Tabu Search for the Dynamic Bipartite Drawing Problem

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

ANNA MARTÍNEZ-GAVARA
Departamento de Estadística e Investigación Operativa,
Universidad de Valencia, Spain
gavara@uv.es

JESÚS SÁNCHEZ-ORO
Dept. Computer Sciences,
Universidad Rey Juan Carlos, Spain.
jesus.sanchezoro@urjc.es

ABRAHAM DUARTE
Dept. Computer Sciences,
Universidad Rey Juan Carlos, Spain.
Abraham.Duarte@urjc.es

Abstract

Drawings of graphs have many applications and they are nowadays well-established tools in computer science in general, and optimization in particular. Project scheduling is one of the many areas in which representation of graphs constitutes an important instrument. The experience shows that the main quality desired for drawings of graphs is readability, and crossing reduction is a fundamental aesthetic criterion to achieve it. Incremental or dynamic graph drawing is an emerging topic in this context, where we seek to preserve the layout of a graph over successive drawings. In this paper, we target the edge crossing reduction in the context of incremental graph drawing. Specifically, we apply a mathematical programming formulation and several heuristic methods based on the tabu search methodology to solve it. In line with the previous paper on this topic, we consider bipartite graphs in our experimentation. The extensive computational experiments with more than 1,000 instances show the superiority of our proposals in both, quality and computing time.

Key Words: graph drawing, incremental drawing, bipartite graphs, dynamic representations.

Version: 27th October, 2017

1. Introduction

Information systems are nowadays commonly represented with a drawing, which makes them easier to interpret and understand. Graphs are the basic modeling unit in a wide variety of areas, like project management, production scheduling, line balancing, business plans or software visualization. This is why graph drawing has become an important research area, with a large number of related publications. We refer the reader to the book by Di Battista et al. (1999) for a survey on graph drawing techniques. In Gibson et al. (2012) and Beck et al. (2016) more recent studies on drawing conventions, models and aesthetic criteria can also be found.

The selection of a measure or criterion to evaluate the quality of a graph drawing is somehow controversial given the many different approaches to this problem. However, the number of edge crossings is a widely-admitted criterion for evaluating the quality of a drawing. As stated by Carpano (1980), the fewer crossings the better the drawing. Since this seminal work, many authors have proposed crossing minimization methods to improve the readability of the drawing. In particular, Garey and Johnson (1983) proved that the problem of minimizing the number of crossings is NP-hard, and Purchase (2002) proved it to be one of the most important measures among seven aesthetic metrics to evaluate the quality of a graph drawing. This author performed a thorough aesthetic analysis of graph drawings produced by traditional layout algorithms, where crossing reduction emerged as a key objective.

Jünger and Mutzel (1997) presented several exact and heuristic algorithms for crossing minimization in bipartite graphs (also called 2-layered straight-line hierarchies). In particular, they compared the results obtained by fixing the ordering of the vertices in one layer and moving only the vertices in the other layer with the results of solving the general problem of moving (reordering) all the vertices in the graph. The authors also proposed a mathematical programming formulation to the general crossing minimization problem that we adapt in this paper to the dynamic case.

1.1 Previous Studies on the Dynamic Problem.

One of the most challenging areas in graph drawing is the one devoted to the so-called dynamic or incremental representations. As mentioned in Diehl and Görg (2002), in dynamic graph drawing we have to compute the layout of a graph evolving over time. A graph is modified by adding and deleting vertices and edges and we have to represent both, the original and the resulting graph. The drawing of the new graph, after the modifications, as an independent problem (i.e., from scratch) would be inefficient, since the graph has been slightly modified from the original drawing. As pointed out by Eades et al. (1991), the user has built up a mental map when reading the original drawing, so he or she expects the new graph to be represented in a similar way (layout) than the original one. This is why researchers in the area (see for example Branke, 2001), established that it would be advantageous to minimize the effort of the user to become familiar with a graph, i.e., to build the mental map. Therefore, minimizing the changes between the original and the new graph is a desired objective in dynamic graph drawing.

As it is well-known in this field, there are many different paradigms for graph drawing, being probably orthogonal and hierarchical the most popular ones. Special attention therefore deserves the paper by Görg et al. (2004), where drawing sequences of orthogonal and hierarchical graphs are studied. In this latter case of hierarchical graph, they proposed a way to capture the idea of preserving the mental map. Specifically, considering that the ordering of

vertices in each layer is responsible for the number of edge crossings, Görg et al. (2004) considered to preserve the relative order of the original vertices in the corresponding layer. Martí and Estruch (2001) proposed independently the same criterion to reflect the idea of stability across drawings: keep the relative ordering among the common vertices. These authors also proposed exact and heuristic methods to obtain solutions to this hard optimization problem. We follow these two works in our approach, and focus in this paper on preserving the relative order of the original vertices when drawing the new graph.

Di Battista et al. (1999) used the term *incremental construction* in the context of planar graphs. Martí and Estruch (2001) introduced the term *incremental graph drawing* to describe their problem on 2-layered graphs. Other authors, such as Branke (2001) and Görg et al. (2004), used the term *dynamic graph drawing* to refer to the same type of problems. We propose to call this problem of incremental or dynamic edge crossing minimization in 2-layered graphs, simply as Dynamic Bipartite Drawing Problem (DBDP). Several algorithms were developed to handle dynamic graphs. For example, Diehl and Görg (2002); Kumar and Garland (2006); and Sallaberry et al. (2012) presented algorithmic techniques in the context of clustering dynamic graphs. Martí and Estruch (2001) proposed an exact procedure to target this NP-hard problem based on the branch and bound methodology, which explores the set of solutions (permutations of the vertices in each layer) with the so-called search tree. This method provides the optimal solution for small size instances of up to 32 vertices. The authors also proposed a heuristic based on Greedy Randomized Adaptive Search Procedure (GRASP) to solve large size instances.

More recently, Burch et al. (2011) presented an eye tracking study for evaluating the quality of node-link tree layout representing hierarchies. They concluded that radial representations are the most space-efficient one, but they usually result in drawings that are difficult to interpret. These authors recommended a traditional tree-diagram with the root on the top, which is similar to bipartite graphs considered in this paper. Additionally, Van den Eltzen et al. (2013) developed an extension for Massive Sequence Views with the aim of analyzing the temporal and structural aspects of dynamic networks. This study allows the user to find anomalies in the network and analyze temporal properties. More recently, Burch et al. (2017) proposed a novel visualization technique for graphs with considerably large number of time steps (more than a thousand, as stated by the authors).

In this paper, we limit our attention to hierarchical graphs, where vertices are arranged in layers (drawn in parallel lines) and edges are drawn as straight lines. In line with Martí and Estruch (2001), we consider the case of two layers (bipartite graphs), where nodes and edges have been added to an original graph already drawn for crossing minimization. The problem is then to insert the new nodes (and the corresponding edges) in the appropriate positions in order to minimize the total number of edge crossings in the final graph. As mentioned above, the relative order among the original nodes is kept. We compare our method, based on tabu search, with their GRASP algorithm on a large set of instances, as well as with the optimal solution for small size instances.

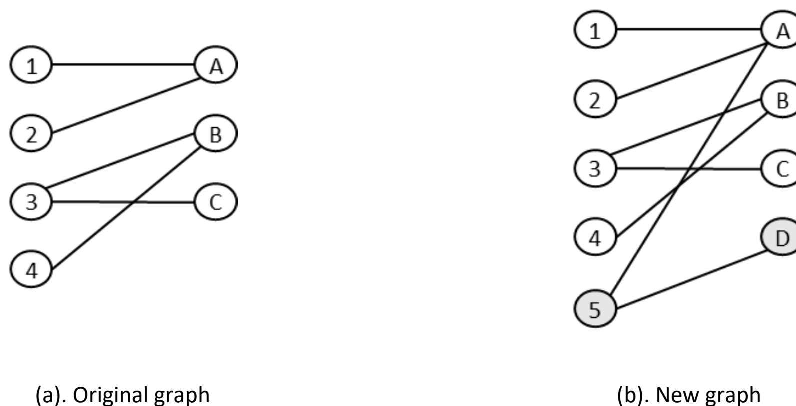
Note that our method can be applied to the general case of a sequence of drawings, not only to the 2-step problem with an original drawing and an incremented one. As a matter of fact, the DBDP can be easily extended to a sequence of n drawings by simply performing one-step optimization and then fix for the next step the new added nodes in the position obtained by the tabu search algorithm. Then, in the next step, we consider the new set of nodes as those that can be moved, and the ones added in the previous iterations as fixed that cannot be moved

again. The next subsection shows an example of a sequence of drawings to illustrate the general application of our method.

1.2 Dynamic Graph Drawing Applications

We can find many applications of dynamic graph drawing in Management Science. In the case of general digraphs, Project Management is probably one of the most well-known areas where this problem finds a very useful application. It has been well document that many changes occur during the development of a large project and they have to be reflected in the associated graph or chart. Dynamic graph drawing is a demand of project managers who need a stable sequence of drawings as the project evolves.

In the context of bipartite graphs, the well-known assignment problem provides interesting applications of dynamic graph drawing. Figure 1 illustrates a so-called affiliation network, where individuals and groups are represented with nodes, and edges represent membership of individuals to that groups. Affiliation networks usually change during the time, since new groups and members are systematically added. Figure 1a shows the original graph in which we can see that individuals 1 and 2 belong to group A, and individual 3 belongs to groups B and C. Figure 1b shows the same network at a later stage where some additions have been performed (nodes highlighted in gray). Specifically, we can see a new individual, labeled as 5, belonging to group A. Additionally, a new group, labeled as D, has also been included. Note that the edge from 5 to A creates 3 crossings in this new graph. It is worth mentioning that the new graph preserves the mental map of the original graph since the original vertices have not been moved. The challenge in this context is therefore to minimize the number of crossings while preserving the mental map.



(a). Original graph (b). New graph

Figure 1. Dynamic graph drawing illustration.

Assignment problems find nowadays many applications in Computer Science. For example, queries on online advertisement (Antonellis et al., 2008) have to be represented as a sequence of graphs for their analysis. In these bipartite graphs, the left layer represents queries performed by users, while the right layer represents advertisements (ads). A link between a query and an ad indicates that a query has been used to reach a specific ad. This kind of bipartite graphs are naturally dynamic, since users are continuously doing queries, and new ads are included by the companies. However, it is recommended by data analyzers to maintain the structure of the graph when new queries and ads are added, so previously performed queries and old ads remain in the same relative position.

Figure 2 shows a sequence of query-advertisement graphs with 9 original queries (numbered from 0 to 8) connected with 9 ads (numbered from 10 to 18). The first drawing in the left of Figure 2 shows the original 18 vertices with black circles where a new query and two ads are added (nodes 9, 19, and 20 represented with white circles). This drawing has 80 edge crossings.

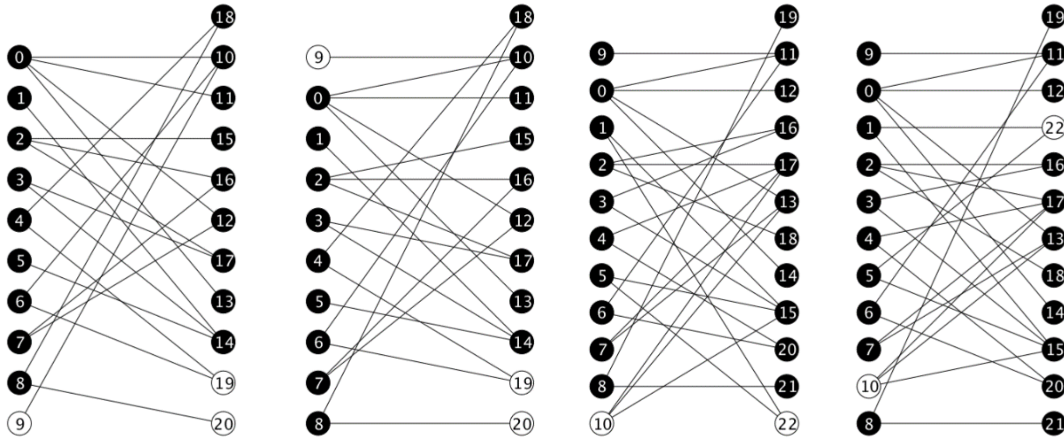


Figure 2. Dynamic graph drawing sequence.

The left drawing in Figure 2 is optimized according to the dynamic graph drawing problem (i.e., by relocating the new three nodes in the position that minimizes the number of crossings while preserving the relative position of the original nodes), resulting in the drawing next to it with 67 edge crossings. In the third drawing (starting from the left) two new vertices are added to the graph (vertex 10 representing a query and vertex 22 representing an ad), while the vertices added in the previous step are now represented with black color since they are now fixed and cannot be moved anymore. This drawing has 111 cuts. To simplify the notation, the label of the vertices in the first layer is always smaller than the ones in the second layer, so node renumbering is made on consecutive drawings. Finally, in the last drawing (the one in the right side of Figure 2) the algorithm is executed again obtaining a drawing with 101 cuts.

In this paper, we first adapt the linear integer formulation proposed for the Bipartite Drawing Problem (Jünger and Mutzel, 1997) to the dynamic variant considered here (Section 2). Then, we first describe the previous heuristic method for this problem (Section 3), and describe our algorithm based on the tabu search methodology (Section 4). In particular, we consider a greedy randomized construction with a short-term tabu search improvement method. We also consider several hybridizations of these methodologies with a path relinking post-processing (Section 5). Our extensive experimentation with more than 1,000 instances shows the superiority of this proposal with respect to the previous method (Section 6). The paper ends with the associated conclusions and future works (Section 7).

2. Mathematical formulation

Jünger and Mutzel (1997) proposed the following linear integer formulation of the bipartite drawing problem (BDP). Let $G=(V_1,V_2,E)$ be a bipartite or 2-layered graph, with $|V_1|=n_1$ and $|V_2|=n_2$. A drawing D (BDP solution), is determined by the ordering π_1 of V_1, and the ordering π_2 of V_2, and it is denoted in mathematical terms as $D=(G,\pi_1,\pi_2)$. The ordering π_1 (symmetrically π_2) assigns a distinct integer from 1 to n_1 (symmetrically to n_2) to each node $v\in V_1$ ($w\in V_2$). For the sake of simplicity, we denote with π the ordering of all vertices of the graph. The authors introduced the binary variable x_{ik} that takes the value 1 if vertex i precedes

vertex k in the ordering of V_1 ($\pi_1(i) < \pi_1(k)$), and 0 otherwise. Symmetrically, for a pair of vertices $l, j \in V_2$, the binary variable y_{lj} takes the value 1 when l precedes j ($\pi_2(l) < \pi_2(j)$).

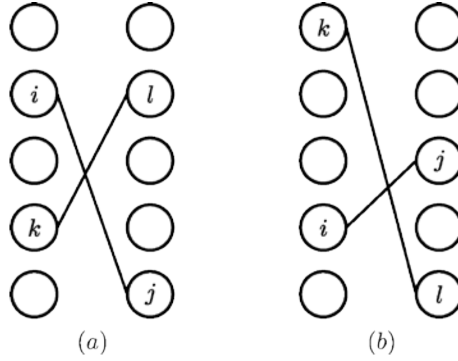


Figure 3. Examples of crossings.

As shown in Figure 3, a crossing between the edges (i, j) and (k, l) takes place when $\pi_1(i) < \pi_1(k)$ and $\pi_2(l) < \pi_2(j)$ (see Figure 3a), or $\pi_1(i) > \pi_1(k)$ and $\pi_2(l) > \pi_2(j)$ (see Figure 3b). Jünger and Mutzel (1997) introduced the binary variable c_{ijkl} that takes the value 1 when a crossing between these two edges occurs. Note that constraints (1) and (2) below force c_{ijkl} to take the value 1 when the variables x_{ik} and y_{lj} indicate a crossing.

(BDP) $Min \sum_{(i,j),(k,l)\in E} c_{ijkl}$

$$x_{ik} + y_{lj} - c_{ijkl} \leq 1 \quad (i, j), (k, l) \in E, i < k, j \neq l \qquad (1)$$

$$x_{ki} + y_{jl} - c_{ijkl} \leq 1 \quad (i, j), (k, l) \in E, i < k, j \neq l \qquad (2)$$

$$x_{ij} + x_{jk} + x_{ki} \leq 2 \quad 1 \leq i < j < k \leq n_1 \qquad (3)$$

$$y_{ij} + y_{jk} + y_{ki} \leq 2 \quad 1 \leq i < j < k \leq n_2 \qquad (4)$$

$$x_{ij} + x_{ji} = 1 \quad 1 \leq i < j \leq n_1 \qquad (5)$$

$$y_{ij} + y_{ji} = 1 \quad 1 \leq i < j \leq n_2 \qquad (6)$$

$$x_{ij}, y_{ij}, c_{ijkl} \in \{0,1\}$$

Constraints (3) and (4) are the so-called 3-dicycle constraints, originally proposed for the Linear Ordering Problem by Grötschel et al. (1984), and adapted by Jünger and Mutzel (1997) to the bipartite drawing problem. Note that in this formulation only a fraction of the large number of 3-dicycle inequalities are included (those with $1 \leq i < j < k \leq n$). These authors solved the formulation above with Cplex 4.0 and proposed a branch-and-cut approach. Within a time limit of 3,600 seconds on a SUN ULTRA2 (167MHZ) workstation they were able to solve to optimality small instances (with up to 44 vertices).

Given a bipartite graph $G = (V_1, V_2, E)$ and a drawing $D = (G, \pi_1, \pi_2)$, we can consider the addition of some nodes and edges as described in the introduction, obtaining an incremental graph. Martí and Estruch (2001) introduced it formally as $IG = (IV_1, IV_2, IE)$ where $V_1 \subseteq IV_1$, $V_2 \subseteq IV_2$, and $E \subseteq IE$ ($|IV_1| = m_1, |IV_2| = m_2$). A drawing $ID = (IG, \varphi_1, \varphi_2)$ is an incremental drawing of $D = (G, \pi_1, \pi_2)$ if the original vertices (those in $V = V_1 \cup V_2$) preserve their relative ordering. In mathematical terms:

$$\varphi_i(v) < \varphi_i(w)\ \ \forall v, w \in V_i \iff \pi_i(v) < \pi_i(w)\ \ i = 1,2. \quad (7)$$

For the sake of brevity, we denote with φ the ordering of the whole set of vertices of the incremental graph.

We can easily adapt the mathematical formulation above to tackle the dynamic bipartite drawing problem. In the DBDP, the relative position between each pair of original vertices (those in V) is already set; therefore, the associated x-variables or y-variables, depending on which layer they belong, can be fixed in the model. Similarly, the c_{ijkl} variables with $(i,j),(k,l) \in E$ can be set as well.

(DBDP) $\quad Min \sum_{(i,j),(k,l)\in IE} \quad c_{ijkl}$

$$x_{ik} + y_{lj} - c_{ijkl} \le 1 \qquad (i,j),(k,l) \in IE, i < k, j \ne l \qquad (8)$$

$$x_{ki} + y_{jl} - c_{ijkl} \le 1 \qquad (i,j),(k,l) \in IE, i < k, j \ne l \qquad (9)$$

$$x_{ij} + x_{jk} + x_{ki} \le 2 \qquad i,j,k \ \in IV_1, i < j, i < k, j \ne k \qquad (10)$$

$$y_{ij} + y_{jk} + y_{ki} \le 2 \qquad i,j,k \ \in IV_2, i < j, i < k, j \ne k \qquad (11)$$

$$x_{ij} + x_{ji} = 1 \qquad 1 \le i < j \le m_1 \qquad (12)$$

$$y_{ij} + y_{ji} = 1 \qquad 1 \le i < j \le m_2 \qquad (13)$$

$$x_{ij} = 1 \qquad i,j \in V_1 : \pi_1(i) < \pi_1(j) \qquad (14)$$

$$y_{ij} = 1 \qquad i,j \in V_2 : \pi_2(i) < \pi_2(j) \qquad (15)$$

$$x_{ij}, y_{ij}, c_{ijkl} \in \{0,1\}$$

Constraints (8) – (13) are straightforward adaptations of (1) – (6) in the BDP formulation described above. New constraints (14) and (15) preserve the ordering of the original vertices. We have empirically found that when (14) and (15) are included in this model, we also need to include all the 3-dicycle constraints (now (10) and (11)) as in the original linear ordering formulation. We will apply this model in our computational experience in Section 6.

3. Previous method

Martí and Estruch (2001) proposed a GRASP method for the DBDP. The GRASP methodology (Festa and Resende, 2011) is based on the statistical sampling of the solution space. The randomization component in the construction has the objective of obtaining relatively diverse solutions, thus having candidate solutions in different regions of the search space. These solutions are then submitted to a local search post-processing to obtain the so-called local optima. We now describe in detail their heuristic method.

Constructive method

The constructive method by Martí and Estruch (2001) starts by creating a list CL of unassigned vertices which, at the beginning, contains all the vertices of the graph. The first vertex v is randomly selected from all those vertices in CL with maximum degree. As it is customary in GRASP, in subsequent construction steps, the next vertex v is randomly selected from a restricted candidate list, RCL, which consists of those vertices in CL with a degree of no less than α times

the maximum degree in CL. Vertex degree is calculated with respect to the partial subgraph under construction (i.e., in which only those vertices previously located are considered).

A selected vertex v is placed in its layer in the position prescribed by the barycenter calculation (Di Battista et al., 1999), except for the first vertex, which is placed in an arbitrary position. The barycenter of a vertex $v \in IV_1$, $bc(v)$, is the arithmetic mean of the current positions of the vertices $w \in IV_2$ adjacent to v (similarly for the barycenter of a vertex in IV_2). If vertex v belongs to the original graph (i.e., $v \in V_1 \cup V_2$), then it can be allocated in positions that are feasible in terms of the original ordering π. In other words, this construction is creating an ordering φ that, as mentioned before, must verify that $\varphi_i(v) < \varphi_i(w)$ for all vertices w such that $\pi_i(v) < \pi_i(w)$. Then, v is placed in the closest *feasible* position to $bc(v)$ with respect to π_i. The method finishes when all the vertices have been allocated.

Local search method

Each step of the improvement phase is based on a probabilistic selection of the vertices, in order to place them in the position that produces the maximum reduction in the number of crossings. The probability $\mathrm{P}(v)$ that a vertex v is selected is proportional to its degree, $\rho(v)$ (i.e., higher degree vertices are more likely to be selected):

$$\mathrm{P}(v) = \frac{\rho(v)}{\sum_{u \in IV} \rho(u)}$$

Then, if v^* is the selected vertex, it is placed in the position that produces the minimum number of crossings considering these three moves: to insert the vertex one position before the barycenter ($\lfloor bc(v^*) \rfloor - 1$), to insert the vertex at the barycenter position ($\lfloor bc(v^*) \rfloor \; or \; \lceil bc(v^*) \rceil$), and finally to insert the vertex one position after the barycenter ($\lceil bc(v^*) \rceil + 1$). As in the constructive method, the procedure is limited to perform feasible moves with respect to the original ordering π. Therefore, if $v^* \in V$, then its new position must be feasible according to constraint (7) of the mathematical programming formulation (i.e., the relative position of v^* in the original drawing). The improvement phase finishes when all vertices $v \in IV$ are considered, and no improving move is found.

4. A Hybrid tabu search method

It must be noted that the construction method described above computes the positions of all vertices, original and added ones. In our view, it involves a relatively large computational effort, considering that in the DBDP the ordering among the original vertices has to be kept. We therefore propose an alternative method in which we consider the vertices in V_1 and V_2 already allocated (according to π_1, π_2 respectively), and explore where to allocate the new vertices to complete the solution.

4.1 Constructive method

The construction phase starts by considering the original drawing D as a partial solution. In the constructive method by Martí and Estruch (2001), a selected vertex is placed in the position prescribed by the barycenter. We propose here a different approach; in which we explore all the possible insertions for a selected vertex.

Let NV be the set of new vertices (i.e, those added to the original graph). In mathematical terms, $NV = IV \setminus V$, where $IV = IV_1 \cup IV_2$ and $V = V_1 \cup V_2$. The constructive method basically selects an element in NV and inserts it in the partial solution under construction (initially D). The procedure evaluates each candidate element v with a greedy function $g(v)$ in order to identify the best elements, and adds them to the Restricted Candidate List (RCL), where one of them will be randomly selected as it is customary in GRASP (Resende and Ribeiro, 2001). Initially, the candidate list consists in the set of new vertices ($CL = NV$). In subsequent iterations, when a vertex v is selected and inserted in the partial solution, the candidate list is updated by removing it ($CL = CL \setminus \{v\}$).

We propose the following greedy function to compute the increment in the objective function if a vertex $v \in CL$ is added to the partial solution. We define $C(v,p)$ as the number of crossings generated by inserting vertex v in position p (in its corresponding layer) in the partial solution. In other words, if the partial solution has c crossings and we insert v in position p, we obtain a partial solution with $c + C(v,p)$ crossings. We examine all the positions to insert v and select the best one, p^*, minimizing the number of crossings:

$$g(v) = C(v,p^*) = \min_{p} C(v,p), \tag{16}$$

In the case that several positions have the same minimum C-value, the position p^* is selected at random among them. We compute $g(v)$ for all the candidate vertices (in CL), and build RCL with those that, according to the greedy function, achieve a relatively low increment in the objective function value, $RCL = \{v \in CL \colon g(v) \leq \tau\}$, where

$$\tau = \min_{v \in CL} g(v) + \alpha \left(\max_{v \in CL} g(v) - \min_{v \in CL} g(v) \right) \tag{17}$$

and α is a search parameter that we will empirically set in our experimentation (see Section 6). The process continues until all new vertices are included in the partial solution, thus obtaining a complete solution that we call incremental drawing ID. This is the standard GRASP design in which, in short, we can say that we first apply greediness and then randomness.

4.2 Tabu Search

Instead of the standard local search improvement method, we coupled our GRASP construction with a short term tabu search (TS). This methodology (Glover and Laguna, 1997) is a metaheuristic that guides a local search procedure to explore the solution space beyond local optimality. One of the main components of TS is the use of adaptive memory, which creates a flexible search behavior. Tabu search begins in the same way as ordinary local or neighborhood search, proceeding iteratively from one solution to another. Each solution, called ID in our problem, has an associated neighborhood $N(ID)$, containing the solutions $ID' \in N(ID)$ that can be reached from ID by an operation called a move. We may contrast TS with a simple descent local search method that only permits moves to neighbor solutions that improve the current objective function value, ending when no further improvement is possible. On the contrary, TS permits moves that deteriorate the current objective function value. Moves are chosen from a modified neighborhood $N^*(ID)$, which is the result of maintaining a selective history of the states encountered during the search. In this section, we limit ourselves to a short-term memory design, which specifies to record recent information (usually solution or moves attributes) to

exclude certain solutions to become part of $N^*(ID)$. We refer the reader to Glover and Laguna (1997) for further details about this methodology and successful applications.

Given a solution ID, we propose a neighborhood $N(ID)$ based on moving a vertex $v \in NV$ to a new position. Note that we only consider moving new vertices, since original vertices cannot change their relative position. In particular, we define $move^-(ID, v)$ to insert vertex v in a previous position to its current one. In other words, if vertex u precedes v in its layer, this move swaps u and v. Similarly, we define $move^+(ID, v)$ to insert vertex v in a posterior position to its current one (i.e., if vertex w succeeds v in its layer, this move swaps v and w).

Given two vertices u and v in IV_1 (usually called the left layer) and a drawing ID, let l_{uv} be the number of crossings between the edges incident to u and the edges incident to v, when u precedes v in its layer (i.e., when $\varphi_1(u) < \varphi_1(v)$). Note that this value depends on the ordering of their adjacent vertices in IV_2. Similarly, given two vertices u and v in IV_2 (usually called the right layer), we define r_{uv} as the number of crossing between their incident edges when u precedes v (and r_{vu} when v precedes u).

To record this information, we define two matrices L (left layer) and R (right layer) with the number of edge crossings between two vertices as described above:

$$L = (l_{uv}) \quad \forall u, v \in IV_1 \tag{18}$$

$$R = (r_{uv}) \quad \forall u, v \in IV_2 \tag{19}$$

Figure 4 shows these two matrices for the example depicted in Figure 1, considering the ordering shown there. Specifically, we can see in the graph drawing showed in Figure 1b that the number of crossings between the edge incident to vertex 4, (4,B), and the two edges incident to vertex 5, (5,A) and (5,D), is 1, since edge (4, B) crosses edge (5,A). Therefor, $l_{45} = 1$, as shown in matrix L (row 4, column 5) of Figure 4. It is also easy to see in Figure 1b that if we swap the position of these two contiguous vertices, we obtain a new drawing in which edges (4,B) and (5,A) are not crossing anymore but, on the other hand, edges (4,B) and (5,D) are crossing now. This is why, in Figure 4, matrix L (row 5, column 4) has $l_{54} = 1$.

$$L = \begin{bmatrix} - & 0 & 0 & 0 & 0 \\ 0 & - & 0 & 0 & 0 \\ 2 & 2 & - & 1 & 2 \\ 1 & 1 & 0 & - & 1 \\ 1 & 1 & 2 & 1 & - \end{bmatrix} \qquad R = \begin{bmatrix} - & 2 & 1 & 0 \\ 4 & - & 1 & 0 \\ 2 & 0 & - & 0 \\ 2 & 2 & 1 & - \end{bmatrix}$$

Figure 4. Number of edge crossing between pairs of nodes.

Given a solution ID and a new vertex $v \in IV_1$, we evaluate the change in the number of crossings if $move^-(ID, v)$ is performed, as $move_value^-(ID, v) = l_{uv} - l_{vu}$, where u is the vertex immediately preceding v in IV_1. If $move_value^-(ID, v) > 0$, it indicates that this is an improving move since the number of crossings of the edges incident with these two vertices in the current solution l_{uv} is larger than the number of crossings of these edges if we swap the vertices (l_{vu}). In short, the move reduces the number of crossings. A key aspect in this computation is that when we swap two consecutive vertices in a layer while keeping the ordering of the vertices in the other layer fixed, the change in the total number of crossing only depends on this amount. In mathematical terms, if $C(ID)$ is the total number of edge crossings of drawing (or solution) ID, and we perform $move^-(ID, v)$, the total number of crossings of the resulting solution is:

$$C(ID) - move_value^-(ID, v). \qquad (20)$$

In a similar way, we evaluate $move^+(ID, v)$ with the expression $move_value^+(ID, v) = l_{vw} - l_{wv}$, where w is the vertex immediately after v in IV_1. As in the previous case, if the move value is positive, it indicates that if we apply the move, we will obtain a solution with a lower number of crossings.

The move and the move value above were introduced for a vertex in the left layer (IV_1) and, in a similar way, we now define the move and its associated value for a vertex in the right layer (IV_2). Given a solution ID, the neighbourhood $N(ID)$ consists of all the solutions that can be obtained by inserting a new vertex in a previous or posterior position in its layer. Mathematically,

$$\forall v \in NV \text{ we consider } move^+(ID, v) \text{ and } move^-(ID, v)$$

and we select the best of them as the move to be performed. Note that if v is the first vertex in its layer, we can only consider one move for it ($move^+(ID, v)$). Symmetrically, if it is the last one, only $move^-(ID, v)$ can be considered. We implement the so-called best strategy, in which we explore the neighborhood of a solution and select the best solution in it.

In the example above (Figure 1b), we added two new vertices, $NV = \{5, D\}$, to the graph shown in Figure 1a. Since both vertices were in the last position of each layer respectively, it is only possible to move them to a previous position (i.e., only $move^-(ID, v)$ are feasible). It is easy to compute their associated move values as: $move_value^-(ID, 5) = -1 - 1 = 0$ and $move_value^-(ID, D) = 0 - 1 = -1$. The neighborhood $N(ID)$ is formed with these two moves (inserting 5 in a previous position in the left layer, and inserting D in a previous position in the right layer), and the best one is $move^-(ID, 5)$ with a move value of 0, which indicates that the number of crossings does not change if we apply it. In mathematical terms, the total number of crossings of the resulting solution is computed as $C(ID) - move_value^-(ID, 5) = 4$, as we can confirm in Figure 5.

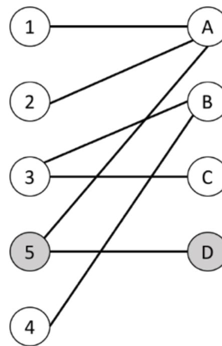


Figure 5. New solution after a move.

Figure 5 shows the resulting solution after applying $move^-(ID, 5)$, where vertex 5 is now in position 4 in the left layer. An interesting implementation detail of our algorithm is the update of the matrices L and R, which store the number of crossings, after performing a move. As mentioned above, the number of crossings computed for the vertices in the left layer, stored in L, only depends on the ordering of the vertices in the right layer. Then, since the vertices in the right layer did not change their position when applying this move, matrix L does not change. We only need to update matrix R.

Considering that we have moved two consecutive vertices, u and v, in the left layer IV_1, to update the matrix R we need to check the vertices adjacent to them. Without loss of generality, assume that initially u preceded v, and after performing the move then v precedes u. Thus, for each vertex a adjacent to u, and each vertex b adjacent to v, we have to increase r_{ab} by one unit, and decrease r_{ba} by one unit (i.e., $r_{ab} = r_{ab} + 1$ and $r_{ba} = r_{ba} - 1$).

In our example in Figure 5, $v = 5$ is in the left layer, so when performing the move matrix L does not change. We have to add or subtract one unit to the elements in matrix R corresponding to vertices incident to $u = 4$ and $v = 5$. B is the adjacent vertex to vertex $u = 4$, and A and D the adjacent vertices to vertex $v = 5$. The new matrix is the result of adding 1 to the elements r_{BA} and r_{BD}, and subtracting 1 to the elements r_{AB} and r_{DB}. Figure 6 shows the new matrix R. Note that only these four elements need to be updated.

$$R = \begin{bmatrix} - & 1 & 1 & 0 \\ 5 & - & 1 & 1 \\ 2 & 0 & - & 0 \\ 2 & 1 & 1 & - \end{bmatrix}$$

Figure 6. Crossing matrix after the move.

Although moving an element to its consecutive position is somehow limited, our neighbor is formed by all the solutions that can be reached by moving any new element to the immediate previous or posterior position, to select the best of them. Therefore, the size of the neighborhood is large enough to permit an efficient exploration of the search space. Additionally, as shown above, moving continuous vertices permit a fast update of the information required to evaluate moves, and to compute the objective function in an incremental way (see Eq. 20).

We include a memory structure in the local search algorithm to create a short-term tabu search method. In particular, when we select a new vertex v and move it, we record in $tabu(v)$ the number of the current iteration, in order to prohibit to move it in the next iterations. In this way, in a given iteration $iter$, we only permit to select a new vertex u for movement if the following condition is met:

$$iter - tabu(u) > tenure \tag{21}$$

where $tenure$ is a search parameter specifying the number of iterations that a *tabu* element cannot be selected. After these number of iterations, the tabu status of u is released, and it can be selected again to be moved. As it is customary in tabu search, we modify the neighborhood described above by excluding from it those solutions involving to move a tabu element. An important characteristic of tabu search is that the best solution in the neighborhood is always performed, even if it deteriorates the objective function (i.e., if the number of crossing increases). However, it is well documented that over a medium to large number of iterations this strategy permits to visit high-quality solutions. The tabu search method terminates after a specific number of iterations.

5. A Path relinking post-processing

Path relinking (Glover and Laguna, 1997) generates new solutions by exploring trajectories that connect elite solutions by starting from one of these solutions, called an *initiating solution*, and generating a path in the neighborhood space that leads toward the other solutions, called *guiding solutions*. This is accomplished by selecting moves that introduce attributes contained

in the guiding solutions. Note that in standard local search, the move selection in the neighborhood is typically guided by the objective function (i.e., the method usually explores the entire neighborhood in search for an improving move). In path relinking (PR) however, the neighborhood of a solution is limited to the solutions that contain attributes present in the guiding solution. Therefore, although we consider the objective function when selecting a move, the primary objective is to get closer to the guiding solution. PR subordinates all other considerations to the goal of choosing moves that introduce the attributes of the guiding solutions, in order to create a "good attribute composition" in the current solution.

The approach may be viewed as an extreme strategy that seeks to incorporate attributes of high quality solutions, by creating inducements to favor these attributes in the moves selected. In evolutionary terms, we can call this strategy a *combination method*, since the final output is a set of solutions (those in the path) that can be viewed as the result of combining the initiating and the guiding solutions (called reference solutions).

Path relinking gives a natural foundation for developing intensification and diversification strategies. Intensification strategies in this setting typically choose reference solutions to be elite solutions that lie in a common region or that share common features. Similarly, diversification strategies, based on path relinking characteristically, select reference solutions that come from different regions or that exhibit contrasting features. Diversification strategies may also place more emphasis on paths that go beyond the reference points.

Laguna and Martí (1999) adapted PR in the context of GRASP as a form of intensification. The relinking, in the context of multi-start algorithms, consists in finding a path between two solutions generated with the constructive method and, eventually, improve the solution in the path with a local search. Therefore, the relinking concept has a different interpretation within GRASP, since the solutions are not originally linked by a sequence of moves. The authors, however, kept the original name of the methodology in spite of the fact that the two solutions are linked for the first time. Resende et al. (2010) explored different implementations to hybridize these two methodologies:

- Greedy path relinking. In this method, the moves in the path from a solution to another one are selected in a greedy fashion, according to the objective function value.
- Greedy randomized path relinking. In this variant, the method creates a candidate list with the good intermediate solutions and randomly selects among them.
- Truncated path relinking. In this application of PR, the path between two solutions is not completed. It is applied, for example, in problems where good solutions are found close to the end points (original solutions) in the path.

In this paper, we consider the greedy randomized path relinking that has given excellent results in previous methods (Resende et al., 2010). Let $ID_x = (IG, \varphi_1^x, \varphi_2^x)$ and $ID_y = \left(IG, \varphi_1^y, \varphi_2^y\right)$ be two solutions of our problem. They are incremental drawings of the original drawing $D = (G, \pi_1, \pi_2)$. The path relinking procedure starts with the first solution ID_x, called *initiating solution*, and gradually transforms it into the second solution ID_y, called *guiding solution*, by selecting a new element in NV and inserting it in the position that occupies in ID_y. Once a new element has been selected and inserted, we do not select it again. When all the new elements have been selected, the method finishes, since the path has reached the guiding solution.

Let $ID_0 = ID_x$ be the initiating solution in the path. As mentioned, we consider for each new vertex, $v \in CL = NV$, its insertion in its position according to the guiding solution ($\varphi_1^y(v)$ if v is

in the left layer, and $\varphi_2^y(v)$ if it is in the right layer). Let $pr(ID_0, v)$ be the move value (change in number of crossings) of performing this insertion in ID_0. If $C(ID_0)$ is the total number of edge crossings of the initial solution in the path, and we move v as indicated above, the number of crossings of the resulting solution ID_1 is $C(ID_1) = C(ID_0) - pr(ID_0, v)$. Then, if $pr(ID_0, v) > 0$, it indicates that this is an improving move.

We compute $pr(ID_0, v)$ for all the candidate vertices ($v \in CL$), and build the restricted candidate list RCL with those that according to the greedy function $pr(ID_0, v)$ achieve a relatively large reduction in the number of crossings: $RCL = \{v \in CL: pr(ID_0, v) \geq \gamma\}$ where:

$$\gamma = \min_{v \in CL} pr(ID_0, v) + \beta \left(\max_{v \in CL} pr(ID_0, v) - \min_{v \in CL} pr(ID_0, v) \right) \tag{22}$$

and β is a search parameter indicating the degree of randomization that we want to include in the process. Following the GRASP methodology, we randomly select a vertex v^* in RCL and move it, obtaining ID_1. Then, we update the candidate list of vertices, $CL = CL \setminus \{v^*\}$, and repeat the steps above replacing ID_0 with ID_1. In short, we build a new RCL from which to select a vertex whose insertion will result in ID_2. In this way, we obtain a path of solutions, ID_0, ID_1, ID_2, and so on, up to we reach the guiding solution. The best solution found in the path is returned as the output of this path relinking step.

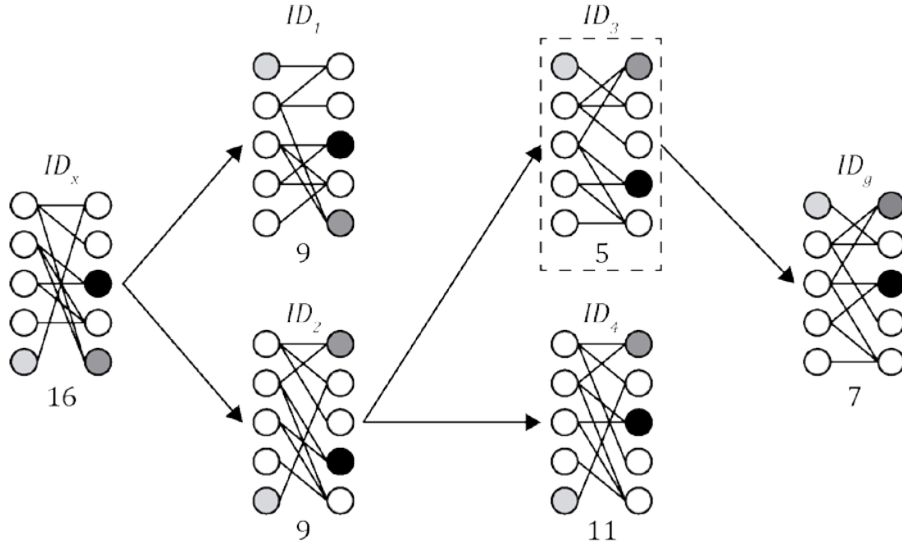


Figure 6. Path generation between initial solution ID_x and guiding one ID_g.

Figure 6 shows a path constructed between the initial solution ID_x and the guiding solution ID_g. The original nodes (that must preserve their relative ordering) are depicted in white, while the new nodes, which can be moved, are represented in light gray, medium gray, and black. The number under each solution represents its number of crossings. Starting in ID_x, the procedure generates the solutions that can be reached from it by locating one of the new nodes in its position in ID_g. In particular, solution ID_1 is generated by inserting the light gray node (the one in the first layer of ID_x) in the first position, while solution ID_2 is generated after the insertion of the medium gray node in the first position of second layer. The black node is located at the same position in ID_x and ID_g, so we do not consider to move it. At the next step, the method selects one of the best solutions to continue the search. In this case, both ID_1 and ID_2 exhibit the same number of crossings, so one of them is selected at random, say ID_2. The search

continues generating solutions ID_3 and ID_4 by locating the light gray node and the black one at their ID_g position, respectively. Finally, ID_g is directly reached from ID_3, thus ending the path. The best solution found during the search (ID_3) is returned as the output of this path construction.

Path relinking operates on a set of solutions, called elite set (ES), constructed with the application of a previous method. In this paper, we apply the hybrid tabu search algorithm described in Section 4 to build ES with the best 10 solutions found. Then, we apply the PR described in this section to all pairs of solutions in ES, returning the best solution found as the output of the method.

6. Computational experiments

The computational experiments described in this section were performed to test the effectiveness and efficiency of the procedures discussed above. The previous GRASP method, called prev_GRASP, by Martí and Estruch (2001), and our new procedures were implemented in Java SE 8, and the experiments were conducted on a computer with a 2.8 GHz Intel Core i7 processor with 16 GB of RAM. In particular, we report the results obtained with our constructive method, tabu search, and path relinking post-processing. Additionally, the mathematical programming formulation described in Section 2 was solved with Gurobi[1].

We employed two sets of instances in our experimentation. The first one contains 120 instances generated according to Martí and Estruch (2001), while the second one has 1000 instances and was proposed by Stallmann et al. (2001). In line with previous papers, we generated the first set of instances based on the original number of vertices in each layer, (n_1, n_2), and the graph density d in the interval [0.065, 0.175]. Additionally, as in Martí and Estruch (2001), the instances are incremented adding vertices and edges up to pre-established numbers. These numbers are calculated as a percentage δ of the quantities in the original graph ($|IV_i| = \delta|V_i|$ for each $i = 1, 2$, and $|IE| = \delta|E|$). We consider the following values in our experiments:

- (n_1, n_2) = (25, 25), (25, 50), (50, 25), and (50, 50).
- d = 0.065, 0.175, and 0.300.
- $\delta = 1.2$ and 1.6.

The generator to create our first set of instances is described in Martí and Estruch (2001). For each vertex u in the left layer, an edge to a randomly chosen vertex v in the right layer is included. Additional edges are added by randomly choosing two vertices of left and right layer. The process is repeated until all additional edges have been included to meet the desired density. Once the original graph has been created, we applied the well-known barycenter algorithm (Di Battista et al., 1999) to obtain the original drawing. Then, it is incremented by adding vertices and edges randomly up to the pre-established numbers. For each new vertex in NV_1 an edge to a randomly chosen vertex in IV_2 is included. Similarly, for each new vertex NV_2 an edge to a randomly chosen vertex in IV_1 is included. This guarantees that each new vertex has a degree of at least one. Additional edges are added by randomly choosing two vertices up to the desired number.

[1] http://www.gurobi.com/

The second set contains 1000 instances obtained with the generator described in Stallmann et al. (2001), which is publicly available[2]. The size of the first layer is in the range [10, 377], while the size of second layer ranges from 10 to 190 nodes. The number of edges is in the range [20, 950]. These instances are bipartite graphs, and we convert them in incremental bipartite graphs by considering a percentage of their nodes as the new nodes added to the original graph. In this way, we kept the structure and density of the instance. In particular, for each original instance we have generated three new instances obtained by selecting as new nodes the 10%, 20%, and 30% percent of the original nodes. To facilitate future comparisons, these instances are publicly available at www.optsicom.es/dbdp.

6.1 Preliminary experimentation

First experiments are devoted to select the best values of the key search parameters of the algorithms to configure our final method. We perform these experiments on a subset of 22 representative instances in the first set, and we do not include instances of the second set. In this way, we avoid over-training of our method in the final comparison with the previous method (prev_GRASP). For each experiment, we report the following metrics to measure the merit of each procedure when generating 10 constructions for each instance: Average number of crossings, *Cross.*, computing time in seconds, *Time* (*s*), average percent deviation from the best solution found in the experiment, *Dev*(%), and the number of best solutions found in the experiment, *#Best*.

The first experiment is intended to select the best value of the parameter α to determine the greediness of the constructive method. We have tested the following α values: 0.25, 0.50, 0.75. Additionally, we tested a variant (labelled RND) in which at each iteration α is randomly selected between 0 and 1. Table 1 reports the solutions of this experiment.

α value	**Cross.**	**Time (s)**	**Dev (%)**	**#Best**
RND	89515.23	40.84	0.20%	10
0.25	89741.32	38.23	0.93%	3
0.50	89518.27	39.57	0.36%	7
0.75	89409.95	40.16	0.38%	6

Table 1. Constructive method on training instances with different α values.

The values in Table 1 show that the best results are obtained when considering α at random in each iteration (first row in the table), which favors the randomness part of the algorithm, thus generating more diverse solutions.

In the second preliminary experiment, we test the effectiveness of the tabu search method. In particular, we generate 10 solutions with the best constructive method identified above, and then apply the tabu search algorithm to them. We stop the tabu search method after 50 iterations without improvement (we will study in the next experiment the influence of this parameter). In this experiment, we consider several values of the *tenure* parameter. In line with previous tabu search experiments (Glover and Laguna, 1997), we consider $tenure = 5$, 10, 15,

[2] https://people.engr.ncsu.edu/mfms/Software/SBG_Software/index.html

and 20. We do not reproduce the results of this experiment to limit the extension of the paper. Experimentally, $tenure = 5$ provides the most effective variant since with this value the algorithm is able to match 55% of best solutions with an average percent deviation of 0.16%, while the other *tenure* values match less than 41% of best solutions and exhibit percent deviations larger than 0.22%.

In the experiment above, we stopped the tabu search after a certain number of iterations without improvement, $MaxIter$, which is a standard way to finish a method. Specifically, we consider $MaxIter = 50$ consecutive iterations without improvement. We extended this experiment and tested different values of $MaxIter$ to stop the tabu search. Table 2 shows the results of this extended study in which we can see that with 50 iterations without improvement the method is able to achieve the best results.

$\boldsymbol{MaxIter}$	**Cross.**	**Time (s)**	**Dev (%)**	**#Best**
10	89333.82	41.68	0.08%	17
25	89330.05	40.53	0.03%	19
50	89326.73	41.91	0.00%	21
100	89335.14	42.96	0.09%	19

Table 2. Tabu search with different stopping values.

An important question when applying tabu search is the contribution of the memory structure to the quality of the final solution. In other words, we create our tabu search method by adding a memory structure (a tabu list) to a standard local search. We can then consider what would be the result if, instead of applying the tabu search to the 10 constructed solution, we apply to them the local search that only performs improving moves. The results of this experiment confirm the contribution of the memory structure. In particular, the version with local search exhibits a lower number of best solutions (11) than the tabu search version (21). From now on, we label as TS our method that first constructs 10 solutions (with the random variant), and then improves them with the tabu search method with $MaxIter = 50$ and $tenure = 5$.

In our last preliminary experiment, we test the path relinking post-processing. Specifically, we apply TS and collect the 10 best solutions found (forming the elite set). Then, we apply path relinking to all pairs in the elite set. Table 3 reports the solutions considering the different values of the β parameter (between 0 and 1) defining the restricted candidate list in this method.

$\boldsymbol{\beta}$	**Cross.**	**Time (s)**	**Dev (%)**	**#Best**
RND	89302.77	52.26	0.03%	15
0.25	89307.45	52.32	0.03%	18
0.50	89305.14	52.33	0.01%	14
0.75	89312.32	52.47	0.02%	14

Table 3. Tabu search with different stopping values.

Table 3 includes a variant in which the parameter is randomly selected in each iteration (RND). If we compare the results in Table 3 with those in Table 2, where path relinking was not applied, we can see that the average number of crossing (Cross.) takes now lower values, showing the contribution of this method. The results in this table indicate that there are small differences among the variants tested, being $\beta = 0.25$ the one which is able to obtain the largest number

of best solutions. We therefore select this variant, simply labelled as TS+PR, to perform the competitive experiments below.

				prev_GRASP			TS+PR			Gurobi		
n_1	n_2	d	δ	Cross.	Time	Dev	Cross.	Time	Dev	Cross.	Time	Dev
25	25	0.065	0.2	316	6.1	3.6%	**305**	0.4	0.0%	**305**	0.4	0.0%
25	25	0.065	0.6	1131	34.5	15.5%	1106	1.8	13.0%	**979**	237.7	0.0%
25	25	0.175	0.2	4037	36.9	2.9%	3927	0.4	0.1%	**3922**	1.4	0.0%
25	25	0.175	0.6	12374	120.0	3.3%	**11982**	3.8	0.0%	12444	1800.6	3.9%
25	25	0.300	0.2	15185	60.9	1.0%	15067	0.6	0.2%	**15036**	9.6	0.0%
25	25	0.300	0.6	40538	252.2	2.2%	**39657**	7.7	0.0%	41705	1801.6	5.2%
25	50	0.065	0.2	2218	27.8	2.1%	2185	0.7	0.6%	**2173**	1.5	0.0%
25	50	0.175	0.2	20112	201.7	1.6%	19861	2.1	0.3%	**19794**	193.3	0.0%
25	50	0.300	0.2	64113	495.3	1.8%	63312	4.8	0.5%	**62986**	1802.3	0.0%
25	50	0.300	0.6	**175764**	519.0	0.0%	176909	52.8	0.0%	184788	1831.6	4.5%
50	25	0.065	0.2	2230	42.2	2.8%	2191	0.8	1.0%	**2169**	1.6	0.0%
50	25	0.065	0.6	6459	267.3	10.8%	**5828**	12.4	0.0%	6155	1801.0	5.6%
50	25	0.175	0.2	20265	228.9	2.2%	19890	2.0	0.3%	**19831**	336.2	0.0%
50	25	0.175	0.6	57004	517.7	5.6%	**54004**	31.3	0.0%	63287	1802.4	17.2%
50	25	0.300	0.2	66253	502.8	1.0%	**65593**	4.4	0.0%	66319	1802.0	1.1%
50	25	0.300	0.6	186429	538.6	4.0%	**179282**	58.6	0.0%	189119	1805.3	5.5%
50	50	0.065	0.2	7859	297.3	2.9%	7664	3.9	0.4%	**7637**	3.5	0.0%
50	50	0.065	0.6	25545	506.5	2.5%	**24933**	67.0	0.0%	27110	1802.8	8.7%
50	50	0.175	0.2	78717	505.5	1.9%	**77253**	14.3	0.0%	77480	1802.7	0.3%
50	50	0.175	0.6	238979	504.1	2.4%	**233326**	209.3	0.0%	256917	1808.6	10.1%
50	50	0.300	0.2	251277	536.8	1.1%	**248454**	26.1	0.0%	258330	1806.3	4.0%
50	50	0.300	0.6	728794	575.7	2.3%	**712459**	416.7	0.0%	757750	1800.3	6.4%
			Avg.	91163.6	308.1	3.3%	89326.7	41.9	0.7%	94374.4	1102.4	3.3%

Table 4. Comparison among Prev_GRASP, Tabu Search with Path Relinking and Gurobi.

6.2 Competitive testing

In the first experiment of this subsection, we test the ability of the previous heuristic (prev_GRASP) and our TS+PR method to match the optimal solutions of the problem. To this end, we solve the mathematical programming formulation described in Section 2 with Gurobi, for a maximum time of 1,800 seconds in each instance. Table 4 reports the individual results on the 22 instances in our training set, where the bold font indicates the best value found for each instance. Note that when Gurobi finds the best value in a running time lower than 1800 seconds we can certify the optimality of the solution found. However, when a heuristic method outperforms Gurobi on an instance, we cannot assure that the best solution found is the optimal one.

Table 4 shows that Gurobi is able to obtain the optimal solution in 10 instances out of the 22 considered, although this method exhibits a relatively long average computational time (1102.4 seconds). The previous heuristic considered, prev_GRASP, is only able to produce one best solution in this experiment, although it is very fast compared with Gurobi (308 seconds, on average). Our heuristic method, TS+PR, obtains 12 best known solutions in an average running time of 41.9 seconds. Considering the average percentage deviations from the best-known solution, the ranking of the methods is TS+PR (0.7%), prev_GRASP (3.3%), and Gurobi (3.3%).

We now compare these three methods on the entire first set of 120 instances. As in the previous experiment, we limit the execution of the methods to a maximum of 1,800 seconds. Additionally, we report two versions of our TS+PR method. The first one, in which 10 constructed solutions are improved with TS+PR, and the second one, labeled TS(500)+PR, in which 500 solutions are constructed and improved with TS+PR if the time limit allows it. Table 5 shows the results on the instances incremented on a 20% of the original size ($\delta = 1.2$), and Table 6 the results with those incremented on a 60% ($\delta = 1.6$).

Algorithm	Cross.	Time (s)	Dev (%)	#Best	#Opt
small size ($n_1 + n_2 = 50$)					
Gurobi	6230.00	27.18	0.00%	15	15
prev_GRASP	6317.47	33.23	2.17%	0	0
TS(500)+PR	6232.20	48.76	0.10%	5	5
TS+PR	6234.33	8.15	0.16%	4	4
medium size ($n_1 + n_2 = 75$)					
Gurobi	28522.20	781.71	0.24%	22	21
prev_GRASP	28776.57	228.98	2.52%	0	0
TS(500)+PR	28379.10	621.53	0.16%	9	1
TS+PR	28390.17	61.11	0.21%	2	0
large size ($n_1 + n_2 = 100$)					
Gurobi	112572.27	1207.01	1.10%	6	5
prev_GRASP	111381.07	423.60	1.94%	0	0
TS(500)+PR	110214.33	1516.66	0.08%	7	0
TS+PR	110233.07	292.30	0.12%	3	0

Table 5. Final comparison of best methods in the first set ($\delta = 1.2$).

Algorithm	Cross.	Time (s)	Dev (%)	#Best	#Opt
small size ($n_1 + n_2 = 50$)					
Gurobi	18553.53	1224.33	4.24%	5	5
prev_GRASP	18056.27	141.83	7.44%	0	0
TS(500)+PR	17459.13	1039.53	1.01%	10	0
TS+PR	17489.07	90.12	1.71%	2	0
medium size ($n_1 + n_2 = 75$)					
Gurobi	84088.57	1808.00	7.50%	2	0
prev_GRASP	80917.97	409.31	5.15%	2	0
TS(500)+PR	78959.23	1718.27	0.27%	19	0
TS+PR	79161.07	386.90	0.61%	10	0
large size ($n_1 + n_2 = 100$)					
Gurobi	343478.67	1891.89	8.70%	0	0
prev_GRASP	328878.53	532.43	3.86%	0	0
TS(500)+PR	322528.33	1687.68	0.00%	15	0
TS+PR	322831.60	514.51	0.23%	0	0

Table 6. Final comparison of best methods in the first set ($\delta = 1.6$).

Results in Tables 5 and 6 confirm the superiority of our proposal with respect to previous methods. As expected, the performance of Gurobi quickly deteriorates when the size of the instances increases. If we focus on the largest instances in Table 5, with $\delta = 1.2$, TS+PR has an average percent deviation of 0.12%, which compares favorably with Gurobi (1.10%) and prev_GRASP (1.94%). Note additionally that TS+PR is the fastest method since it only requires 292.30 seconds on average, while Gurobi and prev_GRASP run for 1207.01 and 423.60 seconds respectively. Table 5 also shows the result of our method run for more iterations. In particular, TS(500)+PR exhibits a remarkable average percentage value of 0.08%, but it also requires the longest execution time (1516.66 seconds on average). Results in Table 6 are in line with those commented on Table 5.

The last two columns in these tables show the number of instances in which the method is able to obtain the best solution (#Best), and the number of instances in which we know that the method matches the optimal solution (#Opt). In Table 5, where we only add a small fraction of new vertices, Gurobi is able to obtain some optimal solutions. Note that for the other instances reported in this table, we do not know how far the heuristic solutions are from the optimal ones. In medium and large instances of Table 6, where a larger fraction of new vertices is added to the original instances, no method is able to certify the optimality of the solutions.

We then applied the Wilcoxon test to compare prev_GRASP and TS+PR on the results reported in Tables 5 and 6. This statistical test answers the question: Do the two samples (solutions obtained with the methods) represent two different populations? The obtained p-value < 0.001 confirms that there are significant performance differences between these two heuristic methods.

We now perform our main experiment in the competitive testing. In particular, we compare our best algorithm, TS+PR, with the best previous heuristic, prev_GRASP, in the second set of 1,000 instances. Note that this set of instances was not used to tune any method, so this experiment tests the adaptability and scalability of both heuristics.

Algorithm	**Cross.**	**Time (s)**	**Dev (%)**	**#Best**
		small size		
prev_GRASP	510.82	0.40	4.09%	119
TS+PR	497.77	0.09	2.03%	293
		medium size		
prev_GRASP	22742.99	18.83	4.69%	25.00
TS+PR	22394.12	7.80	0.06%	278.00
		large size		
prev_GRASP	87844.49	319.53	4.97%	6.00
TS+PR	85816.74	232.27	0.01%	344.00

Table 7. Final comparison of best heuristic methods in the second set of instances.

Table 7 shows the results of each algorithm over the Stallmann et al. (2001) set of instances. We can see that the computing time of both algorithms are equivalent, although our proposal is slightly faster. Notice that both algorithms were executed in the same computer platform, and implemented by using the same programming language. If we analyze the quality of the solutions generated by each algorithm, we can clearly see that TS+PR is able to obtain the best solution in 915 out of 1000 instances, while prev_GRASP only reaches the best solution in 150 instances. This difference is smaller in the case of small instances (with number of vertices

ranging from 21 to 57), where prev_GRASP finds 119 best solutions and TS+PR 293. However, in medium and large instances (with number of vertices ranging from 111 to 471), the superiority of TS+PR is evident, with a number of best solutions found with an order of magnitude larger than those in the previous method. Furthermore, the overall average deviation of TS+PR is close to 0, which means that in the 85 out of 1000 instances in which it is not able to obtain the best value, it is close to it. This compares favorably with the overall deviation of prev_GRASP of 4.58%. The p-value < 0.0001 obtained in the Wilcoxon test performed between both algorithms confirms the superiority of our proposal.

To complete the experimentation, we finally consider a real application in an assignment graph. In particular, we target the 2-layered graph of search queries on online advertisement (Antonellis et al., 2008). The graph in Figure 7 represents the assignment of 24 original queries (in the left layer) to 25 advertisements (in the right layer), where three additional queries (25, 26, and 27) and three ads (Z, a, and b) have been added. It has 311 edge crossings.

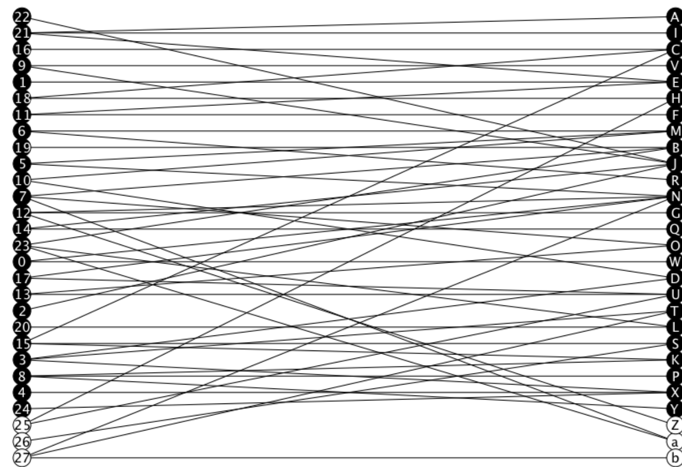


Figure 7. Assignments of queries to advertisements.

We apply our TS+PR procedure to the graph in Figure 7 with 311 edge crossings, and obtain the drawing shown in Figure 8 with 210 crossings. Note that the relative ordering among the original nodes is kept, thus helping the user to easily read the new graph.

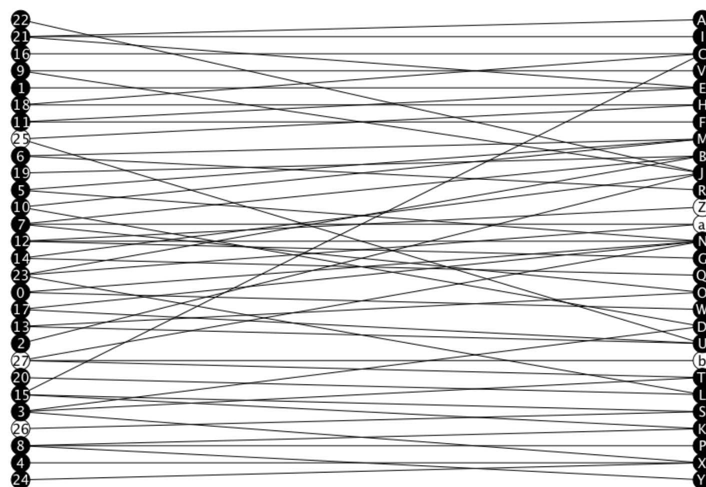


Figure 8. Assignment graph optimized with TS+PR.

7. Conclusions

In this paper, we have considered the dynamic bipartite graph drawing problem, also called incremental bipartite drawing problem in the literature. We propose new heuristic methods based on the tabu search methodology. Our extensive computation shows that the proposed method is able to outperform the previous heuristic for this problem. It is worth mentioning an implementation detail of our method that makes it especially fast. In particular, the update of the objective function when a move is performed is computed by means of two matrices, which store the number of crossings for each pair of vertices. The tabu search method efficiently updates these matrices after each move. On the other hand, we adapted the mathematical programming formulation originally proposed for the bipartite drawing problem to the dynamic case. Our experiments with Gurobi show that it is able to solve small and medium size instances to optimality.

An interesting point when designing a tabu search method is the memory contribution. We can say in plain words that a short term tabu search is simply a local search method in which we added a memory structure (the so-called tabu list). Therefore, one could ask what is the incremental contribution obtained with this addition. Our preliminary experimentation confirms that the tabu search clearly performs better than the simple local search which it is based on. Thus, the hybridization of a constructive GRASP method with a tabu search turns out to be a very effective method to target this problem. Additionally, we learnt that using a combination method such as path relinking for creating paths between two high quality drawings is a good technique to generate new and better solutions. We believe that this approach can be tested in other graph drawing problems.

In this paper we have formulated the stability across a sequence of graph drawings in terms of the relative ordering of their vertices, in line with some previous papers. However, alternative ways to approach stability would be also of interest. We are indeed starting to work on a formulation based on the absolute ordering of the original nodes. Another extensions and future lines of research include dynamic drawings in hierarchies with more than 2 layers, as well as a comparison of the different variants of path relinking for the DBDP.

8. Acknowledgments

This work has been partially supported by the Spanish "Ministerio de Economía y Competitividad" and by "Comunidad de Madrid," grants refs. TIN2015-65460-C02 and S2013/ICE-2894, respectively. We would like to thank the Walnut Brew-Lab in Boulder (CO), and the Spanish researchers there, for the discussions to conceive this paper.

References

Antonellis, I., H.G. Molina, and C. Chao (2008). Simrank++: query rewriting through link analysis of the click graph. 1(1):408-421.

Beck, F., M. Burch S. Diehl, and D, Weiskopf (2016). A Taxonomy and Survey of Dynamic Graph Visualization. Computer Graphics forum. 36(1): 133–159.

Branke, J. (2001). Dynamic Graph Drawing. In: Drawing Graphs. Methods and Models. Kaufmann, M., and Wagner, D. (Eds.), Springer LNCS. 2025: 228-246.

Burch, M., J. Heinrich, N. Konevtsova, M. Höferlin, and D. Weiskopf (2011). Evaluation of Traditional, Orthogonal, and Radial Tree Diagrams by an Eye Tracking Study. IEEE Transactions on Visualization and Computer Graphics. 17(12): 2440-2448.

Burch, M., M. Hlawatsch, and D. Weiskopf (2017). Visualizing a Sequence of a Thousand Graphs (or Even More). Computer Graphics Forum. 36(3):261-271.

Carpano, M. J. (1980). Automatic Display of Hierarchized Graphs for Computer-Aided Decision Analysis. IEEE Transactions on Systems, Man, and Cybernetics. 10(11): 705–715.

Di Battista, G., P. Eades, R. Tamassia, and I. G. Tollis (1999). Graph Drawing: Algorithms for the Visualization of Graphs. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition.

Diehl, S., and C. Görg (2002). Graphs, they are changing. In: 10th International Symposium on Graph Drawing GD 2002. Kobourov, S. G., and Goodrich, M. T. (Eds.), Springer LNCS. 2528: 23–30.

Duarte, A., J. Sánchez-Oro, M. Resende, F. Glover, and R. Martí (2015). GRASP with Exterior Path Relinking for Differential Dispersion Minimization. Information Sciences. 296: 46-60.

Eades, P., W. Lai, K. Misue, and K. Sugiyama (1991). Preserving the Mental Map of a Diagram. Proc. of Compugraphics. 91: 24-33.

Festa, P., and M. G. C. Resende (2011). GRASP: basic components and enhancements. Telecommunication Systems. 46(3): 253–271.

Garey, M. R., and D. S. Johnson (1983). Crossing Number is NP-Complete. SIAM Journal on Algebraic Discrete Methods. 4(3): 312–316.

Gibson, H., J. Faith, and P. Vickers (2012). A survey of two-dimensional graph layout techniques for information visualization. Information Visualization. 12(3-4) 324–357

Glover, F., and Laguna, M. (1997). Tabu search. Kluwer, Norwell, MA.

Görg, C., P. Birke, M. Pohl, and S. Diehl (2004). Dynamic graph drawing of sequences of orthogonal and hierarchical graphs. In: 12th International Symposium on Graph Drawing, GD 2004. Pach, J. (Eds), Springer LNCS. 3383: 228–238.

Grötschel, M., M. Jünger, and G. Reinelt (1984). A Cutting Plane Algorithm for the Linear Ordering Problem. Operations Research. 32(6), 1195-1220.

Jünger, M., and P. Mutzel (1997). 2-Layer Straightline Crossing Minimization: Performance of Exact and Heuristic Algorithms. Journal of Graph Algorithms and Applications. 1(1): 1-25.

Kumar G., and M. Garland (2006). Visual exploration of complex time-varying graphs. IEEE Transactions on Visualization and Computer Graphics. 12(5): 805–812. doi:10.1109/TVCG. 2006.193.

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

Martí, R., and V. Estruch (2001). Incremental Bipartite Drawing Problem. Computers and Operations Research. 28: 1287-1298.

Martí, R. (2001). Arc Crossing Minimization in Graphs with GRASP, IIE Transactions. 33 (10): 913-919.

Purchase, H.C. (2002). Metrics for Graph Drawing Aesthetics, Journal of Visual Languages and Computing. 13: 501-516.

Resende, M.G.C., and C.C. Ribeiro (2001). Greedy randomized adaptive search procedures. In: Metaheuristics. Glover, F., Kochen-Berger, G. (Eds.), Kluwer Academic Publishers. 219-270.

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

Sallaberry, A., C. Muelder, and K. Ma. Clustering, visualizing, and navigating for large dynamic graphs (2012). In: 20th International Symposium on Graph Drawing. Didimo W., and Patrignani,

Stallmann, M., F. Brglez, and D. Ghosh (2001). Heuristics, Experimental Subjects, and Treatment Evaluation in Bigraph Crossing Minimization. Journal of Experimental Algorithmics, 6-8.

Van der Elzen, S., D. Holten, J. Blaas, and J.J. van Wijk (2013). Dynamic Network Visualization with Extended Massive Sequence Views. IEEE Trans. on Visualization and Comp. Graphics. 20(8): 1087-1099.