, S={A, D, E}, E_S ={(A, E), (D, E)} and S_g={ (A, D, E, F, C, B), (A, D, E, C, B, F), (A, D, E, B, F, C), (A, D, E, F, B, C), (A, D, E, C, F, B), (A, D, E, B, C, F)}.

Given a complete solution of the CMP, the contribution of each vertex to the objective function is computed with the equation (1). However, this formula can be adapted to compute the objective function of a partial solution (S, g). Therefore, we can calculate the cutwidth of each labeled vertex in $\vec{G}_S$ with respect to the ordering g and the edges in E_S, $CW_g(v)$ as follows:

$$CW_g(v) = |\{(u,w) \in E_S : g(u) \leq g(v) < g(w)\}| \qquad (19)$$

In Figure 2, we have $CW_g(A) = 1, CW_g(D) = 2$ and $CW_g(E) = 0$. It is clear that the cutwidth values in the partial solution provide a lower bound of their corresponding values than in any complete solution $f \in S_g$. In this example, if f is a complete solution (with 4, 5 and 6 assigned to C, B and F), we have $CW_f(A) \geq CW_g(A) = 1$, $CW_f(D) \geq CW_g(D) = 2$ and $CW_f(E) \geq CW_g(E) = 0$. We can therefore conclude that the cutwidth of $CW_f(\vec{G})$ is larger than $\max\{CW_g(A), CW_g(D), CW_g(E)\} = 2$ and say that this maximum is a lower bound of the cutwidth. In mathematical terms, for any $f \in S_g$:

$$CW_f(\vec{G}) \geq LB\ (S,g) = \max_{v \in S} CW_g(v) \qquad (20)$$

In this section we propose four lower bounds, LB_1, LB_2, LB_3 and LB_4, to the value of $CW_f(\mathrm{G})$ for $f \in S_g$ thus improving this trivial lower bound, $LB(S,g)$. LB_1 is based on the degree of the vertices in $\vec{G}$, LB_2 improves LB by considering the edges between the labeled and unlabeled vertices, LB_3 considers the best vertex to be labeled next in the partial solution, and LB_4 is based on the distribution of the edges in $\vec{G}$ minimizing the cutwidth.

3.1 Lower bound LB_1

Let $N(v)$ be the set of vertices adjacent to v and let $\mathcal{E}(v)$ be the edges with an endpoint in v. Consider a solution f and the vertex u in position $f(v) - 1$ (i.e., u precedes v in the ordering

f). If an edge in $\mathcal{E}(v)$ is incident on a vertex w with $f(w) < f(v)$, then it contributes to $CW_f(u)$; otherwise, it contributes to $CW_f(v)$ (the edge is computed in the cutwidth of the vertex). Then $CW_f(u) + CW_f(v) \geq |N(v)|$. Therefore,

$$\max\{CW_f(u), CW_f(v)\} \geq \left|\frac{N(v)}{2}\right| \quad (21)$$

Considering that the cutwidth of the graph $CW_f(\mathcal{G})$ is the maximum of the cutwidths of all its vertices, we conclude that $|N(v)|/2$ is a lower bound on $CW_f(\mathcal{G})$.

$$CW_f(\mathcal{G}) \geq LB_1 = \max_{v \in \mathcal{V}} \left\lceil \frac{N(v)}{2} \right\rceil \quad (22)$$

In the example in Figure 2, we obtain $LB_1 = 3$. Note that this bound is independent of the labeling, and it actually provides a lower bound on the optimum cutwidth of the graph $CW(\mathcal{G})$.

In order to compute this lower bound, we need to examine all the vertices in the graph and, for each of them, all its neighbors. In a direct implementation, this procedure would be $O(n\,m)$. However, for each vertex we store its number of neighbors, reducing the complexity to $O(n)$. It is important to remark that this lower bound does not need to be updated when (S, g) grows since it only depends on the vertex with maximum degree in the graph. Therefore, it is computed only once.

3.2 Lower bound LB_2

Given a partial solution (S, g) and a complete solution f in S_g, the cutwidth of $v \in S$ with respect to f, $CW_f(v)$, can be computed as:

$$CW_f(v) = CW_g(v) + \sum_{\substack{u \in S \\ 1 \le g(u) \le g(v)}} |N_U(u)| \qquad (23)$$

where $N_U(u)$ is the set of unlabeled vertices adjacent to u. The first term in this expression, $CW_g(v)$, corresponds to the cutwidth of v in $\mathcal{G}_S$ (S, E_S). The second term represents the number of edges with an endpoint in a vertex u labeled with $g(u) \le g(v)$ (i.e., previous to v in the ordering g), and the other endpoint in an unlabeled vertex w. Note that $f(w) > g(v)$ for all w in U and any labeling (solution) f in S_g. This is why we include all the edges with an endpoint in the unlabeled vertices w in the computation of $CW_f(v)$.

Given that (23) provides an expression of $CW_f(v)$ for all v in $S \subseteq \mathcal{V}$, and that $CW_f(\mathcal{G})$ is the maximum of $CW_f(v)$ for all v in $\mathcal{V}$, we can conclude that:

$$CW_f(\mathcal{G}) \ge LB_2 = \max_{v \in S} \left\{ CW_g(v) + \sum_{\substack{u \in S \\ 1 \le g(u) \le g(v)}} |N_U(u)| \right\} \qquad (24)$$

In the partial solution shown in Figure 2, the value of the cutwidth of any solution f in S_g, $CW_f(\mathcal{G})$, satisfies:

$$CW_f(\mathcal{G}) \ge \max\{CW_f(A), CW_f(D), CW_f(E)\} = \max\{4,7,6\} = 7,$$

where:

$$CW_f(A) = CW_g(A) + |N_U(A)| = 1 + 3 = 4$$

$$CW_f(D) = CW_g(D) + |N_U(A)| + |N_U(D)| = 2 + 3 + 2 = 7$$

$$CW_f(E) = CW_g(E) + |N_U(A)| + |N_U(D)| + |N_U(E)| = 0 + 3 + 2 + 1 = 6$$

The computation of this lower bound is performed in an incremental way. In particular, we only

need to check the neighbors of the last vertex included in the partial solution. Therefore the complexity of computing LB_2 is $O(n)$.

3.3 Lower bound LB_3

Consider a partial solution (S, g), an unlabeled vertex $u \in U$, the vertex v_k in S with the largest label, and a solution f in S_g. If the vertex u is labeled in f with $k+1$ (i.e., u follows v_k in the ordering f) its cutwidth can be computed as:

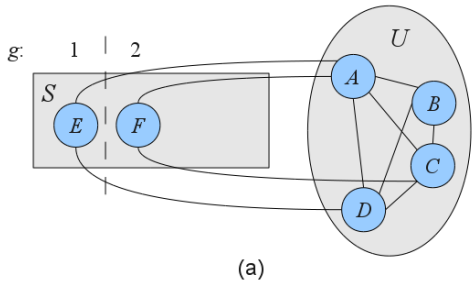
$$CW_f(u) \geq CW_f(v_k) - (|N_S(u)| - |N_U(u)|) \tag{25}$$

Note that $CW_f(v_k)$ is equivalent to the expressions $|E \cap (S \times U)|$, $\sum_{v \in S}|N_U(v)|$ and $\sum_{u \in U}|N_S(u)|$.

We can then compute a lower bound of the CW_f-value for the vertex in position $k+1$, by computing the maximum of $|N_S(u)| - |N_U(u)|$ for all $u \in U$. Thus we obtain:

$$CW_f(\mathcal{G}) \geq LB_3 = CW_f(v_k) - \max_{u \in U}(|N_S(u)| - |N_U(u)|) \tag{26}$$

A partial solution (S, g) of the example given in Figure 1 is shown in Figure 3.a, where $S = \{E, F\}$, $g(E) = 1$, $g(F) = 2$ and $U = \{A, B, C, D\}$ with $CW_f(F) = 4$. In Figure 3.b it is shown the value of $|N_S(u)| - |N_U(u)|$ for each vertex $u \in U$. According to the definition given above, we select the vertex A, giving a value of $LB_3 = 4 - (-1) = 5$. This means that, independently of the labeling of the vertices in U, the value of the final solution is greater than or equal to 5.



(a)

	$\lvert N_S(u)\rvert$ - $\lvert N_U(u)\rvert$
A	|2| - |3| = -1
B	|0| - |3| = -3
C	|1| - |3| = -2
D	|1| - |3| = -2

(b)

Figure 3: (a) Partial solution. (b) $|N_S(u)| - |N_U(u)|$ values for every $u \in U$.

Computing LB_3 requires traversing all the unlabeled vertices and then, checking whether its neighbors are not labeled. Therefore, the complexity of the computation of this lower bound is $O(n^2)$.

3.4 Lower bound LB_4

Given a graph $\vec{G}$ with n vertices and m edges we compute the lower bound LB_4 of its cutwidth $CW(\vec{G})$, by constructing an auxiliary graph $\vec{G}$' with n vertices and m edges distributed in such a way that it has minimum cutwidth. In other words, we "put" the edges in $\vec{G}$' between the appropriate vertices to obtain a minimum cutwidth. In this way, the cutwidth of $\vec{G}$' is a lower bound of the cutwidth of $\vec{G}$ for any labeling of its vertices (it is in fact a lower bound of the cutwidth of any graph with n vertices and m edges).

Consider the case in which $m < n$, we construct $\vec{G}$' as a path (Figure 4) in which some vertices may eventually be disconnected (when $m = n - 1$ it is a connected path). The cutwidth of $\vec{G}$' is equal to 1 and it is clear that regardless how the edges are distributed in $\vec{G}$, given that it has m edges, for any f, its cutwidth $CW_f(\vec{G})$ will be equal to or larger than $CW(\vec{G}') = 1$. Moreover, if we have $m = n$, we need to add an extra edge to the connected path $\vec{G}$' and it necessarily results in a vertex with cutwidth 2; therefore, in this case $CW(\vec{G}') = 2 \leq CW_f(\vec{G})$ for any labeling of the vertices in $\vec{G}$.

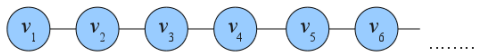


Figure 4: Graph 𝔾' with $m = n - 1$ edges (path).

Let us now consider the case in which $m > n$. The best way to distribute the m edges in a graph with n vertices in order to reduce its cutwidth is as follows: We place the first $n - 1$ edges joining "consecutive" vertices, in the graph (we call them edges of length 1) as shown in Figure 4 (between v_i and v_{i+1} for any i). Then, we can add some edges increasing the cutwidth by only one unit. Specifically, we can add $\lfloor (n-1)/2 \rfloor$ edges between "alternated" vertices (v_i and v_{i+2}) as shown in Figure 5, keeping the cutwidth of 𝔾' with value 2. We shall denote them edges of length 2. Therefore, the cutwidth of a graph 𝔾 with n vertices and m edges with $n \leq m \leq n - 1 + \lfloor (n-1)/2 \rfloor$ satisfies $CW(\mathbb{G}') = 2 \leq CW_f(\mathbb{G})$ for any labeling of the vertices in 𝔾. Any extra edge would result in a cutwidth of 3.

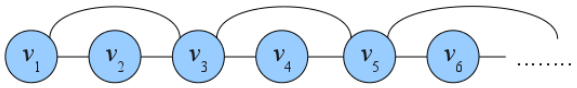


Figure 5: Graph 𝔾' with a length 1 and 2 edges.

In Figure 6 it is shown how can we add $\lfloor (n-2)/2 \rfloor$ edges to the graph in Figure 5 keeping the cutwidth of 𝔾' with value 3. Then, following the same argument described above, the cutwidth of a graph 𝔾 with n vertices and m edges with $n - 1 + \lfloor (n-1)/2 \rfloor < m \leq (n-1) + (n-2)$ satisfies $3 \leq CW_f(\mathbb{G})$ for any labeling of its vertices (it is easy to see that $\lfloor (n-1)/2 \rfloor + \lfloor (n-2)/2 \rfloor = n - 2$).

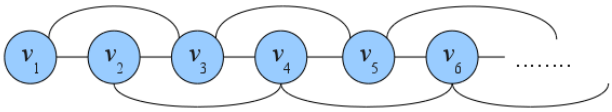


Figure 6: Graph 𝔾' with cutwidth 3.

Generalizing this incremental construction of $\mathcal{G}$', we observe that there is a maximum of $n-k$ edges of length k (between v_i and v_{i+k} for any i) that can be added to $\mathcal{G}$' (in which we have previously added all the edges with lengths t from $t = 1$ to $k-1$). The first $\lfloor (n-1)/k \rfloor$ edges increase the cutwidth of $\mathcal{G}$' by one unit; the second $\lfloor (n-2)/k \rfloor$ by another unit, the third $\lfloor (n-3)/k \rfloor$ in another unit and so on until the $n-k$ edges of length k have been added and the cutwidth of $\mathcal{G}$' increases by k units. The cutwidth of graph $\mathcal{G}$' provides a bound of the cutwidth of any graph with the same number of vertices and edges.

Note that it is possible to compute the cutwidth of such a graph $\mathcal{G}$' without explicitly constructing it, using the following recursive expression:

$$MinCW(M,l,i) = \begin{cases} 0 & if\ M \leq 0 \\ 1 + MinCW\left(M - \left\lfloor \frac{|U|-i}{l} \right\rfloor, l, i+1\right) & if\ (M > 0) \text{ and } (i < l) \\ 1 + MinCW\left(M - \left\lfloor \frac{|U|-i}{l} \right\rfloor, l+1, 1\right) & if\ (M > 0) \text{ and } (i = l) \end{cases} \quad (27)$$

where M is the number of remaining edges that have not been yet placed, $l = |f(v_i) - f(v_j)|$ is the length of the edges considered in the labeling, i is the label of the vertex in which we start to place edges and $\lfloor (|U|-i)/l \rfloor$ computes the maximum number of edges with length l we can place for each recursion level (i.e., placing consecutive edges of length l starting in vertex with label i). Note that each recursion increases the cutwidth value by one unit.

Given a partial solution S and the set of unlabeled vertices $U = \mathcal{V} \backslash S$, we define E_U as the set of edges (v_i, v_j) such that $v_i, v_j \in U$. We can compute LB_4 as follows:

$$LB_4 = MinCW(|E_U|,1,1) \quad (28)$$

For instance, the first call to $MinCW$ incorporate the edges depicted in Figure 4, the second call, the new edges depicted in Figure 5, the third call, the new edges depicted in Figure 6 and so on.

As a result, the value of the cutwidth of any solution f in S_g, $CW_f(\vec{G})$, satisfies:

$$CW_f(\vec{G}) \geq CW_f(\vec{G}') \geq LB_4 = MinCW(|E_U|,1,1) \qquad (29)$$

Since this lower bound only depends on the graph structure, its computation can be done in a simple lookup-table of the problem size (i.e., of size $n \times m$). This table, computed offline, stores for each pair (n, m) the corresponding value of LB_4; therefore, LB_4 is available in constant time, $O(1)$.

3.6 Relative dominance among the lower bounds

In this section we analyze the relative dominance among the lower bounds presented above and conclude that there are no dominance relationships among them. To this end, we present a graph in Figure 7.a and partial orderings in Figure 7.b, each one with a different best lower bound.

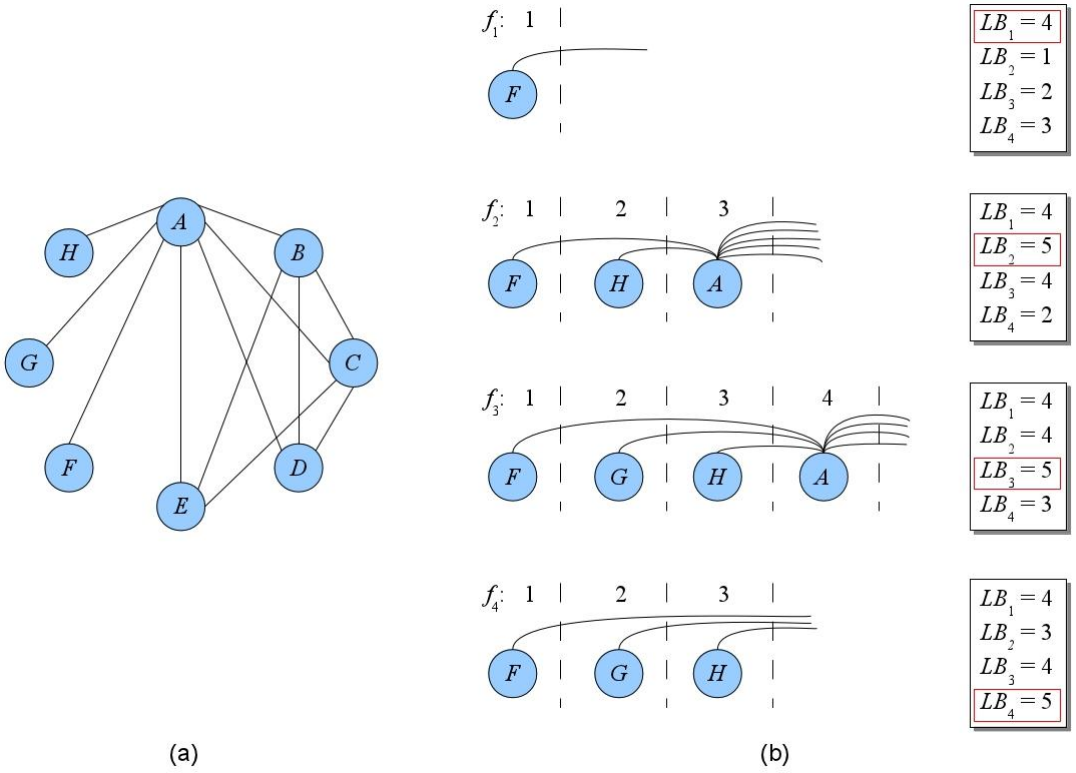


Figure 7. Examples of dominance among lower bounds.

It is easy to see in Figure 7 that in labeling f_1 the lower bound LB_1 dominates the others ($LB_1 = 4$, $LB_2 = 1, LB_3 = 2$ and $LB_4 = 3$). In the second ordering (f_2), LB_2 dominates the rest of bounds ($LB_2 = 5$, which is larger than $LB_1 = 4, LB_3 = 4$ and $LB_4 = 2$). In f_3 the lower bound LB_3 dominates the rest of lower bounds and, finally, in f_4, LB_4 is the best (largest) lower bound.

4. Dominance between partial solutions

In this section we propose a fathoming strategy based on the dominance among partial solutions (nodes in the search tree) to reduce the exploration and consequently the running time of our branch and bound procedure.

Given a partial solution (S, g) and a complete solution f in S_g, we introduced in Section 3.2 the expression (23) to compute the cutwidth of $v \in S$ with respect to f, $CW_f(v)$, in terms of the set of unlabeled vertices adjacent to u, $N_U(u)$. It is clear that the cutwidth of $\vec{G}$ with respect to f can be split into two parts:

$$CW_f(\vec{G}) = max_{v \in V}\, CW_f(v) = max\{max_{v \in S}\, CW_f(v)\,, max_{v \in V \setminus S}\, CW_f(v)\} \qquad (30)$$

Note that the value of each part in (30) is independent with the specific ordering of the vertices in the expression of the other part. For example, the cutwidth of $v \in V \setminus S$ is independent with the ordering of the vertices in S, because in any ordering of S all its vertices would receive a label lower than the label of v. Therefore, the value of the first part in (30) only depends on the ordering of the vertices in S (i.e., on g) and similarly, the value of the second part only depends on the ordering of the vertices in $V \setminus S$. As a matter of fact, the value of the first part is the LB_2 lower bound.

$$CW_f(\vec{G}) = max\{LB_2(g), max_{v \in V \setminus S}\, CW_f(v)\} \qquad (31)$$

Consider now a different partial solution over the same set of vertices (S, h) and a complete solution f' in S_h. From (31) it holds that:

$$CW_{f'}(\vec{G}) = max\{LB_2(h), max_{v \in V \setminus S}\, CW_{f'}(v)\} \quad (32)$$

If $LB_2(g) \leq LB_2(h)$ we can conclude that the value of the best solution in S_g is better (lower) or equal than the value of the best solution in S_h. Let f^{best} be the best solution in S_h, we can express its value, $CW_{f^{best}}(\vec{G})$, as in (32) in terms of $LB_2(h)$ and $CW_{f^{best}}(v)$ for all the $v \in V \setminus S$. If we reorder in f^{best} the elements of S according to g we obtain a solution in S_g in which the value, expressed as in (31), clearly is lower or equal to the value of f^{best} (note that the second part in both expressions have the same value). Therefore, we can skip the examination of the solutions in S_h and only examine S_g to determine the optimal solution.

This dominance rule can be exploited during the search process. If a partial solution (S, g) dominates another partial solution (S, h), (i.e., if $LB_2(g) \leq LB_2(h)$) we do not explore (S, h). To implement this rule in a direct way we would need to store the set S, to eventually fathom any other partial solution of these vertices. However, storing every possible set S would result in an extremely large number of sets in the order of $O(n!)$. An alternative strategy to limit the storage would be to generate any other partial solution with the same vertices assigned but in a different order (for example using a backtracking method). However, this would be highly time-consuming (in the order of $O(n!)$) when the number of vertices considered is relatively large. We therefore have considered a compromise between the time/storage needed to study the dominance rule and the performance obtained. Our strategy only checks the dominance by a reference partial solution in which the vertices are in a lexicographical order. In this way we ensure that at least one solution over this set of vertices is checked, which permits an efficient implementation. In the computational experience (Section 7), we evaluate the effectiveness of

this dominance rule, which complements the effect of the lower bounds. Both together significantly reduce the partial solutions explored and therefore the total running time of the method.

5. GRASP Upper Bound

In this section, we propose a heuristic approach based on the GRASP methodology (Feo and Resende 1994) to obtain an upper bound for the CMP. A Greedy Randomized Adaptive Search Procedure, GRASP, is a multi-start or iterative procedure where each iteration consists of two phases: construction and local search (Feo and Resende, 1989). At each iteration of the construction phase, GRASP maintains a set of candidate elements, *CL*, that can be feasibly added to the partial solution under construction. Every candidate element is evaluated according to a greedy function in order to select the next element to be added to the construction. A restricted candidate list, $RCL \subseteq CL$, is created with the best elements in *CL*. This is the greedy aspect of the method. The element to be added to the partial solution is randomly selected from those in the *RCL*. This is the probabilistic aspect of the heuristic. Once the selected element is added to the partial solution, the candidate list *CL* is updated and its elements evaluated. This is the adaptive aspect of the heuristic. Once a solution is constructed, a local search is applied to reach a local optimum. We refer the reader to Resende et al. (2003; 2010) for two recent reviews of GRASP. In Figure 8 there is a pseudo-code of our GRASP construction method for the cutwidth problem.

The constructive method starts by creating a list of unlabeled vertices U (initially $U = \mathbb{V}$). The first vertex is randomly selected from all those vertices in U and labeled with 1. In subsequent construction steps, a candidate list *CL* is formed with all the vertices in U that are adjacent to at least one labeled vertex. For each vertex u in *CL* we compute its evaluation $e(u)$ as:

$$e(u) = |N_s(u)| - |N_U(u)|. \quad (33)$$

Note that in this step a greedy selection would label the vertex u^* having the maximum *e*-value with the next available label, which would be the minimum $CW_f(u)$ value. However, by contrast, the GRASP methodology computes a restricted candidate list, RCL, with good candidates and selects one at random. Specifically, $RCL = \{v \in CL / e(v) \geq threshold\}$ where:

$$threshold = e_{\min} + \alpha(e_{\max} - e_{\min}) \quad (34)$$

$$e_{\min} = \min_{v \in CL}\{e(v)\} \quad (35)$$

$$e_{\max} = \max_{v \in CL}\{e(v)\}. \quad (36)$$

The search parameter α is computed as a percentage between the maximum, $e_{\max}$, and minimum, $e_{\min}$, values in *CL*. It is randomly selected, at each iteration, for diversification purposes.

PROCEDURE Constructive

1. Let S and U be the sets of labeled and unlabeled vertices of the graph respectively
2. Initially $S = \emptyset$ and $U = \mathcal{V}$
3. Select a vertex u from U randomly
4. Assign the label $k = 1$ to u. $S = \{u\}, U = U \backslash \{u\}$

WHILE ($U \neq \emptyset$)

5. $k = k + 1$
6. Construct $CL = \{v \in U / (w, v) \in \mathcal{C}, w \in S\}$
7. Let $N_s(v)$ and $N_U(v)$ be the set of adjacent labeled and unlabeled vertices to v respectively.
8. Compute $e(v) = |N_s(v)| - |N_U(v)| \forall v \in CL$
9. Construct $RCL = \{v \in CL / e(v) \geq threshold\}$
10. Select a vertex u randomly in RCL
11. Label u with the label k
12. $U = U \backslash \{u\}$, $S = S \cup \{u\}$

Figure 8. Pseudo-code of the constructive method.

Once a solution has been constructed we apply an improving phase based on a local search procedure. Our local search method for the cutwidth problem is based on insertion moves. Given *f*, we define the insertion move $MOVE(f, j, v)$ consisting of deleting 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. In mathematical terms, from $f = (\dots, v_{j-1}, v_j, v_{j+1}, \dots, v_{i-1}, v, v_{i+1}, \dots)$, we obtain the new ordering $f' = (\dots, v_{j-1}, v, v_j, v_{j+1}, \dots, v_{i-1}, v_{i+1}, \dots)$.

- If $f(v) = i < j$, v is inserted just after v_j in position j. Therefore, from the ordering $f = (\dots, v_{i-1}, v, v_{i+1}, \dots, v_{j-1}, v_j, v_{j+1}, \dots)$, we obtain the new ordering $f' = (\dots, v_{i-1}, v_{i+1}, \dots, v_{j-1}, v_j, v, v_{j+1}, \dots)$.

We define the set of critical vertices *CV* as those with a cutwidth value equal or close to the cutwidth of the graph. In mathematical terms, the set of critical vertices is defined as:

$$CV = \bigcup_{i \geq \lceil \beta CW_f(\mathcal{G}) \rceil} S_i \qquad (37)$$

where $S_i = \{v \in V | CW_f(v) = i\}$ is the set of vertices with a cutwidth value equal to i and the threshold value is computed as a percentage β ($0 \leq \beta \leq 1$) of the current objective function value $\lceil \beta CW_f(\mathcal{G}) \rceil$. The search parameter β has been experimentally set to 0.9. These vertices determine the value of the objective function or alternatively are considered likely to do so in subsequent iterations. In each iteration, our local search method selects a vertex v in *CV* and performs the first improving move $MOVE(f, pos, v)$, where the meaning of *improving* is not

limited to the objective function (which provides little information in this problem). The position pos in the move is computed as the median of the positions (according to f) of the vertices adjacent to v. The search procedure explores not only pos, but also positions close to pos, and performs the first improving move. The considered moves are defined as $MOVE(f, j, v)$ with $j \in [pos - w, pos + w]$, being w a search parameter which is calculated as $|\mathcal{V}| \cdot \gamma$, where γ is set to 0.1. An improving move is the one that either reduces $CW_f(G)$ or the number of vertices in *CV*. When a move is performed, the associated vertex is removed from *CV*. When the set becomes empty, we recalculate it and resort to the first element in it. The method cuts off when there is no improving move associated with the vertices in *CV* (i.e., when the solution cannot be further improved).

Note that the set *CV* implements a *candidate list strategy* to scan the neighborhood in an intelligent way. Moreover, note that *CV* is not re-computed after each move. The notion of not updating key values (e.g., move values) after every iteration is based on the *elite candidate list* suggested in Glover and Laguna (1997). The design considers that it is not absolutely necessary to update the value of the moves in a candidate list after an iteration is completed (i.e., the selected move is executed) because most of these move values either remain the same or their relative merit remains almost unchanged. This strategy has been successfully applied in different optimization problems.

6. The search tree

A branch-and-bound procedure generates and explores the entire set of solutions to the problem by means of a search tree. It first starts by running a heuristic algorithm (in our problem we chose the GRASP introduced in Section 5) to obtain an initial solution. The objective function value of this solution is an upper bound UB of the optimal value. Then, at each node of the search tree, we test if it is dominated by the reference solution (see Section 4). If so, the corresponding node is fathomed. Otherwise, a lower bound LB is computed and it is compared

with the UB. If $LB \geq UB$ then we fathom the node (because no better solution than the incumbent one can be found in the subtree rooted at this node); otherwise we branch the node and explore its first *child* node. When the exploration reaches a *leaf* node (which represents a complete solution to our problem), it computes the objective function value of this solution, and updates the upper bound UB if necessary. Then, it performs a backward step, checking its *parent* node again (backtracking) with the new upper bound and continues the exploration. The branch-and-bound algorithm stops when all the nodes have been examined (some of which have been branched and others fathomed), and returns the optimum solution as the output. An early termination, due to time limitations, provides us with a lower bound and an upper bound of the optimal value. The upper bound is obtained as the value of the best solution found, while the lower bound is computed as the minimum of the lower bounds in the active (unexplored) nodes. For a more detailed description on this methodology see for example Martí et al., (2010).

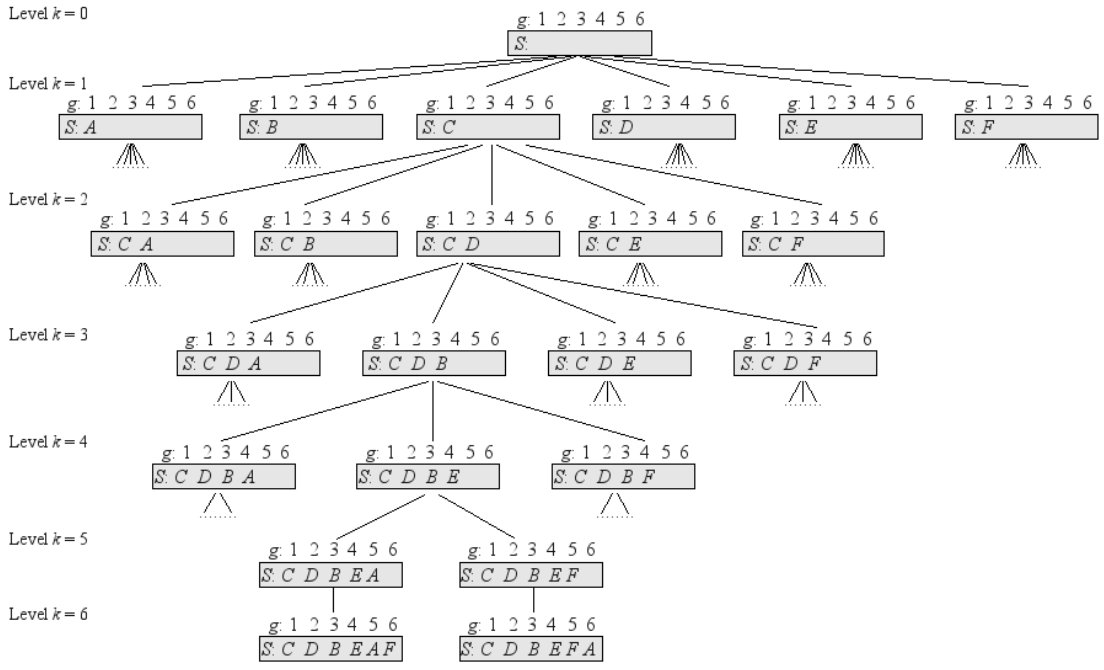


Figure 9: Search tree.

In our search tree, the initial node branches into n nodes labeling each vertex $A, B, C, \ldots$ with

label 1. Then, the node containing the vertex i represents the partial solution (S, g) where $S = \{i\}$ and $g(i) = 1$. Each of these n nodes at the first level branches into $n-1$ nodes (which will be referred to as nodes at level 2). Then, a node at level 2 contains two labeled vertices i and j and represents the partial solution $S = \{i, j\}$ with $g(i) = 1$ and $g(j) = 2$. Therefore, at each level in the search tree, the algorithm extends the current partial solution by labeling one more vertex. Figure 9 represents this search tree for the example given in Figure 1.a.

PROCEDURE $BB1(Node_k, UB)$

1. Let (S, g) be the partial solution associated with $Node_k$, being k the last assigned label

IF ($Node_k$ is a leaf node) /* Complete solution*/

2. Compute $CW_f(\vec{G})$ as the cutwidth of its associated solution

IF ($CW_f(\vec{G}) < UB$)

3. $UB = CW_f(\vec{G})$

ELSE

4. Compute LB

IF ($LB < UB$)

5. Let U be the set of unlabeled vertices

6. $k = k + 1$

WHILE ($U \neq \emptyset$)

7. Select u from U in lexicographical order

8. $U = U \backslash \{u\}$

9. Set $Node_k = \{S = S \cup \{u\}: g(u) = k\}$

10. $BB1(Node_k, UB)$

Figure 10: Pseudo-code of BB1

We propose three different ways to explore the search tree, called BB1, BB2 and BB3. In BB1, the search tree is first explored in depth. This strategy of exploration might benefit of reaching leaf nodes quickly. This could take advantage of the faster update of the upper bound when the

heuristic procedure provides an initial solution which is not close to the optimum value.

A pseudo-code of BB1 is shown in Figure 10 in which we initially call $BB1(Node_k, UB)$ with $Node_k = \{S = \emptyset: g(u) = 0, \forall u \in \mathbb{V}\}$, $k = 0$, and UB being an upper bound, initially computed with the GRASP procedure.

PROCEDURE BB3()

1. Compute UB with the GRASP algorithm
2. $k = 0$
3. Set $Node_k = \{S = \emptyset: g(u) = 0 \ \forall u \in \mathbb{V}\}$
4. $PQ = \emptyset$ /* empty priority queue */
5. Enqueue$(Node_k, PQ)$ /* add an element to the queue with an associated priority */

WHILE ($PQ \neq \emptyset$**)**

4. $Node_k =$ De-queue(PQ) /*return, removing from PQ, the highest priority element*/

IF ($Node_k$ is a leaf node) /* Complete solution*/

5. Compute $CW_f(\mathbb{G})$ as the cutwidth of its associated solution

IF ($CW_f(\mathbb{G}) < UB$)

6. $UB = CW_f(\mathbb{G})$

ELSE

IF ($LB < UB$)

7. Let U be the set of unlabeled vertices
8. Let k be the latest label assigned in the current node $Node_k$

WHILE ($U \neq \emptyset$)

9. Select u from U in lexicographical order
10. $U = U \backslash \{u\}$
11. Set $Node_{k+1} = \{S = S \cup \{u\}: g(u) = k + 1\}$
12. Compute LB

IF ($LB < UB$)

13. In-queue$(Node_{k+1}, PQ)$

Figure 11. Pseudo-code of BB3.

BB2 also performs a depth first search but, instead of exploring the first child node (in lexicographical order) of the latest explored node as BB1, it explores the most promising node at each level (i.e., the one with the lowest *LB* value). We have implemented effective data structures to store the non-branched nodes at each level for a fast back-tracking. Finally, BB3 is based on a breadth first search over the search tree. In order to enhance the performance of the algorithm, we use a priority queue to drive the search where the priority criterion is the same as the above mentioned. Figure 11 provides a pseudo-code of this procedure.

7. Computational Experiments

In this section we describe the computational experiments performed to test the efficiency of our branch-and-bound procedure, as well as to compare it with previous approaches. We have implemented the branch-and-bound algorithm in Java SE 6 and solved the linear integer formulation (shown in Section 2.2) with Cplex 11.1. We used available executable codes (also implemented in Java SE 6) to compute the CMP lower bounds related with maximal fundamental cuts and the second smallest eigenvalue, and we used the values previously reported for the MinLA problem for its associated lower bounds (described in Section 2.1). All the experiments were conducted on an Intel Core 2 Quad CPU and 6 GB RAM.

We have employed three sets of instances in our experimentation. The first one, *Small*, was reported in Martí et al. (2008), the second one, *Grids*, was described in Rolim et al. (1995) and the third one, *Harwell-Boeing*, is a subset of the public-domain Matrix Market library (available at http://math.nist.gov/MatrixMarket/data/Harwell-Boeing/). All these instances are available at http://www.optsicom.es/cutwidth. Each set of instances is described below:

Small: This data set consists of 42 graphs established in the context of the bandwidth reduction problem. We have selected 42 representative graphs (out of 84) from the original set. The number of vertices ranges from 16 to

24, and the number of edges ranges from 18 to 49.

Grids: This data set consists of 36 matrices constructed as the Cartesian product of two paths (Raspaud et al., 2009). They are also called two dimensional meshes and, as documented in Raspaud et al. (2009), the optimal solution of the cutwidth problem for these types of instances is known by construction. For this set of instances, the vertices are arranged on a grid with a dimension $width \times height$ where $width,\ height \in \{3, 4, \dots, 10\}$ and $width \geq height$.

HB: We derived 34 instances from the Harwell-Boeing Sparse Matrix Collection. This collection consists of a set of standard test matrices arising from problems in linear systems, least squares, and eigenvalue calculations from a wide variety of scientific and engineering disciplines. The problems range from small matrices, used as counter-examples to hypotheses in sparse matrix research, to large matrices arising in applications. Graphs are derived from these matrices as follows. Let M_{ij} denote the element of the *i*-th row and *j*-th column of the $n \times n$ sparse matrix M. The corresponding graph has n vertices. Edge (i, j) exists in the graph if and only if $M_{ij} \neq 0$. From the original set we have considered all the graphs with $n \leq 200$. Specifically the number of vertices ranges from 30 to 199 and the number of edges from 46 to 2145.

We have performed a preliminary experimentation over a set of 15 representative instances (five *small*, five *grids* and five *HB* instances referenced in Table 1) in order to test the main characteristics of our procedure. We shall call the set with 15 instances *Set*1. In all the experiments the CPU time is limited to 30 minutes. When the branch-and-bound algorithm is not able to explore the entire search tree within this time limit, we report the absolute gap (*gap*)

and relative gap (%*gap*) between the best lower and upper bounds obtained in the search, *LB* and *UB* respectively. Both gaps provide an evaluation of the branch-and-bound performance on an early termination.

$$gap = UB - LB, \quad \%gap = \frac{UB-LB}{LB} \times 100 \qquad (38)$$

The first experiment compares the performance of the three proposed search algorithms BB1, BB2 and BB3. For each of them we report the number of explored nodes, *Expl*, and the number of fathomed nodes (not explored because of their bound), *Fath*, in the search tree. To complement this information, we also report the number of unexplored and unfathomed nodes (*UnExpl*). Note that if the whole search tree is explored, *UnExpl* equals zero (and *Expl*+*Fath* equals the total number of nodes in the search tree). These values are shown in Table 1 over the 15 instances in *Set*1.

		BB1			BB2			BB3		
		Expl	*Fath*	*UnExpl*	*Expl*	*Fath*	*UnExpl*	*Expl*	*Fath*	*UnExpl*
Small (5)	p51_20_28	1.4E04	6.6E18	0.0E0	1.4E04	6.6E18	0.0E0	1.4E04	6.6E18	0.0E0
	p63_21_42	1.2E06	1.4E20	0.0E0	1.2E06	1.4E20	0.0E0	1.2E06	1.4E20	0.0E0
	p72_22_49	1.3E06	3.1E21	0.0E0	1.3E06	3.1E21	0.0E0	1.3E06	3.1E21	0.0E0
	p81_23_46	7.7E06	7.0E22	0.0E0	7.7E06	7.0E22	0.0E0	7.7E06	7.0E22	0.0E0
	p100_24_34	1.5E06	1.7E24	0.0E0	1.5E06	1.7E24	0.0E0	1.5E06	1.7E24	0.0E0
Grids (5)	Grid5x5	6.5E02	4.2E25	0.0E0	6.5E02	4.2E25	0.0E0	6.5E02	4.2E25	0.0E0
	Grid6x8	1.3E04	3.4E61	0.0E0	1.3E04	3.4E61	0.0E0	1.3E04	3.4E61	0.0E0
	Grid7x9	5.9E05	54E.87	0.0E0	5.9E05	5.4E87	0.0E0	5.9E05	5.4E87	0.0E0
	Grid8x9	3.6E07	3.7E103	1.3E104	3.5E07	2.1E103	1.4E104	3.2E07	1.4E104	3.1E103
	Grid10x10	2.0E07	1.8E148	2.5E158	2.0E07	5.4E142	2.5E158	8.0E05	3.7E157	2.2E158
HB (5)	ibm32	1.6E08	2.9E34	6.9E35	1.5E08	7.1E34	6.4E35	2.5E06	1.3E35	5.9E35
	ash85	4.4E07	4.8E125	7.7E128	4.1E07	1.3E123	7.7E128	4.2E05	2.5E127	7.4E128
	arc130	1.1E07	3.1E197	1.8E220	1.2E07	5.7E148	1.8E220	1.8E05	0.0E0	1.8E220
	west0167	6.0E06	2.6E285	4.1E300	6.6E06	4.7E256	4.1E300	1.1E05	0.0E0	4.1E300
	will199	4.6E06	3.2E318	1.1E373	5.0E06	4.7E256	1.1E373	8.0E04	0.0E0	1.0E373

Table 1: Explored, fathomed and unexplored nodes in the search tree.

Results in Table 1 indicate that BB1, BB2 and BB3 are able to solve the 5 small instances and the first 3 grids optimally. However, in the last two grids, Grid8x9 and Grid10x10, none of the variants is able to finish but BB3 is able to fathom a larger number of nodes than BB1 and BB2. On the other hand, instances in the *HB* set exhibit a different pattern since BB3 is unable to fathom any nodes (while BB1 and BB2 fathom a relatively large number of nodes). This can be partially explained considering the way in which BB3 explores the search tree (i.e., branching

the most promising node). In large instances, there are a lot of promising nodes in the priority queue that are waiting for being branched. This could lead to a low value of the total number of fathomed nodes in an early termination of the method. However, although these nodes are not fathomed, they have been explored and contribute to improve the final lower bound, thus providing the best overall strategy. Table 2 includes the lower bound, *LB*, the absolute gap, *gap*, and the relative gap, *%gap*, obtained with the three methods on the 15 instances of *Set*1. Results in Table 2 seem to confirm that the best strategy to explore the search tree is BB3, in which nodes are ordered according to their bound.

With the goal of supporting our conclusions about the performance of the proposed procedures, we performed a statistical test. Specifically, we applied the non-parametric Friedman test for multiple correlated samples to the values obtained by each of the 3 methods. This test computes, for each instance, the rank value of each method according to solution quality (where rank 3 is assigned to the best method and rank 1 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 of 0.002 obtained in this experiment clearly indicates that there are statistically significant differences among the 3 methods. The rank values produced by this test are 2.40 (BB3), 1.80 (BB1), and 1.80 (BB2). We will therefore consider BB3 in the following experiments.

		BB1			BB2			BB3		
		LB	*gap*	*%gap*	*LB*	*gap*	*%gap*	*LB*	*gap*	*%gap*
Small (5)	p51_20_28	6	0	0.0	6	0	0.0	6	0	0.0
	p63_21_42	12	0	0.0	12	0	0.0	12	0	0.0
	p72_22_49	14	0	0.0	14	0	0.0	14	0	0.0
	p81_23_46	13	0	0.0	13	0	0.0	13	0	0.0
	p100_24_34	7	0	0.0	7	0	0.0	7	0	0.0
Grids (5)	Grid5x5	6	0	0.0	6	0	0.0	6	0	0.0
	Grid6x8	7	0	0.0	7	0	0.0	7	0	0.0
	Grid7x9	8	0	0.0	8	0	0.0	8	0	0.0
	Grid8x9	3	6	200.0	3	6	200.0	8	1	12.5
	Grid10x10	3	8	266.7	3	8	266.7	8	3	37.5
HB (5)	ibm32	6	17	283.3	6	17	283.3	18	6	33.3
	ash85	5	11	220.0	5	11	220.0	9	7	77.8
	arc130	62	140	225.8	62	140	225.8	62	140	225.8
	west0167	10	47	470.0	10	47	470.0	12	45	375.0
	will199	8	134	1675.0	8	134	1675.0	15	123	820.0
Average		11.3	24.2	222.7	11.3	24.2	222.7	13.7	21.7	105.5

Table 2: Lower bound, absolute and relative gaps.

In the second experiment we test the efficiency of each lower bound separately. Specifically, we compute the percentage of nodes that each lower bound is able to fathom. This measure can be interpreted as a success rate for each lower bound. Note that, in some cases, a search tree node can be fathomed by two (or more) different lower bounds; then we compute "this success" in the rates of all the corresponding lower bounds (therefore this measure is independent on the order in which the fathoming tests are applied). Table 3 reports the percentage of fathomed nodes for each lower bound (LB_1 to LB_4) and, in the last row, the CPU times to compute them (as a percentage over the total running time) in the 15 instances of *Set*1 (reporting the average on *Small*, *Grids*, and *HB* instances).

	$\boldsymbol{LB_1}$	$\boldsymbol{LB_2}$	$\boldsymbol{LB_3}$	$\boldsymbol{LB_4}$
Small (5)	0	93.85	96.30	0
Grids (5)	0	99.49	98.82	0
HB (5)	0	92.41	98.42	0
Total	0	95.12	98.53	0
%CPU Time	0.64	1.56	4.78	0.74

Table 3. Average fathoms of each lower bound.

Results presented in Table 3 clearly show that LB_1 and LB_4 are not fathoming a significant number of nodes (the associated percentages are very close to 0, and are represented by 0 in the table for the sake of simplicity). On the other hand, the behavior of LB_2 and LB_3 is very similar, fathoming on average about 95% and 98% respectively of the total fathomed nodes. However, the time required to compute LB_1 and LB_4 is relatively short since LB_1 is computed offline (only once, in the first node of the search tree) and LB_4 is calculated in constant time (see Section 3.4). The time needed to compute LB_1 and LB_4 (see last row of Table 3) represents less than 1% of the total running time. Although LB_2 and LB_3 are more time consuming (1.56 and 4.78 percent of the total running time respectively), they fathom most of the nodes in the search tree.

It is also important to remark that LB_1 and LB_4 are especially relevant when a pre-established time limit is reached and the method does not explore the complete search tree. The

computation of these two bounds contributes to reduce the final gap. To test this point we perform a new experiment reporting the gap values when only LB_2 and LB_3 are computed in the search tree. We shall call this method *BB*. Then, we incorporate the computation of LB_1 and LB_4 at the end of the process, which results in the entire method tested above. We shall call this method *BB+LB*. Table 4 reports the average gaps, absolute and relative, obtained with each of these two methods on the instances in *Set*1.

	BB		***BB+LB***	
	gap	***%gap***	***gap***	***%gap***
Small (5)	0.0	0.0	0.0	0.0
Grids (5)	0.8	10.0	0.8	10.0
HB (5)	73.2	447.4	64.8	310.6
Total	24.7	152.5	21.9	106.9

Table 4. Average gaps of two branch-and-bound variants.

The results in Table 4 show that the addition of LB_1 and LB_4 helps to reduce the final gap of the method on the *HB* instances. Examining Tables 3 and 4 together, we can conclude that the lower bounds complement each other. On one hand, LB_2 and LB_3 fathom a large number of nodes in the search tree. On the other hand, LB_1 and LB_4 reduce the final gap. We shall therefore include the four lower bounds in our final branch-and-bound algorithm.

In our fifth experiment we compare the previous lower bounds described in Section 2.1 with the proposed lower bounds for complete solutions (LB_1 and LB_4). We limit this experiment to the 11 instances reported in Caprara et al. (2011) to obtain the LB_{MinLA} value directly from their experiments (see their Table 2). We compute, for these instances the LB_{FF} and LB_{LM} values and our LB_1 and LB_4. Table 5 reports the values of these bounds and the associated CPU times (in the LB_{MinLA} they correspond to an Intel Core Duo 3.33 GHz and 2GB RAM).

Table 5 shows that the combination between LB_1 and LB_4 (which is applied on the first node of our branch and bound algorithm) obtains, on average, better lower bounds than the three previous bounds considered. Specifically they obtain an average value of 83.8 on 0.02 seconds while LB_{MinLA}, LB_{FF} and LB_{LM} obtain 45.9, 13.09 and 14.0 on 55202.1, 19.4 and 2446.8

seconds respectively.

The sixth experiment focuses on the combination of the GRASP heuristic with the branch-and-bound procedure. We compare the performance of the branch-and-bound procedure with the initial upper bound computed with GRASP, *BB from GRASP*, with the branch-and-bound procedure with an initial upper bound set as the value of a random solution, *BB from Random*. In Table 6 we include the average, absolute and relative gaps of both variants.

	LB_{MinLA}		LB_{LM}		LB_{FF}		$Max(LB_1, LB_4)$	
	Value	*CPU Time*	*Value*	*CPU Time*	*Value*	*CPU Time*	*Value*	*CPU Time*
gd95c	8	68.3	1	0.1	9	0.4	8	0.01
gd96a	72	86400	8	33.8	42	2522.9	56	0.02
gd96b	12	493.5	1	0.1	34	1.8	24	0.01
gd96c	7	218.1	1	0.1	5	0.7	3	0.01
gd96d	12	1642.2	4	0.1	14	6.7	14	0.01
c1y	73	86400	25	11.4	8	1363.6	152	0.01
c2y	78	86400	28	20.3	9	2454.3	164	0.02
c3y	86	86400	27	55.0	10	6.882	182	0.05
c4y	79	86400	24	60.4	10	7603.8	155	0.03
c5y	74	86400	24	16.6	12	4777.9	162	0.03
bintree10	4	86400	1	15.5	1	1300.8	2	0.02
Avg.	45.9	55202.1	13.09	19.4	14.0	2446.8	83.8	0.02

Table 5. Comparison with previous bounds.

As shown in Table 6, the results obtained with the branch-and-bound algorithm coupled with the heuristic initial upper bound are better, as expected, than those obtained with the random variant. We have also computed the number of instances in which the solution obtained with the GRASP algorithm matches the optimum value. This is difficult to compute since we do not know the optimum in all the cases (with the exception of the Grid instances in which, by design, the optimum is known, as documented in Rolim et al., 1995). In this experiment we observed that GRASP is able to obtain the optimum in the 5 *Small* and the 5 *Grid* instances tested in *Set*1. On the other hand, we cannot assess how far the GRASP solutions are from the optimum in the *HB* instances.

	BB from GRASP		***BB from Random***	
	gap	***%gap***	***gap***	***%gap***
Small (5)	0.0	0.0	1.4	11.7
Grids (5)	0.8	10.0	33.0	471.4
HB (5)	64.8	310.6	179.4	1055.2
Total	21.9	106.9	71.3	512.8

Table 6. Comparison of heuristic with random initial solution.

The next experiment focuses on the effectiveness of the dominance rule (proposed in Section 4) in the performance of BB3. To this aim we run BB3 (with the lower bounds LB_1, LB_2, LB_3 and LB_4) and compare it with a new version of BB3 which also includes the dominance test among partial solutions. Both executions start from the upper bound constructed with the GRASP heuristic. The results of this experiment are presented in Table 7.

	BB3			***BB3+ Dominance***		
	#Opt.	***%gap***	***CPU Time***	***#Opt.***	***%gap***	***CPU Time***
Small (5)	5	0.0	28.8	5	0.0	0.23
Grids (5)	3	10.0	727.7	4	7.5	376
HB (5)	0	310.6	1807.5	1	250.9	1591.4
Total	8	106.9	854.7	10	86.2	655.9

Table 7. Performance of the dominance properties.

As it can be seen in Table 7, dominance properties among partial solutions considerably improve the results of BB3. Specifically, *BB3+Dominance* is able to find two new optima (i.e., 10 out of 15 instances), the %*gap* decreases on average about 20% and the CPU Time also decreases about 200s, which represents a saving of 25% of running time.

In our final experiment, we compare our branch-and-bound algorithm with the linear integer formulation (Luttamaguzi et al. 2005) solved with Cplex 11.1. Specifically, we consider our three variants to explore the search tree, BB1, BB2 and BB3. In the three variants we compute LB_1, LB_2, LB_3 and LB_4; the dominance rule among partial solutions; and the initial GRASP upper bound. In Table 8 it is reported the number of optimal solutions found, #opt, the average absolute gap between the final lower and upper bounds, *gap*, the average relative gap (in percentage), %*gap*, and the CPU time in seconds for each method on our entire benchmark set

of 115 instances (42 *Small*, 36 *Grids* and 34 *HB*).

Results in Table 8 show that the Cplex solver with the linear integer formulation is only able to solve 9 small instances ($n \leq 20$) within 30 minutes of CPU time. Alternatively, the three variants tested of our branch-and-bound algorithms clearly outperform Cplex with this formulation since they are able to optimally solve all the small and medium-sized instances, and the average relative gap values in the *HB* instances are below 300% (while the average relative gap value of Cplex is 634.8% in these instances). The three branch-and-bound variants present a similar performance with a marginal improvement of BB3 over BB1 and BB2. Specifically, on the 34 *HB* instances BB3 presents an average relative gap of 173.53% while BB1 and BB2 present a value of 296.45% and 297.04% respectively. However, BB1 and BB2 are able to reach one more optima than BB3 in the hardest set of instances (*HB*). This behavior suggests that when the size of the instance is quite large, BB3 is not able to reach leaf nodes as fast as BB1 and BB2 do (in the time horizon considered).

		BB1	**BB2**	**BB3**	**Cplex**
Small (42)	# opt	42	42	42	9
	gap	0.00	0.00	0.00	1.90
	%*gap*	0.00	0.00	0.00	54.70
	CPU Time	0.04	0.04	0.06	1573.90
Grids (36)	# opt	33	33	33	2
	gap	0.61	0.61	0.14	4.20
	%*gap*	20.37	20.37	1.66	211.10
	CPU Time	156.16	156.37	156.40	1707.90
HB (34)	# opt	9	9	8	0
	gap	50.94	50.97	48.00	97.00
	%*gap*	296.45	297.04	173.53	634.80
	CPU Time	1373.59	1375.66	1431.20	1800.00

Table 8. Branch-and-bound algorithms versus Cplex.

We finally represent the search profile of the three branch-and-bound variants, BB1, BB2 and BB3 when running for 3 hours. Specifically, Figure 12 depicts the progression of the average relative gap of the three methods over the 15 instances in *Set*1. We report the average relative gap values of BB1, BB2 and BB3 every 10 minutes in each execution (and join the points with a

line to observe the trend).

The progression of the average gap represented in Figure 12 confirms that BB3 performs slightly better than BB1 and BB2. Additionally, it also shows that the most significant reduction in the gap value is obtained in the first 30 minutes; then, only a marginal extra improvement can be obtained if we run the method longer.

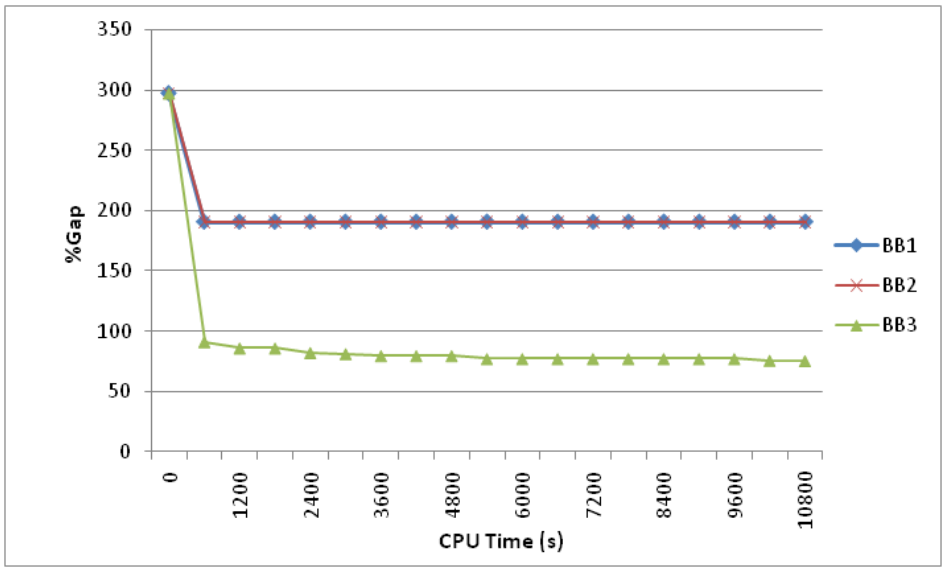


Figure 12. Relative gap profile.

Table 9 in the Appendix contains the best upper and lower bounds obtained for the set of 34 *HB* instances (identified as the hardest to solve in our study). We ran the GRASP for 10 minutes to obtain the initial upper bound and BB3 for 4 hours to obtain the lower bound on each instance (thereby setting a benchmark for future comparisons).

8. Conclusions

We have developed an exact procedure based on the branch-and-bound methodology coupled with a GRASP heuristic to provide solutions for the Cutwidth Minimization Problem. We have introduced the partial solution as the set of solutions that share some vertices, and we have

proposed several approaches for computing lower bounds on partial solutions. These bounds allow us to explore a relatively small portion of the nodes in the search tree when implementing our branch-and-bound procedure. Additionally, we have presented three different strategies to explore the search tree, which we have called BB1, BB2 and BB3.

We have conducted extensive preliminary experimentation to analyze the performance of the proposed lower and upper bounds, as well as the search strategies. The final experiment shows that our branch-and-bound procedures clearly outperform the previous linear integer formulation solved with the well-known Cplex (version 11.1), and that they are able to optimally solve all the small-sized problems as well as some of the larger ones. Finally, we provide detailed results for the hardest instances for future comparisons.

Acknowledgments

This research has been partially supported by the Government of Spain (Grant Refs. TIN2008-06890-C02-02, TIN2009-07516 and TIN2011-28151).

References

Adolphson D. and T. C. Hu. Optimal linear ordering. *SIAM Journal on Applied Mathematics*, 25(3):403-423, 1973.

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

Caprara A., A. N. Letchford and J. J. Salazar-González. Decorous Lower Bounds for Minimum Linear Arrangement. *INFORMS Journal on Computing*, 23(1):26-40, 2011.

Chung M. J., F. Makedon, I.H. Sudborough and J. Turner. Polynomial time algorithms for the min cut problem on degree restricted trees. *In Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, pages 262-271, 1982.

Cohoon, J., S. Sahni. Heuristics for the board permutation problem. *Journal of VLSI and Computer*

Systems, 2:37–61, 1987.

Díaz J., J. Petit and M. Serna. A survey of graph layout problems. *ACM Computing Surveys*, 34(3):313–356, 2002.

Feo T. A., M. G. C. Resende and S. H. Smith. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

Feo T. A., M. G. C. Resende and S. H. Smith. A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, 42:860–878, 1994.

Festa P. and M. G. C. Resende. An annotated bibliography of GRASP – Part I: Algorithms. International Transactions in Operational Research. Volume 16, Issue 1, pages 1-24, 2009a.

Festa P. and M. G. C. Resende. An annotated bibliography of GRASP – Part II: Applications. International Transactions in Operational Research. Volume 16, Issue 2, pages 131–172, 2009b.

Ford, L.R. and D.R. Fulkerson. *Flows in networks*. Princeton University Press, 1962.

Garey M. R., F. Some NP-complete graph problems. *Theoretical Computer Science*, 1, 237-267, 1976.

Gavril F. Some NP-complete problems on graphs. *In Proceedings of the 11th conference on information Sciences and Systems*, 91-95, 1977.

Glover, F. and M. Laguna (1997) Tabu Search, Kluwer Academic Publishers, Boston.

Harper L. H. Optimal numberings and isoperimetric problems on graph. *Journal of Combinatorial Theory*, 1:385-393, 1966.

Juvan M. and B. Mohar. Optimal linear labeling and eigenvalues of graphs. *Discrete Applied Mathematics*, 36:153-168, 1992.

Karger D. R. A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem. *SIAM Journal on Computing*, 29(2):492-514, 1999.

Luttamaguzi J., M. Pelsmajer, Z. Shen and B. Yang. Integer programming solutions for several optimization problems in graph theory. Technical report, DIMACS, 2005.

Makedon F., C. Papadimitriou, I. H. Sudbourough. Topological bandwidth. *SIAM Journal on Algebraic and Discrete Methods*, 6(3):418-444, 1985.

Makedon F. and I.H. Sudborough. On minimizing width in linear layouts. *Discrete Applied Mathematics*,

23(3):243-265, 1989.

Martí, R., V. Campos and E. Piñana. Branch and bound for the matrix bandwidth minimization, *European Journal of Operational Research* 186:513-528, 2008.

Martí R., M. Gallego, A. Duarte. An Exact Method for the Maximum Diversity Problem. *European Journal of Operational Research*. 200: 36-44, 2010

Mutzel, P. A polyhedral approach to planar augmentation and related problems. *In Proceedings of the 3rd Annual European Symposium on Algorithms*, Lecture Notes in Computer Science, 979:497-507, 1995.

Pantrigo, J.J., A. Duarte, R. Martí, E.G. Pardo. Scatter Search for the cutwidth problem. *Annals of Operations Research*. Accepted. DOI: DOI: 10.1007/s10479-011-0907-2, 2010

Petit J. Experiments on the minimum linear arrangement problem. *ACM Journal of Experimental Algorithmics*, 8, 2003.

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

Resende, M. G. C. and C. C. Ribeiro. Greedy randomized adaptive search procedures, In *Handbook of Metaheuristic.* 219–250. Kluwer Academic Publishers, 2003.

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

Resende, M. G. C. and C. C. Ribeiro. Greedy randomized adaptive search procedures: advances and applications, In *Handbook of Metaheuristic.* 2nd edition. 281-317. Springer 2010.

Rolim J., O. Sýkora and I. Vrt'o. Cutwidth of the de Bruijn graph. *RAIRO Informatique Théorique et Applications*, 29(6):509-514, 1995.

Thilikos D. M, M.J. Serna, H. Bodlaender. Cutwidth I: A constructive linear time algorithm for small cutwidth. Journal of Algorithms, 56:1-24, 2005.

Yannakakis M. A polynomial algorithm for the min-cut linear arrangement of trees. *Journal of the ACM*, 32(4):950-988, 1985.

Appendix

	n	***m***	***LB***	***UB***
pores_1	30	103	**17**	**17**
ibm32	32	90	**23**	**23**
bcspwr01	39	46	**5**	**5**
bcsstk01	48	176	27	32
bcspwr02	49	59	**5**	**5**
curtis54	54	124	10	13
will57	57	127	7	11
impcol_b	59	281	24	55
bcsstk02	66	2145	**1089**	**1089**
steam3	80	424	**20**	**20**
ash85	85	219	11	16
nos4	100	247	**12**	**12**
gent113	104	549	27	87
bcsstk22	110	254	6	13
gre__115	115	267	12	36
dwt__234	117	162	6	12
bcspwr03	118	179	6	10
lns__131	123	275	6	30
arc130	130	715	62	202
bcsstk04	132	1758	107	310
west0132	132	404	18	71
impcol_c	137	352	14	46
can__144	144	576	**25**	**25**
lund_a	147	1151	43	113
lund_b	147	1147	42	111
bcsstk05	153	1135	42	115
west0156	156	371	14	56
nos1	158	312	**4**	**4**
can__161	161	608	23	52
west0167	167	489	17	55
mcca	168	1662	58	390
fs_183_1	183	701	52	190
gre__185	185	650	22	48
will199	199	660	21	132

Table 9. Lower and upper bounds for the *HB* instances.