GRASP WITH PATH RELINKING HEURISTICS FOR THE ANTIBANDWIDTH PROBLEM

A. DUARTE, R. MARTÍ, M.G.C. RESENDE, AND R.M.A. SILVA

Abstract. This paper proposes a linear integer programming formulation and several heuristics based on GRASP and path relinking for the antibandwidth problem. In the antibandwidth problem, one is given an undirected graph with n nodes and must label the nodes in a way that each node receives a unique label from the set $\{1, 2, \ldots, n\}$, such that, among all adjacent node pairs, the minimum difference between the node labels is maximized. Computational results show that only small instances of this problem can be solved exactly (to optimality) with a commercial integer programming solver and that the heuristics find high-quality solutions in much less time than the commercial solver.

1. Introduction

Let $G = (V, E)$ be an undirected graph, where V denotes the set of vertices and E the set of edges. Let $n = |V|$ and $m = |E|$. A *labeling* f of the vertices of G is a one-to-one mapping from the set V onto the integers $\{1, 2, \ldots, n\}$, i.e. each vertex $v \in V$ has a unique label $f(v) \in \{1, 2, \ldots, n\}$. Note that each labeling can be identified with a permutation of $\{1, 2, \ldots, n\}$. Given a graph G, the *bandwidth* of a vertex v, $B_f(v)$, is the maximum of the differences between $f(v)$ and the labels of its adjacent vertices (i.e. vertices in $N(v)$). That is:

$$B_f(v) = \max\{|f(u) - f(v)| : u \in N(v)\}.$$

The bandwidth of a graph G with respect to a labeling f is then:

$$B_f(G) = \max\{B_f(v) : v \in V\}$$

Then, the bandwidth reduction problem consists in minimizing the value of $B_f(G)$ over all possible labelings, i.e.

$$B(G) = \min_{f \in \pi_n} B_f(v),$$

where π_n is the set of all permutations of $\{1, 2, \ldots, n\}$. The bandwidth minimization problem has been studied extensively. See, e.g. Gibbs et al. (1976), Martí et al. (2001), Piñana et al. (2004), and Rodriguez-Tello et al. (2008). An important application of this problem arises in the solution of non-singular systems of linear algebraic equations. Preprocessing of the coefficient matrix is done to reduce its bandwidth, resulting in substantial savings in the computational effort associated

Date: April 3, 2009. Revised March 15, 2010.

Key words and phrases. Metaheuristics, bandwidth, antibandwidth, GRASP, path-relinking, evolutionary path-relinking.

AT&T Labs Research Technical Report.

with solving the system of equations. Such systems occur in the design of aircraft structures, liquid nitrogen gas tanks, propeller blades, and submarines. The bandwidth minimization problem is known to be NP-hard (Papadimitriou, 1976).

In some practical problems it is interesting to maximize the minimum difference between adjacent vertices. For example, if the vertices of the graph G represent sensitive facilities or chemicals, then placing them too close together can be risky. This optimization problem is dual to the *bandwidth* problem and it is commonly known as the *antibandwidth* problem, also known as the *separation* problem (Leung et al., 1984) and the *dual bandwidth* problem (Yixun and Jinjiang, 2003). It can be defined as follows. Given a graph G and a labeling f, the *antibandwidth* $AB_f(G)$ of f is defined as

$$AB_f(G) = \min\{AB_f(v) : v \in V\},$$

where

$$AB_f(v) = \min\{|f(u) - f(v)| : u \in N(v)\}$$

is the antibandwidth of vertex v and $N(v)$ is the set of its adjacent vertices. The antibandwidth $AB(G)$ of G is

$$AB(G) = \max_{f \in \pi_n} AB_f(G).$$

The antibandwidth problem is NP-hard (Leung et al., 1984). However, special cases can be solved in polynomial time, including the complements of intervals, arborescent comparability, and on threshold graphs (Donnelly and Isaak, 1999).

This problem arises in different contexts such as in scheduling (Leung et al., 1984), in some variations of the multiprocessor scheduling problems (MSRRD, SSUPS, and MSIRF), in relation to obnoxious facility location (Raspaud et al., 2009), and in radio frequency assignment (Hale, 1980).

Previous papers on the antibandwidth problem where devoted to the theoretical study of its properties to find optimal solutions for special cases. In the variant studied in Leung et al. (1984), the problem consists in finding a labeling f with a value $AB_f(G)$ greater than a predetermined value k. The authors determined the NP-completeness of this problem and give polynomial time algorithms for several classes of graphs.

Yixun and Jinjiang (2003) proposed several upper bounds for $AB(G)$. Some of them are related to the independent and chromatic numbers of G. The bound UB_1 is computed as a function of the minimum degree ($mind$) and maximum degree ($maxd$) of G.

$$UB_1 = \min\left\{\left\lfloor \frac{n - mind + 1}{2} \right\rfloor, n - maxd\right\}, \tag{1}$$

where

$$mind = \min_{u \in V} |N(u)|,$$
$$maxd = \max_{u \in V} |N(u)|.$$

A weaker upper bound, UB_2, is computed as a function of the number n of vertices and the number m of edges:

$$UB_2 = \left\lfloor n - \frac{\sqrt{8m+1} - 1}{2} \right\rfloor. \tag{2}$$

Raspaud et al. (2009) solve the antibandwidth problem for several classes of special graphs: two dimensional meshes (Cartesian product of two paths), tori (Cartesian product of two cycles), and hypercubes. Török and Vrt'o (2007) extended these results to the case of three dimensional meshes. More recently, Dobrev et al. (2009) derive tight upper bounds for general Hamming graphs and optimal solution values for a special class of these graphs.

Until now, no heuristic has been proposed to obtain high-quality solutions for the antibandwidth problem on general graphs. In this paper, we first propose a linear integer programming formulation for the antibandwidth problem and then propose some heuristics based on GRASP and evolutionary path relinking to find optimal or near-optimal solutions on general graphs. Finally, we test our formulation with CPLEX and study the efficiency of these heuristics on a set of medium- and large-scale instances, including meshes and Hamming graphs to measure the deviation of the best solution found with our methods with respect to the optimal solution on large size instances.

2. Integer programming model

We next propose an integer programming model for the antibandwidth problem. Let x_{ik} be a binary variable that takes on the value 1 if and only if $f(i) = k$, i.e. node i takes label k. Define the integer variable l_i to be the label of node i, i.e. $l_i = f(i) \in \{1, 2, \ldots, n\}$. Finally, let $b = AB_f(G) = \min_{(u,v)\in E} |f(u) - f(v)|$. An integer programming formulation for the antibandwidth problem is:

$$\max b$$

subject to

$$\sum_{i=1}^{n} x_{ik} = 1, \quad \forall k = 1, \ldots, n, \tag{3}$$

$$\sum_{k=1}^{n} x_{ik} = 1, \quad \forall i = 1, \ldots, n, \tag{4}$$

$$\sum_{k=1}^{n} k \cdot x_{ik} = l_i, \quad \forall i = 1, \ldots, n, \tag{5}$$

$$b \leq |l_i - l_j|, \quad \forall (i,j) \in E, \tag{6}$$

$$x_{ik} \in \{0,1\}, \forall i, k = 1, \ldots, n, \tag{7}$$

$$l_i \in \{1, \ldots, n\}, \quad \forall i = 1, \ldots, n. \tag{8}$$

Constraints (3) assign one label to each node while constraints (4) assign one node to each label. Constraints (5) explicitly compute the value of the labels from the x_{ik} variables. Taken together with the objective function, constraints (6) imply that $b = \min\{|l_i - l_j| : (i,j) \in E\}$. Constraints (7)–(8) define the domain of the variables.

Note that constraints (6) are nonlinear. To linearize these constraints, we employ a standard artifact. In (6), if $l_i \geq l_j$, then the constraint can be represented as

$$b \leq l_i - l_j. \tag{9}$$

Otherwise, the constraint is

$$b \leq -(l_i - l_j). \tag{10}$$

For each $(i,j) \in E$, we introduce binary variables y_{ij} and z_{ij} that indicate whether $l_i \geq l_j$ holds. If $l_i \geq l_j$ then $y_{ij} = 0$ and $z_{ij} = 1$. Otherwise, $y_{ij} = 1$ and $z_{ij} = 0$. With these variables, (9) and (10) can be replaced, respectively, by

$$b - (l_i - l_j) \leq 2y_{ij}(n-1), \ \ \forall (i,j) \in E$$

and

$$b + (l_i - l_j) \leq 2z_{ij}(n-1), \ \ \forall (i,j) \in E$$

with the additional constraints

$$y_{ij} + z_{ij} = 1, \ \ \forall (i,j) \in E, \tag{11}$$

$$b \geq 1. \tag{12}$$

Constraints (11) imply that only one of the alternatives $l_i \geq l_j$ or $l_i < l_j$ holds. Since the absolute value is removed in the linearization, we must add constraint (12) to guarantee positiveness.

This results in the following linear integer programming formulation for the antibandwidth problem:

$$\max b$$

subject to

$$\begin{aligned}
&\sum_{i=1}^{n} x_{ik} = 1, \ \ \forall k = 1, \ldots, n, \\
&\sum_{k=1}^{n} x_{ik} = 1, \ \ \forall i = 1, \ldots, n, \\
&\sum_{k=1}^{n} k \cdot x_{ik} = l_i, \ \ \forall i = 1, \ldots, n, \\
&b - (l_i - l_j) \leq 2y_{ij}(n-1), \ \ \forall (i,j) \in E, \\
&b + (l_i - l_j) \leq 2z_{ij}(n-1), \ \ \forall (i,j) \in E, \\
&y_{ij} + z_{ij} = 1, \ \ \forall (i,j) \in E, \\
&b \geq 1, \\
&x_{ik} \in \{0,1\}, \ \ \forall i,k = 1, \ldots, n, \\
&y_{ij} \in \{0,1\}, \ \ \forall (i,j) \in E, \\
&z_{ij} \in \{0,1\}, \ \ \forall (i,j) \in E, \\
&1 \leq l_i \leq n, \ \ \forall i = 1, \ldots, n.
\end{aligned}$$

The integer programming model has $n^2 + n + 2m + 1 = O(n^2)$ variables and $3n + 3m + 1 = O(n^2)$ constraints.

3. GRASP

The GRASP metaheuristic was developed in the late 1980s (Feo and Resende, 1989) and the acronym was coined in Feo et al. (1994). We refer the reader to Resende and Ribeiro (2003; 2009) for recent surveys of this metaheuristic. Each GRASP iteration consists in constructing a trial solution and then applying local

begin *C1*
1 $U \leftarrow \{1, \ldots, n\}$ Set of unlabeled vertices;
2 $L \leftarrow \{1, \ldots, n\}$ Set of available labels;
3 $u_0 \leftarrow Select\_First\_Vertex(U)$;
4 $l_0 \leftarrow Select\_First\_Label(V)$;
5 $f(u_0) \leftarrow l_0$; $L \leftarrow L \setminus \{l_0\}$; $U \leftarrow U \setminus \{u_0\}$;
6 $CL \leftarrow \{u \in U \mid (u, u_0) \in E\}$;
7 Select α_1 randomly from $(0, 1)$;
8 **while** $L \neq \emptyset$ **do**
9 $\quad d_{min} \leftarrow \min_{u \in CL} d(u)$;
10 $\quad d_{max} \leftarrow \max_{u \in CL} d(u)$;
11 $\quad RCL \leftarrow \{v \in CL \mid d(v) \geq d_{min} + \alpha_1 \cdot (d_{max} - d_{min})\}$;
12 $\quad$ Select u randomly from RCL;
13 $\quad l \leftarrow Best\_Available\_Label(L)$;
14 $\quad f(u) \leftarrow l$;
15 $\quad N^*(u) \leftarrow$ Unlabeled vertices adjacent to u;
16 $\quad CL \leftarrow CL \setminus \{u\} \cup N^*(u)$;
17 $\quad L \leftarrow L \setminus \{l\}$;
18 $\quad U \leftarrow U \setminus \{u\}$;
19 **end-while**;
end

FIGURE 1. Constructive heuristic *C1*.

search from the constructed solution. The construction phase is iterative, randomized greedy, and adaptive. In this section we propose several adaptations of the GRASP metaheuristic for the antibandwidth problem.

3.1. **Construction procedures.** We have designed two constructive algorithms *C1* and *C2* for the antibandwidth problem. *C1* implements a typical GRASP construction where each candidate element is initially evaluated by a greedy function to construct a Restricted Candidate List (*RCL*) and one element is selected at random from the *RCL*. *C1* tries to assign a relatively "small" label to the selected vertex and a "large" label to its adjacent vertices. The rationale behind this is to maximize the value of the antibandwidth (i.e. maximizing the minimum difference of adjacent labels). Similarly, we can also assign a "large" label to the selected vertex and a relatively "small" label to its adjacent vertices.

Given the set U (with $|U| = n$) of unlabeled vertices and L (with $|L| = n$) of available labels, the construction procedure *C1* performs n steps to obtain a solution, as shown in Figure 1. It starts by selecting a vertex u_0 at random (see step 3 in Figure 1) and then assigning a label l_0 to u_0 (step 5). Additionaly sets U and L are updated. We have implemented three different ways of selecting the first label l_0: (1) select l_0 at random in the range $[1, n]$ ($l_0 = random(1, n)$); (2) assign to l_0 the average range value ($l_0 = n/2$); and (3) set l_0 equal to one ($l_0 = 1$). We will discuss in the experimental section the impact of these strategies in the performance of the method.

Once the first label l_0 is assigned to vertex u_0, *C1* constructs the candidate list *CL* (step 6) with the vertices adjacent to u_0. It then computes d_{min} and d_{max} as, respectively, the minimum and maximum degrees of the elements in *CL*. The algorithm next constructs a *RCL* (step 11) with all the unselected vertices in *CL* having degree greater than or equal to a specified cutoff value

$$d_{min} + \alpha_1 \cdot (d_{max} - d_{min}),$$

where

$$d_{min} = \min_{u \in CL} d(u) \text{ and } d_{max} = \max_{u \in CL} d(u).$$

It randomly selects an element u from the *RCL* (step 12) and assigns the best available label to it. To this end, we compute the smallest ($l_{min}(u)$) and largest ($l_{max}(u)$) labels of the already labeled vertices adjacent to u, as well as the best label

$$Best\_l(u) = \text{argmax}_{1 \le l \le n} \min\{|l - l_{max}(u)|, |l - l_{min}(u)|\}$$

for u, where

$$l_{max}(u) = \max_{v \in LN(u)} f(v)$$

and

$$l_{min}(u) = \min_{v \in LN(u)} f(v),$$

where $LN(u)$ is the set of labeled vertices adjacent to u. The closest available label to $Best\_l(u)$ is assigned to u (step 14). *CL*, L, and U are updated in steps 16, 17, and 18 respectively. The method iterates as long as the set L contains available labels.

Consider the partial representation of a graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, depicted in Figure 2, consisting of five vertices with a labeling given by the numbers shown next to each vertex (vertex u is not labeled yet). To obtain a label for vertex u, construction method *C1* first explores the set of its adjacent labeled vertices, $LN(u) = \{v, w, x, y\}$. It then computes the minimum and maximum labels among the adjacent labeled vertices which are respectively $l_{min}(u) = 4$ and $l_{max}(u) = 10$. Finally, *C1* computes the best label for u, $Best\_l(u)$ with the above expression, which basically tries labels from 1 to n and keeps the one which maximizes the difference with respect to the labels of its neighbors. In this example, considering $n = 10$, label 1 maximizes the antibandwidth of vertex u obtaining a value of $AB_f(u) = 3$.

We now consider *C2*, based on another construction strategy introduced in Resende and Werneck (2004) in which randomization takes place before the greedy selection is applied in each construction step. In *C2* we first randomly choose candidates and then evaluate them according to a greedy function to make the greedy choice. *C2* first constructs a restricted candidate list *RCL2* with a fraction α_2 ($0 \le \alpha_2 \le 1$) of the elements in *CL* selected at random. Then, it evaluates all the elements in *RCL2*, computing $d(u)$ for all $u \in RCL2$, and selects the best one, i.e. the element u_0 such that

$$d(u_0) = \max_{u \in RCL2} d(u).$$

Figure 3 shows the pseudo-code of the constructive procedure *C2*. The main difference with respect to *C1* is that greedy evaluation and random selection are switched. However, the computation of the best available label for a selected vertex

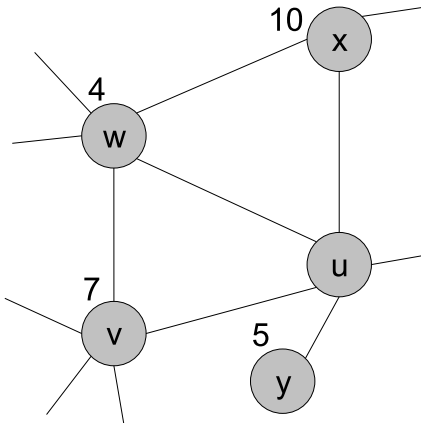

FIGURE 2. Labeled partial graph

begin *C2*
1 $U \leftarrow \{1, \ldots, n\}$ Set of unlabeled vertices;
2 $L \leftarrow \{1, \ldots, n\}$ Set of available labels;
3 $u_0 \leftarrow Select\_First\_Vertex(U)$;
4 $l_0 \leftarrow Select\_First\_Label(L)$;
5 $f(u_0) \leftarrow l_0$; $U \leftarrow U \setminus \{u_0\}$; $L \leftarrow L \setminus \{l_0\}$;
6 $CL \leftarrow \{v \in U \mid (u_0, v) \in E\}$;
7 Select α_2 randomly from $(0, 1)$;
8 **while** $L \neq \emptyset$ **do**
9 $size \leftarrow \alpha_2 \cdot |CL|$;
10 $RCL2 \leftarrow size$ elements randomly selected from CL;
11 Select $u \in RCL \mid d(u) = \max_{v \in RCL} d(v)$;
12 $l \leftarrow Best\_Available\_Label(L)$;
13 $f(u) \leftarrow l$;
14 $U \leftarrow U \setminus \{u\}$;
15 $N^*(u) \leftarrow$ Unlabeled vertices adjacent to u;
16 $CL \leftarrow CL \setminus \{u\} \cup N^*(u)$;
17 $L \leftarrow L \setminus \{l\}$;
18 **end-while**;
end

FIGURE 3. Constructive heuristic *C2*.

is the same in both methods. In the computational study, we will discuss how variations in the search parameters α_1 and α_2 affect *C1* and *C2*, respectively.

3.2. **Local search procedure.** It is worthwhile noting that optimizing a max-min objective function implies that there may be many different solutions with the same objective function value, i.e. the solution space has a flat landscape.

The bandwidth reduction (Piñana et al., 2004), the max-min diversity (Resende et al., 2010), and the antibandwidth problems are examples where this occurs. For this reason these problems are challenging for solution methods based on heuristic optimization. This issue is especially relevant in local search procedures where the improvement is based on movements (most of them with a null associated move value).

In the case of the antibandwidth problem, there may be many labelings f with identical $AB_f(G)$ value. Moreover, for a given labeling there may be multiple vertices with antibandwidth equal to $AB_f(G)$. Consequently, changing a labeling to increase the antibandwidth $AB_f(u)$ of a particular vertex u does not necessarily imply that $AB_f(G)$ will also increase. Moreover, vertices with unequal antibandwidth, but close to $AB_f(G)$, can be crucial in future iterations. They do not determine the value of the objective function $AB_f(G)$ in the current labeling, but they are considered likely to do so in subsequent iterations. We therefore define the set $C(f)$ of *crucial vertices* of a labeling f to be

$$C(f) = \{u : AB_f(u) \leq \beta \times AB_f(G)\},$$

where β is a parameter that takes values in the range $1 < \beta < 2$.

We define a move as the swap of labels of a pair of vertices. That is, the operator $move(v, u)$ assigns the label $f(u)$ to vertex v and the label $f(v)$ to vertex u obtaining a new labeling f'. To significantly increase the value of the current labeling f, the local search procedure scans the set of crucial vertices $C(f)$ changing their labels to increase their antibandwidth. To find a good label for a vertex $u \in C(f)$ we proceed in a way similar to the construction procedures above. Define

$$Best\_l(u) = \text{argmax}_{1 \leq l \leq n} \min\{|l - l_{max}(u)|, |l - l_{min}(u)|\}$$

to be the best label for u, where

$$l_{max}(u) = \max_{v \in N(u)} f(v)$$

and

$$l_{min}(u) = \min_{v \in N(u)} f(v).$$

Once we establish the best label $Best\_l(u)$ for vertex u, we need to determine the vertex v with this label ($f(v) = Best\_l(u)$) to evaluate $move(v, u)$. Note that we know that the label $Best\_l(u)$ is "good" for u, but we need to check whether $f(u)$ is "good" for v. We therefore extend the search for a good label for u to include not only the vertex v with $f(v) = Best\_l(u)$, but also those vertices with labels close to $Best\_l(u)$.

If $Best\_l(u) > l_{max}(u)$, then $Best\_l(u) = n$. We scan the labels $f'(u) = n, n - 1, n - 2, \ldots, l_{max}(u) + AB_f(u) + 1$ sequentially until we find the first available label or verify that none of these labels is available. It should be noted that if we were to choose $f'(u)$ to be one of labels $l_{max}(u) + AB_f(u) - 1, \ldots, l_{max}(u) + 1$, then this would result in $AB_{f'}(u) < AB_f(u)$, thus deteriorating the current solution.

Symmetrically, if $Best\_l(u) < l_{min}(u)$, then $Best\_l(u) = 1$. A label that is close to 1 will improve the antibandwidth of u if is less than $l_{min}(u) - AB_f(u)$. Note that if were to consider $f'(u)$ to be one of labels $l_{min}(u) - AB_f(u) + 1, \ldots, l_{min}(u) - 1$, then the antibandwidth of vertex u would be less than $AB_f(u)$. We therefore sequentially examine labels $f'(u) = 1, \ldots, l_{min}(u) - AB_f(u) - 1$ until we find the first available label or verify that none of these labels is available.

Finally, the remaining possibility is when $l_{min}(u) < Best\_l(u) < l_{max}(u)$. In this case good labels for u are located close to $Best\_l(u)$ but far from both $l_{min}(u)$ and $l_{max}(u)$. We scan these labels sequentially in the following order: $Best\_l(u) + 1, Best\_l(u) - 1, Best\_l(u) + 2, Best\_l(u) - 2, \ldots$, until we reach the upper bound $l_{max}(u) - AB_f(u) - 1$ and the lower bound $l_{min}(u) + AB_f(u) + 1$ until we find the first available label or verify that none of these labels is available.

To summarize, depending on the value of $Best\_l(u)$ with respect to $l_{min}(u)$ and $l_{max}(u)$, the set $N'(u)$ of suitable swapping vertices for $u \in C(f)$ is

$$N'(u) = \begin{cases} \{v \in V : 1 \leq f(v) < l_{min}(u) - AB_f(u)\} \\ \qquad\qquad \text{if } Best\_l(u) < l_{min}(u); \\ \{v \in V : l_{max}(u) + AB_f(u) < f(v) \leq n\} \\ \qquad\qquad \text{if } Best\_l(u) > l_{max}(u); \\ \{v \in V : l_{min}(u) + AB_f(u) < f(v) < l_{max}(u) - AB_f(u)\} \\ \qquad\qquad \text{if } l_{min}(u) < Best\_l(u) < l_{max}(u). \end{cases} \tag{13}$$

The candidate list $CL(u)$ of moves associated with a vertex $u \in C(f)$ is defined as

$$CL(u) = \{move(v, u) : v \in N'(u)\}.$$

Note that if $N'(u) = \emptyset$, then $AB_f(u)$ cannot be increased in a single step by changing the current label of u.

Let us consider again the example introduced in Figure 2 with vertex u labeled with 1 and not considering the edges with a single endpoint. The antibandwidth of each vertex depicted in the partial representation of Figure 2 is

$$\begin{aligned} AB_f(u) &= \min\{|1-7|, |1-4|, |1-10|, |1-5|\} = 3, \\ AB_f(v) &= \min\{|7-1|, |7-4|\} = 3, \\ AB_f(w) &= \min\{|4-1|, |4-7|, |4-10|\} = 3, \\ AB_f(x) &= \min\{|10-1|, |10-4|\} = 6, \\ AB_f(y) &= \min\{|5-1|\} = 4. \end{aligned}$$

Consequently, the antibandwidth of the graph, computed as the minimum of these values for all vertices, is less than or equal to 3 (we only depict in the figure a fraction of the vertices). As noted above, we can see that the solution space for this problem has a flat landscape because relabeling one vertex does not necessarily improve the objective function. In other words, we need to apply several moves to change the value of a solution. If we consider that in this example $AB_f(G) = 3$ and $\beta = 1.4$, the set of crucial vertices is $C(f) = \{u, v, w, y\}$. The goal of the local search procedure consists in increasing the antibandwidth of the vertices in $C(f)$. To do that, we compute the best label $Best\_l(u)$ for each vertex u. Let us consider in our example of Figure 2 the vertex v with $l_{min}(v) = 1$ and $l_{max}(v) = 4$. In this case, $Best\_l(v) = 10$ (assuming again that $n = 10$). Labels 8, 9, and 10 can be assigned to vertex v to improve its antibandwidth. Therefore, $N'(v) = \{z \in V : l_{max}(v) + AB_f(v) < f(z) \leq Best\_l(v)\} = \{z \in V : 7 < f(z) \leq 10\}$.

One of the key elements in heuristic search is the definition of the value of a move. The most common practice is to define the move value as the change in the objective function value. However, in the context of the antibandwidth problem, as we have mentioned, the change in the objective function value usually provides little or no information. Therefore, given a vertex $u \in C(f)$, we define the value of

$move(u, v) \in CL(u)$ as the difference in the antibandwidth of vertex u. If f is the original labeling and f' the resulting labeling after performing $move(u, v)$, then

$$MoveValue(u, v) = AB_{f'}(u) - AB_f(u).$$

Note that we do not consider in the definition of the move value the change in the antibandwidth of vertex v. We therefore check whether this change affects $AB_{f'}(G)$ and only perform $move(u, v)$ if $AB_{f'}(v) > AB_f(G)$. On the other hand, note that the calculation of $AB_f(G)$ requires the examination of all the vertices in the graph and is computationally expensive. We therefore do not update this value after performing a single move (as is customary in local search heuristics). As suggested in Glover and Laguna (1997), we do not compute this value every iteration but rather only when we compute $C(f)$. This strategy is particularly useful when move value updates are computationally expensive, as in our context.

Specifically, after the vertices in $C(f)$ have been explored and their antibandwidths have eventually been increased, we examine all the vertices in the graph to first update the $AB_f(G)$ value and then compute the associated $C(f)$ set. It must be noted that when we modify the label of a vertex, the antibandwidth of its neighbors may change. This is why when we examine a vertex in $C(f)$, we first check whether its antibandwidth has increased above the $\beta \times AB_f(G)$ threshold as a result of the modification of the label of one of its neighbors previously selected in $C(f)$. In that case, we do not change its label and resort to the next vertex in $C(f)$.

As can be seen in the pseudo-code in Figure 4, the local search algorithm LS starts by constructing the set $C(f)$ of crucial vertices (step 5). It then selects at random a vertex u in $C(f)$ (step 7). In step 9, LS selects the best label for u and in step 10 it computes the set of candidate vertices $N'(u)$ (as in equation (13)) for swapping, with a label close to the best computed above. The procedure scans this neighborhood in the inner while loop (steps 12 to 19) in search for an improving move. The set $N'(u)$ is scanned from best to worst label. The first improving move $move(u, v)$ is made. If no improving move is found, no move is performed and we move on to the next vertex in $C(f)$. When all of the vertices in $C(f)$ have been examined, the objective function value of the new labeling f' is recomputed. If the solution improves, a new set $C(f')$ is computed and a new global iteration is performed. Otherwise, LS stops.

4. Path Relinking

Path relinking (PR) was suggested as an approach to integrate intensification and diversification strategies in the context of tabu search (Glover, 1996; Glover and Laguna, 1997). This approach generates new solutions by exploring trajectories that connect high-quality solutions – by starting from one of these solutions, called the *initiating solution*, and generating a path in the neighborhood space that leads toward the other solutions, called *guiding solutions*. This is accomplished by selecting moves that introduce attributes contained in the guiding solutions, and incorporating them in an *intermediate solution* initially originated in the initiating solution. In this section we explore different adaptations of PR to the antibandwidth problem.

Let X and Y be two solutions to the antibandwidth problem and let f_x and f_y be their associated representations (permutations). Given these two solutions,

```
begin LS
1   prevAB ← −∞;
2   currAB ← AB_f(G);
3   while currAB > prevAB do
4       prevAB ← currAB;
5       C(f) ← {v ∈ V | AB_f(v) ≤ β · currAB} with 1 ≤ β ≤ 2;
6       while C(f) ≠ ∅ do
7           Select u randomly in C(f);
8           C(f) ← C(f) \ {u};
9           Best_l(u) ← argmax_{1≤l≤n} min{|l − l_max(u)|, |l − l_min(u)|};
10          Compute N'(u) as in equation (13);
11          improve = FALSE ;
12          while improve = FALSE and N'(u) ≠ ∅ do
13              Select the best vertex v in N'(u);
14              N'(u) ← N'(u) \ {v};
15              if AB_f'(v) > AB_f(G) then
16                  Perform move(u, v);
17                  improve = TRUE;
18              end-if;
19          end-while;
20      end-while;
21      currAB ← UpdateAB() = AB_f'(G);
22  end-while;
end
```

FIGURE 4. Local search – *LS*.

the PR method operates over the set D of vertices that are not allocated in each permutation in the same position. D is thus considered the candidate list of vertices to be examined. To create a path from X to Y, PR selects at each step a vertex v from D and labels it with its label $f_y(v)$ in the guiding solution Y. To do this, we look in the initiating solution X for the vertex u with label $f_x(u) = f_y(v)$ and perform $move(u, v)$. Then the vertex v is removed from D and the next vertex is selected from D. PR performs a greedy selection in each step (i.e., PR evaluates all possible movements $move(u, v)$ for all $v \in D$ and performs the best one in terms of the objective function).

In this paper we explore the PR variant called *Mixed Path Relinking* (MPR) (Glover, 1996; Ribeiro and Rosseti, 2002). Instead of starting from a solution X and gradually transforming it into the solution Y by swapping elements in the permutation, this variant performs one step from X to Y, obtaining an intermediate solution A. Then Y becomes the initiating solution and A the guiding solution, obtaining a new intermediate solution B. In the next step of the procedure A becomes the initiating solution and B the guiding solution, obtaining C and so on. This logic is maintained until both paths joint in the middle. Figure 5 graphically shows how MPR builds this path. The main advantage of this strategy is that it explores deeply neighborhoods of both input solutions.

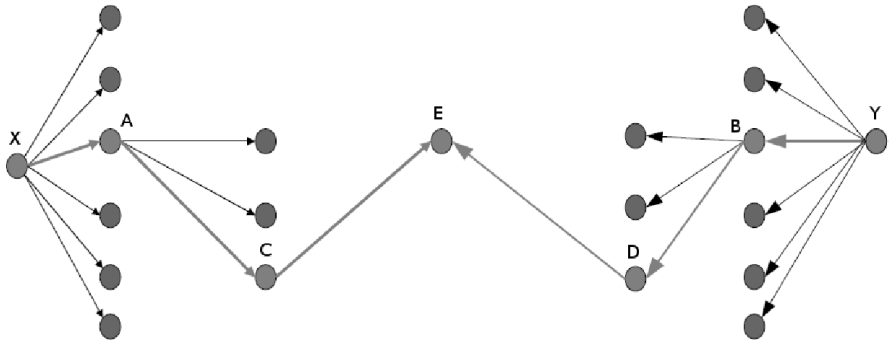

FIGURE 5. Mixed Path Relinking

The PR algorithm operates on a set of solutions, called *elite set*

$$ES = \{f^1, f^2, \ldots, f^b\},$$

ordered from best (f^1) to worst (f^b). It is constructed with the application of a previous method. In this paper, we apply GRASP to build the elite set. If we only consider a quality criterion to populate the elite set, we could simply populate it with the the best $|ES|$ solutions generated with GRASP. However, previous studies (Resende and Werneck, 2004) have empirically found that an application of PR to a pair of solutions is likely to be unsuccessful if the solutions are very similar. Therefore, to construct ES we will consider both quality and diversity including in ES a new solution f' when it improves the best solution in ES ($AB_{f'}(G) > AB_{f^1}(G)$) or it improves upon the worst in ES and its distance to ES is larger than a pre-established threshold dth ($AB_{f'}(G) > AB_{f^b}(G)$ and $d(f', ES) \geq dth$)

A distance function, d, is used to measure how diverse one solution is with respect to a set of solutions. Specifically, for the antibandwidth problem we consider the distance d between two permutations as:

$$d(f^i, f^j) = \sum_{k=1}^{n} |f^i(k) - f^j(k)|$$

and the distance between a permutation and the set ES as:

$$d(f, ES) = \min_{i \in ES} d(f, f^i).$$

The parameter dth is a distance threshold value that reflects the term "sufficiently different" and it is empirically adjusted (see Section 5). To keep the size of ES constant and equal to b, whenever we add a solution to this set, we remove another one. To maintain the quality and the diversity, we remove the closest solution to f' in ES among those worse than it in value.

The design in Figure 6 is called *static* since we first apply GRASP to construct the elite set ES (see steps 4 to 13) and then we apply PR to generate solutions between all the pairs of solutions in ES (see steps 15 to 21). During the realization of the GRASP phase we only replace a solution in the ES when the quality and diversity criteria are satisfied. Given two solutions in ES, f^i and f^j, we apply the mixed path relinking described above, obtaining f, the best solution found in the path (we represent it as $MPR(f^i, f^j) = f$ in step 16). We then apply the local

```
begin StaticGRASP+PR
1   GlobalIter ← number of global iterations;
2   Apply GRASP (construction and local search) for b = |ES| iterations
    to populate ES ← {f^1, f^2, ..., f^b} which are kept ordered from f^1 (best) to f^b (worst);
3   iter ← b + 1;
4   while iter ≤ GlobalIter do
5       f ← GRASP construction phase;
6       f' ← GRASP local search starting at f;
7       if AB_f'(G) > AB_f^1(G) or
                (AB_f'(G) > AB_f^b(G) and d(f', ES) ≥ d_th) then
8           f^k ← closest solution to f' in ES with AB_f'(G) > AB_f^k(G);
9           ES ← ES \ {f^k};
10          Insert f' into ES so that ES remains ordered;
11      end-if;
12      iter ← iter + 1;
13  end-while;
14  f^best ← f^1;
15  for (i = 1 to b − 1 and j = i + 1 to b do
16      Apply MPR(f^i, f^j) = f;
17      f' ← local search phase of GRASP starting from f;
18      if AB_f'(G) > AB_f^best(G) then
19          f^best ← f';
20      end-if;
21  end-for;
22  return f^best;
end
```

FIGURE 6. GRASP with PR in a static variant.

search method to f obtaining f' (step 17) and check whether the improved solution f' improves or not upon the best solution so far (steps 18 to 20). The algorithm terminates when PR is applied to all the pairs in ES and the best overall solution f^{best} is returned as the output.

An alternative implementation of GRASP with PR consists in a *dynamic* update of the elite set as introduced in Laguna and Martí (1999). Figure 7 shows the pseudo-code for this dynamic variant in which each solution f' generated with GRASP is directly subjected to the PR algorithm (see step 8), which is applied between f' and a solution f^j probabilistically selected from ES according to the values of solutions. In our computational experience, described on Section 5, we compare the static variant versus the dynamic variant with respect to both quality and speed.

Resende and Werneck (2004) introduced the *evolutionary path relinking* (evPR) as a post-processing phase for GRASP with PR (see also Andrade and Resende (2007) and Resende et al. (2010)). In evPR, the solutions in the elite set (ES) are evolved in a way that is similar to how the reference set evolves in scatter search (SS) (Laguna and Martí, 2003). We refer the reader to Resende et al. (2010) to see similarities and differences between both methods.

As in the dynamic variant of GRASP with path relinking, in evPR we apply in each iteration the construction and the improvement phase of GRASP as well as the PR method to obtain the elite set (see steps 5 to 9 in the pseudo-code shown in Figure 8). After a pre-established number of iterations the GRASP with dynamic path relinking stops. However, in evPR, a post-processing phase based on path

```
begin DynamicGRASP+PR
1   GlobalIter ← number of global iterations;
2   Apply GRASP (construction and local search) for b = |ES| iterations
    to populate ES ← {f^1, f^2, ..., f^b} which are kept ordered from f^1 (best) to f^b (worst);
3   iter ← b + 1;
4   while iter ≤ GlobalIter do
5       f ← GRASP construction phase;
6       f' ← GRASP local search starting at f;
7       Select f^j from ES according to its quality
            (randomly selected according to its quality);
8       Apply MPR(f', f^j) = f;
9       f' ← GRASP local search starting at f;
10      if AB_f'(G) > AB_f^1(G) or
                (AB_f'(G) > AB_f^b(G) and d(f', ES) ≥ d_th) then
11          f^k ← closest solution to f' in ES with AB_f'(G) > AB_f^k(G);
12          ES ← ES \ {f^k};
13          Insert f' into ES so that ES remains ordered;
14      end-if;
15  end-while;
16  f^best ← f^1;
17  return f^best;
end
```

FIGURE 7. GRASP with PR in a dynamic variant.

relinking is applied to each pair of solutions in *ES*. The solutions obtained with this latter application of PR are considered to be candidates to enter *ES*, and PR is again applied to them as long as new solutions enter *ES*. This way we say that *ES* evolves. Figure 8 shows the pseudo-code of GRASP with evPR in which this process is repeated for *GlobalIter* iterations.

In our computational experience, described on Section 5, we compare these variants of path relinking with respect to both quality and speed.

5. Computational results

This section describes the computational experiments performed to test the efficiency of the GRASP with path relinking heuristics. We implemented the methods in C and solved the integer linear programming formulation described in Section 2 with CPLEX 11.2, the most recent version of CPLEX when the experiments were carried out. All experiments with the heuristics were conducted on a Pentium 4 computer running at 3 GHz with 3 Gb of RAM while the CPLEX runs were done on a computer running 32 1.5 GHz Itanium 2 processors with 244 Gb of RAM. CPLEX used a single processor for each run. The GRASP implementations make use the Mersenne Twister (Matsumoto and Nishimura., 1998) for random number generation.

5.1. **Test problems.** We used three sets of test problems in our experiments. A total of 72 instances[1] were considered.

[1]These instances are available for download at `http://www.uv.es/rmarti`.

```
begin GRASP+evPR
1   GlobalIter ← number of global iterations;
2   Apply GRASP (construction and local search) for b = |ES| iterations
    to populate ES ← {f^1, f^2, ..., f^b};
3   iter1 ← b + 1;
4   while iter1 ≤ GlobalIter do
3       iter2 ← 1;
5       while iter2 ≤ LocalIter do
6           f ← GRASP construction phase;
7           f' ← GRASP local search starting at f;
8           Select f^j from ES according to its quality
                    (randomly selected according to its quality);
9           Apply MPR(f', f^j) = f;
10          f' ← GRASP local search starting at f;
11          if AB_f'(G) > AB_f^1(G) or
                    (AB_f'(G) > AB_f^b(G) and d(f', ES) ≥ d_th) then
12              f^k ← closest solution to f' in ES with AB_f'(G) > AB_f^k(G);
13          ES ← ES \ {f^k};
14          Insert f' into ES so that ES remains ordered;
15          end-if;
16      end-while;
17      NewSols ← True;
18      while NewSols do
19          NewSols ← False;
20          for (i = 1 to b − 1 and j = i + 1 to b do
21              Apply MPR(f^i, f^j) = f;
22              f' ← local search phase of GRASP starting from f;
23              if AB_f'(G) > AB_f^1(G) or
                        (AB_f'(G) > AB_f^b(G) and d(f', ES) ≥ d_th) then
24                  f^k ← closest solution to f' in ES with AB_f'(G) > AB_f^k(G);
25                  Add f' to ES and remove f^k;
26                  Sort ES from best f^1 to worst f^b;
27                  NewSols ← True;
28              end-if;
29          end-for;
30      end-while;
31  end-while;
32  f^best ← f^1;
33  return f^best;
end
```

FIGURE 8. Evolutionary Path Relinking.

5.1.1. *Harwell-Boeing.* We derived 24 instances from the Harwell-Boeing Sparse Matrix Collection[2]. 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 $M_{ij} \neq 0$. We considered two subsets in the Harwell-Boeing set.

[2] `http://math.nist.gov/MatrixMarket/data/Harwell-Boeing`

The first consists of 12 smaller instances of size $n \in [30, 100]$: `bcspwr01`, `bcspwr02`, `ibm32`, `pores1`, `curtis54`, `will57`, `bcsstk01`, `dwt234`, `ash85`, `bcspwr03`, `impcol.b`, and `nos4`. The second subset is made up of 12 larger instances with $n \in [400, 900]$: `494bus`, `662bus`, `685bus`, `bcsstk07`, `bcsstk06`, `can445`, `can715`, `dwt503`, `dwt592`, `impcol.d`, `nos6`, and `sherman4`.

5.1.2. *Grid graphs.* This data set consists of 24 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 solutions of the antibandwidth problem for these types of instances are known by construction. As in the previous set, we considered two subsets, the first with 12 smaller instances ($n \in [81, 120]$): `mesh9x9`, `mesh50x2`, `mesh34x3`, `mesh25x4`, `mesh20x5`, `mesh10x10`, `mesh17x6`, `mesh13x8`, `mesh15x7`, `mesh12x9`, `mesh11x11`, and `mesh12x12`, and the second with 12 larger instances ($n \in [960, 1170]$): `mesh-130x7`, `mesh120x8`, `mesh110x9`, `mesh100x10`, `mesh50x20`, `mesh40x25`, `mesh60x17`, `mesh34x30`, `mesh80x13`, `mesh70x15`, `mesh90x12`, and `mesh33x33`.

5.1.3. *Hamming graphs.* This data set consists of 24 graphs constructed as the Cartesian product of d complete graphs K_{n_k}, for $k = 1, 2, \ldots, d$ (Dobrev et al., 2009). The vertices in these graphs are d-tuples $(i_1, i_2, \ldots, i_d)$, where $i_k \in \{0, 1, \ldots, n_k - 1\}$. Two vertices $(i_1, i_2, \ldots, i_d)$ and $(j_1, j_2, \ldots, j_d)$ are adjacent if and only if the two tuples differ in exactly one coordinate. These graphs as referred to as Hamming graphs. It is shown in Dobrev et al. (2009) that if $d \geq 2$ and $2 \leq n_1 \leq n_1 \leq \cdots \leq n_d$, then the optimal solution of the antibandwidth problem for this type of instance is given by

$$AB(\prod_{k=1}^{d} K_{n_k}) = \begin{cases} n_1 n_2 \cdots n_{d-1} & \text{if } n_{d-1} \neq n_d \\ n_1 n_2 \cdots n_{d-1} - 1 & \text{if } n_{d-1} = n_d \text{ and } n_{d-2} \neq n_{d-1} \text{ and } d \geq 3. \end{cases}$$

As in the previous sets, we considered two subsets, the first with 12 smaller instances ($n \in [80, 180]$): `hamming4x4x5`, `hamming3x5x6`, `hamming4x5x5`, `hamming3x5x7`, `hamming4x5x6`, `hamming3x5x8`, `hamming4x4x8`, `hamming4x5x7`, `hamming5x5x6`, `hamming4x5x8`, `hamming5x5x7`, and `hamming5x6x6`, and the second with 12 larger instances ($n \in [840, 1152]$): `hamming3x3x4x4x6`, `hamming2x3x4x5x7`, `hamming3x3x4x-5x5`, `hamming2x3x4x6x6`, `hamming2x2x5x6x7`, `hamming3x4x4x4x5`, `hamming3x3x4x-4x7`, `hamming2x3x4x5x8`, `hamming3x3x4x5x6`, `hamming2x3x3x7x8`, `hamming2x3x5x-6x6`, and `hamming3x4x4x4x6`.

5.2. **Experiments.** Four main experiments were carried out. We first compute the upper bounds UB_1 and UB_2 as defined by Yixun and Jinjiang (2003) in (1) and (2), respectively. Then, we attempt to solve the instances using the integer linear programming formulation, given in Section 2, with the commercial solver CPLEX 11.2. In the third part of the experiment, we determine which construction procedure, local search parameter values, and path-relinking strategy to use in the full experiment with the heuristics. Finally, we compare GRASP with path relinking and GRASP with evolutionary path relinking.

5.2.1. *Upper bounds.* In this experiment, we evaluate the quality of the two bounds (UB_1 and UB_2) introduced in Section 1. Table 1 reports, for each subset of instances, the average values of both bounds as well as LB, the average values of the best known, or optimal, solutions. For the case of the grid graph and Hamming

TABLE 1. Average upper bounds (UB_1 and UB_2) and best known solution values (LB) for each subset of test problems.

	UB_1	UB_2	LB
12 Smaller Grid graphs	50.42	83.75	48.00
12 Larger Grid graphs	505.66	951.17	498.58
12 Smaller Harwell-Boeing	32.08	48.75	19.75
12 Larger Harwell-Boeing	273.67	487.75	140.17
12 Smaller Hamming graphs	58.58	89.17	19.58
12 Larger Hamming graphs	473.67	838.52	153.33

graph instances the values of LB are optimal (known by construction), while for the Harwell-Boeing instances they are the best solutions found by our heuristics. With the exception of two of the smaller Harwell-Boeing instances solved by CPLEX, these values are not necessarily optimal. Upper bounds are shown in detail in Tables 7, 8, and 9.

Table 1 clearly shows that both bounds for the grid instances are not tight while for the Harwell-Boeing and Hamming graph instances they are far from the best known or optimal values. Although bounds UB_1 are more accurate than UB_2, they are still far from their corresponding LB values. Note, however, that in the case of the Harwell-Boeing instances this does not necessarily mean that the bounds are weak. For these instances, the LB values are mostly produced by our heuristics and it may be the case that the heuristics are finding weak lower bounds. The only indication we have that the upper bounds are indeed weak is the fact that on the two small Harwell-Boeing instances (`bcspwr01` and `ibm32`) for which CPLEX proved optimality, both bounds were weak (for `bcspwr01`, $UB_1 = 19$ and $UB_2 = 29$ while the optimum is 17; for `ibm32`, $UB_1 = 15$ and $UB_2 = 19$ while the optimum is 9). For the two classes for which optimal solutions are known, bounds UB_1 are relatively tight for the grid graph instances while they are relatively weak for the Hamming graph instances. This suggests that the Hamming graph instances are the most difficult in our testbed. In the experiments to follow, we compare the solutions of our heuristic methods with LB as well as several upper bounds.

5.2.2. *Integer programming solutions.* In this experiment, we investigate the integer linear programming formulation proposed in Section 2. We solve the integer programs with CPLEX 11.2. Tables 2 and 3 report, for each instance in the Harwell-Boeing and grid graph sets, respectively, the number of rows (n), columns (m), and nonzeroes (nz) of the integer program, the number of iterations (itrs $\times 10^6$), the number of nodes in the branch and bound tree (B&B nodes), the CPU time in seconds (time), the value of the best solution found (value), and the best known upper bound (UB). The upper bounds produced by CPLEX are listed when they are better than both UB_1 and UB_2. We limit each CPLEX run to at most 24 hours of CPU time.

TABLE 2. CPLEX 11.2 integer programming solver on *Harwell-Boeing* instances.

	dimension of integer program			itrs				
Problem	n	m	nz	(10^6)	B&B nodes	time	value	UB
bcspwr01	209	1607	4931	10.4	268428	1987.67s	**17**	17
bcspwr02	265	2510	7675	425.3	8830998	$*$	21	22
ibm32	276	1147	3792	27.5	549004	5701.56s	**9**	9
pores1	296	1034	3524	350.1	16760248	$*$	6	8
curtis54	410	3095	9740	215.0	6867978	$*$	10	13
will57	425	3434	10763	214.6	3984994	$*$	12	14
bcsstk01	496	2529	8320	218.2	4906025	$*$	6	11
dwt234	675	13969	42363	88.9	1674757	$*$	22	58
ash85	693	7530	23427	115.1	3816727	$*$	12	27
bcspwr03	712	14222	43204	73.8	1628719	$*$	22	57
impcol.b	739	3822	12691	143.1	3035712	$*$	4	14
nos4	794	10348	31976	88.5	1553986	$*$	10	47
494bus	2654	245117	736796	4.5	949	$*$	12	247
662bus	3798	439813	1321980	1.3	408	$*$	16	331
685bus	4619	471193	1417931	1.5	10	$*$	3	342
bcsstk06	8700	180541	558960	6.0	406	$*$	1	210
bcsstk07	8700	180541	558960	5.8	401	$*$	1	210
can445	4699	200153	607531	3.3	321	$*$	1	221
can715	8095	514916	1557475	1.9	16	$*$	1	357
dwt503	7033	256275	781123	2.4	103	$*$	1	250
dwt592	6288	353313	1069440	3.4	84	$*$	2	295
impcol.d	3809	182318	552011	4.6	466	$*$	2	212
nos6	4605	457591	1377195	2.4	48	$*$	4	337
sherman4	4320	300004	905076	3.2	107	$*$	5	272

$*$ – stopped after 24 hours of CPU time

Tables 2 and 3 show that CPLEX 11.2 only solves two small instances (highlighted in Table 2) out of 24 in the Harwell-Boeing set in the 24 hour limit and it solves none of the grid graph instances. In the rest of the instances shown in Tables 2 and 3, we can observe a gap between the value of the best solution found and the best computed upper bound. For example, for instance bcspwr03, the best solution found has a value of 22 while the best upper bound is 57. Note also that on the smaller instances, the dimensions of the integer programs were small enough to permit CPLEX to explore a large number of branches of the branch and bound tree while this was not the case for the larger instances. Consequently, as can be observed in Tables 7 and 8, CPLEX found very poor quality lower bounds on the larger instances, often finding unit-valued solutions, i.e. the worst possible solutions for an antibandwidth problem. Since CPLEX performed poorly on the grid graph instances, we chose not to run CPLEX on the more difficult Hamming graph instances.

5.2.3. *Components and parameter tuning.* A series of preliminary experiments were done to set the values of the key search parameters of our heuristic methods. In each experiment, we compute the following statistics for each method: the average of the objective function value of the best solutions found (*Value*), the average percentage deviation (*Dev.*) between the solutions found and the best known value (LB), the number *#Best* of instances in which the value of the best solution obtained matches the best known solution, and the average CPU time in seconds (*Time*).

TABLE 3. CPLEX 11.2 integer programming solver on *grid graph* instances.

	dimension of integer program			itrs				
Problem	n	m	nz	(10^6)	B&B nodes	time	value	UB
mesh9x9	531	6787	20835	140.	1686369	$*$	30	36
mesh50x2	596	10249	31184	113.1	1822105	$*$	23	49
mesh34x3	640	10674	32548	100.0	1195115	$*$	37	50
mesh25x4	642	10272	31368	104.4	1008569	$*$	38	48
mesh20x5	650	10276	31400	100.7	1093909	$*$	35	48
mesh10x10	660	10281	31440	105.5	1194209	$*$	24	45
mesh17x6	668	10688	32660	92.3	1278383	$*$	23	48
mesh13x8	686	11108	33944	106.7	1113069	$*$	39	48
mesh15x7	691	11319	34579	97.0	1139144	$*$	27	49
mesh12x9	714	11968	36552	96.0	786289	$*$	41	50
mesh11x11	803	14983	45683	76.6	762275	$*$	26	55
mesh12x12	960	21145	64320	57.2	547845	$*$	23	106
mesh130x7	6096	830694	2497764	0.3	0	$*$	1	452
mesh120x8	6464	924353	2779136	0.8	0	$*$	1	476
mesh110x9	6692	982952	2955188	0.6	0	$*$	2	491
mesh100x10	6780	1002891	3015120	0.6	0	$*$	2	495
mesh50x20	6860	1002931	3015440	0.7	0	$*$	2	490
mesh40x25	6870	1002936	3015480	0.4	0	$*$	1	488
mesh60x17	6986	1043384	3136904	0.5	0	$*$	1	502
mesh34x30	7012	1043397	3137008	0.6	0	$*$	1	495
mesh80x13	7094	1084628	3260696	0.4	0	$*$	1	514
mesh70x15	7180	1105566	3323620	0.3	0	$*$	1	518
mesh90x12	7356	1169539	3515664	0.3	0	$*$	1	534
mesh33x33	7491	1189123	3574659	0.3	0	$*$	1	528

$*$ – stopped after 24 hours of CPU time

In the first preliminary experiment we study the labeling strategy for the first vertex in construction procedures *C1* and *C2* described in Subsection 3.1. Specifically, we compare three variants of each procedure labeled as "1", "random," and "$n/2$," indicating that the first vertex selected in each construction is labeled with 1, a random number, or $n/2$, respectively. Table 4 reports, for the set of 48 instances and each construction variant, the values of *Value*, *Dev.*, *#Best*, and *Time*.

The results in Table 4 show that the best outcomes are obtained when the construction method *C2* is used along with the "$n/2$" strategy. Although the

TABLE 4. Comparison of construction methods *C1* and *C2*.

	C1			*C2*		
	random	1	$n/2$	random	1	$n/2$
Value	19.12	19.30	25.35	21.06	20.31	27.93
Dev.	66.55%	66.76%	63.52%	64.96%	65.31%	62.34%
#Best	8	8	23	16	14	30
Time	11.4s	11.3s	11.4s	51.7s	53.6s	50.3s

TABLE 5. Effect of parameter β on construction with local search.

β	1.5	1.4	1.3	1.2	1.1
Value	170.52	169.73	169.15	168.29	167.63
Dev.	4.05%	5.48%	5.51%	5.99%	6.55%
#Best	32	31	28	22	20
Time	196.92s	184.08s	161.02s	141.76s	97.58s

running times employed by *C2* are longer than those of *C1*, they are still moderate (less than one minute). Therefore, we use this method in the remaining experiments.

In the second preliminary experiment, we study the effect of local search parameter β. We run construction method *C2* in combination with the local search method with different values of β. Table 5 reports, for the set of 48 instances and each value of β tested, the values of *Value*, *Dev.*, *#Best*, and *Time*.

Results in Table 5 clearly show that much better outcomes are obtained when the construction method is coupled with the local search. If we compare the average deviation with respect to the best known solution (*Dev.*), we observe a significant reduction from 62.34% (see Table 4) to 5.48% (see Table 5). Furthermore, Table 5 also shows that the quality of the solution obtained with the local search method improves as the value of the parameter β increases. As expected, CPU time also increases. As a compromise, we select $\beta = 1.4$ since it presents a good trade-off point between quality and CPU time. Therefore, the GRASP heuristic is formed with *C2* as the construction method and the local search with $\beta = 1.4$.

In the third preliminary experiment, we compare the two variants of the path relinking algorithm described in Section 4. We consider both the static version, in which PR is applied after GRASP (pseudo-code shown in Figure 6), and the dynamic version, in which PR is executed within each iteration of GRASP (pseudo-code shown in Figure 7). The path relinking method depends on the parameter *dth* that specifies the minimum difference with respect to the elite solutions required for a solution (that is not better than the best elite solution) to be accepted as a member of the elite set. We compute this value as a fraction of the maximum distance

$$d_{max} = \sum_{i=1}^{n} |i - (n - i)| = \sum_{i=1}^{n} |2i - n|$$

between two solutions (permutations).

Table 6 reports, for the static and dynamic variants of PR and three different values of *dth*, the average objective function value (*Value*), the average percentage deviation from the best solution obtained overall (*Dev.*), the number of best solutions in this experiment (*#Best*) and the average CPU time (*Time*) in seconds.

Table 6 clearly shows that path relinking with its dynamic scheme obtains better solutions than the static variant, although it uses more CPU time (on average about 431 seconds compared with the 306 seconds for the static version). Moreover, this table also shows that the best value of *dth* in the dynamic version is $5d_{max}/1000$, since the method obtains an average percentage deviation of 4.15% and 42 best solutions (out of the 48 instances), which compares favorably with the other values

TABLE 6. Comparison of static and dynamic path relinking variants.

	Static PR			Dynamic PR		
dth	$\frac{d_{max}}{1000}$	$\frac{5d_{max}}{1000}$	$\frac{d_{max}}{100}$	$\frac{d_{max}}{1000}$	$\frac{5d_{max}}{1000}$	$\frac{d_{max}}{100}$
Value	169.33	169.73	169.73	170.56	170.88	170.56
Dev.	5.29%	5.48%	5.48%	4.35%	4.15%	4.35%
#Best	16	20	20	36	42	36
Time	302.83s	306.24s	308.80s	425.11s	441.30s	427.70s

TABLE 7. Comparison of lower bounds and upper bounds on Harwell-Boeing instances. Boldface indicates best (lower and upper) bound values.

	Lower bounds (solutions)			Upper bounds		
Problem	G+evPR	G+PR	CPLEX	UB_1	UB_2	CPLEX
bcspwr01	**17**	**17**	**17**	19	29	**17**
bcspwr02	**21**	**21**	**21**	24	38	**22**
ibm32	**9**	**9**	**9**	15	19	**9**
pores1	**6**	**6**	**6**	13	16	**8**
curtis54	**12**	**12**	10	26	38	**13**
will57	**13**	**13**	12	28	41	**14**
bcsstk01	**8**	**8**	6	22	29	**11**
dwt234	**51**	**51**	22	**58**	99	**58**
ash85	**22**	21	12	42	64	**27**
bcspwr03	**39**	**39**	22	59	99	**57**
impcol.b	**8**	**8**	4	29	35	**14**
nos4	**35**	34	10	50	78	**47**
494bus	**228**	227	12	**247**	460	410
662bus	**220**	**220**	16	**331**	619	660
685bus	**136**	**136**	3	**342**	634	680
bcsstk06	**33**	32	1	**210**	334	372
bcsstk07	**33**	32	1	**210**	334	372
can445	**85**	82	1	**221**	387	439
can715	**121**	115	1	**357**	638	709
dwt503	**58**	53	1	**250**	429	497
dwt592	**112**	108	2	**295**	525	554
impcol.d	**105**	104	2	**212**	375	381
nos6	**328**	326	4	**337**	624	674
sherman4	**261**	**261**	5	**272**	494	544

shown. We therefore set the value of *dth* to $5d_{max}/1000$ in our final path relinking algorithm and restrict our attention to the dynamic variant.

5.2.4. *Comparing GRASP heuristics.* We ran both GRASP with path relinking heuristics 30 times. GRASP with path relinking was run for 250 global iterations while GRASP with evolutionary path relinking was run for four global iterations and 25 local iterations during each global iterations. Our goal was to try to make both heuristics have about the same running time. CPLEX was run for a maximum of 24 hours of CPU.

TABLE 8. Comparison of lower bounds and upper bounds on grid graph instances. Boldface indicates best solution value.

	Lower bounds (solutions)			Upper bounds			
Problem	G+evPR	G+PR	CPLEX	UB_1	UB_2	CPLEX	OPT
mesh9x9	**36**	**36**	30	40	64	54	**36**
mesh50x2	**48**	**48**	23	**49**	83	71	**49**
mesh34x3	**48**	**48**	37	**50**	84	61	**50**
mesh25x4	**46**	**46**	38	49	82	70	**48**
mesh20x5	**46**	**46**	35	49	81	68	**48**
mesh10x10	**45**	**45**	24	49	81	64	**45**
mesh17x6	**47**	46	23	50	83	68	**48**
mesh13x8	**47**	46	39	51	85	69	**48**
mesh15x7	**48**	47	27	52	86	67	**49**
mesh12x9	**49**	48	41	53	88	74	**50**
mesh11x11	**55**	**55**	26	60	100	79	**55**
mesh12x12	**66**	**66**	23	71	212	106	**66**
mesh130x7	**442**	441	1	454	852	909	**452**
mesh120x8	**468**	467	1	479	900	958	**476**
mesh110x9	**482**	481	2	494	929	998	**491**
mesh100x10	**485**	**485**	2	499	939	998	**495**
mesh50x20	**474**	473	2	499	938	998	**490**
mesh40x25	**480**	477	1	499	938	998	**488**
mesh60x17	**481**	**481**	1	509	957	1019	**502**
mesh34x30	**491**	490	1	509	957	1018	**495**
mesh80x13	**500**	**500**	1	519	977	1039	**514**
mesh70x15	**502**	501	1	524	987	1049	**518**
mesh90x12	**522**	521	1	539	1016	1079	**534**
mesh33x33	**524**	523	1	544	1024	1088	**528**

Tables 7, 8, and 9, compare the lower and upper bounds on the Harwell-Boeing, grid graph, and Hamming graph instances, respectively. While the lower bounds correspond to the results obtained with the GRASP heuristics and CPLEX, the upper bounds are the UB_1, UB_2 bounds, and the bounds found by the CPLEX solver. For the grid graph and Hamming graph instances, the optimal solutions (known by construction) are also included. In the case of the Hamming graph instances, the CPLEX solver was not run since these instances are the most difficult in our set of benchmark instances. The first two tables show that the GRASP with evolutionary path relinking found the best or optimal solutions for all instances. Although the GRASP with path relinking reached almost all the best or optimal solutions on the smaller Harwell-Boeing and grid graph instances, on the larger problems it only found the best solution on two of the Harwell-Boeing instances (`662bus` and `685bus`) and the optimal on three of the grid graph instances (`mesh100x10`, `mesh60x17`, and `mesh80x13`).

CPLEX is far from the best known lower and upper bounds for the larger Harwell-Boeing and grid graph instances. It is also not tight for the best known lower and upper bounds for the smaller instances, except for four small Harwell-Boeing instances (`bcspwr01`, `bcspwr02`, `ibm32`, and `pores2`).

Tables 10, 11, and 12 compare the running times for the GRASP heuristics on the Harwell-Boeing, grid graph, and Hamming graph instances, respectively. Each table lists the instance name, and for each GRASP heuristic, the minimum, average, and maximum running times computed over the 30 runs. Even though the

TABLE 9. Comparison of lower bounds and upper bounds on Hamming graph instances. Boldface indicates best solution value.

	Lower bounds (solutions)		Upper bounds		
Problem	G+evPR	G+PR	UB_1	UB_2	OPT
hamming4x4x5	**12**	11	35	52	**16**
hamming3x5x6	10	**11**	40	59	**15**
hamming4x5x5	12	**13**	45	67	**19**
hamming3x5x7	**11**	**11**	47	70	**15**
hamming4x5x6	**14**	**14**	54	82	**20**
hamming3x5x8	11	**12**	54	81	**15**
hamming4x4x8	12	**13**	58	87	**16**
hamming4x5x7	14	**15**	64	97	**20**
hamming5x5x6	**16**	**16**	69	106	**25**
hamming4x5x8	**15**	**15**	73	113	**20**
hamming5x5x7	**17**	**17**	81	126	**25**
hamming5x6x6	18	**19**	83	130	**29**
hamming3x3x4x4x6	**94**	**94**	425	750	**144**
hamming2x3x4x5x7	**80**	**80**	412	724	**120**
hamming3x3x4x5x5	**99**	**99**	443	784	**179**
hamming2x3x4x6x6	**88**	**88**	424	746	**143**
hamming2x2x5x6x7	**84**	75	412	721	**120**
hamming3x4x4x4x5	**111**	109	473	840	**192**
hamming3x3x4x4x7	**105**	100	496	881	**144**
hamming2x3x4x5x8	**93**	84	472	832	**120**
hamming3x3x4x5x6	**123**	107	532	949	**180**
hamming2x3x3x7x8	**92**	81	495	873	**126**
hamming2x3x5x6x6	**114**	99	532	945	**180**
hamming3x4x4x4x6	**123**	119	568	1016	**192**

number of iterations for both GRASP heuristics were the same on both classes of problems, the relative running times of the GRASP heuristics varied greatly. For the Harwell-Boeing instances, the ratios of the GRASP with evolutionary path relinking average running time to that of the GRASP with path relinking varied from 3.50 (for bcspwr01) to 6.35 (for can445), while for the grid graph instances this ratio varied from 1.13 (for mesh12x12) to 2.66 (for mesh70x15). This is because the number of pairs combined in the evolutionary path relinking stage was greater for the Harwell-Boeing instances than for the grid graph instances. For the Hamming graph instances, the ratios of the GRASP with evolutionary path relinking average running time to that of the GRASP with path relinking varied from 0.7 (for hamming3x5x7) to 1.42 (for hamming2x3x4x6x6).

Table 13 reports the results obtained with the two methods, GRASP+PR and GRASP+evPR, on the set of 72 instances. We split this table into six main rows corresponding to the six subsets of instances according to their type and size.

The GRASP with evolutionary path relinking consistently produces the best solutions with percent deviations smaller than those of the competing methods (and with number of best solutions found larger than the others). GRASP+evPR presents a marginal improvement when compared with GRASP with path relinking but requires longer running times (especially for large instances).

TABLE 10. Comparison of total running times (in seconds) for GRASP with path relinking and GRASP with evolutionary path relinking on Harwell-Boeing instances.

	GRASP+PR			GRASP+evPR		
Problem	min	avg	max	min	avg	max
bcspwr01	0.17	0.18	0.18	0.61	0.63	0.70
bcspwr02	0.29	0.30	0.30	0.98	1.06	1.18
ibm32	0.24	0.25	0.25	0.84	0.93	1.07
pores1	0.24	0.25	0.25	0.85	0.98	1.12
curtis54	0.56	0.58	0.59	2.02	2.17	2.48
will57	0.63	0.64	0.65	2.25	2.37	2.69
bcsstk01	0.71	0.72	0.73	2.62	2.91	3.26
dwt234	1.97	2.02	2.07	7.73	8.43	9.82
ash85	1.72	1.74	1.77	6.54	7.47	8.79
bcspwr03	2.16	2.22	2.26	8.30	8.62	8.89
impcol.b	1.47	1.51	1.54	5.43	6.10	7.03
nos4	2.39	2.45	2.50	9.13	10.16	11.62
494bus	66.18	68.42	71.45	333.89	375.61	434.39
662bus	114.52	116.14	118.09	610.17	662.06	736.84
685bus	120.56	121.04	121.68	591.44	594.70	597.75
bcsstk06	212.54	213.91	215.45	1018.4	1184.72	1380.83
bcsstk07	212.75	213.86	215.66	985.24	1160.86	1336.31
can445	102.32	104.53	106.61	527.95	663.90	1005.81
can715	371.99	377.51	381.38	1867.53	2259.50	2810.00
dwt503	189.01	190.31	191.04	888.29	1045.85	1384.71
dwt592	204.61	208.68	211.47	1047.53	1321.90	1808.50
impcol.d	73.64	74.72	76.16	337.57	409.19	489.10
nos6	369.58	380.45	389.98	2145.00	2303.03	2803.00
sherman4	293.35	302.22	312.28	1495.62	1568.87	1672.52

6. SUMMARY AND CONCLUDING REMARKS

In this paper we propose an integer programming formulation and several heuristics based on GRASP and path relinking for the antibandwidth problem. The proposed GRASP heuristics consist of two randomized greedy construction procedures and a parametrized local search procedure. A mixed path relinking intensification algorithm was also proposed and tested in two path relinking schemes. In the static scheme, path relinking is done once between all pairs of elite set solutions after the last GRASP iteration. In the dynamic scheme, after each GRASP local search phase, path relinking is done between the local maximum and a solution selected at random from the pool. We also proposed a hybrid GRASP with evolutionary path relinking heuristic which periodically carries out path relinking between all pairs of solutions in the elite set.

The experimental part of the paper consists of four parts. The benchmark test set was made up of 72 instances, 24 derived from the Harwell-Boeing matrix collection, 24 on grid, or two dimensional mesh, graphs, and 24 on Hamming graphs. Each set of 24 instances had 12 smaller and 12 larger instances. The grid graph and Hamming graph instances have optimal solution values known by construction.

In the first part of the experiments, both upper bounds proposed by Yixun and Jinjiang (2003) were computed for all instances. Bound UB_1 was clearly better than bound UB_2, in spite of the fact that it was only tight for two of the 72 instances.

TABLE 11. Comparison of total running times (in seconds) for GRASP with path relinking and GRASP with evolutionary path relinking on grid graph instances.

	GRASP+PR			GRASP+evPR		
Problem	min	avg	max	min	avg	max
mesh9x9	1.46	1.48	1.50	2.35	2.54	2.86
mesh50x2	2.12	2.15	2.21	3.46	3.85	4.46
mesh34x3	2.45	2.52	2.67	4.04	4.62	5.27
mesh25x4	2.40	2.43	2.48	3.95	4.44	5.08
mesh20x5	2.39	2.45	2.50	3.95	4.39	4.79
mesh10x10	1.48	2.44	2.48	2.33	4.33	5.32
mesh17x6	2.46	2.51	2.58	4.02	4.66	5.68
mesh13x8	2.59	2.63	2.67	4.06	4.55	5.37
mesh15x7	2.60	2.76	2.84	4.31	5.01	5.81
mesh12x9	2.40	2.90	2.96	4.27	5.08	5.71
mesh11x11	3.90	3.98	4.04	6.16	6.95	8.31
mesh12x12	3.94	9.40	9.99	7.34	10.66	13.85
mesh130x7	777.19	806.29	830.17	1655.74	1816.21	2011.33
mesh120x8	878.53	917.44	942.75	1692.03	1968.86	2195
mesh110x9	941.84	976.41	1008.18	1825.23	2146.27	2549.00
mesh100x10	958.01	991.9	1041.29	1785.98	2227.91	2687.00
mesh50x20	923.79	947.07	968.64	1576.15	1809.99	2163.00
mesh40x25	948.02	975.17	1006.61	1651.47	1866.68	2129.00
mesh60x17	936.38	977.75	1017.15	1606.77	2030.33	2586.00
mesh34x30	1002.99	1040.2	1070.88	1653.04	1751.98	1990.57
mesh80x13	1012.27	1049.39	1084.07	2275.00	2609.57	3237.00
mesh70x15	1027.08	1061.48	1093.15	2276.00	2826.23	3418.00
mesh90x12	1130.42	1180.82	1228.47	2443.00	2829.10	3398.00
mesh33x33	1339.43	1398.09	1435.88	8488.00	1968.86	9740.00

In the second part of the experiments, we attempted to solve each of the 48 Harwell-Boeing and grid graph instances with the CPLEX 11.2 integer programming solver. Each run was limited to at most 24 hours of CPU time. CPLEX was only able to provably solve two of these 48 instances (two smaller Harwell-Boeing problems). The upper bounds produced by CPLEX were better than those of UB_1 in the smaller Harwell-Boeing instances but worse than UB_1 for the smaller grid graph instances. However, for the 24 larger instances, the CPLEX bounds were worse than both UB_1 and UB_2, with the exception of one instance.

In the third part of the experiments, we determined the GRASP and path relinking strategies used for testing the GRASP with path relinking heuristics. The chosen construction scheme was the sampled greedy heuristic which sets the initial label at random to $n/2$. The selected path relinking procedure uses the dynamic scheme. Local search is used since, on average, it improved the constructed solutions.

In addition to CPLEX, we compared a GRASP with dynamic path relinking and a GRASP with dynamic and evolutionary path relinking.

In addition to optimally solving the two smaller Harwell-Boeing instances, CPLEX found near-optimal solutions for about half of the smaller Harwell-Boeing instances. However, CPLEX found poor-quality solutions on all larger Harwell-Boeing and grid graph instances, including many unit-valued solutions. This is probably due to the fact that for all larger Harwell-Boeing instances CPLEX explores few branch

TABLE 12. Comparison of total running times (in seconds) for GRASP with path relinking and GRASP with evolutionary path relinking on Hamming graph instances.

	GRASP+PR			GRASP+evPR		
Problem	min	avg	max	min	avg	max
hamming4x4x5	1.41	1.43	1.46	0.75	1.04	1.52
hamming3x5x6	2.06	2.08	2.11	1.05	1.63	3.22
hamming4x5x5	2.59	2.62	2.65	1.48	2.06	3.17
hamming3x5x7	3.30	3.32	3.36	1.47	2.33	3.21
hamming4x5x6	4.44	4.47	4.50	2.63	3.89	5.51
hamming3x5x8	4.88	4.91	4.95	2.82	4.11	6.03
hamming4x4x8	5.63	5.67	5.71	3.42	5.14	7.31
hamming4x5x7	6.86	6.92	6.98	4.01	5.68	7.65
hamming5x5x6	8.00	8.10	8.22	5.10	7.56	12.54
hamming4x5x8	9.86	9.93	9.99	5.81	8.75	12.92
hamming5x5x7	12.07	12.17	12.29	7.09	10.99	15.18
hamming5x6x6	12.82	12.95	13.04	8.33	12.14	18.06
hamming3x3x4x4x6	548.71	555.02	563.48	439.62	807.80	1582.69
hamming2x3x4x5x7	536.06	541.27	546.84	451.72	885.33	1633.96
hamming3x3x4x5x5	614.07	620.01	627.68	548.20	926.41	1742.80
hamming2x3x4x6x6	582.40	587.52	594.27	563.51	840.88	1279.91
hamming2x2x5x6x7	570.82	577.61	583.26	621.18	930.11	1706.58
hamming3x4x4x4x5	722.97	732.04	740.08	678.33	1148.48	1984.44
hamming3x3x4x4x7	836.80	844.88	853.38	801.30	1335.64	2264.00
hamming2x3x4x5x8	774.17	782.87	791.73	830.46	1236.72	1856.83
hamming3x3x4x5x6	1016.33	1027.73	1038.44	1015.8	1678.05	2673.00
hamming2x3x3x7x8	922.79	933.88	939.92	844.04	1371.09	2278.00
hamming2x3x5x6x6	1070.24	1082.27	1097.83	997.86	1656.37	3499.00
hamming3x4x4x4x6	1202.53	1212.72	1227.94	1303.71	2158.28	4097.00

and bound nodes, while for all larger grid graph instances it did not go beyond the root node of the branch and bound tree. CPLEX found no optimal solution for the instances in the set of grid graph problems.

GRASP with evolutionary path relinking found solutions that were as good or better than those found by GRASP with path relinking, but at the expense of longer running times. GRASP with path relinking found the best known solutions for almost all of the smaller Harwell-Boeing and grid graph instances. In five of the 24 larger instances, GRASP with path relinking produced the best found solution. It found provably optimal solutions for two of the smaller Harwell-Boeing test set and three of the smaller grid graph instances. GRASP with evolutionary path relinking found all best known solutions in the Harwell-Boeing test set and all best found solutions in the grid graph test set. In the Hamming graph test set, GRASP with evolutionary path relinking consistently found the best solutions on the 12 larger instances. However, on half of the smaller Hamming graph instances, it did not find the best solution. GRASP with evolutionary path relinking found better solutions than GRASP with path relinking in 30 of the 72 instances and solutions that were of equal quality in 36 instances. It found provably optimal solutions for five of the 48 Harwell-Boeing and grid graph instances, being two in the set of smaller Harwell-Boeing instances and three in the set of smaller grid graph instances. However, running times for GRASP with evolutionary path relinking

TABLE 13. Comparison of heuristics methods. Each instance is run 30 times by each GRASP heuristic. For each instance the minimum, average, and maximum solution values, percent deviation from best solution value, and running times are computed. These values are averaged over the 12 instances in each of the four problem groups and are shown in the table. For each of the four problem groups and for each of the two GRASP heuristics, the table also shows the minimum and the maximum number of best or optimal solutions that the 30 runs found, as well as %best, the total number of runs that found the best or optimal solution for problems in the group divided by the total number of runs (12 instances $\times$ 30 runs).

		GRASP+PR			GRASP+evPR		
		min	avg (%best)	max	min	avg (%best)	max
12 Smaller grid graph	**Value**	45.4	46.1	46.6	45.7	46.3	46.9
	Dev.	2.9%	3.8%	5.9%	2.2%	3.4%	4.8%
	Time	2.4s	2.6s	2.7s	4.0s	4.7s	5.4s
	#Best	0	(23%)	30	0	(25%)	30
12 Larger grid graph	**Value**	483.7	486.5	490.8	484.7	487.7	491.7
	Dev.	2.4%	3.3%	3.8%	2.2%	3.0%	3.6%
	Time	1009.0s	1046.8s	1081.6s	2479.3s	2822.1s	3281.1s
	#Best	0	(0%)	0	0	(0%)	0
12 Smaller *Harwell-Boeing*	**Value**	18.9	19.3	19.9	18.9	19.5	20.1
	Dev.	0.6%	3.8%	5.9%	0.0%	3.1%	5.9%
	Time	1.0s	1.1s	1.1s	3.9s	4.3s	4.9s
	#Best	0	(51%)	30	1	(57%)	30
12 Larger *Harwell-Boeing*	**Value**	138.5	139.8	141.3	139.3	140.9	143.3
	Dev.	1.0%	2.7%	3.9%	0.0%	2.1%	3.4%
	Time	194.3s	197.6s	200.9s	987.4s	1129.2s	1371.6s
	#Best	0	(12%)	30	1	(17%)	30
12 Smaller *Hamming*	**Value**	13.3	13.7	14.3	12.9	13.6	14.4
	Dev.	26.0%	29.2%	31.8%	25.6%	29.9%	33.2%
	Time	6.2s	6.2s	6.3s	3.7s	5.4s	8.0s
	#Best	0	(0%)	0	0	(0%)	0
12 Larger *Hamming*	**Value**	91.3	95.9	103.0	91.6	98.0	108.6
	Dev.	32.1%	36.8%	39.9%	28.5%	35.5%	39.7%
	Time	669.5s	887.9s	1033.9s	758.0s	1247.9s	2216.5s
	#Best	0	(0%)	0	0	(0%)	0

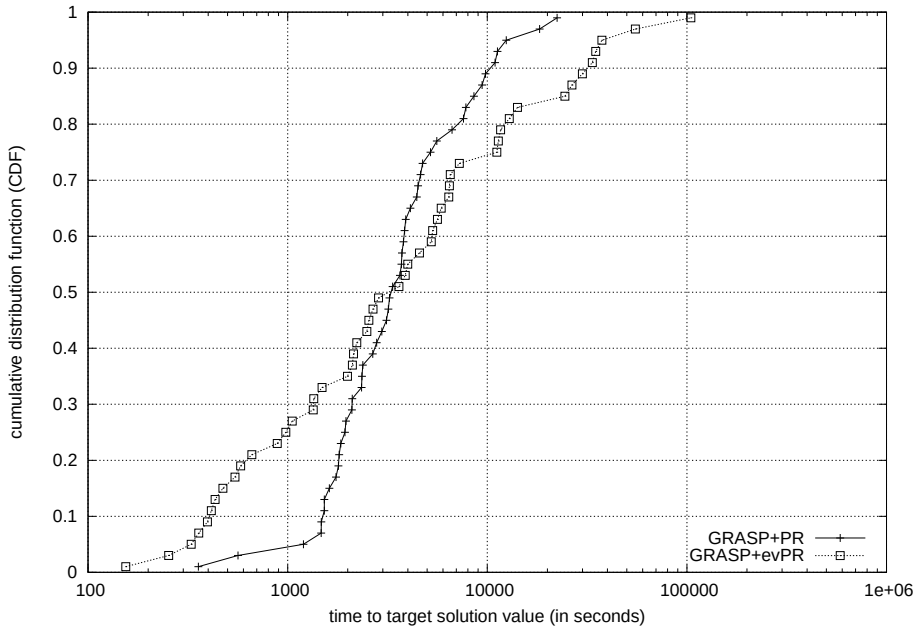

FIGURE 9. Time to target plots (run time distributions) for GRASP+PR and GRASP+evPR on `dwt503` with target solution 56 or more.

were about 3 to 6 times longer than GRASP with path relinking on the Harwell-Boeing instances and about 1.1 to 2.7 times longer on the grid graph instances. However, on the smaller Hamming graph instances, the GRASP with evolutionary path relinking was from 1.07 to 1.42 times faster than GRASP with path relinking. On the larger Hamming graph instances, the GRASP with path relinking was from 1.43 to 1.78 times faster than the GRASP with evolutionary path relinking. Times could be be equalized by either giving more global iterations to GRASP with path relinking, having fewer local or global iterations in GRASP with evolutionary path relinking, or limiting path relinking in the evolutionary stage to only pairs where one solution is one of the two or three top-quality solutions. However, consider the following experiment on the Harwell-Boeing instance `dwt503`. We ran GRASP with PR and GRASP with evPR 50 times on this instance, stopping when a solution with objective 56 or more was found. For each run we recorded the running time. Each run was independent of the other, using a different initial seed for the random number generator. For these runs, GRASP with evPR was configured to carry out 100 local iterations (instead of the 25 carried out before) during each global iteration. With these 50 running times, we plot the time-to-target plots (run time distributions), shown in Figure 9.

In the experiments with fixed number of iterations, GRASP+evPR found a solution with objective value 58 while GRASP+PR only found a 53. However, in those experiments, GRASP+evPR took, on average, 5.5 times longer than GRASP+PR. By running GRASP+PR for more iterations, it manages to find better solutions. In particular, in all 50 runs carried out, it found a solution of objective 56. However, as the figure shows, the running times were much longer and GRASP+PR

was no longer faster than GRASP+evPR to find a solution with the target value. For example, half of the runs of GRASP+evPR took less than 2853s while about 41% of the runs of GRASP+PR took less than that. Let T_{G+evPR} and T_{G+PR} be random variables representing the times required for GRASP with evPR and GRASP with PR, respectively, to find solutions with objective function value at least 56. Using the methodology proposed in Ribeiro et al. (2009), we compute $\text{Prob}(T_{G+evPR} \leq T_{G+PR}) = 0.5048$.

References

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

S. Dobrev, R. Královič, D. Pardubská, L. Török, and I. Vrto. Antibandwidth and cyclic antibandwidth of Hamming graphs. *Electronic Notes in Discrete Mathematics*, 34:295–300, 2009.

S. Donnelly and G. Isaak. Hamiltonian powers in threshold and arborescent comparability graphs. *Discrete Mathematics*, 202:33–44, 1999.

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

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

N.E. Gibbs, W.G. Poole, and P.K. Stockmeyer. An algorithm for reducing the bandwidth and profile of a sparse matrix. *SIAM Journal of Numerical Analysis*, 13:236–250, 1976.

F. Glover. Tabu search and adaptive memory programing – Advances, applications and challenges. In R.S. Barr, R.V. Helgason, and J.L. Kennington, editors, *Interfaces in Computer Science and Operations Research*, pages 1–75. Kluwer Academic Publishers, 1996.

F. Glover and M. Laguna. *Tabu search*. Kluwer Academic Publishers, 1997.

W.K. Hale. Frequency assignment: Theory and applications. In *Proceedings of the IEEE*, volume 68, pages 1497–1514, 1980.

M. Laguna and R. Martí. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11:44–52, 1999.

M. Laguna and R. Martí. *Scatter search: Methodology and implementations in C*. Kluwer Academic Publishers, 2003.

J.Y.-T. Leung, O. Vornberger, and J.D. Witthoff. On some variants of the bandwidth minimization problem. *SIAM J. on Computing*, 13:650–667, 1984.

R. Martí, M. Laguna, F. Glover, and V. Campos. Reducing the bandwidth of a sparse matrix with tabu search. *European J. of Operational Research*, 135: 211–220, 2001.

M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8:3–30, 1998.

C.H. Papadimitriou. The NP-completeness of the bandwidth minimization problem. *Computing*, 16:263–270, 1976.

E. Piñana, I. Plana, V. Campos, and R. Martí. GRASP and path relinking for the matrix bandwidth minimization. *European J. of Operational Research*, 153: 200–210, 2004.

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

M.G.C. Resende and C.C. Ribeiro. Greedy randomized adaptive search procedures. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 219–250. Kluwer Academic Publishers, 2003.

M.G.C. Resende and C.C. Ribeiro. Greedy randomized adaptive search procedures: Advances and applications. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*. Springer, 2nd edition, 2009. To appear.

M.G.C. Resende and R.F. Werneck. A hybrid heuristic for the p-median problem. *Journal of Heuristics*, 10:59–88, 2004.

M.G.C. Resende, M. Gallego, A. Duarte, and R. Martí. GRASP and Path Relinking for the Max-Min Diversity Problem. *Computers and Operations Research*, 37: 498–508, 2010.

C.C. Ribeiro and I. Rosseti. *A parallel GRASP for the 2-path network design problem*. Springer, 2002.

C.C. Ribeiro, I. Rosseti, and R. Vallejos. On the use of run time distributions to evaluate and compare stochastic local search algorithms. In T. Stützle, M. Birattari, and H.H. Hoos, editors, *Engineering Stochastic Local Search Algorithms*, volume 5752 of *Lecture Notes in Computer Science*, pages 16–30. Springer, 2009.

E. Rodriguez-Tello, H. Jin-Kao, and J. Torres-Jimenez. An improved simulated annealing algorithm for the matrix bandwidth minimization. *European J. of Operational Research*, 185:1319–1335, 2008.

L. Török and I. Vrt'o. Antibandwidth of 3-dimensional meshes. *Electronic Notes in Discrete Mathematics*, 28:161–167, 2007.

L. Yixun and Y. Jinjiang. The dual bandwidth problem for graphs. *J. of Zhengzhou University*, 35:1–5, 2003.

(A. Duarte) Departamento de Ciencias de la Computación, Universidad Rey Juan Carlos, Spain.

E-mail address: abraham.duarte@urjc.es

(R. Martí) Departamento de Estadística e Investigación Operativa, Facultad de Matematicas, Universidad de Valencia, Dr. Moliner 50, 46100 Burjassot (Valencia), Spain

E-mail address: rafael.marti@uv.es

(M.G.C. Resende) Algorithms and Optimization Research Department, AT&T Labs Research, 180 Park Avenue, Room C241, Florham Park, NJ 07932 USA.

E-mail address: mgcr@research.att.com

(R.M.A. Silva) Computational Intelligence and Optimization Group, Dept. of Computer Science, Federal University of Lavras, C.P. 3037, CEP 37200-000, Lavras, MG, Brazil

E-mail address, R.M.A. Silva: rmas@dcc.ufla.br