J Heuristics (2012) 18:919–938
DOI 10.1007/s10732-012-9213-7

ORIGINAL PAPER

Variable neighborhood search with ejection chains for the antibandwidth problem

Manuel Lozano · Abraham Duarte · Francisco Gortázar · Rafael Martí

Received: 2 July 2011 / Accepted: 3 October 2012 / Published online: 26 October 2012

Abstract In this paper, we address the optimization problem arising in some practical applications in which we want to maximize the minimum difference between the labels of adjacent elements. For example, in the context of location models, the elements can represent sensitive facilities or chemicals and their labels locations, and the objective is to locate (label) them in a way that avoids placing some of them too close together (since it can be risky). This optimization problem is referred to as the *antibandwidth maximization problem* (AMP) and, modeled in terms of graphs, consists of labeling the vertices with different integers or labels such that the minimum difference between the labels of adjacent vertices is maximized. This optimization problem is the dual of the well-known *bandwidth problem* and it is also known as the *separation* problem or directly as the *dual bandwidth problem*. In this paper, we first review the previous methods for the AMP and then propose a heuristic algorithm based on the variable neighborhood search methodology to obtain high quality solutions. One of our neighborhoods implements ejection chains which have been successfully applied in the context of tabu search. Our extensive experimentation with 236 previously reported

M. Lozano
Departamento de Ciencias de la Computación e Inteligencia Artificial,
Universidad de Granada, Granada, Spain
e-mail: lozano@decsai.ugr.es

A. Duarte · F. Gortázar
Departamento de Ciencias de la Computación, Universidad Rey Juan Carlos, Madrid, Spain
e-mail: abraham.duarte@urjc.es

F. Gortázar
e-mail: francisco.gortazar@urjc.es

R. Martí (✉)
Departamento de Estadística e Investigación Operativa, Universidad de Valencia, Valencia, Spain
e-mail: rafael.marti@uv.es

instances shows that the proposed procedure outperforms existing methods in terms of solution quality.

Keywords Metaheuristics · VNS · Layout problems

1 Introduction

In recent years there has been a growing interest in studying graph layout problems where the main objective is to find a labeling of a graph in such a way that a specific objective function is maximized or minimized. The Linear Arrangement (Rodriguez-Tello et al. 2008), Bandwidth (Piñana et al. 2004), Cutwidth (Pantrigo et al. 2011) or Vertex Separation (Duarte et al. 2012) fall into this class of optimization problems. In this paper, we tackle the antibandwidth maximization problem (AMP), which consists of labeling the vertices of a graph with distinct integers in such a way that the minimum difference between labels of adjacent vertices is maximized.

To formulate the AMP in mathematical terms, we first define the labeling f of a graph G. Given an undirected graph $G(V, E)$, where V ($|V| = n$) and E ($|E| = m$) represent the set of vertices and edges respectively, a labeling f of its vertices is a one-to-one mapping from the set V to the set $\{1, 2, \ldots n\}$ where each vertex $v \in V$ has a unique label $f(v) \in \{1, 2, \ldots, n\}$. Given the labeling f, the antibandwidth $AB_f(G)$ of graph G can be computed as:

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

where

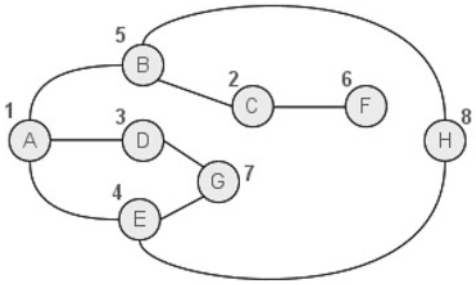
$$AB_f(v) = \min\{|f(v) - f(u)| : (v, u) \in E\}.$$

The AMP consists of finding a labeling f that maximizes $AB_f(G)$. This is the dual of the well-known bandwidth problem (Yixun and Jinjiang 2003), in which the value

$$\max\{|f(v) - f(u)| : (v, u) \in E\}$$

is minimized over all f labelings (Piñana et al. 2004).

This NP-hard problem was originally introduced in Leung et al. (1984) in connection with multiprocessor scheduling problems. An important motivation appears in the context of radio frequency assignment (Hale 1980). In particular, transmitters should be assigned to different frequencies in such a way that the physically neighboring transmitters have as different frequencies as possible (where frequency neighborhood is given by a graph). Other applications include obnoxious facility location problems (Cappanera 1999; Burkard et al. 2001) as they appear in the location of nuclear reactors, garbage dumps, or water purification plants. In those applications, customers no longer consider the facility desirable and try to have it as far as possible to their own location.



(a)

v	$AB_f(v)$
A	min{|1-5|,|1-3|,|1-4|}=2
B	min{|5-1|,|5-2|,|5-8|}=3
C	min{|2-5|,|2-6|}=3
D	min{|3-1|,|3-7|}=2
E	min{|4-1|,|4-8|}=3
F	min{|6-2|}=4
G	min{|7-3|,|7-4|}=3
H	min{|8-4|,|8-5|}=3

(b)

Fig. 1 **a** Graph example and **b** antibandwidth of G for a labeling f

Figure 1a shows an example of an undirected graph with eight vertices and nine edges. The number close to each vertex represents the label assigned to it. For example, the label of vertex A is $f(A) = 1$, the label of vertex B is $f(B) = 5$ and so on. Figure 1b shows the antibandwidth of each vertex, calculated as the minimum difference between the label of the corresponding vertex and all its neighbors' labels. Computing the minimum of these antibandwidth values we conclude that $AB_f(G) = 2$.

The antibandwidth problem can be optimally solved for specific classes of graphs. Raspaud et al. (2009) solved it for two dimensional meshes (cartesian product of two paths), tori (cartesian product of two cycles), and hyper-cubes. Török and Vrt'o (2007) extended these results to the case of three-dimensional meshes. Dobrev et al. (2009) proposed an exact algorithm for Hamming graphs (Cartesian product of d-complete graphs).

Recently, two heuristic procedures have been independently and simultaneously presented for the AMP (and therefore they did not compare each other). Duarte et al. (2011) presented two randomized greedy constructive procedures and a local search algorithm based on exchanges. Combining these heuristics the authors derived several GRASP methods. Additionally, a static and a dynamic path relinking post-processing procedures were also proposed for search intensification. In the static scheme, path relinking is performed once between all pairs of elite set solutions previously found with GRASP. In the dynamic scheme, after each GRASP local search phase, path relinking is executed between the corresponding local maximum and a solution selected at random from the elite set. The authors also proposed a GRASP with evolutionary path relinking heuristic, EvPR, which periodically applies path relinking between all pairs of solutions in the elite set. This later method obtains the best results although it consumes longer running times than the other variants.

Bansal and Srivastava (2011) proposed a Memetic Algorithm, MA, for the AMP. The algorithm starts by creating an initial population of solutions using a randomized breadth-first search, BFS. This method produces a spanning tree in which adjacent vertices belong to either same level or to adjacent levels. This ensures that the vertices belonging to alternate levels are not adjacent. Non-adjacent vertices belonging to alternative levels are labeled sequentially, and the remaining vertices are labeled in

a greedy fashion. As it is customary in evolutionary methods, the initial population evolves by applying three steps: selection, combination and mutation. The selection strategy is implemented by means of a classical tournament operator. The combination operator is implemented using a modified version of the BFS procedure, in which a solution is obtained by copying part of its "father" (up to a random point) and then completing it with the BFS constructive procedure. The mutation strategy is implemented by swapping two positions of a solution. These three main steps are repeated until a maximum number of iterations (generations) is reached.

In this paper, we propose a Variable Neighborhood Search procedure (Hansen et al. 2010) for the AMP. Section 2 introduces the three neighborhood structures and the associated local search methods. One of the neighborhoods implements ejection chains (Glover and Laguna 1997) which have been successfully applied in the context of tabu search. Section 3 is devoted to describe the VNS procedure itself, and how the neighborhoods interact. Computational experiments are described in Sect. 4 and concluding remarks are made in Sect. 5.

2 Neighborhood structures

Solutions to graph arrangement problems are typically represented as permutations, where each vertex occupies the position given by its label. For example, the labeling of the graph depicted in Fig. 1a,

$$f(A) = 1;\ f(B) = 5;\ f(C) = 2;\ f(D) = 3;$$
$$f(E) = 4;\ f(F) = 6;\ f(G) = 7;\ f(H) = 8,$$

can be expressed with the permutation $f = (A, C, D, E, B, F, G, H)$. In short, the first vertex in the permutation receives the label 1, the second vertex receives the label 2, and so on. In this section, we define three neighborhood structures based on permutations. Associated to each neighborhood a local search procedure can be defined to visit the solutions in the search space. In the next section we will describe how these three neighborhoods, and their associated local searches, interact within the VNS methodology.

The first two neighborhoods, N_1 and N_2, implement classic moves in permutation-based problems, general exchanges and consecutive swappings, respectively. Given a solution $f = (v_1, \ldots, v_i, \ldots, v_j, \ldots, v_n)$, we define $exchange(f, j, i)$ as exchanging in f the vertex in position i with the vertex in position j, producing a new solution $f' = (v_1, \ldots, v_{i-1}, v_j, v_{i+1}, \ldots, v_{j-1}, v_i, v_{j+1}, \ldots, v_n)$. For the sake of simplicity we denote $f' = exchange(f, j, i)$. The associated neighborhood N_1 has size $n(n-1)/2$, which can be considered relatively large, so instead of an exhaustive exploration, we apply candidate list strategies (Glover and Laguna 1997) for its improved scan. In particular, we order the vertices according to their AB_f-value (where the vertex with the minimum antibandwidth comes first), and examine them in this order. For each vertex v_i, we search for the first position j resulting in an improving $exchange(f, j, i)$ move. If we find it, we apply the move; otherwise we do not change the current solution f. In any case we resort to the next vertex in the ordered

list. When all the vertices have been examined and eventually some moves have been performed, we re-compute the antibandwidth of all of them and update their order. (Update key information only at certain points is considered an implementation of the *elite candidate list* introduced in the context of tabu search by Glover and Laguna 1997.) This local search method finishes when all the vertices have been examined and no improving move has been found.

The second neighborhood N_2 is defined by means of two symmetric moves, $swap_+(f, v_i)$ and $swap_-(f, v_i)$. The first one consists of removing the vertex v_i from its current position i in f and inserting it in position $i+1$ (i.e., swapping v_i and v_{i+1} in f). Symmetrically, the second move swaps v_i and v_{i-1} in f. In mathematical terms, given a solution $f = (v_1, \ldots, v_{i-1}, v_i, v_{i+1} \ldots, v_n)$ we have that:

$$swap_+(f, v_i) = (v_1, \ldots, v_{i-1}, v_{i+1}, v_i \ldots, v_n),$$
$$swap_-(f, v_i) = (v_1, \ldots, v_i, v_{i-1}, v_{i+1} \ldots, v_n).$$

We explore the associated neighborhood N_2 as we described above for N_1 (i.e., vertices are scanned in ascending order of their antibandwidth value and examined in search for an improving move). Given a vertex v_i, we first find its closest neighbor v_j in terms of labels (i.e., its adjacent vertex in which the antibandwidth of v_i is reached):

$$AB_f(v_i) = \min\{|f(v_i) - f(u)| : (v_i, u) \in E\} = |f(v_i) - f(v_j)|.$$

We want to change the label of v_i to increase $AB_f(v_i)$, therefore if $j > i$ we try $swap_-(f, v_i)$; otherwise, we try $swap_+(f, v_i)$. Without loss of generality consider that we try $swap_-(f, v_i)$. If it is an improving move the procedure performs it, obtains the new solution $f' = (v_1, \ldots, v_i, v_{i-1}, v_{i+1} \ldots, v_n)$ and tries the consecutive swap: $swap_-(f', v_i)$. The procedure performs consecutive swaps until no further improvement is possible or $j = 1$ (symmetrically $j = n$). At that point vertex v_i is discarded and the method resorts to the following vertex in the ordering list.

The third neighborhood N_3 is based on the ejection chain methodology. This strategy is often used in connection with tabu search (Glover and Laguna 1997) and consists of generating a compound sequence of moves, leading from one solution to another by means of a linked sequence of steps. In each step, the changes in some elements cause other elements to be ejected from their current state. See for instance Martí et al. (2009) and Rego (2001), for successful implementations of this strategy.

In the context of the AMP, suppose that we want to exchange the label $f(u)$ of a vertex u with the label $f(v)$ of another vertex v because this exchange results in an increment of the antibandwidth of u, but we found that it deteriorates the antibandwidth of v. We can therefore consider labeling u with $f(v)$ but, instead of labeling v with $f(u)$, examine another vertex w and check whether the label $f(u)$ may be advantageously assigned to w and whether, to complete the process, the label $f(w)$ is appropriate to v. In terms of ejection chains, we may say that the assignment of $f(v)$ to u caused $f(u)$ to be "ejected" from u to w (and concluding by assigning $f(w)$ to v). The outcome defines a compound move of depth two. We can repeat this logic to build longer chains.

To restrict the size of N_3 and to reduce the computational effort we only scan a subset W of the possible labels selected at random. As in the previous neighborhoods, vertices are scanned in ascending order of their antibandwidth value. Let u be the first vertex in the list, the chain starts by selecting the best label $f(v) \in W$ for u and evaluating the exchange of both labels ($f(v)$ and $f(u)$). If this is an improving move, it is applied and the chain stops (with depth one). Otherwise, we search for a vertex w with a label $f(w)$ in W adequate for v. If the compound move of depth two is an improving one, it is applied and the chain stops; otherwise the chain continues until the compound move becomes an improving one or the length of the chain reaches the prespecified limit depth. This local search is therefore restricted by two parameters: the size of W and the depth of the ejection chain, which control both the number of vertices involved in the move and the distance between the labels. We consider a maximum depth, $depth$, of compound moves that will be empirically adjusted in our experiments (it represents a balance between computational effort and search intensification). If none of the compound moves from depth 1 to $depth$ is an improving move, no move is performed and the exploration continues with the next vertex in the ordered list.

In the following section we define what we understand by an improving move. Considering that many vertices can have an antibandwidth equal to the graph's antibandwidth, a straightforward definition of move value would result in a poor guided local search method and therefore we propose an alternative, and richer, move value definition.

3 Variable neighborhood search

VNS is a methodology for solving optimization problems based on changing neighborhood structures. In recent years, a large amount of VNS variants have been proposed. Just to mention a few: Variable Neighborhood Descent (VND), Reduced VNS (RVNS), Basic VNS (BVNS), Skewed VNS (SVNS), General VNS (GVNS) or Reactive VNS. We refer the reader to Hansen et al. (2010) for an excellent review of this methodology.

In this paper, we focus on the VND variant, in which a predefined set of neighborhoods $\{N_1, N_2, \ldots, N_{k_{max}}\}$ is available and the change between them is performed in a deterministic and sequential way. Our method (see Fig. 2) implements a multi-start

```
PROCEDURE VND(MaxIter, k_max)
  1.    FOR i = 1, ..., MaxIter DO
  2.        f = Construct_Solution()
  3.        k = 1
  4.        WHILE (k < k_max)
  5.            f' = LocalSearch(f, N_k)
  6.            IF f' improves upon f THEN
  7.                f = f'
  8.                k = 1
  9.            ELSE
  10.               k = k + 1
  11.   RETURN f
END
```

Fig. 2 Pseudo-code of the VND method

procedure where in each iteration we first construct an initial solution with a level algorithm (Bansal and Srivastava 2011), and then apply an improvement method with a VND strategy. The algorithm starts by constructing an initial solution f (step 2), and then applies in step 5 a local search over the first neighborhood, $LocalSearch(f, N_k)$ with $k = 1$. If the resulting local optimum, f', does not improve upon f, we set $k = k + 1$ (step 10) and apply again $LocalSearch(f, N_k)$, but now with $k = 2$. We proceed in this way until k reaches k_{max} or the resulting local optimum, f', improves upon the initial solution. In the former case, we finish this step since f cannot be improved with any of the neighborhoods considered. In the later case, we resort to $k = 1$, update the current solution f (steps 7 and 8), and apply the local search with the first neighborhood to the obtained solution. The VND method terminates when a maximum number of construction steps, $MaxIter$, is performed. It is worth mentioning that when we compare two solutions in the step 6 of Fig. 2, we do not restrict our attention to the objective function, but we also consider additional evaluators (described in the next subsections).

3.1 Constructive method

Level algorithms (Díaz et al. 2002) are constructive procedures based on the partition of the vertices of a graph in different levels, $L_1, \ldots, L_s$, such that the endpoints of each edge in the graph are either in the same level L_i or in two consecutive levels, L_i and L_{i+1}. This level structure guarantees that vertices in alternative levels are not adjacent. Level structures are usually constructed using a BFS method, providing a root of the corresponding spanning tree and an order in which the vertices of the graph are visited.

Bansal and Srivastava (2011) applied a level procedure to the AMP. Specifically, they proposed a randomized breadth-first search wherein the spanning tree is constructed by selecting the root vertex (in level 1) as well as the neighbors of the visited vertices at random. Starting from an odd level (even level) the constructive procedure labels all the non-adjacent vertices of odd levels (even levels) sequentially. The remaining vertices are labeled using a greedy approach in which a vertex v is labeled with the unused label that produces the minimum increasing of its antibandwidth value $AB_f(v)$. Figure 3a shows a spanning tree of the graph introduced in Fig. 1a. Figure 3b depicts vertices in odd levels labeled sequentially. Finally, Fig. 3c shows the entire labeling, where the unlabeled vertices have been labeled in a greedy fashion.

3.2 Local search

In the AMP there may be many different solutions with the same objective function value. We could say that the solution space presents a "flat landscape". In this kind of problems, such as the min–max or max–min, local search procedures typically do not perform well because most of the moves have a null value. In the AMP there may be multiple vertices with an antibandwidth value equal to $AB_f(G)$. Then, changing the label of a particular vertex u to increase its antibandwidth $AB_f(u)$, does not necessarily imply that $AB_f(G)$ also increases. We therefore consider that

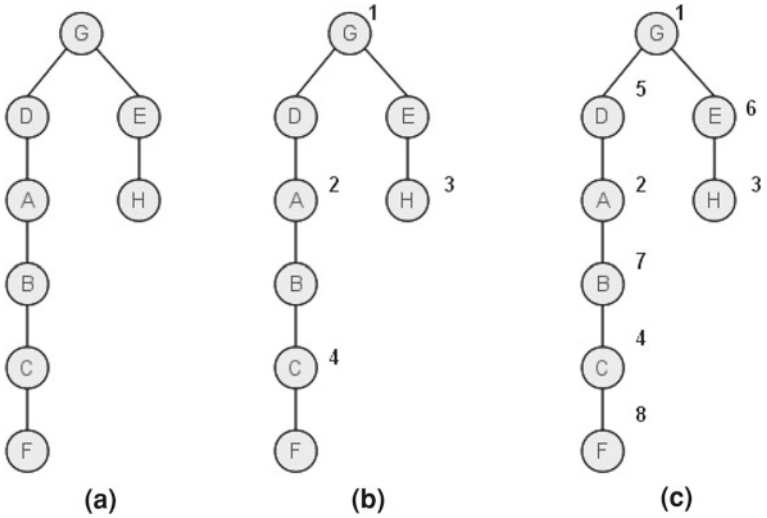


Fig. 3 **a** Spanning tree, **b** sequential labeling of odd levels, and **c** greedy labeling of remaining nodes

a move improves the current solution if the number of vertices with a relative small antibandwidth value is reduced (even if the objective function is not improved). With this extended definition of "improving" we overcome the lack of information provided by the objective function. Specifically, we classify the vertices in the sets S_i for $i = 1$ to $maxAB$ where set S_i contains the vertices with antibandwidth i. In mathematical terms, $S_i = \{v \in V | AB_f(v) = i\}$. For example, considering the graph depicted in Fig. 1a and the corresponding labeling f, we obtain the sets $S_1 = \emptyset$, $S_2 = \{A, D\}$, $S_3 = \{B, C, E, G, H\}$, and $S_4 = \{F\}$.

We consider that a move changing the label of a vertex v improves the current solution if it reduces the cardinality of any set S_i with $i \geq AB_f(v)$ without increasing the cardinality of any set S_k with $k \leq i$. We have empirically found that this criterion allows the local search procedure to explore a larger number of solutions than a typical implementation that only performs moves when the objective function is improved. Figure 4a shows an improving move for the labeling of Fig. 1a. The move consists of exchanging the labels of vertices A and H, obtaining a new solution f'. In the new labeling, vertex A is removed from set S_2 and included in set S_3. It means that the antibandwidth value of vertex A is increased by one unit (from 2 to 3). On the other hand vertex H remains in set S_3. This move does not increase the antibandwidth of the graph, but we reduce the number of vertices with a small antibandwidth value.

Considering the graph shown in Fig. 1a if we now exchange the labels of vertices G and H (see Fig. 4b) vertex G is removed from set S_3 and included in set S_4 (i.e., its antibandwidth is improved by one unit), but vertex H is removed from set S_3 and included in set S_2. We then consider that this move does not improve the incumbent solution.

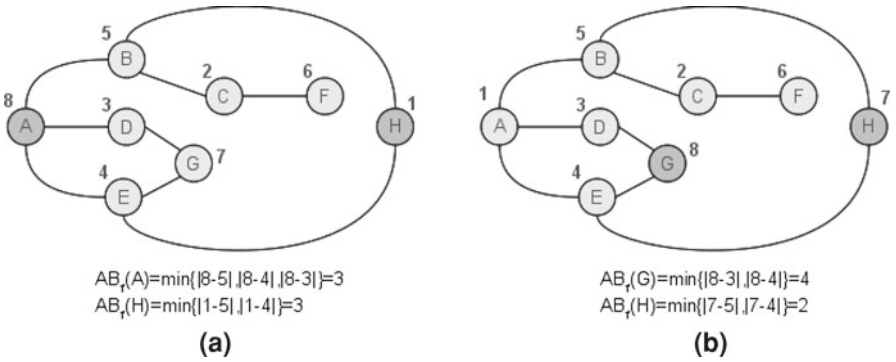


Fig. 4 **a** Improving move and **b** non-improving move

4 Experimental results

This section describes the computational experiments that we performed to compare our procedure with the state-of-the-art methods for solving the AMP. The VNS procedure described in Sect. 3, the Memetic Algorithm (Bansal and Srivastava 2011), and the Evolutionary Path Relinking (Duarte et al. 2011) were run on the same personal computer to perform a fair comparison. Specifically, we used an Intel Core 2 Quad CPU Q 8300 with 6 GiB of RAM and Ubuntu 9.04 64 bits OS. We have considered 10 sets with a total of 236 instances classified in two groups: 164 instances with known optimum, and 72 instances with unknown optimum. All these instances are available at http://www.optsicom.es/abp/. A detailed description of each set of instances follows.

Instances with known optimum

- *Paths* This data set consists of 24 graphs constructed as a linear arrangement of vertices such that every vertex has a degree of two, except the first and the last vertex that have a degree of one. The size of these instances ranges from 50 to 1000. For a path P_n with n vertices, Yixun and Jinjiang (2003) proved that the optimal antibandwidth is $\lfloor \frac{n}{2} \rfloor$.
- *Cycles* This data set consists of 24 graphs constructed as a circular arrangement of vertices such that every vertex has a degree of two. The size of these instances ranges from 50 to 1000. For a cycle C_n with n vertices the optimal antibandwidth is $\lfloor \frac{n-1}{2} \rfloor$, as shown in Bansal and Srivastava (2011).
- *Grids* This data set consists of 24 graphs constructed as the Cartesian product of two paths P_{n_1} and P_{n_2} (Raspaud et al. 2009). The size of these instances ranges from 81 to 1170. They are also called two dimensional meshes. For a grid $P_{n_1} \times P_{n_2}$ witn $n = n_1 n_2$ vertices the optimal antibandwidth is $\left\lceil \frac{(n_1-1)n_2}{2} \right\rceil$ with $n_1 \geq n_2 \geq 2$.
- *Toroidal grids (Tori)* This data set consists of 37 graphs constructed as the Cartesian product of two cycles (i.e., $C_n \times C_n$). The size of these instances ranges from 16 to 1600. The optimal antibandwidth is $\frac{(n-2)n}{2}$ if n is even, and $\frac{(n-2)(n+1)}{2}$ if n is odd (see Raspaud et al. 2009).

- *Hypercubes* This data set consists of seven hypercubes Q_d. Each vertex is represented as a binary vector of length d. Two vertices are adjacent if their associated binary vectors only differ in one element. A hypercube is a graph with $n = 2^d$ vertices and $m = d * 2^{d-1}$ edges. The optimal antibandwidth is $2^{n-1} - \sum_{j=0}^{n-2} \binom{j}{\lfloor \frac{j}{2} \rfloor}$ as shown in Wang et al. (2009).
- *Complete binary trees (CBT)* This data set consists of 24 trees, where every tree-level is completely filled except possibly the last level and all nodes are as far left as possible. The size of these instances ranges from 30 to 950. The optimal antibandwidth is $\lfloor \frac{n}{2} \rfloor$ (see Miller and Pritikin 1989).
- *Hamming graphs* This data set consists of 24 graphs constructed as the Cartesian product of d complete graphs K_{nk}, for $k = 1, 2, \ldots, d$ (Dobrev et al. 2009). The size of these instances ranges from 80 to 1152. The vertices in these graphs are d-tuples $(i_1, i_2, ..., i_d)$, where $i_k \in \{0, 1, ..., n_{k-1}\}$. 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 are referred to as Hamming graphs. It is shown in Dobrev et al. (2009) that if $d \geq 2$ and $2 \leq n_1 \leq n_2 \leq \cdots \leq n_d$, then the optimal solution of the antibandwidth problem for this type of instance is given by:

$$AB\left(\prod_{k=1}^{d} K_{nk}\right) = \begin{cases} n_1 n_2 \ldots n_{d-1} & \text{if } n_{d-1} \neq n_d \\ n_1 n_2 \ldots n_{d-1} - 1 & \text{if } n_{d-1} = n_d \text{ and } d \geq 3 \end{cases}$$

Instances with unknown optimum

- *Caterpillars* This data set consists of 40 graphs. Each caterpillar, $P_{n_1 n_2}$ is constructed using the path P_{n_1} and n_1 copies of the path P_{n_2} (usually referred to as "hairs"), where each vertex i in P_{n_1} is connected to the first vertex of the i-th copy of the path P_{n_2}. The size of these instances ranges from 20 to 1000.
- *3D grids* This data set consists of 8 graphs constructed as the Cartesian product of three paths P_{n_1}, P_{n_2} and P_{n_3} (Raspaud et al. 2009). The size of these instances ranges from 27 to 1000. They are also called three dimensional meshes or cuboidal meshes. For a 3D grid $P_{n_1} \times P_{n_2} \times P_{n_3}$ with $n = n_1 n_2 n_3$ vertices, the optimal antibandwidth is lower than or equal to the upper bound $UB = \frac{k^3}{2} - \frac{2k^2}{8}$, where $k = n_1 = n_2 = n_3$ and $k \geq 3$ (see Török and Vrt'o 2007).
- *Harwell-Boeing* We derived 24 matrices from the Harwell-Boeing Sparse Matrix Collection (Harwell-Boeing 2011). The size of these instances ranges from 30 to 900. 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 science and engineering. The problems range from small matrices, used as counter-examples to hypotheses in sparse matrix research, to large matrices arising in practical 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 $M_{ij} \neq 0$.

Table 1 Width of the ejection chain procedure

$\|W\|$	Dev. (%)	Score	Time
$0.1n$	25.27	67	78.72
$0.2n$	24.37	50	120.26
$0.3n$	23.42	37	159.92
$0.4n$	22.10	18	198.34
$0.5n$	21.73	25	236.43

We have divided our experimentation into two parts: preliminary experimentation and final experimentation. The preliminary experiments were performed to set the values of the key search parameters of our heuristic method as well as to show the merit of the proposed search strategies. We consider a representative subset of instances (12 Harwell-Boeing, 12 Grids and 12 Hamming instances with different densities and sizes).

In our preliminary experimentation, we first consider the ejection chain method implemented in neighborhood N_3 (see Sect. 2). In particular, the first experiment is devoted to adjust the size of the set W of candidate labels of the ejection chain (EC) procedure. To do that, we generate a set of 100 solutions with the constructive procedure described in Sect. 3.1 and apply to them the EC method with different values of $|W| \in \{0.1n, 0.2n, 0.3n, 0.4n, 0.5n\}$. Table 1 shows the average percentage deviation (Dev.) between the best solutions found and the best known value (which in the case of the Grids and Hamming instances are the optimal values). We also report the so-called Score associated with each method (Resende et al. 2010). This statistic is based on the *nrank* of each algorithm over each instance. The *nrank* of algorithm A is defined as the number of methods that found a better solution than the one found by A. In the event of ties, the methods receive the same *nrank*, equal to the number of methods strictly better than all of them. The value of Score is the sum of the *nrank* values for all the instances in the experiment, thus, the lower the Score the better the method. Finally, for each value of $|W|$ we show the CPU time in seconds (Time) needed to construct and improve 100 solutions.

Table 1 shows that size of W has an important effect in both the quality of the solution obtained and the running time. As it was expected, the larger the $|W|$, the lower the deviation (and the larger the CPU time). Taking into account that the EC method is part of a master procedure, we set $W = 0.3n$ as a compromise selection for the rest of our experimentation.

In the second preliminary experiment, we adjust the $depth$ parameter of the ejection chain method. We again construct 100 solutions and improve them with the EC method considering $W = 0.3n$ and $depth \in \{1, 10, 25, 0.1n, 0.05n\}$, where the last two values ($0.1n$ and $0.05n$) are linearly dependent with the instance size. Table 2 shows the same statistics than above.

Table 2 shows that the value of $depth$ has a large impact on the behaviour of the method in both, running time and average deviation. Specifically, average deviation ranges from 29.44 % (with $depth = 1$, indicating that we restrict the search to single moves and there is no chained move), to 21.75 % (with $depth = 25$ chained moves).

Table 2 Depth of the ejection chain procedure

Depth	Dev. (%)	Score	Time
1	29.44	112	18.16
10	22.37	62	32.41
25	21.75	43	42.56
$0.1n$	21.88	46	85.14
$0.05n$	22.42	51	56.44

Table 3 Comparison of different neighborhoods

Method	Dev. (%)	Score	Time
LS1	11.97	21	58.98
LS2	16.11	44	123.00
LS3	24.87	67	55.62
VNS	9.32	2	122.88

We therefore set $depth = 25$ for which the method exhibits the best performance (lower deviation and moderate CPU time).

In our third preliminary experiment, we study the contribution of each neighborhood to the VNS method. Specifically, we study the quality that each neighborhood is able to obtain when it works in isolation (implemented in a local search) and when they all work in combination within the VNS. We compare the LS1 local search procedure (in which we apply neighborhood N_1 until no further improvement is possible), with the LS2 (that applies N_2) and LS3 (that applies N_3). As in the previous experiments, we construct 100 solutions with the aforementioned procedure and improve them with any of the three variants as well as with the VNS method (based on the three neighborhoods as described in Sect. 3). Table 3 summarizes the results of these four methods.

Results in Table 3 confirm that the VNS procedure compares favorably with simple local search methods. Specifically, VNS achieves the lowest deviation (9.32 %) compared with the three local search methods tested (11.97 %, 16.11 % and 24.87 %, for LS1, LS2, and LS3, respectively). Additionally, the score value of VNS is 2 which means that VNS was only outperformed twice while the second best method (LS1) was outperformed in 21 times. On the other hand, the CPU time of LS2 and VNS is significantly larger than the one invested by LS1 and LS3. To complement this information, we show in Fig. 5 how the average deviation value of these four methods improves over time (we consider a time horizon of 150 s reporting values every second).

The time-profile depicted in Fig. 5 shows that VNS clearly outperforms LS2 and LS3 and presents a marginal improvement with respect to LS1 (this improvement is consolidated as the search progresses). Note that the AMP is a max–min problem, where the objective function consists of maximizing a minimum value. As it is well documented, this kind of problems presents a "flat landscape" in which simple local search methods usually get trapped (because the associated moves have a null value). In this context, the VNS turns to be a good option since the change of the neighborhood can help to disclose which are the "good" moves. Although not shown in this figure, it

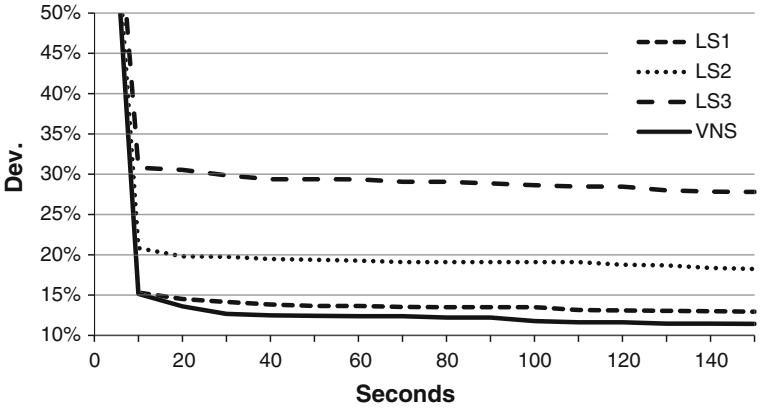


Fig. 5 Average deviation of the best solution found over the time

is worth mentioning that even if we run LS1 for longer running times (up to 1800 s), it is not able to reach the objective function values found by VNS in the first 100 s in 8 out of 36 instances.

Considering that the differences between LS1 and VNS are relatively small, we have applied two statistical tests for pairwise comparisons, the Wilcoxon test and the Sign test. The former test computes if the two samples (the solutions of both methods) come from different populations, while the latter computes the number of instances on which an algorithm improves upon the other. The resulting p values of 0.000 and 0.001 respectively, indicate that there are significant differences between the results of both methods, resulting VNS as the best method of this experiment.

In the final experimentation, we compare our VNS method with the best previous methods. As described in Sect. 1, they are the Memetic Algorithm, MA, proposed by Bansal and Srivastava (2011) and the Evolutionary Path Relinking, EvPR, due to Duarte et al. (2011). Table 4 presents the performance of each algorithm over the group of 164 instances with known optimum. Each main row reports the results on each group of instances. In particular, we compute the average deviation (Dev.), number of optimum (#Opt) and score of the best solutions found with each method on each group of instances. The three methods are run for a similar running time (150 s) on the same computer. Small differences on running times are due to the stopping criteria (based on the number of iterations of each method). It is important to note that the three methods considered are able to find most of the optima if they run for long CPU times. However, in the context of heuristic optimization, the ability to find good solutions in short running times is a key factor. In order to find significant differences among the methods and test this ability, we limit the CPU time to 150 s.

Results in Table 4 show that the VNS obtains the best solutions in four sets of instances (in terms of number of optima and average deviation). In particular, it is able to match the optima in the 24 instances in the Paths set, 16 optima (out of 24) and 0.84 % average deviation in the Cycles set, 1 optimum (out of 24) and 3.92 % average deviation in the CBT set, and 1 optimum (out of 24) and 17.06 % average deviation in the Hamming set, which compare favorably with the results obtained with

Table 4 Comparison of best methods on the 164 instances with known optimum

	Dev. (%)	# Opt	Score	Time (avg)
Paths				
MA	0.04	21	3	175.69
EvPR	1.88	4	40	160.23
VNS	0.00	24	0	150.08
Cycles				
MA	0.88	16	2	171.28
EvPR	2.46	4	32	158.70
VNS	0.84	16	1	150.08
Grids				
MA	0.13	11	0	154.18
EvPR	3.02	3	38	366.32
VNS	0.92	5	17	150.83
Tori				
MA	38.59	11	51	191.44
EvPR	5.48	21	3	201.25
VNS	6.94	20	8	150.65
Hypercubes				
MA	0.00	7	0	218.01
EvPR	1.35	4	5	174.06
VNS	1.15	5	3	150.29
CBT				
MA	18.79	0	46	150.66
EvPR	3.94	1	7	152.27
VNS	3.92	1	4	150.05
Hamming				
MA	57.99	0	42	173.11
EvPR	30.80	0	26	416.12
VNS	17.06	1	0	150.25

MA and EvPR in these sets. On the other hand, the MA obtains the best results in Grids and Hypercubes, and the EvPR in the Tori. Overall, VNS obtains 56 optima, out of 164 instances, while MA and EvPR obtain 50 and 33 optima, respectively. The score statistic also indicates that VNS systematically ranks in the first positions when comparing the three methods, since it exhibits a value of 33 when considering the 164 instances, while MA and EvPR present 144 and 151 respectively. In line with these values, we have performed a non-parametric statistical comparison, the Friedman test, also based on ranks (where rank 3 is assigned to the best method and rank 1 to the worst one). The resulting p value of 0.000 obtained in this experiment clearly indicates that there are statistically significant differences among the three methods tested and the rank values produced by this test are 2.41 (VNS), 1.80 (MA), and 1.79 (EvPR), confirming the previous results.

Table 5 Comparison of best methods on the 72 instances with unknown optimum

	Dev. (%)	# Best	Score	Time (avg)
3D grids				
MA	0.00	8	0	223.08
EvPR	1.17	2	7	154.30
VNS	0.92	3	5	150.50
Caterpillars				
MA	3.68	0	80	201.99
EvPR	0.02	38	2	155.12
VNS	0.05	38	2	150.10
HarwellBoeing				
MA	48.10	3	42	152.02
EvPR	0.94	16	9	156.30
VNS	0.21	20	4	150.04

We conduct the same study over the group of instances with unknown optima (72 instances) and present the associated results in Table 5. In this case, the average deviation, Dev., is computed with respect to the best known solution and #Best indicates the number of times that each method matches the best known solution. We include in the Appendix Tables 6, 7, and 8 where we report, for each instance with unknown optimum, the best known value, the tightest upper bound (Yixun and Jinjiang 2003), the relative deviation (in percentage) between the best known value and the upper bound, and the heuristic method (or methods) achieving these results.

Table 5 shows that MA obtains the best results in the 3D grids set, and the worst ones in the Caterpillars and HarwellBoeing sets. On the other hand, VNS obtains the best results in the HarwellBoeing set and ranks second in the other two sets. Overall results on the 72 instances show that VNS presents an average percentage deviation of 0.39 % and 61 best-known values, followed by EvPR (with 0.71 % and 56 respectively), and MA (with 25.89 % and 11). The resulting p value of 0.000 obtained with the Friedman test in this experiment certifies that there are differences among these three methods, and the associated rank values of 2.42 (VNS), 2.33 (EvPR) and 1.26 (MA) are in line with the pattern shown above.

In order to evaluate the behavior of these methods over a long-term time horizon, we run MA, EvPR and VNS for 30 min, reporting the average deviation of the best found solution every minute. We consider the set of 36 representative instances used in the preliminary experimentation. Figure 6 shows the corresponding average time profile, in which we can see that VNS consistently produces better results than its competitors although the three of them are able to produce good results. Specifically, the three methods quickly improve its average deviation in the first two minutes in which VNS clearly establishes its superiority. From this point, the methods are only able to produce a marginal improvement (around 1 %), which on the other hand is a difficult task considering the max–min of this optimization problem.

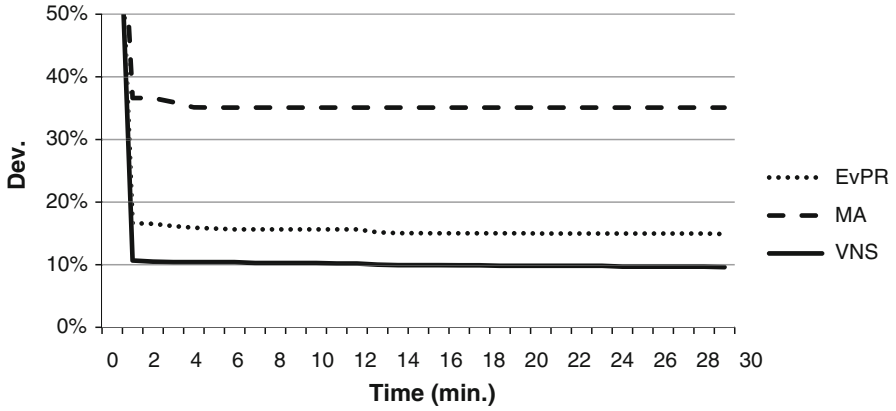


Fig. 6 Performance profile for a 30 min run

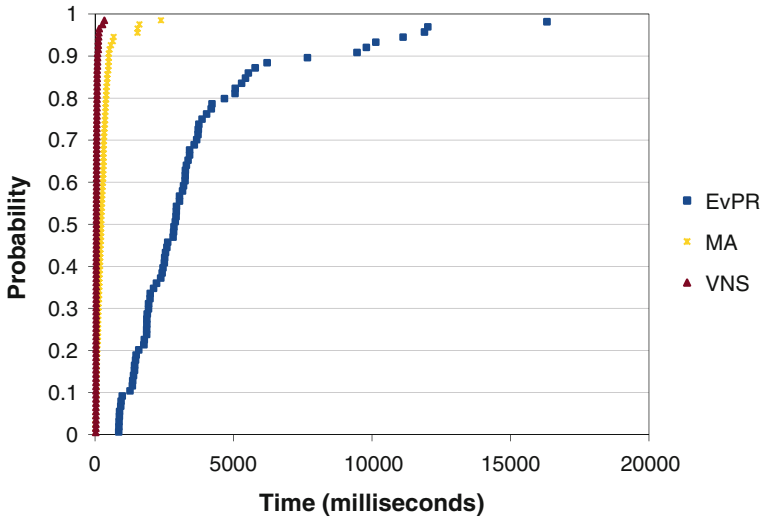


Fig. 7 Time to target plots

To finish our experiments, we consider a time-to-target plot (Aiex et al. 2002). We ran the three competing methods 100 times on a representative instance, Hamming 5×6×6, stopping when a solution with objective value equal to the best known for this instance, 16, 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. With these 100 running times, we plot the time-to-target plots (run time distributions), which is depicted in Fig. 7. This figure shows that VNS is able to find the target solution faster than the other two methods. Moreover, this experiment illustrates the exponential runtime distribution for these methods. Therefore, linear speed is expected if they are implemented in parallel.

5 Conclusions

In this paper, we report our research on the use of different neighbours and candidate list strategies that allows us to cope with a computationally expensive objective function evaluation within a VNS procedure with ejection chains. Our procedure selects moves from a candidate list of moves whose move values are not updated after every iteration. The list follows the tabu search principle that the values of a set of elite moves do not drastically change from one iteration to the next and therefore it is not necessary to update them after the execution of every move. In addition to the application of this candidate list strategy, our procedure employs an unconventional definition of move value, which is not based on the change of the objective function value to direct the search. In this way, our move value definition conveys information that is not available when the change in the objective function value is calculated.

The performance of the procedure has been assessed using 236 problem instances of several types and sizes. Our preliminary experimentation clearly proves the merit of combine neighborhoods, as VNS does, in the context of max–min problems. Moreover, the procedure has been shown robust in terms of solution quality within a reasonable computational effort. The proposed method was compared with two recently developed procedures due to Duarte et al. (2011) and Bansal and Srivastava (2011) respectively. The comparisons favor the proposed variable neighborhood search implementation.

According to our experimentation, the Paths, Cycles, Grids and Hypercube instances can be considered "easy to solve". On the contrary, the CBT, Hamming and HarwellBoeing instances are actually a challenge for modern heuristic methods.

Acknowledgements This research has been partially supported by the Ministerio de Ciencia e Innovación of Spain within the OPTSICOM project (http://www.optsicom.es/) with grant codes TIN2008-05854, TIN2009-07516, TIN2012-35632, and P08-TIC-4173.

Appendix

Tables 6, 7 and 8 report the comparison of the state-of-the-art methods over the set of instances with unknown optimum. We consider a time limit of 150 s per instance. Each table shows for each instance the best known value, *Best val.*, the tightest upper bound (Yixun and Jinjiang 2003), *UB*, the relative deviation (in percentage) between the best known value and the upper bound, *Dev*, and finally the heuristic method (or methods) achieving these results.

Table 6 Best values, best methods and upper bounds per instance for Harwell-Boeing matrices

	Best val.	UB	Dev	Method
ash85.mtx.rnd	21	42	50.0	VNS
bcspwr01.mtx.rnd	17	19	10.5	VNS, EvPR
bcspwr02.mtx.rnd	21	24	12.5	VNS, EvPR
bcspwr03.mtx.rnd	39	59	33.9	VNS, EvPR
bcsstk01.mtx.rnd	8	22	63.6	VNS, EvPR
pores_1.mtx.rnd	6	13	53.8	VNS, MA, EvPR
will57.mtx.rnd	13	28	53.6	VNS, EvPR
curtis54.mtx.rnd	13	26	50.0	VNS
dwt__234.mtx.rnd	50	58	13.8	VNS, EvPR
ibm32.mtx.rnd	9	15	40.0	VNS, EvPR
impcol_b.mtx.rnd	8	29	72.4	VNS, EvPR
nos4.mtx.rnd	34	50	32.0	VNS, EvPR
494_bus.mtx.rnd	227	247	8.1	VNS
662_bus.mtx.rnd	220	331	33.5	VNS
685_bus.mtx.rnd	136	342	60.2	VNS, EvPR
can__445.mtx.rnd	82	221	62.9	VNS
can__715.mtx.rnd	115	357	67.8	EvPR
bcsstk06.mtx.rnd	32	210	84.8	EvPR
bcsstk07.mtx.rnd	31	210	85.2	VNS, EvPR
dwt__503.mtx.rnd	53	250	78.8	VNS, EvPR
dwt__592.mtx.rnd	113	295	61.7	VNS
impcol_d.mtx.rnd	103	212	51.4	VNS, EvPR
nos6.mtx.rnd	329	337	2.4	MA
sherman4.mtx.rnd	261	272	4.0	MA

Table 7 Best values, best methods and upper bounds per instance for 3D grids

	Best val.	UB	Dev. (%)	Method
3dmesh_3.txt	9	12	25.0	VNS, MA, EvPR
3dmesh_4.txt	25	31	19.4	VNS, MA, EvPR
3dmesh_5.txt	51	61	16.4	VNS, MA
3dmesh_6.txt	92	107	14.0	MA
3dmesh_7.txt	149	170	12.4	MA
3dmesh_8.txt	228	255	10.6	MA
3dmesh_9.txt	328	363	9.6	MA
3dmesh_10.txt	454	499	9.0	MA

Table 8 Best values, best methods and upper bounds per instance for Caterpillars

	Best val.	UB	Dev. (%)	Method
caterpillar_5_4.txt	10	10	0.0	VNS, EvPR
caterpillar_5_5.txt	12	12	0.0	VNS, EvPR
caterpillar_5_6.txt	15	15	0.0	VNS, EvPR
caterpillar_5_7.txt	17	17	0.0	VNS, EvPR
caterpillar_9_4.txt	18	18	0.0	VNS, EvPR
caterpillar_9_5.txt	22	22	0.0	VNS, EvPR
caterpillar_9_6.txt	27	27	0.0	VNS, EvPR
caterpillar_9_7.txt	31	31	0.0	VNS, EvPR
caterpillar_10_4.txt	20	20	0.0	VNS, EvPR
caterpillar_10_5.txt	25	25	0.0	VNS, EvPR
caterpillar_10_6.txt	30	30	0.0	VNS, EvPR
caterpillar_10_7.txt	35	35	0.0	VNS, EvPR
caterpillar_15_4.txt	30	30	0.0	VNS, EvPR
caterpillar_15_5.txt	37	37	0.0	VNS, EvPR
caterpillar_15_6.txt	45	45	0.0	VNS, EvPR
caterpillar_15_7.txt	52	52	0.0	VNS, EvPR
caterpillar_20_4.txt	40	40	0.0	VNS, EvPR
caterpillar_20_5.txt	50	50	0.0	VNS, EvPR
caterpillar_20_6.txt	60	60	0.0	EvPR
caterpillar_20_7.txt	69	70	1.4	VNS, EvPR
caterpillar_20_10.txt	99	100	1.0	VNS, EvPR
caterpillar_20_15.txt	148	150	1.3%	VNS, EvPR
caterpillar_20_20.txt	197	200	1.5	VNS, EvPR
caterpillar_20_25.txt	247	250	1.2	EvPR
caterpillar_25_10.txt	124	125	0.8	VNS, EvPR
caterpillar_25_15.txt	185	187	1.1	VNS, EvPR
caterpillar_25_20.txt	247	250	1.2	VNS, EvPR
caterpillar_25_25.txt	309	312	1.0	VNS
caterpillar_30_10.txt	149	150	0.7	VNS, EvPR
caterpillar_30_15.txt	222	225	1.3	VNS, EvPR
caterpillar_30_20.txt	297	300	1.0	VNS, EvPR
caterpillar_30_25.txt	371	375	1.1	VNS, EvPR
caterpillar_35_10.txt	174	175	0.6	VNS, EvPR
caterpillar_35_15.txt	260	262	0.8	VNS
caterpillar_35_20.txt	347	350	0.9	VNS, EvPR
caterpillar_35_25.txt	433	437	0.9	VNS, EvPR
caterpillar_40_10.txt	199	200	0.5	VNS, EvPR
caterpillar_40_15.txt	297	300	1.0	VNS, EvPR
caterpillar_40_20.txt	397	400	0.8	VNS, EvPR
caterpillar_40_25.txt	496	500	0.8	VNS, EvPR

References

Aiex, R.M., Resende, M.G.C., Ribeiro, C.C.: Probability distribution of solution time in GRASP: An experimental investigation. J. Heuristics **8**, 343–373 (2002)

Bansal, R., Srivastava, K.: Memetic algorithm for the antibandwidth maximization problem. J. Heuristics **17**, 39–60 (2011)

Burkard, R.E., Donnani, H., Lin, Y., Rote, G.: The obnoxious center problem on a tree. SIAM J. Discret. Math. **14**(4), 498–590 (2001)

Cappanera, P.: A survey on obnoxious facility location problems. Technical Report TR-99-11, Dipartimento di Informatica, Università di Pisa (1999)

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

Dobrev, S., Královic, R., Pardubská, D., Török, L., Vrt'o, I.: Antibandwidth and cyclic antibandwidth of Hamming graphs. Electron. Notes Discret. Math. **34**, 295–300 (2009)

Duarte, A., Martí, R., Resende, M.G.C., Silva, R.M.A.: GRASP with path relinking heuristics for the antibandwidth problem. Networks **58**(3), 171–189 (2011)

Duarte, A., Escudero, L.F., Martí, R., Mladenovic, N., Pantrigo, J.J., Sánchez-Oro, J.: Variable neighborhood search for the vertex separation problem. Comput. Oper. Res. **39**(12), 3247–3255 (2012)

Glover, F., Laguna, M.: Tabu Search. Kluwer, Norwell (1997)

Hale, W.K.: Frequency assignment: theory and applications. Proc. IEEE **68**, 1497–1514 (1980)

Hansen, P., Mladenovic, N., Brimberg, J., Moreno-Pérez, J.A.: Variable neighborhood search. In: Gendreau, M., Potvin, J.-Y. (eds.) Handbook of Metaheuristics, 2nd edn, pp. 61–86. Springer, Heidelberg (2010)

Harwell-Boeing: http://math.nist.gov/MatrixMarket/data/Harwell-Boeing/ (2011)

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

Martí, R., Duarte, A., Laguna, M.: Advanced scatter search for the max-cut problem. INFORMS J. Comput. **21**(1), 26–38 (2009)

Miller, Z., Pritikin, D.: On the separation number of a graph. Networks **19**, 651–666 (1989)

Pantrigo, J.J., Martí, R., Duarte, A., Pardo, E.G.: Scatter search for the cutwidth minimization problem. Ann. Oper. Res. **199**(1), 285–304 (2012)

Piñana, E., Plana, I., Campos, V., Martí, R.: GRASP and path relinking for the matrix bandwidth minimization. Eur. J. Oper. Res. **153**, 200–210 (2004)

Raspaud, A., Schröder, H., Sykora, O., Török, L., Vrt'o, I.: Antibandwidth and cyclic antibandwidth of meshes and hypercubes. Discret. Math. **309**, 3541–3552 (2009)

Rego, C.: Node ejection chains for the vehicle routing problem: sequential and parallel algorithms. Parallel Comput. **27**(3), 201–222 (2001)

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

Rodriguez-Tello, E., Jin-Kao, H., Torres-Jimenez, J.: An effective two-stage simulated annealing algorithm for the minimum linear arrangement problem. Comput. Oper. Res. **35**(10), 3331–3346 (2008)

Török, L., Vrt'o, I.: Antibandwidth of 3-dimensional meshes. Electron. Notes Discret. Math. **28**, 161–167 (2007)

Yixun, L., Jinjiang, Y.: The dual bandwidth problem for graphs. J. Zhengzhou Univ. **35**, 1–5 (2003)

Wang, X., Wu, X., Dumitrescu, S.: On explicit formulas for bandwidth and antibandwidth of hypercubes. Discret. Appl. Math. **157**(8), 1947–1952 (2009)