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

Original version: January 6, 2010

Abstract — The cutwidth minimization problem consists of finding a linear layout of a graph so that the maximum linear cut of edges (i.e., the number of edges that cut a line between consecutive vertices) is minimized. This paper starts by reviewing previous exact approaches for special classes of graphs as well as a linear integer formulation for the general problem. We propose a branch and bound algorithm based on different lower bounds on the cutwidth of partial solutions. Empirical results with a collection of previously reported instances indicate that the proposed algorithm is able to solve all the small-sized instances (up to 32 vertices) as well as some of the large-sized instances tested (up to 158 vertices) in less than 30 minutes of CPU time. We compare our method with the best previous linear integer formulation solved with the well-known software Cplex. The comparison favors the proposed procedure.

KeyWords: Cutwidth, Branch and Bound, 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 of G 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 a vertex v, with respect to a labeling f, $CW_f(v)$, is given by the number of edges $(u,w) \in E$ in the graph satisfying $f(u) \leq f(v) < f(w)$.

In mathematical terms:

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

Given a labeling f, the cutwidth of G is defined as:

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

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

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

This problem is NP-hard as stated in Gavril (1977) even for graphs with a maximum degree of three (Makedon et al., 1985). Some special cases have been solved optimally; for example, Harper (1966) solved the cutwidth for hypercubes, Chung et al. (1982) presented a $O(\log^{d-2} n)$ time algorithm for the cutwidth of trees with n vertices and with maximum degree d. Yannakakis (1985) improved the aforesaid results by giving a $O(n \log n)$ time algorithm to determine the cutwidth of trees with n vertices. In particular, for k-level, t-ary trees $T_{t,k}$, it holds that:

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

Exact methods to obtain the optimal cutwidth of grids have been proposed in Rolim et al. (1995). Specifically, for *width*, *height* ≥ 2 the authors proved that

$$CW(L_{width,height}) = \begin{cases} 2, & \text{if } width = height = 2 \\ \min\{width+1, height+1\}, & \text{otherwise} \end{cases}$$

Finally, Thilikos et al. (2001) presented an algorithm to compute the cutwidth of bounded degree graphs with small tree-width in polynomial time.

Practical applications of the cutwidth problem can be traced back 35 years. 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).

Figure 1.a shows an example of an undirected graph with 6 vertices and 10 edges. Figure 1.b shows a labeling, f, of the graph in Figure 1.a, setting the vertices in a line in the order of the labeling, 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 graph *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.

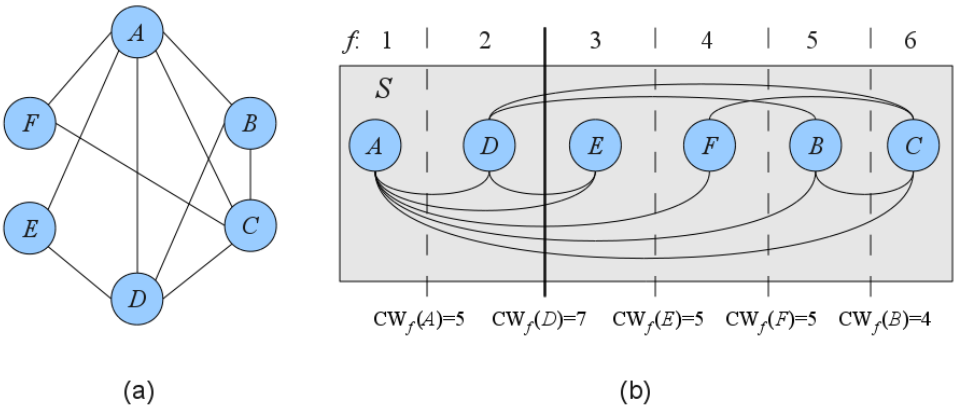


Figure 1: (a) Graph example, (b) Cutwidth of $\vec{G}$ for a labeling *f*.

Luttamaguzi et al. (2005) proposed the following linear integer formulation to solve the cutwidth minimization problem:

Min *b*

s.t.

$$x_i^k \in \{0,1\} \quad (1)$$

$$i,k \in \{1,\ldots,n\} \quad (2)$$

$$\sum_{k\in\{1,\ldots,n\}} x_i^k = 1, \quad \forall i \in \{1,\ldots,n\} \quad (3)$$

$$\sum_{i\in\{1,\ldots,n\}} x_i^k = 1, \quad \forall k \in \{1,\ldots,n\} \quad (4)$$

$$y_{i,j}^{k,l} \le x_i^k \quad (5)$$

$$y_{i,j}^{k,l} \le x_j^l \quad (6)$$

$$x_i^k + x_j^l \le y_{i,j}^{k,l} + 1 \quad (7)$$

$$\sum_{\substack{(k\le c<l)\vee \\ (l\le c<k)}} y_{i,j}^{k,l} \le b, \quad \forall c \in \{1,\ldots,n-1\} \quad (8)$$

where x_i^k is a decision binary variable whose indices are $i, k \in \{1,2,\ldots,n\}$. This variable specifies whether *i* is placed in position *k* in the ordering. In other words, for all x_i^k $(i, k \in \{1, 2,\ldots,n\})$they take on value 1 if and only if *i* occupies the position *k* in the ordering; otherwise x_i^k takes on value 0. Constraints (3) and (4) ensure that each vertex is only assigned to one position and one position is only assigned to one vertex respectively. Consequently, constraints (1), (2), (3) and (4) together imply that a solution of the problem is an ordering.

The decision binary variable $y_{i,j}^{k,l} \in \{0,1\}$ is defined in terms of x_i^k and x_j^l as follows:

$$y_{i,j}^{k,l} = x_i^k \wedge x_j^l$$

where $i, j, \in \{1,2,...,n\}$, $(v_i, v_j) \in \mathcal{E}$ and $k, l, \in \{1,2,...,n\}$ the labels associated to vertex v_i and v_j respectively. In the linear formulation above this conjunction is computed with constraints (5), (6) and (7).

Constraint (8) computes for each position c in the ordering, the number of edges whose origin is placed in any position k ($1 \leq k < c$) 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, $c \in \{1,...,n - 1\}$ of the labeling. Therefore, the objective function b must be larger than or equal to this quantity.

In this paper we propose a branch and bound algorithm for the cutwidth minimization problem. It consists of a systematic enumeration of all its solutions (labeling) based on the definition of *partial solutions*, set out in Section 2. In this section we also propose lower bounds that will enable us to discard a large number of solutions in the enumeration process. In Section 3 we introduce a simple heuristic to obtain an upper bound for the cutwidth problem. Section 4 describes the search tree for an efficient enumeration of the problem solutions and its associated strategies. Finally, the paper concludes with the computational experiments, in which we include our experience with the linear integer formulation above, and the associated conclusions.

2. Lower bounds for partial solutions

Given a subset S of $\mathcal{V}$ with $k < n$ vertices and an ordering $g \in \Pi_k$ assigning the integers {1, 2, ..., k} to the vertices in S, we define a partial solution as the pair (S, g). A complete solution of the cutwidth problem in the graph $\mathcal{G}=(\mathcal{V}, \mathcal{E})$ can be obtained by adding n-k elements from $\mathcal{V} \setminus S$ to S, assigning them the integers {k+1, k+2, ..., 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. Figure 2 shows the partial solution (S, g) of the example introduced in Figure 1.a (see Section 1) where the vertices in S={A, D, E} have been 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.

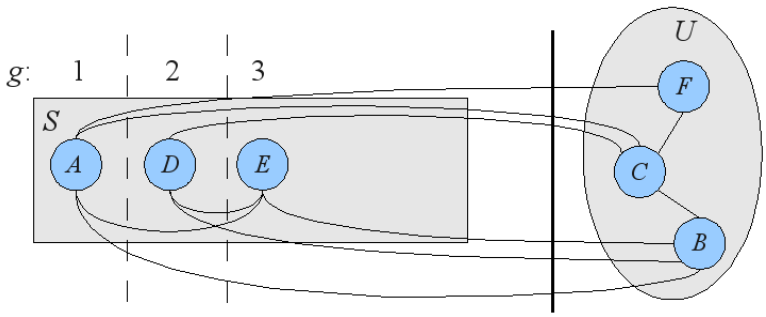


Figure 2: Partial solution.

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)}.

Particularizing the expression to compute the cutwidth shown in Section 1 to a partial solution (S, g), we can calculate the cutwidth of each labeled vertex in $\mathcal{G}_S$ with respect to the ordering g

and the edges in E_S, $CW_g(v)$ as follows:

$$CW_g(v) = \left|\left\{(u,w) \in E_S : g(u) \le g(v) < g(w)\right\}\right|$$

In the example 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 $CW_f(A) \ge CW_g(A)=1$, $CW_f(D) \ge CW_g(D)=2$ and $CW_f(E) \ge CW_g(E)=0$. We can therefore conclude that the cutwidth of the graph $CW_f(\vec{G})$ 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 $f \in S_g$:

$$CW_f(\vec{G}) \ge LB(S,g) = \max_{v \in S} CW_g(v).$$

In this section we propose five lower bounds, LB_1, LB_2, LB_3, LB_4 and LB_5, to the value of $CW_f(\vec{G})$ 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 computes the edges between the labeled and unlabeled vertices, LB_3 is a refinement of LB_1, LB_4 considers the best vertex to be labeled next in the partial solution and LB_5 is based on the distribution of the edges in $\vec{G}$ minimizing the cutwidth.

2.1 Lower bound LB_1

Let $N(v)$ be the set of adjacent vertices to vertex 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 $N(v)$ is adjacent to 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 $CW_f(u) + CW_f(v) \ge |N(v)|$. Therefore,

$$\max\{CW_f(u), CW_f(v)\} \ge |N(v)| / 2$$

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

$$CW_f(\vec{G}) \ge LB_1 = \max_{v \in \mathcal{V}} \left\lceil \frac{|N(v)|}{2} \right\rceil$$

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

2.2 Lower bound LB_2

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

$$CW_f(v) = CW_g(v) + \sum_{\substack{u \in S \\ 1 \le g(u) \le g(v)}} |N_U(u)| \qquad (9)$$

where $N_U(u)$ is the set of unlabeled adjacent vertices to u. The first term in this expression, $CW_g(v)$, corresponds to the cutwidth of v in $\vec{G}_S = (S, E_S)$. The second term computes 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 a 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 (9) provides an expression of $CW_f(v)$ for all v in $S \subseteq \mathcal{V}$, and that $CW_f(\vec{G})$ is the maximum of $CW_f(v)$ for all v in $\mathcal{V}$, we can conclude that:

$$CW_f(\vec{G}) \geq LB_2 = \max_{v \in S} \left\{ CW_g(v) + \sum_{\substack{u \in S \\ 1 \leq g(u) \leq g(v)}} |N_U(u)| \right\}$$

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

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

where:

$$CW_f(A) = CW_g(A) + |N_U(A)| = 1 + 3 = 4$$
$$CW_f(D) = CW_g(D) + |N_U(A)| + |N_U(D)| = 2 + 3 + 2 = 7$$
$$CW_f(E) = CW_g(E) + |N_U(A)| + |N_U(D)| + |N_U(E)| = 0 + 3 + 2 + 1 = 6$$

2.3 Lower bound LB_3

Given a partial solution (S, g) and an unlabeled vertex $u \in U$, let $N_S(u)$ be the set of labeled adjacent vertices to u. Let v_k be the vertex in S with the largest label (i.e, $g(v_k) = k = |S|$). It is clear that for any f in S_g and any v in S, $f(v) \leq f(v_k) < f(u)$. Then, $CW_f(v_k) \geq |N_S(u)|$. On the other hand, we can also apply the same argument to the vertices in U as in LB_1, obtaining an improved lower bound LB_3 for the vertices in U:

$$CW_f(\vec{G}) \geq LB_3 = \max_{u \in U} \left\{ \left\lceil \frac{1}{2} |N(u)| \right\rceil, N_S(u) \right\}$$

In the example in Figure 2, we can see that the value of LB_3 for vertices B, C and F:

$$LB_3(B) = \max\left\{ \left\lceil \frac{1}{2} |N(B)| \right\rceil, N_S(B) \right\} = \max\{2,3\} = 3$$
$$LB_3(C) = \max\left\{ \left\lceil \frac{1}{2} |N(C)| \right\rceil, N_S(C) \right\} = \max\{2,2\} = 2$$
$$LB_3(F) = \max\left\{ \left\lceil \frac{1}{2} |N(F)| \right\rceil, N_S(F) \right\} = \max\{1,1\} = 1$$

Therefore, LB_3 will be for this graph:

$$LB_3 = \max\{LB_3(B), LB_3(C), LB_3(F)\} = \max\{3,2,1\} = 3$$

2.4 Lower bound LB_4

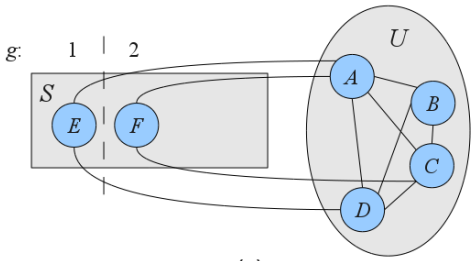
As in the previous case, consider a partial solution (S, g), an unlabeled vertex $u \in U$, the vertex v_k in S with the largest label, and a solution f in S_g. If the vertex u is labeled in f with k+1 (i.e., u follows v_k in the ordering f) its cutwidth can be computed as:

$$CW_f(u) = CW_g(v_k) - \left(|N_S(u)| - |N_U(u)| \right)$$

where $N_S(u)$ is the set of labeled adjacent vertices to u, and $N_U(u)$ is the set of unlabeled adjacent vertices to 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 the term $\left|N_S(u)\right| - \left|N_U(u)\right|$ for all $u \in U$. Thus we obtain:

$$CW_f(\mathcal{G}) \geq LB_4 = CW_g(v_k) - \max_{u \in U}\left(\left|N_S(u)\right| - \left|N_U(u)\right|\right)$$

Figure 3.a shows a partial solution (S, g) of the example given in Figure 1, where $S = \{E, F\}$, $g(E) = 1$, $g(F) = 2$ and $U = \{A, B, C, D\}$ with $CW_g(F) = 4$. Figure 3.b shows 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 $LB_4 = 4 - (-1) = 5$. 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	$\lvert 2\rvert - \lvert 3\rvert = -1$
B	$\lvert 0\rvert - \lvert 3\rvert = -3$
C	$\lvert 1\rvert - \lvert 3\rvert = -2$
D	$\lvert 1\rvert - \lvert 3\rvert = -2$

(b)

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

2.5 Lower bound LB_5

Given a graph $\mathcal{G}$ with n vertices and m edges we compute the lower bound LB_5 of its cutwidth $CW(\mathcal{G})$, by constructing an auxiliary graph $\mathcal{G}$' with n vertices and m edges distributed in such a way that it has minimum cutwidth. In other words, we "put" the edges in $\mathcal{G}$' between the appropriate vertices to obtain a minimum cutwidth. In this way, the cutwidth of $\mathcal{G}$' is a lower bound of the cutwidth of $\mathcal{G}$ 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 the auxiliary graph $\mathcal{G}$' 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 $\mathcal{G}$' is equal to 1 and it is clear that regardless how the edges are distributed in $\mathcal{G}$, given that it has m edges, for any labeling f, its cutwidth $CW_f(\mathcal{G})$ will be equal to or larger than $CW(\mathcal{G}')=1$. Moreover, if we have m=n, we need to add an extra edge to the connected path $\mathcal{G}$' and it necessarily results in a vertex with cutwidth 2; therefore, in this case $CW(\mathcal{G}')=2 \leq CW_f(\mathcal{G})$ for any labeling f of the vertices in $\mathcal{G}$.

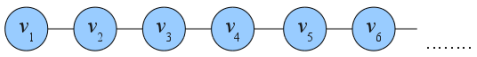


Figure 4: Graph $\mathcal{G}$' with m=n-1 edges (path).

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 a few extra edges increasing the cutwidth by only one unit. Specifically, we can add $\lfloor (n\text{-}1)/2 \rfloor$ edges between "alternated" vertices (v_i and v_{i+2}) as shown in Figure 5, keeping the cutwidth of $\mathcal{G}$' with value 2. We shall denote them edges of length 2. Therefore, the cutwidth of a graph $\mathcal{G}$ with n vertices and m edges with $n \leq m \leq n\text{-}1+\lfloor (n\text{-}1)/2 \rfloor$ satisfies $CW(\mathcal{G}') = 2 \leq CW_f(\mathcal{G})$ for any labeling f of the vertices in $\mathcal{G}$. Any extra edge would result in a cutwidth of 3.

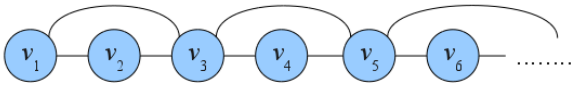


Figure 5: Graph $\vec{G}$' with a length 1 and 2 edges.

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

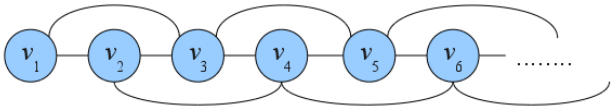


Figure 6: Graph $\vec{G}$' with cutwidth 3.

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

3. Initial Upper Bound

In this section, we propose a heuristic approach to obtain an upper bound for the cutwidth problem based on GRASP methodology (Feo et al. 1994). Each GRASP iteration involves constructing a trial solution and then applying a local search from the constructed solution. Figure 7 shows a pseudo-code of our GRASP construction method for the cutwidth problem.

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

WHILE ($U \neq \emptyset$)
5. $k = k + 1$
6. Construct $CL = \{v \in U / (w,v) \in \vec{E} \ \ \forall w \in S\}$
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 $RCL = \{v \in CL / e(v) \geq th\}$
10. Select a vertex u randomly in RCL
11. Label u with the label k
12. $U = U \setminus \{u\}$, $S = S \cup \{u\}$

Figure 7. Pseudo-code of the constructive method.

The constructive method starts by creating a list of unlabeled vertices U (initially $U = \vec{V}$). The first vertex v is randomly selected from all those vertices in U and labeled with 1. In subsequent construction steps, a candidate list CL is formed by 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:

$$e(u) = |N_S(u)| - |N_U(u)|$$

where $N_S(u)$ is the set of labeled adjacent vertices to u, and $N_U(u)$ is the set of unlabeled adjacent vertices to u. 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, $RCL = \{v \in CL \,/\, e(v) \geq th\}$ where the parameter *th* is a threshold to establish the "good" elements for selection as shown in Figure 7.

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 a labeling 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 the vertex v is inserted just before the vertex v_j in position j. In mathematical terms, from $f = (\ldots, v_{j-1}, v_j, v_{j+1}, \ldots, v_{i-1}, v, v_{i+1}, \ldots)$, we obtain the new ordering $f' = (\ldots, v_{j-1}, v, v_j, v_{j+1}, \ldots\ v_{i-1}, v_{i+1}, \ldots)$.

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

We define the set of critical vertices *CV* as those with a cutwidth value equal or close to the cutwidth of the graph. These vertices determine the value of the objective function or 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, j, v), where the meaning of *improving* is not limited to the objective function (which provides little information in this problem). The position j in the move is computed as the median of the positions (according to f) of the adjacent vertices to v (a search mechanism explores only positions close to j). An improving move is the one that either reduces $CW_f(\mathcal{G})$ 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. The method cuts off when there is no improving move associated with the vertices in *CV*.

4. The search tree

Branch and bound 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 consider the GRASP introduced in Section 3) to obtain an initial solution, whose objective function value gives an upper bound *UB* of the optimal value. Then, at each node of the search tree, it computes a lower bound *LB* (in our problem the maximum among LB_1, LB_2, LB_3, LB_4, and LB_5 introduced in Section 2) and compares it with *UB*. If $LB \geq UB$ then we fathom the node (because no better solution than the incumbent one can be found in this node); otherwise we branch the node and explore its first *child* node. When the exploration reaches a *leaf* node (that represents a complete solution of 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 continuing 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 as the output, the optimum solution to the problem. An early termination, due to time limitations, provides us with a lower bound and an upper bound of the optimum (this latter bound is obtained as the minimum of the lower bounds in the unexplored nodes).

In our search tree, the initial node branches into n nodes labeling each vertex *A*, *B*, *C*, … 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 vertex. Figure 8 represents this search tree for the example given in Figure 1.a.

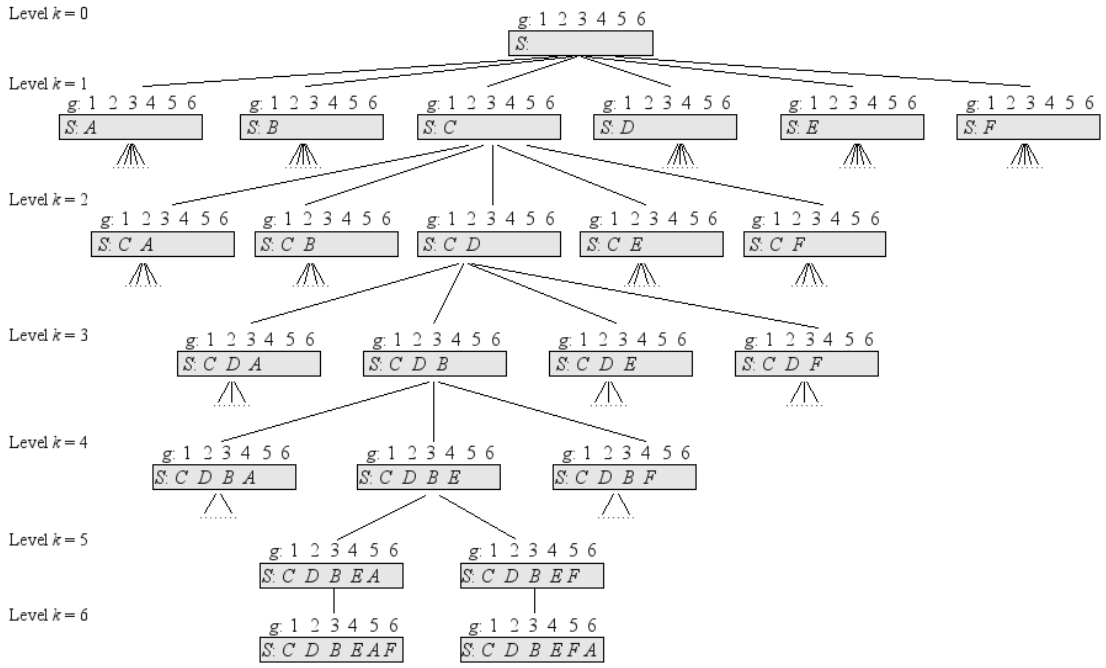


Figure 8: Search tree.

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. Figure 9 shows a pseudo-code of the BB1 in which we initially call BB1($Node_k$) with $Node_k = \{S = \emptyset;\ g(u) = 0\ \forall u \in \mathring{V}\}$ and set $k = 0$.

```
PROCEDURE BB1(Node_k)
  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(G) as the cutwidth of its associated solution
        IF (CW_f(G) < UB)
              3. UB = CW_f(G)
  ELSE
        4. Compute LB = max {LB_1, LB_2, LB_3, LB_4, LB_5}
        IF (LB < UB)
              5. Let U be the set of unlabeled vertices
              6. k = k+1
              WHILE (U ≠ Ø)
                    7. Select u from U in lexicographical order
                    8. U = U \ {u}
                    9. Set Node_k = {S = S ∪ {u}; g(u) = k}
                    10. BB1(Node_k)
```

Figure 9: Pseudo-code of BB1

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 10 provides a pseudo-code of this procedure.

```
PROCEDURE BB3()
  1. Compute UB with the GRASP algorithm
  2. k = 0
  3. Set Node_k = {S = ∅; g(u) = 0 ∀u∈V}
  4. PQ = ∅ /* empty priority queue */
  5. Enqueue(Node_k, PQ) /* add an element to the queue with an associated priority */
  WHILE (PQ ≠ Ø)
      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(G) as the cutwidth of its associated solution
          IF (CW_f(G) < UB)
              6. UB = CW_f(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 ≠ Ø)
                  9. Select u from U in lexicographical order
                  10. U = U \ {u}
                  11. Set Node_{k+1} = {S = S ∪ {u}; g(u) = k+1}
                  12. Compute LB = max {LB_1, LB_2, LB_3, LB_4, LB_5}
                  IF (LB < UB)
                      13. In-queue (Node_{k+1}, PQ)
```

Figure 10. Pseudo-code of BB3.

5. Computational Experiments

This section describes the computational experiments performed to test the efficiency of our branch and bound procedure, as well as to compare it with the linear integer formulation proposed previously. We have implemented the branch and bound algorithm in Java SE 6 and solved the linear integer formulation with Cplex 11.1. All the experiments were conducted on an Intel Core 2 Quad CPU and 6 GB RAM. We limit the running time on each instance to 30 minutes of CPU.

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://heur.uv.es/optsicom/cutwidth.

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., 2008). They are also called two dimensional meshes and, as documented in Raspaud et al. (2008), 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* × *height* where *width*, *height* ∈ {3, 4, …, 10} and *width* ≥ *height*.

HB: We derived 37 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.

$$gap = UB - LB \qquad \%gap = \frac{UB - LB}{LB} \times 100$$

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). Table 1 shows these values 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 show that BB1, BB2 and BB3 are able to solve the 5 small instances optimally. Consequently, as shown in the five rows corresponding to these instances, the number of explored and fathomed nodes is exactly the same in all three methods. This always

occurs if the search tree is fully explored since the only difference among BB1, BB2 and BB3 is the branching order. In the three grids, Grid5x5, Grid6x8 and Grid7x9, the three variants of the branch and bound are able to finish. However, in the other two, Grid8x9 and Grid10x10, none of the variants is able to finish. In the former cases, we can see that 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 partially be 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" to be 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 improving the final lower bound, thus providing the best overall strategy. Table 2 shows 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 clearly confirm that the best strategy to explore the search tree is BB3, in which nodes are ordered according to their bound. We will therefore consider this variant 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.

The next experiment undertakes to 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_5, in the 15 instances of *Set*1 (reporting the average on *Small*, *Grids*, and *HB* instances).

	LB_1	LB_2	LB_3	LB_4	LB_5
Small	0	93.85	0	96.30	0
Grids	0	99.49	0	98.82	0
HB	0	92.41	0	98.42	0
Total	0	95.12	0	98.53	0

Table 3. Average fathoms of each lower bound.

Results presented in Table 3 clearly show that LB_1, LB_3 and LB_5 are not fathoming a significant number of nodes (the associated percentages are very close to 0, and represented by 0 in the table for the sake of simplicity). On the other hand, the behavior of LB_2 and LB_4 is very similar, fathoming on average about 95% and 98% of the fathomed nodes respectively. Therefore we shall not compute LB_1, LB_3 and LB_5 when exploring the nodes in the search tree. However,

when the pre-established time limit is reached and the method terminates, to report the final gap, the computation of these three bounds in the current nodes could contribute to increasing the final lower bound, thus reducing the gap. To test this point we perform a new experiment reporting the gap values when only LB_2 and LB_4 are computed in the search tree. We shall call this method *BB*. Then, we incorporate the computation of LB_3 and LB_5 at the end of the process, which results in the entire method tested above (note that LB_3 contains LB_1 in its definition). 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	0.0	0.0	0.0	0.0
Grids	0.8	10.0	0.8	10.0
HB	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 clearly show that the addition of LB_3 and LB_5 helps to reduce the final gap of the method. Examining the two previous experiments together, we can conclude that the lower bounds complement each other. On one hand, LB_2 and LB_4 fathom a large number of nodes in the search tree. On the other hand, LB_3 and LB_5 reduce the final gap. We shall therefore include the four lower bounds in our final branch and bound algorithm.

The fifth experiment focuses on the combination of the GRASP heuristic (described in Section 3) 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*. Table 5 shows the average, absolute and relative gaps of both variants.

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

Table 5. Comparison of heuristic with random initial solution.

As shown in Table 5, the results obtained with the branch and bound coupled with the heuristic initial upper bound are better, as expected, than those obtained with the random variant. Specifically, the former obtains an average absolute gap value of 21.9 and an average relative gap value of 106.9%; both values compare favorably with the respective gap values of 71.3 and 512.8% obtained for the branch and bound with the initial upper bound from a random solution.

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.

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 the five lower bounds, LB_1 to LB_5, and the initial GRASP upper bound. Table 6 reports 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*). As in the previous experiments we limit the CPU time to 1800 seconds.

		BB1	**BB2**	**BB3**	**Cplex**
Small (42)	# opt	42	42	42	9
	gap	0.0	0.0	0.0	1.9
	%*gap*	0.0	0.0	0.0	54.7
	CPU Time	2.1	2.2	4.9	1573.9
Grids (36)	# opt	30	30	30	2
	gap	1.1	1.0	0.28	4.2
	%*gap*	37.1	29.5	3.5	211.1
	CPU Time	301.5	301.5	302.4	1707.9
HB (37)	# opt	5	5	4	0
	gap	51.8	51.8	49.7	97.0
	%*gap*	314.4	314.4	210.3	634.8
	CPU Time	1574.7	1573.9	1594.4	1800.0

Table 6. Branch and bound versus Cplex.

Table 6 shows 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. On the other hand, 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 350% (while the average relative gap value of Cplex is 634.8% in these instances). On the other hand, 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 210.3% while BB1 and BB2 present a value of 314.4%.

In our last experiment we represent the search profile of the three branch and bound variants, BB1, BB2 and BB3 when running for 3 hours. Specifically, Figure 11 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 the branch and bound execution (and join the points with a line to observe the trend).

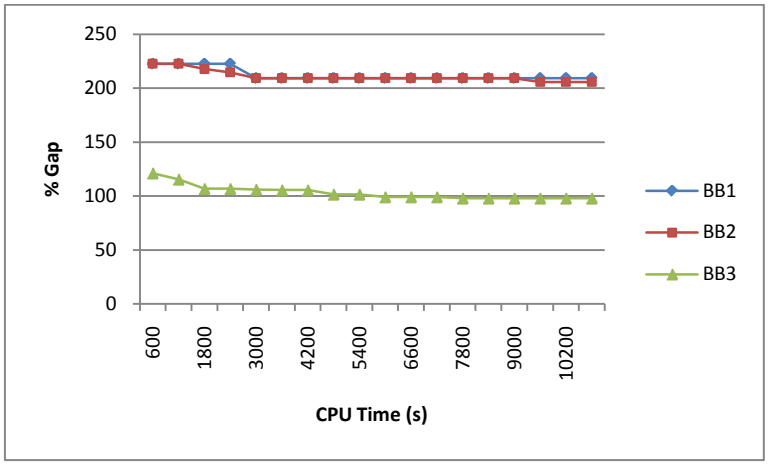


Figure 11. Relative gap profile

The progression of the average gap represented in Figure 11 confirms that BB3 performs slightly better than BB1 and BB2. On the other hand, 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.

Table 7 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 upper bound and the branch and bound for 4 hours to obtain the lower bound on each instance (thereby setting a benchmark for future comparisons).

6. Conclusions

We have developed an exact procedure based on the branch and bound methodology 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 to 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 an 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 it is able to solve all the small-sized problems and some of the larger ones optimally. We finally provide detailed results for the hardest instances for future comparisons.

Acknowledgments

This research has been partially supported by the *Ministerio de Ciencia e Innovación* of Spain (Grant Ref. TIN2006-02696, TIN2008-06890-C02-02, TIN2009-07516) and by the *Comunidad de Madrid – Universidad Rey Juan Carlos* projects (Ref. URJC-CM-2008-CET-3731, URJC-CM-2008-CET-3625).

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.

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.

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.

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

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

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.

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.

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.

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 and H. L. Bodlaender. A polynomial time algorithm for the cutwidth of bounded degree graphs with small treewidth. *In proceedings of the 9th Annual European Symposium on Algorithms*, Lecture Notes in Computer Science, 2161:380-390, 2001.

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	20	23
bcspwr01	39	46	**5**	**5**
bcsstk01	48	176	22	32
bcspwr02	49	59	**5**	**5**
curtis54	54	124	10	13
will57	57	127	7	11
impcol_b	59	281	18	55
bcsstk02	66	2145	**1089**	**1089**
steam3	80	424	**20**	**20**
ash85	85	219	10	16
nos4	100	247	**12**	**12**
gent113	104	549	19	87
bcsstk22	110	254	6	13
gre__115	115	267	10	36
dwt__234	117	162	6	12
bcspwr03	118	179	5	10
lns__131	123	275	6	30
arc130	130	715	62	202
bcsstk04	132	1758	107	310
west0132	132	404	14	71
impcol_c	137	352	11	46
can__144	144	576	**25**	**25**
lund_a	147	1151	37	113
lund_b	147	1147	37	111
bcsstk05	153	1135	34	115
west0156	156	371	12	56
nos1	158	312	**4**	**4**
can__161	161	608	21	52
west0167	167	489	12	55
mcca	168	1662	58	390
fs_183_1	183	701	52	190
gre__185	185	650	19	48
will199	199	660	16	132

Table 7. Lower and upper bounds for the *HB* instances.