Variable Neighborhood Search with Ejection Chains for the Antibandwidth Problem

MANUEL LOZANO
Departamento de Ciencias de la Computación e Inteligencia Artificial, Universidad de Granada, Spain.
lozano@decsai.ugr.es

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

FRANCISCO GORTÁZAR
Departamento de Ciencias de la Computación, Universidad Rey Juan Carlos, Spain.
francisco.gortazar@urjc.es

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

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 72 previously reported instances shows that the proposed procedure outperforms existing methods in terms of solution quality employing a third of computing time.

Keywords: metaheuristics, VNS, layout problems.

Version: June 20, 2011

1. Introduction

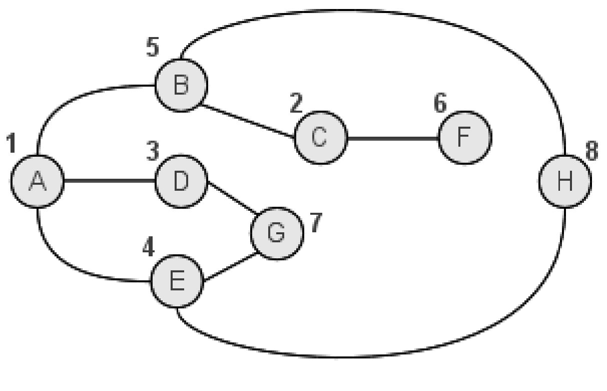
In recent years there has been a growing interest in studying graph layout problems where the main objective is to find a labeling of a graph in such a way that a specific objective function is maximized or minimized. The Linear Arrangement (Rodriguez-Tello et al. 2008), Bandwidth (Piñana et al. 2004) or Cutwidht (Pantrigo et al. 2011) 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 respectively, the set of vertices and edges, a labeling f of its vertices is a one-to-one mapping from the set V to the set $\{1,2, \dots n\}$ where each vertex $v \in V$ has a unique label $f(v) \in \{1,2, \dots, n\}$. Given the labeling f, the antibandwidth $AB_f(G)$ of graph G can be computed as:

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

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

The antibandwidth maximization problem (AMP) consists of finding a labeling f that maximizes $AB_f(G)$. This NP-hard problem was originally introduced in Leung et al. (1984) in connection with multiprocessor scheduling problems. AMP has also applications in relation to obnoxious facility location and radio frequency assignment (see Hale, 1980; Cappanera, 1999; Burkard et al., 2001; and Yixun and Jinjiang, 2003).



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)

Figure 1: (a) Graph example, (b) Antibandwidth of G for a labeling f.

Figure 1.a shows an example of an undirected graph with 8 vertices and 9 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 1.b 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. (2010) 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 breath 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 antibandwidth maximization problem. Section 2 introduces the three neighborhood structures and the associated local search methods in which our VNS method is based on. 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 Section 4 and concluding remarks are made in Section 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 Figure 1.a,

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

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 $f = (v_1, \dots, v_i, \dots, v_j, \dots, v_n)$, 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 $f' = (v_1, \dots, v_{i-1}, v_j, v_{i+1}, \dots, v_{j-1}, v_i, v_{j+1}, \dots, v_n)$. 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 have 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 $f = (v_1, \dots, v_{i-1}, v_i, v_{i+1} \dots, v_n)$ we have that:

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

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

We explore the associated neighborhood N_2 as we described above for N_1 (i.e., vertices are scanning in ascending order of their antibandwidth value and examining in search for an improving move). Given a vertex v_i, we first compute 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 $f' = (v_1, \dots, v_i, v_{i-1}, v_{i+1} \dots, v_n)$ 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. 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 scanning 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 pre-specified 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 straight forward 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 $\{N_1, N_2, \dots, N_{k_{max}}\}$ is available and the change between them is performed in a deterministic and sequential way. Our method (see Figure 2) implements a multi-start 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 (step 2), obtaining a labeling f, 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 construction 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, we do not restrict our attention to the objective function, but we also consider additional evaluators (described in the next subsections). In this broad sense has to be interpreted the step 6 of the algorithm (f' improves upon f).

```
PROCEDURE $VND(MaxIter, k_{max})$
  1.    FOR $i = 1, \dots, MaxIter$ 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
```

Figure 2. Pseudo-code of the VND method.

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, $L_1, \dots, L_s$, such that the endpoints of each edge in the graph are either in the same label 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 breath first search (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 this 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 3.a shows a spanning tree of the graph introduced in Figure 1.a. Figure 3.b depicts vertices in odd levels labeled sequentially. Finally, Figure 3.c shows the entire labeling, where the unlabeled vertices have been labeled in a greedy fashion.

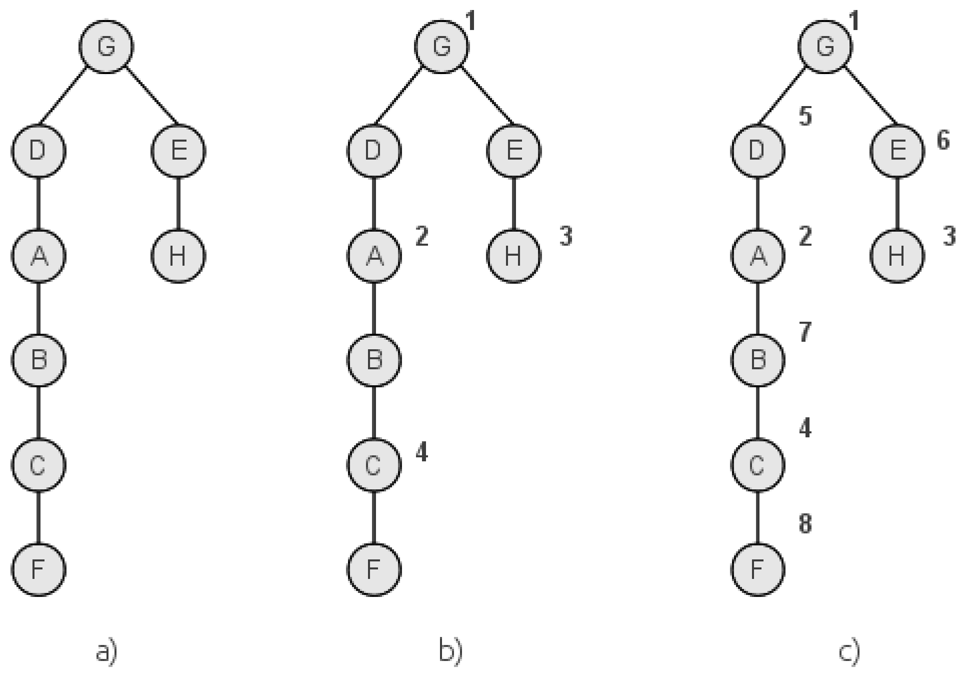


Figure 3: (a) Spanning tree, (b) Sequential labeling of odd levels, (c) Greedy labeling of remaining nodes.

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 a move improves the current solution if the number of vertices with a relative small antibandwidth value is reduced, regardless whether the objective function improves or not. 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, $S_i = \{v \in V \mid AB_f(v) = i\}$. For example, considering the graph depicted in Figure 1.a 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 $k \leq i$. In other words, if vertex v is removed from S_i and included in S_x with $x > i$. 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 4.a shows an improving move for the labeling of Figure 1.a. 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 Figure 1.a if we now exchange the labels of vertices G and H (see Figure 4.b) 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.

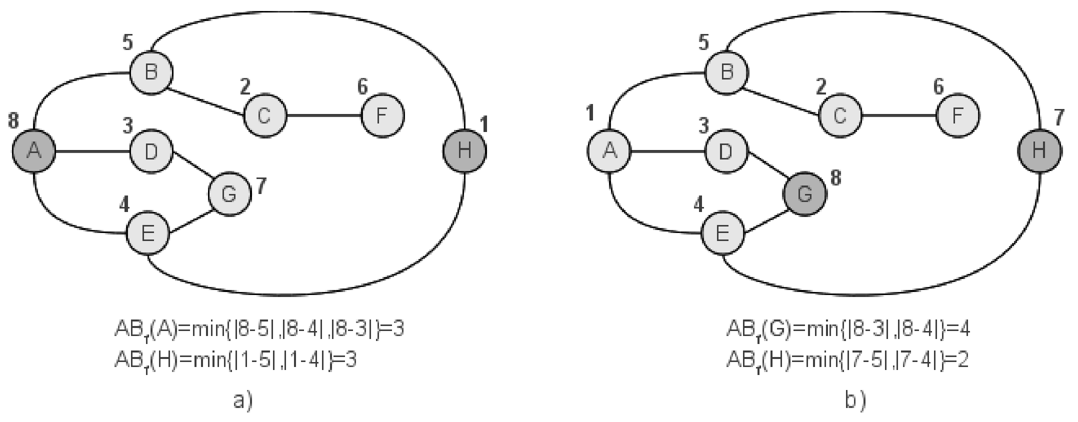

5
B
2
C
6
F
8
A
3
D
1
H
G
7
4
E
AB_f(A)=min{|8-5|,|8-4|,|8-3|}=3
AB_f(H)=min{|1-5|,|1-4|}=3
a)
5
B
2
C
6
F
1
A
3
D
7
H
G
8
4
E
AB_f(G)=min{|8-3|,|8-4|}=4
AB_f(H)=min{|7-5|,|7-4|}=2
b)

Figure 4: (a) Improving move, (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 Section 3 was implemented in Java SE 6 and all the experiments were conducted on an Intel Core 2 Quad CPU Q 8300 with 6 GiB of RAM and Ubuntu 9.04 64 bits OS. We have considered three sets with a total of 72 instances. They have been previously used in several linear layout problems (Pantrigo et al. 2011; Duarte et al. 2010). The first set, *Harwell-Boeing*, is a subset of the public domain Matrix Market library (Harwell-Boeing, 2011); the second one, *Grid graphs*, was introduced in Rolim et al. (1995) and the third one, *Hamming graphs*, was introduced in Dobrev et al. (2009). All these instances are available at http://www.optsicom.es/abp/. A detailed description of each set of instances follows:

Harwell-Boeing. We derived 24 instances from the Harwell-Boeing Sparse Matrix Collection (Harwell-Boeing, 2011). 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 $M_{ij} \neq 0$. We consider two subsets in the Harwell-Boeing set. 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 contains 12 larger instances with $n \in$ [400, 900]: 494bus, 662bus, 685bus, bcsstk07, bcsstk06, can445, can715, dwt503, dwt592, impcol.d, nos6, and sherman4.

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 consider 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]): mesh130x7, mesh120x8, mesh110x9, mesh100x10, mesh50x20, mesh40x25, mesh60x17, mesh34x30, mesh80x13, mesh70x15, mesh90x12, and mesh33x33.

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 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 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:

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

As in the previous sets, we consider 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`, `hamming3x3x4x5x5`, `hamming2x3x4x6x6`, `hamming2x2x5x6x7`, `hamming3x4x4x4x5`, `hamming3x3x4x4x7`, `hamming2x3x4x5x8`, `hamming3x3x4x5x6`, `hamming2x3x3x7x8`, `hamming2x3x5x6x6`, and `hamming3x4x4x4x6`.

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 Section 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 Section 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.

$\lvert \boldsymbol{W} \rvert$	**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

Table 1: Width of the Ejection Chain procedure.

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 $depth \in \{0.06n,\ 0.07n,\ 0.08n\}$. Table 2 shows the same statistics than above.

$depth$	Dev.	Score	Time
$0.06n$	24.93%	2	62.90
$0.07n$	24.83%	5	66.81
$0.08n$	24.83%	1	70.82

Table 2: Depth of the Ejection Chain procedure.

Table 2 clearly shows that the impact of d is less relevant than W. In this case, average deviation ranges from 24.93% (for $d = 0.06$) to 24.83% (for $d \geq 0.07$). We set $d = 0.06$ since the method requires shorter CPU time and the difference in average percentage deviation is insignificant.

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 Section 3). Table 3 summarizes the results of these four methods.

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

Table 3: Comparison of different neighborhoods.

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 Figure 5 how the average deviation value of these four methods improves over time (we consider a time horizon of 150 seconds reporting values every second).

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 tests 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.

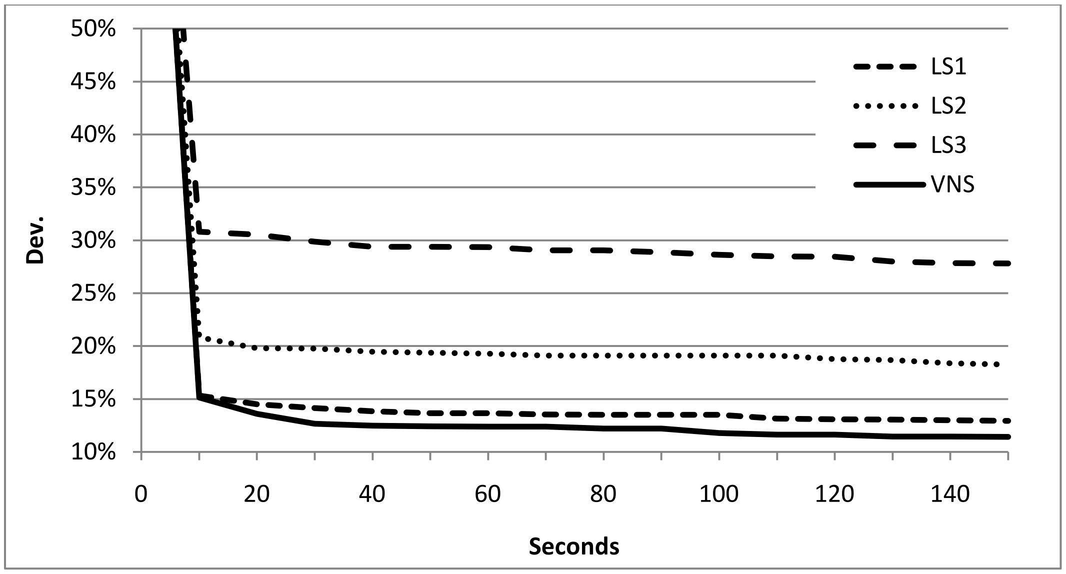


Figure 5: Average deviation of the best solution found over the time.

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

In the final experimentation we compare our VNS method with the best previous methods. As described in Section 1, they are the Memetic Algorithms, MA, proposed by Bansal and Srivastava (2011) and the Evolutionary Path Relinking, EvPR, due to Duarte et al., (2010). Table 4 presents the performance of each algorithm over the whole set of 36 small instances: 12 Harwell-Boeing instances with size n∈[30,100], 12 Grid graphs with $n \in$ [81, 120], and 12 Hamming graphs with $n \in [80,180]$.

Method	Value	Dev.	Score	Time
MA	25.77	15.86%	35	8.00
EvPR	26.45	11.97%	28	3.28
VNS	27.08	9.01%	17	1.38

Table 4: Comparison of state of the art methods for small instances.

The results presented in this table clearly show that our proposed VNS outperforms the previous competitors in both average percent deviation and score, consuming shorter running times. Specifically, VNS obtains an average deviation of 9.01% while EvPR obtains 11.97% and MA 15.86% investing less than a third of their CPU time. Note that in the case of Grid and Hamming graphs, deviations are computed with respect to the optimal value known by construction.

We conduct the same study over the set of 36 large instances: 12 Harwell-Boeing instances with size $n \in [400,900]$, 12 Grid graphs with $n \in [960, 1170]$, and 12 Hamming graphs with $n \in [840,1152]$. Table 5 compares the same methods reporting the same statistics.

Method	Value	Dev.	Score	Time
MA	209.33	40.16%	40	2734.87
EvPR	239.08	14.96%	45	670.30
VNS	247.49	11.58%	23	269.30

Table 5: Comparison of state of the art methods for large instances.

Results reported in Table 5 are in line with those shown in Table 4 and confirm that the VNS method outperforms previous algorithms. We applied the non-parametric Friedman test for multiple correlated samples to the best solutions obtained by each of the 3 methods on our entire set of 72 instances. This test computes, for each instance, the rank value of each method according to solution quality (where rank 3 is assigned to the best method and rank 1 to the worst one). Then, it calculates the average rank values of each method across all the instances solved. If the averages differ greatly, the associated p-value or significance will be small. The resulting p-value of 0.001 obtained in this experiment clearly indicates that there are statistically significant differences among the three methods tested. Specifically, the rank values produced by this test are 2.33 (VNS), 1.86 (EvPR), and 1.81 (MA).

In order to evaluate the behavior of these methods over a long-term time horizon, we run MA, EvPR and VNS for 30 minutes, 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.

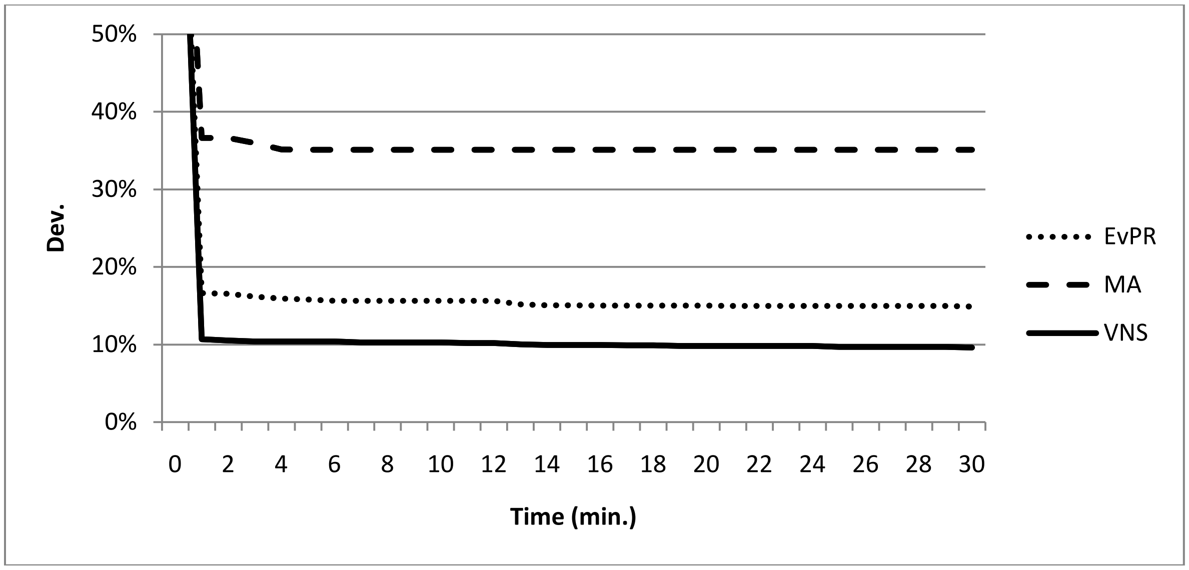


Figure 6: Performance profile for a 30 minutes.

Figure 6 shows 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.

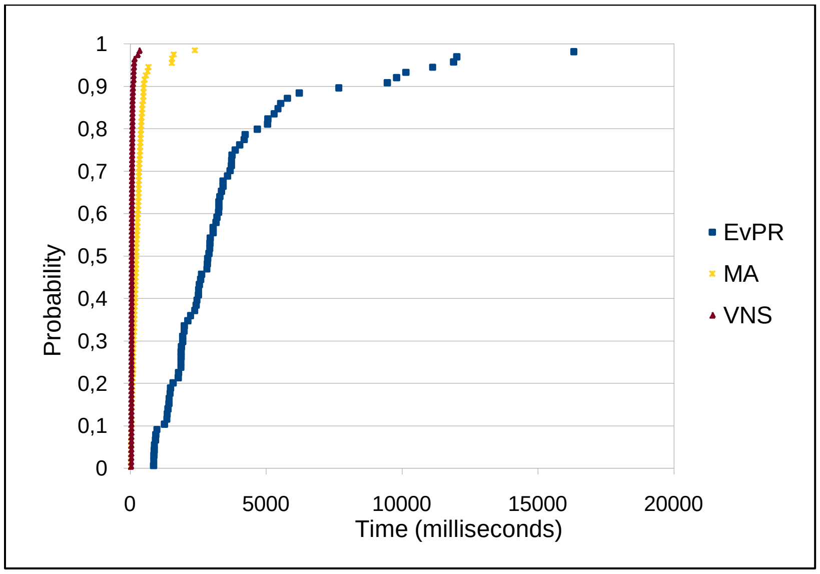


Figure 7: Time to target plots.

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 5x6x6, 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), shown in Figure 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 72 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. (2010) and Bansal and Srivastava (2011) respectively. The comparisons favor the proposed variable neighborhood search implementation.

Acknowledgments

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 and P08-TIC-4173.

References

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

Bansal, R., K. Srivastava (2009) "Memetic algorithm for the antibandwidth maximization problem". *Journal of Heuristics*, 17:39-60.

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

Cappanera, P. (1999) "A survey on obnoxious facility location problems". *Technical report* TR-99-11, Dipartimento di Informatica, Uni. Di Pisa.

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

Dobrev, S., R. Královic, D. Pardubská, L. Török, I. Vrt'o (2009) "Antibandwidth and cyclic antibandwidth of Hamming graphs". *Electronic Notes in Discrete Mathematics*, 34:295–300.

Duarte, A., R. Martí, M.G.C. Resende, R.M.A. Silva (2010) "GRASP with path relinking heuristics for the antibandwidth problem". *Networks*, In press.

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

Hale, W.K. (1980) "Frequency assignment: theory and applications". In *Proceedings of the IEEE*, 68: 1497–1514.

Hansen, P., N. Mladenovic, J. Brimberg, J.A. Moreno-Pérez (2010) "Variable neighborhood search". In *Handbook of Metaheuristics*, Springer. pp. 61-86.

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

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

Pantrigo, J.J., R. Martí, A. Duarte, E.G. Pardo (2011) "Scatter search for the cutwidth minimization problem". *Annals of Operations Research*. In press.

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

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

Resende, M., R. Martí, M. Gallego, A. Duarte (2010) "GRASP and path relinking for the max-min diversity problem". Computers and Operations Research, 37:498-508.

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

Rolim, J., O. Sykora, I. Vrt'o (1995) "Optimal cutwidths and bisection widths of 2- and 3-dimensional meshes". In Graph Theoretic Concepts in Computer Science. *Lecture Notes in Computer Science*, Springer, 1017: 252–264.

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

Yixun, L., Y. Jinjiang (2003) "The dual bandwidth problem for graphs". *Journal of Zhengzhou University*, 35:1–5.