Tabu Search for Min-Max Edge Crossing in Graphs

Tommaso Pastore[1], Anna Martínez-Gavara[2], Antonio Napoletano[3], Paola Festa[1], and Rafael Martí[2]

[1]Department of Mathematics and Applications. University of Napoli Federico II, Italy.
{tommaso.pastore, paola.festa}@unina.it
[2]Department of Statistics and Operations Research. University of Valencia, Spain.
{gavara, rafael.marti}@uv.es
[3]Optit srl, Via Mazzini, 82, 40138 Bologna, Italy
{antonio.napoletano}@optit.net

September 2, 2019

Abstract

Graph drawing is a key issue in the field of data analysis, given the ever-growing amount of information available today that require the use of automatic tools to represent it. Graph Drawing Problems (GDP) are hard combinatorial problems whose applications have been widely relevant in fields such as social network analysis and project management. While classically in GDPs the main aesthetic concern is related to the minimization of the total sum of crossing in the graph (min-sum), in this paper we focus on a particular variant of the problem, the Min-Max GDP, consisting in the minimization of the maximum crossing among all egdes. Recently proposed in scientific literature, the Min-Max GDP is a challenging variant of the original min-sum GDP arising in the optimization of VLSI circuits and the design of interactive graph drawing tools. We propose a heuristic algorithm based on the tabu search methodology to obtain high-quality solutions. Extensive experimentation on an established benchmark set with both previous heuristics and optimal solutions shows that our method is able to obtain excellent solutions in short computation time.

Keywords: combinatorial optimization, graph drawing, metaheuristics.

1 Introduction

Graph drawing problems consist in obtaining an automatic representation of a given graph, described in terms of its vertices and edges. Many aesthetic criteria have been proposed to identify the desirable properties that a good representation has to fulfill, with the most common being: edge crossings, graph area, edge length, edge bends, and symmetries. Their main objective is to achieve readable drawings in which it is easy to obtain or extract information. This goal is particularly critical in graphs with hundreds of vertices and edges, in which an improper layout could be extremely hard to analyze. Graph drawing is an active area of research. An excellent resource on the topic is the book by Di Battista et al. [3], where many graph drawing models and related applications are introduced.

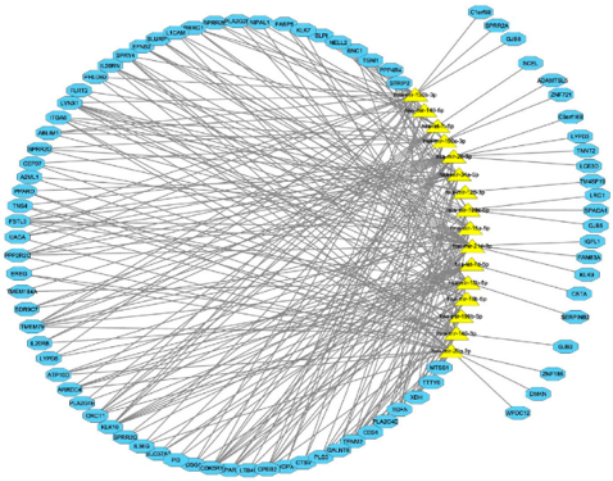


Figure 1: Circular representation of a graph [7].

There are different drawing conventions for representing a graph. In some settings, the vertices of the graph are drawn with a circle, such as for example in genetic interaction networks. Figure 1 represents the two hundred and forty-five interactions found among 16 micro-RNAs and 84 genes (see [7]). In other cases, the data has a clear sequential nature, like in workforce scheduling problems, where a set of tasks cannot be undertaken before the completion of some previous ones. Whenever the data is characterized by this kind of precedence relationships, the natural representation for the corresponding network is given by a layered graph. Practical applications fields which benefit from this graph layout can be found in Table 1. For example, in the field of operational inter-period material activities, a set of operational multi-period scenarios can be represented with a special graph rooted with replicas of the related strategic node, as it can be seen in Figure 2.

The Layered Graph (LG) representation [3] is obtained by arranging the vertices on a series of equidistant vertical lines called layers in such a way that all edges point in the same direction. Note that working in layers is not a limitation, since there exists a number of procedures to transform any directed acyclic graph (DAG) into a layered graph [2]. The Sugiyama's method to obtain a good representation of a graph has become a standard in the field [32]. As pointed out in the excellent survey on graph drawing by Healy and Nikolov in [16], Sugiyama's framework is oriented to obtain a good drawing by representing the edges according to certain aesthetics that induce readability. Specifically, the method arranges vertices in such a way that the drawing has short straight lines, pointing in a uniform direction with low number of crossings. To this end, the authors proposed a method with three steps: assign vertices to layers, reduce edge crossings, and assign coordinates to vertices. We apply this method to the graph in Figure 3 to illustrate its performance.

After a pre-processing to reverse the direction of some edges if needed, the first step, called *layer assignment*, assigns vertices to layers. In this step, edges between vertices in non-contiguous layers are replaced with a chain of dummy vertices and edges between consecutive layers, thus obtaining a layering. Different objectives can be achieved in this initial step, such as minimizing the number of dummy vertices, or reducing the number of layers. This step is shown with an example in Figure 4. In the second step, called *crossing minimization*, vertices are reordered within their corresponding layers to reduce the number of edge crossings, which

Context	References	Description
Workflow visualization	[33]	Representation of the work to be executed by the project team.
Software engineering	[8, 5]	Representation of calling relationships between subroutines in a computer program.
Database modeling	[17]	Definition of data connections and system processing and storing diagrams.
Bioinformatics	[21]	Representations of proteins and other structured molecules with multiple functional components.
Process modeling	[15, 10]	Analytical representation or illustration of an organization's business processes.
Network management	[28, 20]	Representation of the set of actions that ensures that all network resources are put to productive use as best as possible.
VLSI circuit design	[4]	Representation of the design of integrated circuits (ICs) which are essential to the production of new semiconductor chips.
Decision diagrams	[25, 26]	Definition of logic synthesis and formal verification of logic circuits.

Table 1: Layered Graphs applications as listed in [27].

is an NP-hard optimization problem (see Figure 5). Finally, in the third step called *coordinate assignment*, vertices are moved within their layer to obtain a good final drawing, modifying their coordinates without changing their order. The objectives are that long edges are straight and vertices are centered among their neighbors. In this step, long length edges are brought back from the chains of dummy vertices introduced in step 1 to model them, as shown in our example in Figure 6.

The crossing minimization problem in layered digraphs has received a lot of attention. The problem in bipartite graphs has been also extensively studied for more than 40 years, beginning with the Relative Degree Algorithm introduced in Carpano [6]. Early heuristics were based on simple ordering rules, reflecting the goal of researchers and practitioners of quickly obtaining solutions of reasonable quality. However, the field of optimization has recently introduced complex methods, both in exact and heuristic domains. The crossing problem has benefited from these techniques, and advanced solution strategies have been proposed in the last 20 years to solve it. We refer the reader to Martí [23], Di Battista et al. [3], or Chimani et al. [9] for relatively recent developments.

In this paper we focus on a variant of the crossing problem recently introduced in Stallmann [31]. In particular, this author identified some applications in the context of VLSI circuits design in which it is more appropriate to minimize the maximum number of crossings for any edge (min-max) rather than the sum of the edge crossings over all the graph (min-sum). As stated in Bhatt and Leighton [4], an undesirable feature of VLSI layouts is the

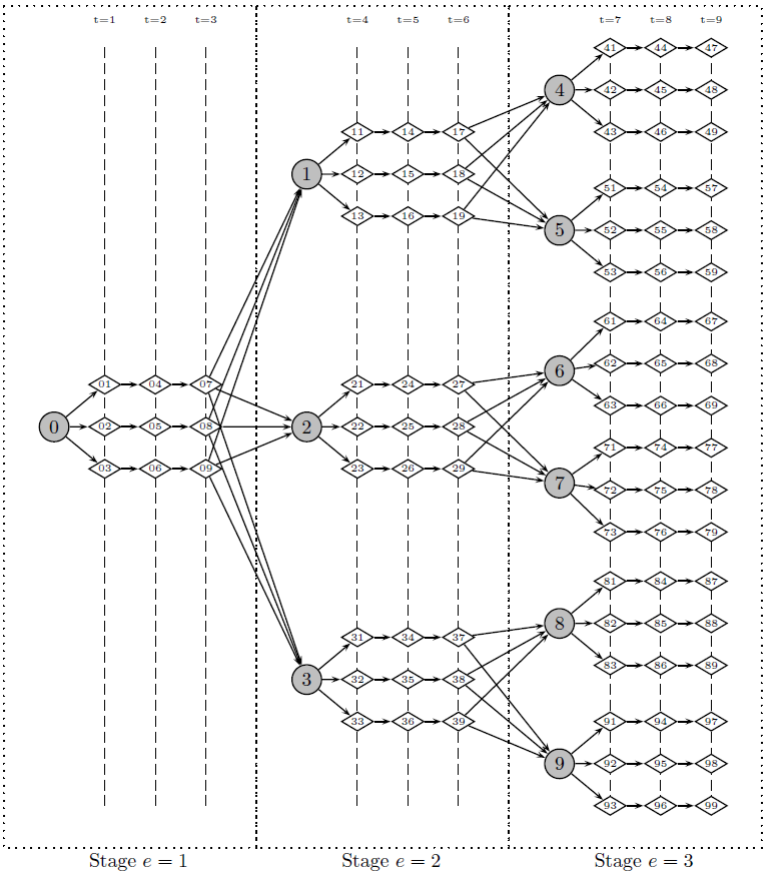


Figure 2: Multistage scenario tree [1].

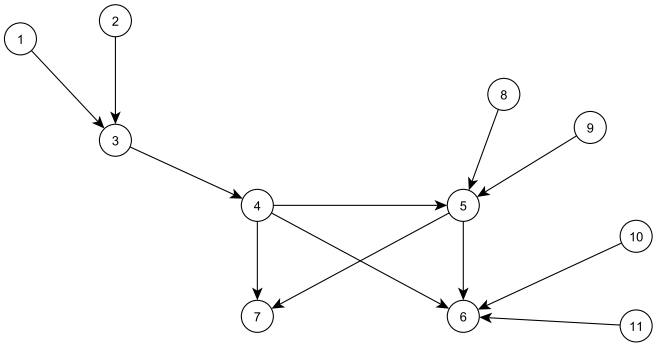


Figure 3: Input graph given to Sugiyama's method.

presence of a large number of wire crossings. More specifically, wires that are crossed by many others are susceptible to cross-talk, when all the crossing wires simultaneously carry the same signal, thus deteriorating the circuit performance. On the other hand, if the number of wire crossings is small, the number of contact-cuts is also small, thus providing a better signal. Therefore, in order to attain a good performance over the network it is critical that no edge has a large number of crossings, more than obtaining a small overall sum of crossings. This application motivated the work by Stallman [31], where a heuristic algorithm is proposed to

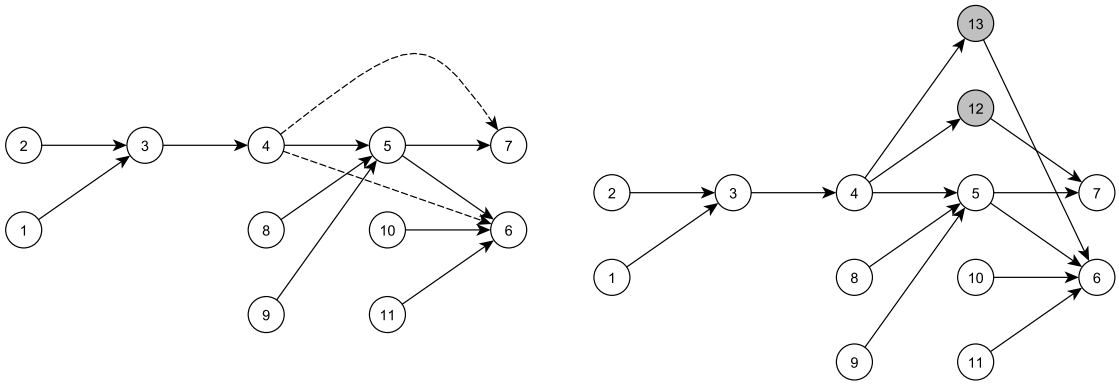


Figure 4: Step 1 in Sugiyama's method: layer assignment.

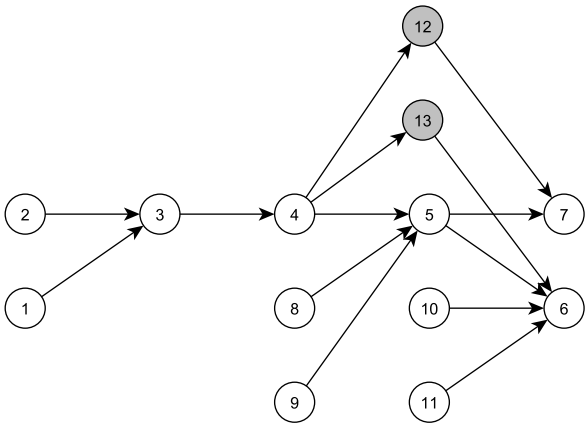


Figure 5: Step 2 in Sugiyama's method: crossing minimization.

minimize the maximum number of crossings among all edges. More recently, Martí et al. [24] applied a metaheuristic algorithm to solve the problem defined in Stallman's work.

Note that the local nature of the min-max problem is also useful in any graph drawing software, where zooming highlights a specific area. In this sense, the min-max objective function reduces crossings for each zoomed edge, while the min-sum problem solution does not imply a low number of crossings in the zoomed area. It is then clear that the min-max problem deserves to be studied and heuristic methods are needed to obtain high-quality solutions quickly. The objective of this paper is to propose a competitive method for this problem.

The main contributions of this work are: (i) extension of the mathematical programming model in the case of multilayer layered graphs, (ii) implementation of a Tabu Search heuristic with a long term memory strategy, (iii) adaptation of specific move evaluation function proposed for bandwidth minimization problems [29], and (iv) improvement of the state-of-the-art in solving the min-max graph-drawing problem.

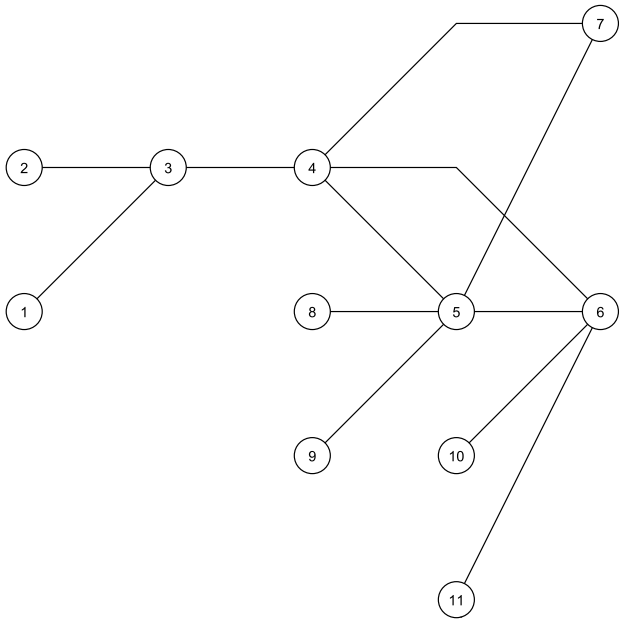


Figure 6: Step 3 in Sugiyama's method: final drawing.

2 Mathematical Programming Model

Crossing minimization is a well-known problem in graph drawing [3], with many optimization algorithms proposed over the last twenty years to obtain good solutions. As described in the previous section, in this paper we consider a recent variant proposed by Stallmann [31], where we seek to minimize the maximum number of crossings across all edges.

A layered graph $H = (G, k, L)$ is a graph $G = (V, E)$, where the set of vertices V and the set of edges E are partitioned into $k-$layers as follows:

$$V = \bigcup_{l=1,\ldots,k} V^l; \tag{1}$$

$$E = \bigcup_{l=1,\ldots,k-1} E^l \subset \bigcup_{1 \leq l < k} V^l \times V^{l+1}, \tag{2}$$

where all edges connect vertices on consecutive layers. Additionally, the function $L : V \rightarrow \{1, \ldots, k\}$ indicates the layer where each vertex $v \in V$ resides, implicitly defining $V^l = \{v \in V : L(v) = l\}$. Let n_l be the number of elements in layer l ($n_l = |V^l|$). A drawing of H is defined as $D = (H, \Phi)$, where $\Phi = \{\varphi^1, \varphi^2, \ldots, \varphi^k\}$, and φ^l is the ordering (permutation) of the vertices in layer l. Let $c(u, v)$ be the number of edges that cross edge $e = (u, v)$ for a given D, called the *crossing number* of e. Given a drawing D, the aim of the *min-max crossing problem* is to minimize its maximum crossing number $C(D)$,

$$C(D) = \max\{c(e) : e \in E\}. \tag{3}$$

For the sake of simplicity, we will use C instead of $C(D)$ whenever there is no ambiguity among different drawings.

The min-max crossing problem in a bipartite graph was formulated as a linear integer model (Martí et al. [24]) by adapting the classic formulation for the min-sum problem proposed by Jünger and Mutzel [19].

We define binary variables $x^l_{uu'}$ and $c^l_{uvu'v'}$ such that

$$x^l_{uu'} = \begin{cases} 1, & \text{if } \varphi^l(u) < \varphi^l(u'); \\ 0, & \text{otherwise,} \end{cases}$$

and $c^l_{uvu'v'}$ takes value 1 if edges (u, v) and (u', v') cross, i.e.,

$$c^l_{uvu'v'} = \begin{cases} 1, & \text{if } (\varphi^l(u) - \varphi^l(u'))(\varphi^{l+1}(v) - \varphi^{l+1}(v')) < 0; \\ 0, & \text{otherwise.} \end{cases}$$

where $u, u' \in V^l$ and $v, v' \in V^{l+1}$ for all layers $l = 1, \ldots, k-1$.

The classical formulation described in [18] (and [19]) models the traditional total crossing minimization problem in multi-layer graphs as follows:

$$\min \quad \sum_{l=1}^{k-1} \sum_{(u,v),(u',v') \in E} c^l_{uvu'v'} \tag{GDP}$$

s.t.

$$-c^l_{uvu'v'} \le x^{l+1}_{vv'} - x^l_{uu'} \le c^l_{uvu'v'} \qquad (u,v),(u',v') \in E^l, u < u', v < v', l < k \tag{4}$$

$$1 - c^l_{uvu'v'} \le x^{l+1}_{v'v} + x^l_{uu'} \le 1 + c^l_{uvu'v'} \qquad (u,v),(u',v') \in E^l, u < u', v > v', l < k \tag{5}$$

$$0 \le x^l_{uv} + x^l_{vw} - x^l_{uw} \le 1 \qquad \forall\ 1 \le u < v < w \le n_l,\ \forall l \tag{6}$$

$$x^l_{uu'} \in \{0,1\} \qquad \forall\ u, u' \in V^l, u < u',\ \forall l \tag{7}$$

$$c^l_{uvu'v'} \in \{0,1\} \qquad \forall\ (u,v),(u',v') \in E^l, u < u',\ \forall l. \tag{8}$$

In the model above, constraints (4) and (5) force $c^l_{uvu'v'}$ to take the value 1 when the variables $x^l_{uu'}$ and $x^{l+1}_{vv'}$ indicate a crossing. Constraints (6) guarantee that the x-variables represent an ordering by introducing precedence relationships (i.e., if u precedes v, and v precedes w, then u has to precede w). We adapt this formulation to the min-max objective function, collecting the crossings for each edge (u, v) in the integer variable $c(u, v)$, and introducing the upper bound variable C. In this way, we replace the objective function above with min C, and we add the following two constraints to the formulation:

$$c(u,v) = \sum_{\substack{(u',v') \in E^l \\ u < u'}} c^l_{uvu'v'} + \sum_{\substack{(u',v') \in E^l \\ u' < u}} c^l_{u'v'uv} \qquad \forall (u,v) \in E^l \tag{9}$$

$$c(u,v) \le C \qquad \forall (u,v) \in E^l \tag{10}$$

When minimizing C, constraint (10) forces this variable to take the maximum of the crossing numbers $c(u, v)$, thus providing the objective function value of the min-max problem. Table 2 summarizes the notation introduced in this section.

Symbol	Definition
G	Graph: $G = (V, E)$
V	Set of vertices of G
E	Set of edges of G
k	Number of layers
H	Layered graph: $H = (V, E, k, L)$
L	Function that indicates the layer in which a vertex resides
V^l	Set of vertices in layer l in H
n_l	Number of vertices in layer l
D	Drawing: $D = (H, \Phi)$
Φ	Set of permutation $\{\varphi^1, \ldots, \varphi^k\}$
$c(e)$	Number of edges that cross edge $e \in E$
$C(D)$	Maximum number of edge crossings of a drawing D

Table 2: Symbols and Definitions.

3 Solution Methods

In this section, we first review the state-of-the-art methods published to solve the min-max crossing problem, and then we propose a new one. In particular, we describe a tabu search method (`TS`) which implements the so-called short term and long term memory structures and their associated strategies.

3.1 Previous algorithms

The min-max crossing problem is addressed by two earlier papers. The first is due to Stallmann [31], who developed the `MCE` heuristic. It is a very efficient method based on the shifting principle, which performs passes over the layers of the graph, until a termination criterion is met. Stallman's method considers the edges as the main element in its design, in contrast to most classic crossing reduction methods [3], usually based on vertices. In particular, `MCE` identifies the edge with the largest number of crossings and tries to reduce it, reallocating its endpoint vertices in their layer. The method starts with an initial drawing D, determined by the barycenter method. Then, it sorts the edges $e \in E$ in descending order according to their number of crossings $c(e)$. At each step, following this order, each edge e is examined, and its endpoints are checked in search for their best position.

As pointed out by the author, an interesting feature of the `MCE` method is that it performs improving moves (vertex relocation) based on a local principle. This means that, instead of using the objective function value $c(D)$ to determine the move, only the edges incident to the chosen vertex are considered to find its best position. Thus, `MCE` determines the best location for a vertex v as the one that minimizes the maximum number of crossings among the edges incident to v. In practical terms, this implies that a drawing could be poorer after a vertex reallocation since the number of crossings of a non-adjacent edge could eventually increase. The experiments showed that this local strategy achieves good results, working as a diversification element and preventing the search from being trapped in a local optimum. A post-processing procedure of exchanging adjacent vertices is also applied to further improve the solutions.

Martí et al. [24] proposed an iterated greedy heuristic based on the Strategic Oscillation (SO) methodology [30]. After the construction and improvement of an initial solution, this technique alternates between destructive and constructive phases. The SO algorithm is based on the interaction of these three steps, constructive phase, neighborhood search and destructive phase.

The construction step starts by randomly selecting a vertex v from all the vertices, and placing it in an arbitrary position in its layer. In subsequent construction steps, the next vertex v is randomly chosen from those in the restricted candidate list RCL, that consists in all unassigned vertices with a degree of no less than a percentage of the maximum degree found among unassigned vertices. Then, the selected vertex is placed in the position prescribed by the barycenter, $bc(v)$ in its layer. The barycenter is computed as the arithmetic mean of the positions of the already adjacent vertices to the chosen vertex. The construction phase ends when all the vertices have been positioned.

Important ingredients in this algorithm are the sets of near-critical edges ($NCE(p_{\texttt{IG}})$) and vertices ($NCV(p_{\texttt{IG}})$). These sets depend on a search parameter $p_{\texttt{IG}} \in [0, 1]$, such that

$$NCE(p_{\texttt{IG}}) = \{e \in E : c(e) \geq p_{\texttt{IG}}\, C(D)\}.$$

On the other hand, the set of near-critical vertices is defined as the set of vertices that are endpoints of near-critical edges.

Once a solution D has been constructed, the neighborhood search is applied. This procedure attempts to move each vertex v in $NCV(p_{\texttt{IG}})$ in positions: $bc(v) - 2$, $bc(v) - 1$, $bc(v)$, $bc(v) + 1$, or $bc(v) + 2$, if they exist. The vertex is placed in the position that produces the minimum of the total (sum) number of crossings. In order to save computational time, the algorithm does not recompute the crossings after every move, and consequently does not update near critical sets and the min-max objective function. This neighborhood search phase ends when all vertices have been considered for insertions. Then, the objective function is re-evaluated.

During the destructive phase, all the vertices in the set $NCV(p_{\texttt{IG}})$ are removed from drawing D. In order to achieve further diversification of the solution, an additional number of vertices randomly selected from the set $V \backslash NCV(p_{\texttt{IG}})$ is removed.

A new iteration in the SO algorithm starts by reinserting the removed vertices in the solution, according to the same greedy criterion used in the constructive phase. Then, as final result, the algorithm returns the best solution found in the entire search process. See [24] for further details.

3.2 A new Tabu Search algorithm

Tabu Search (TS) is a well-known metaheuristic for solving combinatorial optimization problems designed to guide a search method to escape from local optimality. This methodology was first proposed by Glover [11], and precise descriptions appeared in many studies, as for example [12, 13, 14].

The TS framework is built from two main principles: intensification, in which the search tries to achieve a local optimum, and diversification, applied to escape from the actual basin of attraction to find a global optimum. These two strategies let the algorithm explore the search space in different ways. In the intensification phase the movements are guided by the objective function, to improve the current solution. Meanwhile, in the diversification phase, movements are evaluated using both the objective function and frequency functions

recording past information, to guide the search towards unexplored regions. In this paper, we implement these two concepts by incorporating memory functions of different time spans, namely short and long term.

It is well documented, see [22], that min-max and max-min optimization problems are a challenge for heuristic algorithms because those problems have many solutions with the same objective value, thus making local search based methods relatively inefficient. Given a solution, most of its neighbors have the same objective function value, thus making the local search "almost blind" in terms of approaching a local optimum. To overcome this difficulty, we consider in the intensification phase of our tabu search method, the δ function, proposed by Rodriguez-Tello et al. [29] in the context of the Bandwidth Minimization problem (BMP). The idea that motivates us to use this evaluation function lies in the similarities that both problems share: (i) a min-max objective function, (ii) the correspondence between a solution and a permutation of the vertices, and (iii) trivial feasibility.

In the cases of min-max crossing and BMP ([29]), the size of the solution space presents a factorial growth in the size of the instance input. At the same time, in both problems, the objective functions can be bounded from above by a polynomial in the size of the input. For example, in the case of the *min-max crossing* problem, a trivial bound can be obtained by counting the maximum number of edges among two consecutive layers minus one. Since, in both min-max crossing minimization and BMP, the upper bound on the optimal solution is small with respect to the size of the search space, there is a plethora of solutions with the same cost. This means that, during the local search, a move evaluation function solely based on the *min-max* objective hardly detects improving moves.

The δ-evaluation function for a Drawing D, depending on its ordering Φ, is defined in equation (11), where d_x is the number of edges with x crossings, ϑ an upper bound, and β defines the maximum number of crossing ($\beta = c(D)$, as defined in (3)). The upper bound on the number of crossings, ϑ, is computed as the maximum number of edges between consecutive layers minus one. We refer the reader to [29] for further details on this expression.

$$\delta(D) = \beta + \vartheta! \sum_{x=1}^{\beta} \frac{d_x}{(\vartheta + \beta - x + 1)!}. \tag{11}$$

Let us consider the graph in Figure 7 to illustrate the rationale behind the δ-evaluation function (11). This drawing presents two edges with 3 crossings, which happens to be the maximum: the edge that connects vertex 3 of the first layer and vertex 0 of the second, and the edge connecting vertex 2 of the second layer with vertex 0 of the third layer. The histogram on the right hand side of Figure 7 shows the number of crossings (cost) distribution in the drawing, collecting for each possible cost value the number of edges with that number of crossings. For this specific graph representation the objective function (maximum number of crossings) is 3, and the δ function is 3.440476.

Swapping vertices 1 and 0 in the first and in the last layer of Figure 7, we achieve the drawing depicted in Figure 8. As the previous one, this layered representation presents a maximum crossing value of 3, corresponding to the edge connecting vertex 3 of the first layer with vertex 0 of the second. The objective function in this drawing is therefore 3, however δ function attains a value of 3.205357.

Regardless of the max-crossing, which gets the same value for both drawings, the representation provided by Figure 8 is arguably clearer than the one of Figure 7. This feature can be deduced by means of a comparison of the two cost distributions, since the number of edges

with a relatively high cost decreased. In this way, we can consider that in the neighborhood of the new drawing it is more likely to find an improving solution. This important characteristic for a local search method is properly detected by the decrease of the δ function, but overlooked by the plain min-max objective function.

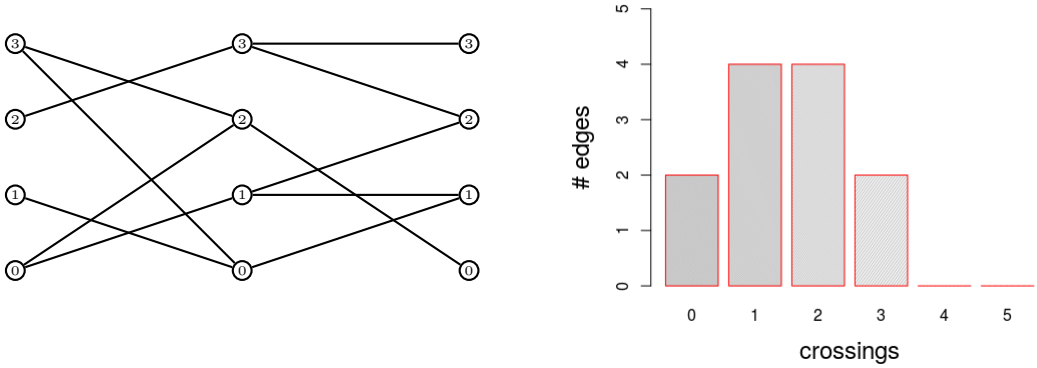


Figure 7: δ-evaluation function example.

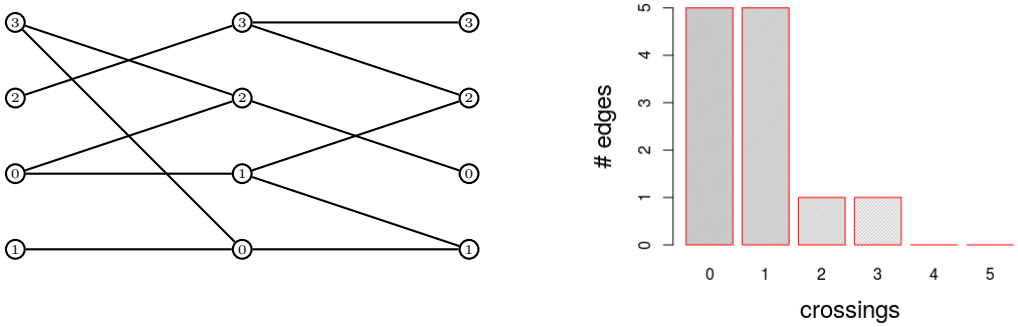


Figure 8: δ-evaluation function example after swap.

Our two-phase `TS` method is outlined in Figure 9. The first phase, Short-Term `TS`, is focused on an intensification strategy. The second phase, Long-Term `TS`, complements the first one by performing search diversification to explore new regions of the solution space. Each phase is executed for a maximum number of iterations without improving the current solution, `MaxIntensification` and `MaxDiversification` respectively. The overall `TS` method stops when the `TimeLimit` is reached.

Intensification Phase

The exploration in the intensification phase is based on swapping vertices in the same layer. As it is customary in tabu search, when a move is performed, the vertices involved become *tabu-active* for a certain number of iterations, their *tabu tenure*, and we call them simply tabu vertices. Tabu vertices cannot be moved during their tabu tenure. Non-tabu vertices are then those that are not tabu-active (i.e., those that can be moved).

```
Data: HDAG D, α, TimeLimit, MaxIntensification, MaxDiversification
Result: Optimized design for the min-max crossing.
bestSolution = D;
noImprov = 0 ;
while time spent < TimeLimit do
    if noImprov < MaxIntensification then
        IntensificationPhase(D,α);
        if the current best is improved then
            noImprov = 0 ;
        else
            noImprov++ ;
    else
        Diversification(S, MaxDiversification);
        noImprov = 0 ;
return bestSolution;
```

Figure 9: Main Tabu algorithm.

For each layer l, from $l = 1$ to $l = k$, we compute a list of candidate elements to be swapped (CL^l). This list collects all the non-tabu vertices in the layer with an edge with number of crossing larger than or equal a threshold, computed as a percentage (α) of a reference maximum cost c^l_{max}. The c^l_{max} value is the current maximum number of crossings in edges incident with a non-tabu vertex in l:

$$CL^l = \{u \in V^l \text{ non-tabu} : c_u \geq \alpha\, c^l_{max}\}, \tag{12}$$

with c_u denoting the maximum number of crossings found in edges incident to vertex u,

$$c_u = \max\{c(e) : e = (u, v) \in E^l\},$$

and

$$c^l_{max} = \max\{c_u : u \in V^l \text{ non-tabu}\}.$$

Starting from a drawing D, for each $u \in CL^l$, the method searches for its best swap with another non-tabu vertex, and performs it whenever the move improves (i.e. decreases) the δ-value of the current solution D. If for all the vertices no improving solution has been found, the algorithm performs the least worsening swap. The swapped vertices become tabu-active and remain in such a status for a specified number of iterations controlled by the search parameter *tenure*. Swaps that involve tabu-active vertices are declared tabu and therefore are not allowed. The intensification phase stops after `MaxIntensification` iterations without improvement. The structure of our intensification phase is illustrated in Figure 10.

Diversification Phase

In some `TS` applications, short term memory strategies produce very high quality solutions by themselves. However, in general, `TS` becomes significantly stronger by including longer term memory, implementing diversification strategies that allow the search process to escape from the current basin of attraction.

```
   Data: HDAG D, α
   Result: Locally optimal drawing.
 1 for l = 1, ..., k do
 2 |   build CL^l;
 3 |   for u ∈ CL^l do
 4 |   |   v ← best_swap_vertex(u);
 5 |   |   if swap(u,v) decreases δ then
 6 |   |   |   update solution;
 7 |   |   |   update δ;
 8 |   |   |   declare tabu u, v;
 9 |   if no swaps performed then
10 |   |   perform least worsening;
11 |   |   update tabu statuses;
12 return;
```

Figure 10: Pseudo-code of the Intensification Phase.

As in classical implementations, our diversification procedure is executed when the intensification reaches the maximum number of iterations without improving the best-found solution, i.e., when we consider that the search is trapped in a specific region of the solution space. During both intensification and diversification the algorithm records, for each vertex u, the number of times u has been moved. Two vertices are randomly selected for a swap with bias related to this count. More specifically, let M^l be the total number of swaps performed in layer l, and let m_u^l indicate the number of swaps involving vertex u of layer l. Since the swap is a pairwise move, we have $\sum_u m_u^l = 2\,M^l$. We define the probability of being selected for a swap in the diversification phase (pr_u^l) as:

$$pr_u^l = \frac{M^l - m_u^l}{\sum_v (M^l - m_v^l)}, \quad \forall u \in V^l. \tag{13}$$

Given a vertex u in layer l, the lower its number of moves, m_u^l, the larger the probability pr_u^l. In this way, the method favors the selection of vertices with low counts. The pseudo-code of the diversification phase is shown in Figure 11. This phase is executed sweeping through the layers of the graph for `MaxDiversification` iterations in each layer if no improving solution is encountered. If the algorithm does encounter a better solution, the diversification phase ends. In any case the `TS` method restarts with the intensification phase in order to obtain a locally optimal solution in the new area under exploration.

Min-Sum post-processing

As discussed in Section 1, the optimization of the min-max crossing function can be found in different real world scenarios, such as the VLSI circuit design or the development of interactive graph drawing tools. As we pointed out in the introduction, the use of the min-max objective function in comparison with the min-sum, reasonably guarantees that edges with a high number of crossing are unlikely to be found in the layout. This feature is particularly valuable when the algorithm is embedded in a drawing tool that allows to zoom onto user-specified areas of the graph.

```
   Data: HDAG D, MaxDiversification
 1 step = 0 ;
 2 while step < MaxDiversification do
 3 |   for l = 0, ..., k do
 4 |   |   for u ∈ V^l do
 5 |   |   |   compute pr_u^l ;
 6 |   |   u ←random_node(pr^l);
 7 |   |   v ←random_node(pr^l);
 8 |   |   swap(u, v) in V^l ;
 9 |   |   update swap frequencies;
10 |   |   update tabu statuses;
11 |   |   if the δ decreases then
12 |   |   |   return;
13 |   step + + ;
14 return;
```

Figure 11: Pseudo-code of the Diversification Phase.

One of the main characteristics of the min-max problems is that we can find many solutions with the same objective function value, so we can expect that it has many alternative optima. Therefore, it is possible to consider a post-processing procedure to reduce the total (sum) number of crossings, without increasing the min-max value. In this way, we are improving the max-sum as a secondary objective function without deteriorating the primary objective min-max function. Note that this approach is in line with Stallmann's [31] and with classical drawing approaches that make use of the min-sum as reference function.

Our post-processing consists of a local search method guided by the min-sum function with a swap-based neighborhood structure. Note that the `TS` method described in the previous subsections is guided by the min-max objective, and more specifically by the δ function. This post-processing only considers moves that do not worsen the solution in terms of min-max, thus preserving solution quality, while attaining better min-sum values. In particular, the method sequentially sweeps layers (from 1 to k), and scans their vertices in search for an improving swap with respect to the sum of crossings. This procedure stops whenever, after an exploration of all the layers (and all the vertices in each one), no improving move is performed.

4 Computational Experiments

In this section, we perform extensive computational experiments to analyze the key elements in our proposed method (`TS`) and to compare it with previous methods. In particular, we compare `TS` with the maximum crossing edge heuristic method (`MCE`) by Stallmann [31] and the strategic oscillation method (`SO`) proposed by Martí et al. [24]. `MCE` and `SO` are implemented in C, and `TS` in C++, and the experiments are conducted on a computer with a 2.6 GHz Intel Core i7 processor with 8 GB of RAM, and a three level cache with levels of $32K$, $256K$, and $6144K$. The `MCE` algorithm has been downloaded from `https://people.engr.ncsu.edu/mfms`. The set of instances we use is available online in the webpage `http://grafo.etsii.urjc.es/optsicom/`. This site also includes the individual

results of the methods and their associated running times for future comparisons.

We divide the experimentation into two main parts, preliminary experiments and comparative study. In the first one we set the values of the search parameters and finish with a study of the δ-function contribution in the search process. As mentioned above, in our comparative study we consider `CPLEX`, and the two previous heuristics. The benchmark set of instances is created with the Stallmann's generators [31], *uniform* and *connected*, and two well-known public domain sets of instances in graph drawing: Rome and North. ~~The *uniform* generator has four parameters: number of layers, l, number of nodes per layer, n_i, graph density, d, and the bias in the random number generator, b.~~ **The *uniform* generator has four parameters: number of layers, l, number of nodes per layer, n_i, the probability that a potential edge between two layers exists p, and a bias value, b.** We generate 299 instances with the uniform generator with similar parameter values as in [24] and [31], which are detailed in Table 3. The bias takes the values $b = 1, 5, 20$, and 40 as in [31]. The *connected* generator also has four parameters: number of nodes, n, number of edges, m, number of layers, l, and skew factor, σ. We generated 95 instances with this generator with the same skew values as in [31], $\sigma = 2, 4$, and 8. Both bias and skew introduce irregularities in the uniform and connected graphs, respectively: in each layer the bias favors the introduction of edges adjacent to a small subset of the nodes, thus affecting the max degree of the generated uniform graph. On the other hand the skew controls the different sizes of the layers of a connected graph, the lower σ the closer the graph is to a uniform network. See [31] for further details. Additionally, we consider 58 instances from the North set and 88 from the Rome set. Note that the three methods in our comparison, tabu search, SO, and MCE, minimize crossings in layers; we therefore transformed these general graphs into layered graphs by first applying Sugiyama's method. We classified these instances in Table 3 according to the experiments where $u(n_i, l, d)$ is the set of uniform instances with n_i number of nodes per layer, l number of layers, and d approximately($\approx$) the graph density. Similarly, let $c(n_i, l, d)$ be the set of connected instances.

Note that `CPLEX` is only able to target medium size instances, and for the heuristic comparison we consider the 301 instances already used and described in [24]. Regarding the δ-function experimentation, we consider three subsets of instances generated varying three independent parameters, namely: the number of vertices per layers n_i, the number of layers l, and edge-density d. The first subset, called **var-density**, has 70 instances obtained considering a fixed number of vertices and layers, and different densities. In the second subset, **var-nodes**, 60 instances are collected with the same density and number of layers, but they differ in terms of number of vertices per layer, $n_i \in \{10, 15, 20, 25\}$. The third subset, called **var-layers**, has 60 instances generated fixing d and n_i, and varying the number of layers in the graph. Moreover, whenever fixed n_i, l and d, are approximately equal to their median value in, respectively, var-nodes, var-layers, and var-density. The aim of the δ test is to study the impact of the δ-evaluation function on a broad set of different graphs, and possibly relate the usefulness of this move evaluation with d, l, and n_i.

We evaluate the results of our experiments with the following statistics:

- $\bar{C}$: the average of the max edge-crossing value;
- s: standard deviation value;
- *Best*: the number of best solutions found;
- *Opt*: the number of optimal solutions found;

Type	#inst.	Vertices (n)	n_i	Layers (l)	Density (d)	Class
			49 instances solved with CPLEX			
uniform	29	$100 - 200$	10	$10 - 20$	≈ 1.5	Low
uniform	20	450	15	30	$\approx 1.7 - 3.6$	Low
			301 instances in the heuristic comparison			
uniform	60	$60 - 1000$	20	$3 - 50$	$\approx 10.0 - 14.0$	High
connected	95	1000	$10 - 40$	$25 - 100$	$\approx 2.0 - 5.0$	Low
North	58	$30 - 553$	$1 - 51$	$2 - 39$	≈ 1.0	Low
Rome	88	$12 - 102$	$1 - 16$	$5 - 19$	≈ 1.0	Low
			190 uniform instances in the δ test			
var-density	70	450	15	30	$1.70 - 10.85$	All
var-nodes	60	$300 - 750$	$10 - 25$	30	≈ 5.0	Medium
var-layers	60	$150 - 900$	15	$10 - 60$	≈ 5.0	Medium

Table 3: Benchmark set.

- % *dev*: the average percent deviation with respect to the best solution found in the experiment. The deviation (in %) is computed as

$$\frac{c_h - c_{best}}{c_{best}} \cdot 100,$$

where c_h is the cost of the solution of the heuristic algorithm evaluated and c_{best} is the cost of the best solution found on a given instance. This measure shows how far the heuristic is to the best value. Smaller percent deviations therefore mean better results.

- % *gap*: the average percent deviation of C value of the heuristic method with respect to the CPLEX solution found in the experiment;

- *Time*: average total time in seconds to execute the method.

4.1 Preliminary Experiments

In this section we first perform a preliminary test to fine tune the parameters of the algorithm. To avoid the over training of the methods, we consider a fraction of the instances (26 graphs) with different number of layers and densities. We call this subset the *training set*, as opposed to the entire set of instances called the *testing set*.

The first tuning experiment is devoted to the study of the parameters used in the intensification phase, α and *tenure*. The first parameter considered is α, which controls the construction of the vertex candidate list in our intensification phase. We evaluate the results obtained by considering five different values of α: 0, 0.1, 0.5, 0.9 and 1, while *tenure* is initially fixed to a reference value of 5.

Table 4 shows the average results over the 26 instances obtained on the training set with a time limit of 60 seconds for each execution. In particular, it shows the average number of crossing ($\overline{C}$), the average percentage deviation (% *dev*), the number of instances for which the method is able to match the best solution found (*Best*), and the CPU time in seconds. It is clear from these results that the best setup is obtained with $\alpha = 1$, in terms of number of best solution found, and in terms of both average max-crossings and deviations.

training set, 26 instances

α	$\overline{C}$	% *dev*	*Best*	*Time*
0	143.69	4.20	7	63.20
0.1	143.77	5.43	6	62.50
0.5	143.62	4.30	7	62.53
0.9	141.04	3.61	9	61.30
1	**139.96**	**2.21**	**18**	60.75

Table 4: Fine-Tune parameter α for the `TS` procedure.

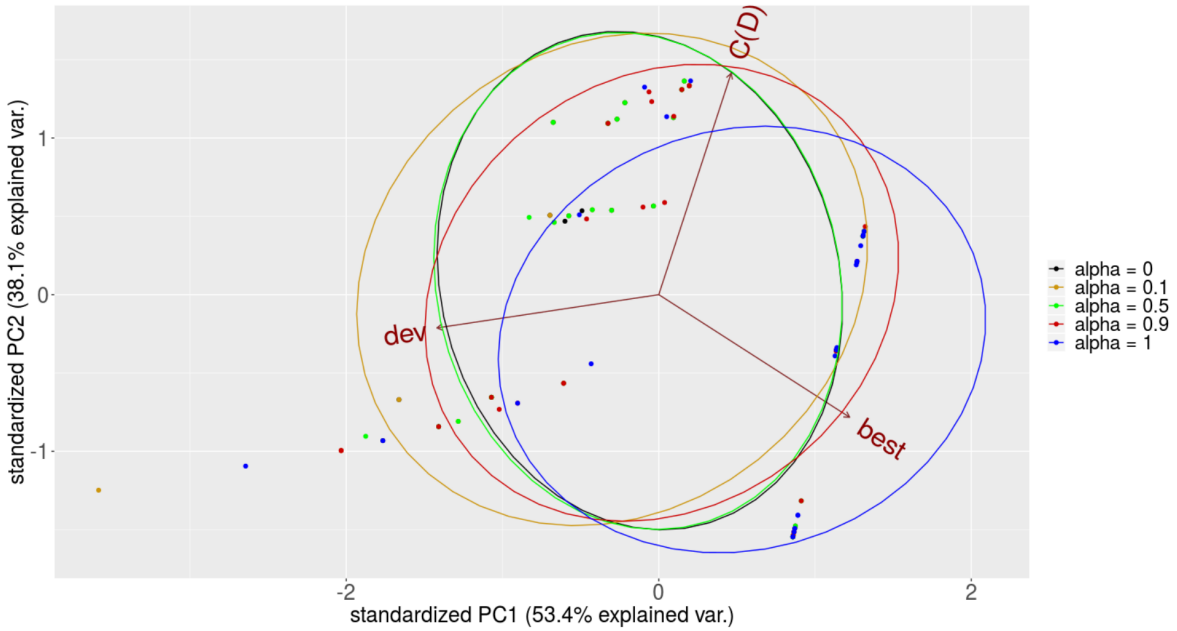


Figure 12: Biplot of the PCA performed in the analysis of the parameter α.

Additionally, to have a better understanding of how the value of α affects the algorithm performance, we conduct a principal component analysis (PCA) on the results reported in Table 4. More specifically, for each instance and each configuration of α, we consider the outcomes of the algorithms as features: $C(D)$, *best*, and $\%dev$. Where $C(D)$ is the maximum number of crossing found, while *best* and $\%dev$ are used to relate the performance of a specific configuration with the others. Namely, *best* is a boolean variable that is equal to 1 when that specific configuration reaches the best objective function value on that specific instance, and $\%dev$ measures the percentage deviation from the best value. As shown in this table, all configurations have comparable computing times, so *Time* is not included in this experiment. The PCA is performed with the `prcomp` function of the `stats R` package. The results collected in the PCA assess how two principal components are able to explain over 90% of the total variance. Moreover, in accordance to the performances evidenced in Table 4, we can observe in the biplot of the PCA, shown in Figure 12, how the cluster corresponding to the value $\alpha = 1$, represented by the blue ellipse, evidences a strong positive correlation with the boolean variable *best*. Likewise, we can observe how this same cluster exhibits negative correlations with respect to $\%dev$ and $C(D)$.

In our second preliminary experiment, we test the short term `TS` method described in Section 3.2 with several *tenure* values. The competing configuration tested are *tenure* = 2, 3, 5 and 10. The results in Table 5 show that there is not a clear winner. Small values of the tenure appear to lead to better results with *tenure* = 3 (highlighted in bold in the table), providing the best combination of minimum $\overline{C}$ and standard deviation and this is the value we use in our experiments.

training set, 26 instances

tenure	$\overline{C}$	% *dev*	*Best*	*Time*
2	139.58	2.41	14	61.04
3	**139.42**	**1.68**	**15**	60.86
5	139.77	4.89	10	60.83
10	142.31	17.93	2	60.63

Table 5: Fine-Tune parameter *tenure* for the `TS` procedure.

In the last tuning experiment, we compare different configurations for the parameters `MaxIntensification` and `MaxDiversification`, used to measure, respectively, the maximum lengths of the intensification and diversification phases. To achieve the right balance between intensification and diversification, we test the parameters coupled in pairs (`MaxIntensification`, `MaxDiversification`), generating four different configurations: $(10, 3)$, $(20, 10)$, $(30, 15)$, and $(50, 20)$. Since the training set is extremely diverse in terms of density and size of the graph, as can be observed in Table 3, in this experiment we divide the training instances into two classes: low-density graphs and high density graphs.

`MaxIntensification`	`MaxDiversification`	$\overline{C}$	% *dev*	*Best*	*Time*
10	3	15.70	6.96	3	60.14
20	10	15.30	3.11	6	60.27
30	15	15.50	7.39	5	60.27
50	20	**15.20**	**1.11**	**7**	60.20

Table 6: `TS` fine-tune on 10 low density instances.

`MaxIntensification`	`MaxDiversification`	$\overline{C}$	% *dev*	*Best*	*Time*
10	3	**216.75**	**0.66**	**9**	61.53
20	10	218.06	1.18	6	62.50
30	15	217.44	0.74	7	61.18
50	20	219.31	1.73	6	61.20

Table 7: `TS` fine-tune on 16 high density instances.

As expected, the differences found in the training set are reflected in tuning results shown in Table 6 (low density) and Table 7 (high density). In particular, the best parameters

combination seems to be $(50, 20)$ in the case of low density graphs, and $(10, 3)$ for high density graphs. In smaller graphs, characterized by low density, the algorithm is able to perform more intensification steps within the same time limits. Moreover, when the density is low, even with an high number of diversification steps, random swaps can strongly diversify the solution but at the same time worsening its quality in a way that can still be repaired in the next intensification phase. On the other hand, on high density graphs, a large number of random swaps could generate a worsening in the solution quality that the intensification phase can not significantly improve.

We conclude our preliminary experimentation with an empirical study on the contribution of the δ-evaluation function in the search efficiency. The tests are designed to run the `TS` with two different move evaluation functions in the intensification phase: the δ-function, and the plain min-max objective function. More specifically, with the former evaluation, the algorithm performs a move from solution D to the neighboring drawing $\bar{D}$ if $\delta(\bar{D}) < \delta(D)$. On the other hand, in the latter move evaluation setup, intensification moves are performed only when the objective function value decreases. We study in this experiment the percentage of the relative difference of the two solution values:

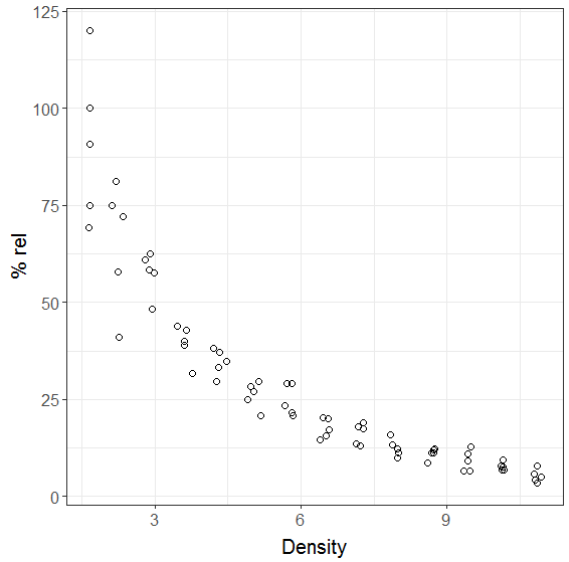
$$\% \; rel = \frac{C(D_{mM}) - C(D_{\delta})}{C(D_{\delta})} \cdot 100, \tag{14}$$

where D_{δ} and D_{mM} are the optimized drawings obtained respectively by using the δ function and the plain min-max objective as move evaluation functions.

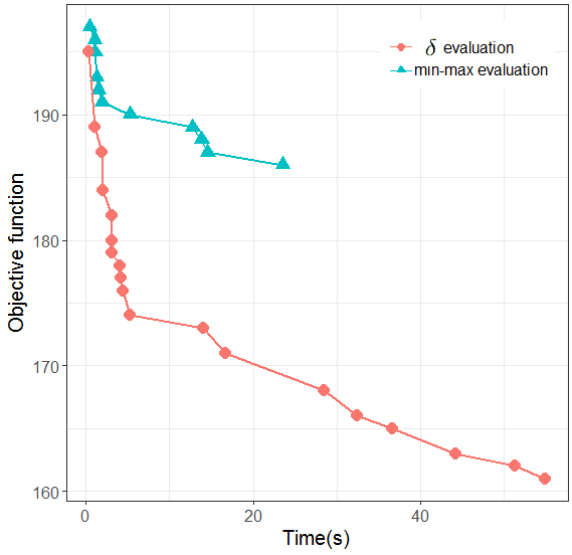
As discussed in Section 3.2, the evaluation of neighboring solutions is particularly hard in large sparse graphs, so we firstly perform our comparison with respect to the density of the graph. Henceforth, to properly relate the performances of the two setups with network density, in this experimental phase we address 70 instances (var-density set) with increasing density and fixed size, both in terms of total vertices and layers. For a summary of the network features used in these experiments see Table 3. The algorithm is executed in both setups for a total time of 60 seconds and using the same parameter configuration for both cases.

As expected, the results in Figure 13-(a) show that the δ-evaluation function is of critical importance for drawing graphs with low density. Moreover, we can remarkably observe how the percentage ratio between the two solutions is always greater than zero, meaning that in all instances the use of δ as evaluation function is always favorable. In addition, Figure 13-(b) depicts an example of the two search profiles obtained for an instance in our benchmark set. We can note how the use of δ as evaluation function lets the algorithm improve when the plain min-max stalls, as well as allowing the procedure to reach earlier good quality solution, as argued in Section 3.2.

We now study of the effectiveness of δ with respect to the variation of the number of vertices in each layer (n_i), and the number of layers (l). As can be observed in the two scatter plots of Figure 14, the advantages related to the use of δ as move evaluation function in the local search are positively correlated with the number of layers and number of vertices in each layer. Additionally, as in the case of the var-density instances, all the $\%rel$ values observed are greater than zero, which confirms that in all the instances tested, it is better to perform the exploration in the search space based on the δ function than on the standard min-max objective function.

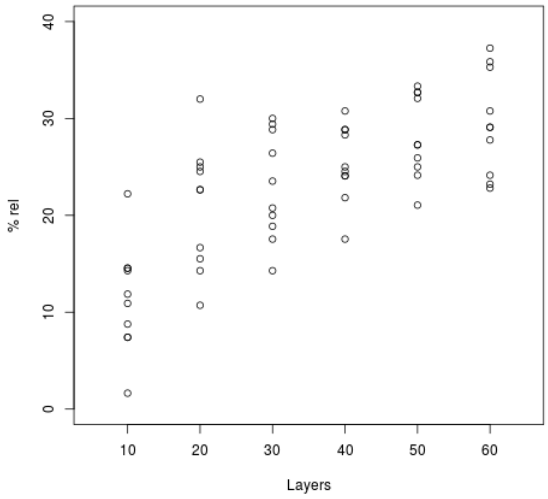


(a) Scatter plot: var-density instances.

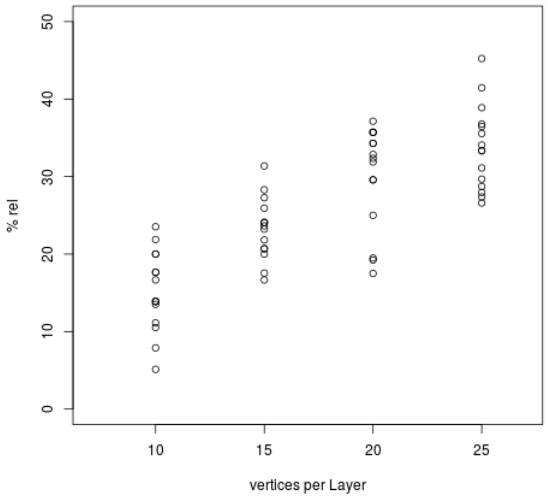


Comparison of two search profiles.

Figure 13: Comparison of δ and min-max evaluation functions on instances of variable density



(a) Scatter plot: var-layers instances.



(b) Scatter plot: var-nodes instances.

Figure 14: Comparison of δ and min-max evaluation functions on the var-layers and var-nodes subsets.

4.2 Comparative Testing

In this section, we first compare our `TS` method with `MCE` and `SO` heuristics, and the solutions obtained with `CPLEX` (v 12.8) with the linear integer formulation. The two heuristic algorithms are executed with a time limit of 60 seconds, and `CPLEX` is configured to stop after 1 hour. Note that when `CPLEX` employs the entire 1 hour of running time, and we force

its early termination, the returning solution is not necessarily optimal. We consider in this experiment the 49 instances of small-to-medium size and low density in the testing set. For a better analysis of the results, the graphs considered in this testing are divided into a sequence of six different groups, growing both in size and density.

As can be seen in Table 8, CPLEX is able to optimally solve almost all the smallest instances belonging to the class $u(10, 10, 1.5)$ and $u(10, 20, 1.5)$. On the other hand, the two heuristics can reach sub-optimal solution in the very short running time of 60 seconds considered in this experiment (which is in line with graph drawing applications). When we move to larger graphs with increasing density, then CPLEX is not able to obtain competitive results while the heuristics are still obtaining good drawings. Note that the negative values of % *gap* in this table, indicate that heuristics outperforms CPLEX on average in those set of instances. In particular, TS clearly outperforms MCE in the three sets with the largest densities (last three columns in the table), as it is shown by the lower percentage deviations from the best known solutions. On small density graphs, $u(15, 30, 1.7)$, MCE performs slightly better than TS.

In our second experiment in this section, we undertake to evaluate the performance of our TS with respect to the two previous heuristics, MCE and SO. We consider 60 instances of mixed size and high density of the *testing set* (uniform set in our benchmark), so according to our previous results, we cannot include CPLEX in this experiment. Our TS is run for 60 seconds in all the cases (i.e., in each instance of the *testing set*). Table 9 reports the solutions of this round of experiments, summed up in Table 10. For the three algorithms we report the average number of crossings ($\overline{C}$), the average total time, the number of best solutions found, and the average percentage deviation from the best solution found.

Analyzing the computational results, we can see how TS obtains the best solution in 57 out of the 60 instances, and shows the minimum average number of crossings among the three procedures. These remarkable performance is reflected in the average percent deviation, which amounts to zero percent in 5 out of 6 instance types, and always lower than 0.60 percent. In particular, we can see in Table 10 how the average deviation over the uniform set is 0.01 percent, compared to 4.77 and 5.76, respectively for MCE and SO. Moreover, we observe how TS achieves high quality solutions in the case of graphs of large size.

On most instances, either MCE or TS performs at least as well as SO. Therefore, to simplify the comparison, we report in Table 11 the results of TS and MCE run for 60 seconds over the four sets of instances in our heuristic benchmark: uniform, connected, North, and Rome. This table shows that, in general terms, we can say that TS improves upon MCE. Specifically, in the uniform set, described in the previous experiment, TS is able to obtain a significant larger number of best solutions and lower deviation than MCE. This is also true in the connected set with low skew. However, in the connected set with higher skew, MCE is better than TS by a close margin. In the small instances in this comparison, North and Rome sets, both methods are comparable, being TS slightly better than MCE. In particular, in the 58 North instances, MCE obtains 46 best solutions and TS 41, but their average percent deviations are 16.0 for MCE and 14.7 for TS. Finally, in the Rome instances, both statistics, number of best solutions and average percent deviation, favor TS.

To complement the information in Table 11, we analyze the percentage deviation as a function of the density and type of instance. In particular, Figure 15 shows a scatter-plot of the connected instances (a) and the uniform instances (b). For each instance, these diagrams show the percentage deviation value (y-axis) of the two solutions obtained with MCE and TS respectively, represented in the corresponding density column of the instance (x-axis). In this way, we can observe that both methods present different patterns when comparing their

	Instance Class					
	$u(10,10,1.5)$	$u(10,20,1.5)$	$u(15,30,1.7)$	$u(15,30,2.3)$	$u(15,30,2.9)$	$u(15,30,3.6)$
$\overline{C}$						
CPLEX	4.33	5.64	26.00	42.20	53.40	64.60
MCE	5.20	6.36	9.20	18.80	29.80	41.00
SO	5.47	7.36	15.40	23.40	32.80	42.40
TS	5.27	6.50	11.20	17.80	26.00	34.40
s						
CPLEX	0.62	1.15	6.75	4.09	3.58	4.10
MCE	0.68	1.28	1.10	1.64	2.39	1.58
SO	0.74	1.01	0.89	1.34	1.10	1.95
TS	0.80	0.94	1.10	2.95	1.41	1.82
Best						
CPLEX	15/15	14/14	0/5	0/5	0/5	0/5
MCE	4/15	4/14	5/5	1/5	0/5	0/5
SO	4/15	3/14	0/5	0/5	0/5	0/5
TS	3/15	3/14	0/5	4/5	5/5	5/5
Opt						
CPLEX	14/15	7/14	0/5	0/5	0/5	0/5
MCE	4/15	1/14	-	-	-	-
SO	4/15	1/14	-	-	-	-
TS	3/15	1/14	-	-	-	-
% *gap*						
MCE	21.44	13.06	-60.67	-55.32	-44.22	-36.44
SO	29.00	33.45	-35.66	-44.29	-38.45	-34.24
TS	22.22	16.55	-52.40	-57.99	-51.27	-46.66
% *dev*						
MCE	21.44	13.06	0.00	8.61	14.53	19.30
SO	29.00	33.45	68.81	35.65	26.33	23.31
TS	22.22	16.55	22.25	2.00	0.00	0.00
Time						
CPLEX	449.04	2487.51	3600.00	3600.00	3600.00	3600.00
MCE	60.00	60.00	60.00	60.00	60.00	60.00
SO	60.00	60.00	60.00	60.00	60.00	60.00
TS	60.07	60.14	60.21	60.26	60.32	60.70

Table 8: Comparison of TS, SO, MCE, and CPLEX on 49 instances (low density).

deviation values with the density. Specifically, MCE exhibits a similar variability of % *dev* in all density values, but TS reduces its variability with larger density values. These results are in line with the previous experiments in which we already observed that MCE performs better in lower density values, while TS is better suited for medium density instances.

	Instance Class					
	$u(20, 15, 14)$	$u(20, 3, 10)$	$u(20, 5, 12)$	$u(20, 25, 14.5)$	$u(20, 40, 10)$	$u(20, 50, 10)$
$\overline{C}$						
MCE	269.30	246.80	254.00	270.30	175.00	177.50
SO	267.20	241.80	254.70	273.30	181.80	184.70
TS	259.30	238.20	247.80	261.80	164.10	163.00
s						
MCE	9.88	11.24	4.81	5.52	2.12	2.46
SO	4.96	7.61	6.90	2.83	1.69	2.21
TS	2.16	7.91	4.98	4.24	3.73	3.20
Best						
MCE	0/10	2/10	1/10	1/10	0/10	0/10
SO	0/10	2/10	1/10	0/10	0/10	0/10
TS	10/10	7/10	10/10	10/10	10/10	10/10
% dev						
MCE	3.85	4.31	2.51	3.27	6.99	8.92
SO	3.04	2.12	2.79	4.41	10.84	13.34
TS	0.00	0.59	0.00	0.00	0.00	0.00
Time						
MCE	60.00	60.00	60.00	60.00	60.00	60.00
SO	80.60	32.20	68.50	145.80	63.90	40.60
TS	61.25	60.53	60.74	62.02	62.02	61.87

Table 9: Comparison of MCE, SO, and TS on 60 instances (high density).

Procedure	$\overline{C}$	*s*	*% dev*	*Best*	*Time*
MCE	232.23	41.13	4.98	4	60.00
SO	233.92	37.78	6.09	3	71.93
TS	**222.37**	42.90	**0.09**	**57**	61.40

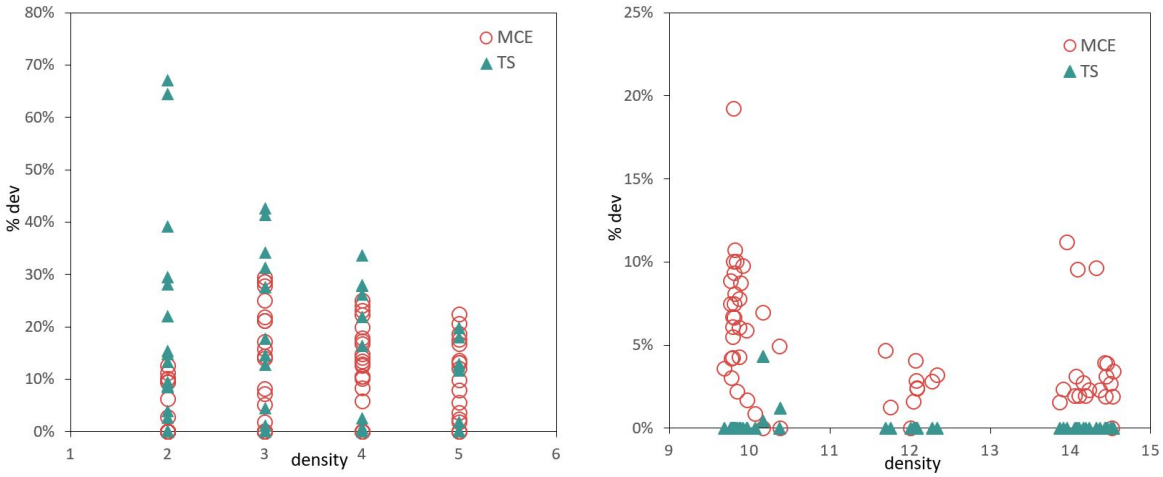
Table 10: Summary of heuristics performance.

5 Conclusions

We had a twofold goal for this work: to experiment with the implementation of tabu search on a min-max problem and, in the process, to develop a state-of-the-art procedure for the crossing reduction problem. The framework that we consider in this work is based on edge-crossing minimization, as this is one of the most common ways of creating good graph representations. In particular, we target a hard graph drawing variant recently proposed: minimizing the maximum number of edge crossings in layered graphs. Additionally, we adapt a mathematical formulation to model the problem in the multilayer case.

	Instance Class				
	uniform	connected $\sigma = 2$	connected $\sigma = 4, 8$	North	Rome
$\overline{C}$					
MCE	232.23	62.34	139.52	3.05	1.63
TS	222.37	55.23	157.72	2.90	1.51
s					
MCE	41.13	48.58	122.99	3.87	1.05
TS	42.90	46.31	151.44	2.82	0.80
Best					
MCE	6/60	3/35	36/60	46/58	67/88
TS	57/60	35/35	25/60	41/58	74/88
% *dev*					
MCE	4.77	14.69	4.81	16.0	10.5
TS	0.01	0.00	13.08	14.7	8.7

Table 11: Comparison between MCE and TS on the heuristic benchmark.



(a) Scatter plot: connected instances.

(b) Scatter plot: uniform instances.

Figure 15: Comparison of density and the percentage of deviation on connected and uniform subsets.

The emphasis on responsive exploration in tabu search derives from the supposition that, in a system that uses memory, a bad choice based on strategy can often yield more information about how the strategy may advantageously change than a good random choice, [13]. Our tabu search method implements two memory structures, short term and long term, for an efficient search exploration based on the search history. The experimentation shows that they are indeed very effective compared with the randomized design of the previous heuristic SO. The use of an auxiliary evaluation function to guide the search in the flat landscape presented by

	Instance Class					
	$u(20, 15, 14)$	$u(20, 3, 10)$	$u(20, 5, 12)$	$u(20, 25, 14.5)$	$u(20, 40, 10)$	$u(20, 50, 10)$
cross-sum						
`MCE`	276040.8	37764.6	76643.5	475970.2	334284.9	421127.6
`SO`	250323.1	34821.3	70594.3	429655.6	278458.1	349641.2
`TS-p`	254133.4	35704.0	72282.8	434861.6	286283.5	260115.7
% dev_s						
`MCE`	10.28	8.55	8.58	10.78	20.05	20.21
`SO`	0.00	0.00	0.00	0.00	0.00	0.00
`TS-p`	1.52	2.54	2.39	1.21	2.81	3.00

Table 12: Analysis of the performances of the post-processing procedure on 60 high-density instances.

the min-max crossing problem is a key feature of our heuristic, as our preliminary experiments confirm. Our testings also confirms that it is responsible of reaching high-quality solutions in the search method. The comparison with two previous heuristics, with moves are based only on the objective function, demonstrates the good performance of our tabu search method. The experimentation also reveals that `CPLEX`, using our mathematical formulation, is able to solve only instances with low density and very small size, and, as expected, requires long running times.

Acknowledgments. This work has been partially supported by the Spanish Ministerio de Ciencia, Innovación y Universidades (MCIU/AEI/FEDER, UE) with grant ref. PGC2018-095322-B-C21.

References

[1] A. Alonso-Ayuso, L. F. Escudero, M. Guignard, and A. Weintraub. On dealing with strategic and tactical decision levels in forestry management under uncertainty. *Submitted.*

[2] O. Bastert and C. Matuszewski. *Layered Drawings of Digraphs*, pages 87–120. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

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

[4] S. Bhatt and F. T. Leighton. A framework for solving vlsi graph layout problems. *Journal of Computer and System Sciences*, 28:300–343, 1984.

[5] M. Burch, C. Müller, G. Reina, H. Schmauder, M. Greis, and D. Weiskopf. Visualizing Dynamic Call Graphs. In M. Goesele, T. Grosch, H. Theisel, K. Toennies, and B. Preim, editors, *Vision, Modeling and Visualization.* The Eurographics Association, 2012.

[6] M-J. Carpano. Automatic display of hierarchized graphs for computer-aided decision analysis. *IEEE Transactions on Systems, Man, and Cybernetics*, 10(11):705–715, 1980.

[7] L. De Cecco, M. Giannoccaro, E. Marchesi, P. Bossi, F. Favales, L.D. Locati, L. Licitra, S. Pilotti, and S. Canevari. Integrative mirna-gene expression analysis enables refinement of associated biology and prediction of response to cetuximab in head and neck squamous cell cancer. *Genes*, 8(1):35, 2017.

[8] J. Chen and I. T. Chau. The hierarchical dependence diagram: improving design for reuse in object-oriented software development. In *Proceedings of 1996 Australian Software Engineering Conference*, pages 155–166, Jul 1996.

[9] M. Chimani, C. Gutwenger, P. Mutzel, M. Spönemann, and H. Wong. Crossing minimization and layouts of directed hypergraphs with port constraints. *Lecture Notes in Computer Science 6502 LNCS*, pages 141–152, 2011.

[10] M. Fernández-Ropero, R. Pérez-Castillo, and M. Piattini. Graph-based business process model refactoring. In *SIMPDA*, pages 16–30, 2013.

[11] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Science*, 8:156–166, 1977.

[12] F. Glover. Tabu search: Part i. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[13] F. Glover and M. Laguna. *Tabu Search*, pages 70–150. Blackwell Scientific Publications, Oxford, 1993.

[14] F. Glover, M. Laguna, E. Taillard, and D. de Werra. Tabu search. *Annals of Operations Research*, 41, 1993.

[15] T. Gschwind, J. Pinggera, S. Zugal, H. A. Reijers, and B. Weber. A linear time layout algorithm for business process models. *J. Vis. Lang. Comput.*, 25(2):117–132, April 2014.

[16] P. Helay and N. S. Nikolov. *Hierarchical Drawing Algorithms*, pages 409–453. Chapman and Hall/CRC, 2013.

[17] C. Hu, Y. Li, X. Cheng, and Z. Liu. A virtual dataspaces model for large-scale materials scientific data access. *Future Generation Computer Systems*, 54:456 – 468, 2016.

[18] M. Jünger, E. K. Lee, P. Mutzel, and T. Odenthal. A polyhedral approach to the multilayer crossing minimization problem. In *International Symposium on Graph Drawing*, pages 13–24. Springer, 1997.

[19] M. Jünger and P. Mutzel. 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms. *Journal of Graph Algorithms and Applications*, 1:Paper 1, 25 p., 1997.

[20] H-P. Kriegel, P. Kröger, M. Renz, and T. Schmidt. Hierarchical graph embedding for efficient query processing in very large traffic networks. In B. Ludäscher and N. Mamoulis, editors, *Scientific and Statistical Database Management*, pages 150–167. Springer Berlin Heidelberg, 2008.

[21] N. W. Lemons, B. Hu, and W. S. Hlavacek. Hierarchical graphs for rule-based modeling of biochemical systems. *BMC Bioinformatics*, 12(1):45, Feb 2011.

[22] M. Lozano, A. Duarte, F. Gortázar, and R. Martí. Variable neighborhood search with ejection chains for the antibandwidth problem. *Journal of Heuristics*, 18(6):919–938, 2012.

[23] R. Martí. A tabu search algorithm for the bipartite drawing problem. *European Journal of Operational Research*, 106:558–569, 1998.

[24] R. Martí, V. Campos, A. Hoff, and J. Peiró. Heuristics for the min-max arc crossing problem in graphs. *Expert Systems with Applications*, 2018.

[25] R. Mateescu, R. Dechter, and R. Marinescu. And/or multi-valued decision diagrams (aomdds) for graphical models. *J. Artif. Int. Res.*, 33(1):465–519, December 2008.

[26] C. Matuszewski, R. Schönfeld, and P. Molitor. Using sifting for k-layer straightline crossing minimization. In J. Kratochvíyl, editor, *Graph Drawing*, pages 217–224, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[27] A. Napoletano, A. Martínez-Gavara, P. Festa, T. Pastore, and R. Martí. Heuristics for the constrained incremental graph drawing problem. *European Journal of Operational Research*, 2018.

[28] B. Oselio, A. Kulesza, and A. O. Hero. Multi-layer graph analytics for social networks. In *2013 5th IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, pages 284–287, Dec 2013.

[29] E. Rodriguez-Tello, J. K. Hao, and J. Torres-Jimenez. An improved simulated annealing algorithm for bandwidth minimization. *European Journal of Operation Research*, 185(3):1319–1335, 2008.

[30] R. Ruiz and T. Stützle. An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research*, 187(3):1143–1159, 2008.

[31] M. F. Stallmann. A heuristic for bottleneck crossing minimization and its performance on general crossing minimization: Hypothesis and experimental study. *ACM Journal of Experimental Algorithms*, 17(1):1–30, 2012.

[32] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Trans. Syst. Man, Cybern.*, 11:109–125, 1981.

[33] J. Vanhatalo, H. Völzer, F. Leymann, and S. Moser. Automatic workflow graph refactoring and completion. In A. Bouguettaya, I. Krueger, and T. Margaria, editors, *Service-Oriented Computing – ICSOC 2008*, pages 100–115. Springer Berlin Heidelberg, 2008.