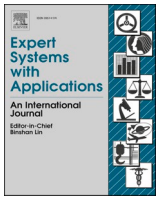
Expert Systems With Applications 243 (2024) 122668

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa



Strategic oscillation tabu search for improved hierarchical graph drawing

Sergio Cavero [a], Eduardo G. Pardo [a,*], Fred Glover [b], Rafael Martí [c]

[a] Departamento de Informática y Estadística, Universidad Rey Juan Carlos, Spain
[b] Entanglement, Inc., CO, USA
[c] Departamento de Estadística e Investigación Operativa, Universitat de València, Spain

ARTICLE INFO

Keywords:
Graph drawing
Arc crossing
Strategic oscillation
Tabu search
Metaheuristics

ABSTRACT

In the last years, many areas in science, business, and engineering have experienced an enormous growth in the amount of data that they are required to analyze. In many cases, this analysis relies intimately on data visualization and, as a result, graph drawing has emerged as a new field of research. This paper addresses the challenge of drawing hierarchical graphs, which is one of the most widely used drawing standards. We introduce a new mathematical model to automatically represent a graph based on the alignment of long arcs, which we combine with the classic arc crossing minimization objective in hierarchical drawings. We complement our proposal with a heuristic algorithm that can obtain high-quality results in the short computational time required by graph drawing systems. Our algorithm joins two methodologies, tabu search and strategic oscillation (SOS), to perform a fast and effective exploration of the search space. We conduct extensive experimentation that integrates our new mathematical programming formulation and the SOS tabu search that targets large instances. Our statistical analysis confirms the effectiveness of this proposal.

1. Introduction

Graph drawing can be defined as the discipline that deals with the representation of a graph data structure in a particular metric space (Battista et al., 1998; Kaufmann & Wagner, 2001). A graph is made of two sets, a vertex set, and an edge set (West, 2001), and it is often used for modelling real systems that can be processed by a computer, where the vertices represent the elements of the system, and the edges represent the connections among the elements. Those models usually need to be graphically depicted, especially when they represent a solution to a particular problem. In those situations, graph drawing appears to be an important tool. In fact, graph drawing is applied to represent many real applications such as: maps, circuit designs, social networks, neural network representation, control systems, or facilities layouts, among others (Carpano, 1980; Chen et al., 2021; Napoletano et al., 2019; Preitl, 2006; Rigatos, 2017; Tan, 2014; Zamfirache, 2023).

Solving large problems usually makes impossible to perform the graph drawing by hand and it is mainly performed by using specialized software such as yEd (yWorks, 2023), GraphViz (Gansner & North, 2000), dot (Gansner et al., 2015), AGD (Paulisch & Tichy, 1990), or OGDF (Chimani et al., 2013), among others. This kind of software applies general procedures to position nodes and arcs to produce graphs representations with desired properties.

Graphs can be represented using different standards (Battista et al., 1998) with the aim of improving the readability of the graph. Based on that, a wide range of applications have been developed. In this paper, we focus on hierarchical representations. In Fig. 1(a) we depict an example of a graph G with 10 vertices and 10 arcs, and in Fig. 1(b) we show a hierarchical representation of the same graph with the vertices arranged in three layers (L_1, L_2, and L_3).

General graphs can be represented as a hierarchy using the Sugiyama's framework (Sugiyama et al., 1981), which obtains a drawing of the graph following some aesthetic criteria, with the aim of improving its readability, such as: the minimum number of crossings, uniform direction, and straight lines, among others. However, satisfying some of the aesthetic criteria aforementioned is not an easy task, especially because some of these criteria are in conflict. Furthermore, some of them constitute difficult (NP-hard) optimization problems. However, it is well accepted that the most important and difficult problem is arc crossing minimization (Carpano, 1980), which has been widely studied in the scientific literature, either from the exact (Glover et al., 2021; Jünger & Mutzel, 2002) and heuristic (Laguna et al., 1997; Martí, 2001) points of view.

In this research we focus on drawing of hierarchical graphs with the

* Corresponding author.
E-mail addresses: sergio.cavero@urjc.es (S. Cavero), eduardo.pardo@urjc.es (E.G. Pardo), fred@entanglement.com (F. Glover), rafael.marti@uv.es (R. Martí).

https://doi.org/10.1016/j.eswa.2023.122668
Received 6 January 2023; Received in revised form 5 July 2023; Accepted 16 November 2023
Available online 23 November 2023

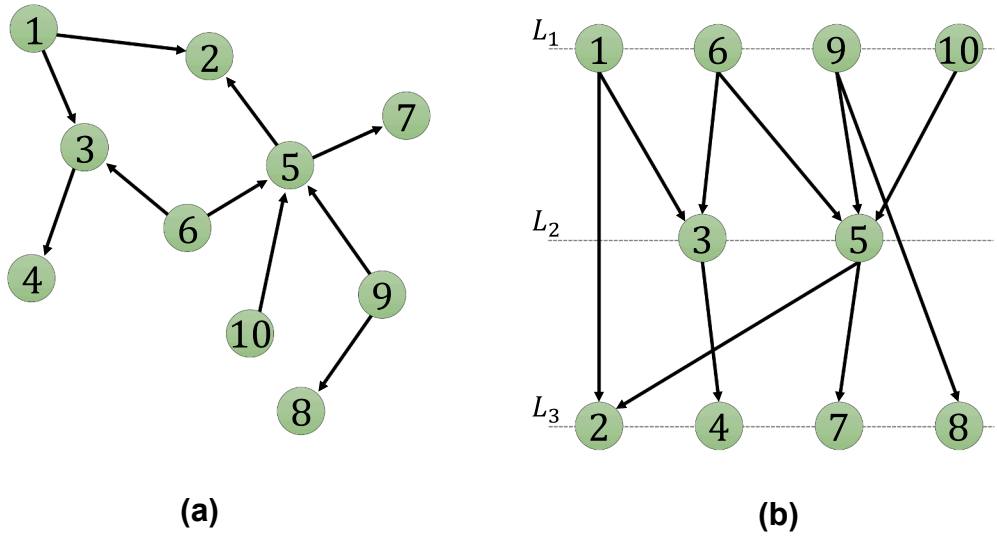


Fig. 1. (a) Original graph G. (b) Hierarchical representation H of graph G.

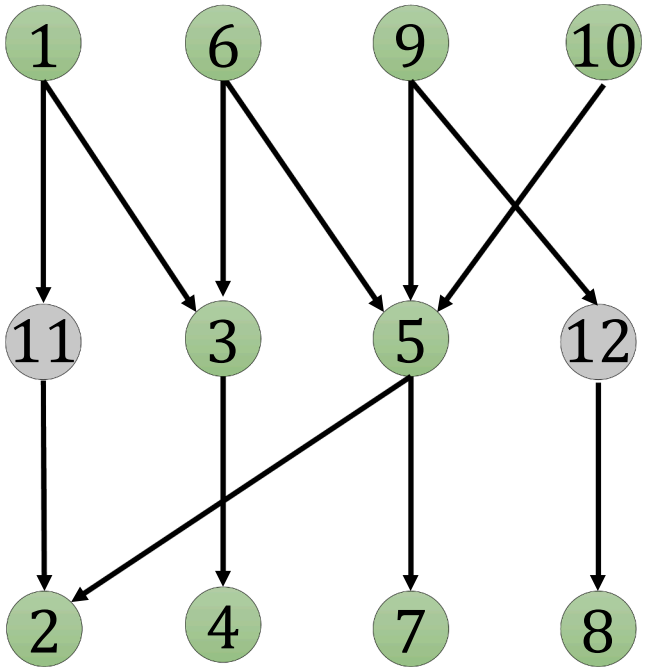


Fig. 2. Proper Hierarchy graph PH of H.

aim of improving the readability of its representation through the minimization of arc crossings, subject to the additional constraint recently introduced in Glover et al. (2021), which restrict the arcs between nodes placed in non-consecutive layer (denoted as long arcs) to be straight lines (i.e., no bends are allowed). Sugiyama's framework is based on replacing long arcs connecting nodes in non-consecutive layers with chains of dummy nodes. This constraint forces them to be placed in the same relative position in their layers, thus forming a straight long arc with no bends.

The main contributions of this work are summarized next: First, we propose an alternative mathematical model to minimize arc crossings including the long arc constraint to avoid bends. As it will be shown, this model is able to solve small problems to optimality. Second, we introduce a multi-start metaheuristic to obtain high quality solutions to target large instances. This method has two components, a constructive method based on tabu search (Glover and Laguna, 1998) to generate good initial solutions, and a strategic oscillation (Glover et al., 1984) based on multiple neighborhoods to obtain improved outcomes. Finally, both, exact and heuristic approaches are compared through extensive computational tests over a benchmark set of test instances exhibiting varied characteristics.

The rest of the paper is organized as follows. Section 2 formalizes the graph drawing problem, and Section 3 presents the mathematical model implemented to solve the problem with an exact commercial optimization software (Gurobi). Our proposed metaheuristic procedure is introduced in Section 4: first, presenting the overall scheme of our multi-start method, followed in Section 4.1 with a detailed description of the different constructive components. The core of the paper comes in Section 4.2, where we introduce an intensification procedure based on the strategic oscillation strategy (SOS), applied to improve the initial solutions and to perform an intelligent exploration of the solution space. Section 5 reports an extensive computational study in which we compare the best variant of our approach with the best previous algorithm in the literature. The paper ends with the customary conclusions section in which we summarize our findings.

2. Problem statement

Formally, let $H = (V, A, nl, L)$ and $G = (V, A)$, be a hierarchical and a general graph respectively, with V representing the set of vertices and A representing the set of arcs. Also, let $L(v) : V \rightarrow \{1, 2, \ldots, nl\}$ be the layering function which associates a layer (i.e., an index number) to a node v. Therefore, $L(v) - L(u)$ calculates the length of the arc $(u, v) \in A$. Hence, L implicitly defines the sets of layers $L_h = \{v \in V : L(v) = h\}$ with $h = 1, 2, \ldots, nl$. We denote as long arcs to those arcs linking two nodes placed in non-consecutive layers in H. Therefore, an arc (u, v) is a long arc if $L(v) > L(u) + 1$.

We illustrate these concepts and definitions with the examples depicted in Fig. 1. Particularly, the graph G in Fig. 1(a), is composed of $|V| = 10$ and $|A| = 10$, where $V = \{1,2,3,4,5,6,7,8,9,10\}$, and $A = \{(1, 2), (1, 3), (3, 4), (5, 2), (5, 7), (6, 3), (6, 5)(9, 5), (9, 8), (10, 5)\}$. Similarly, Fig. 1(b) depicts a 3-layer hierarchical graph H obtained from G, where the three layers are differentiated by a gray dashed line, where $L_1 = \{1, 6, 9, 10\}$, $L_2 = \{3, 5\}$, and $L_3 = \{2, 4, 7, 8\}$. Observing, for example, nodes 8 and 9, they are allocated in L_3 and L_1 respectively. Therefore, $L(8) = 3, L(9) = 1$, and arc $(9, 8)$ has a length of $L(8) - L(9) = 3 - 1 = 2$. Since $L(8) > L(9) + 1$, arc $(9, 8)$ is considered a long arc.

Since we are trying to find representations of hierarchy graphs

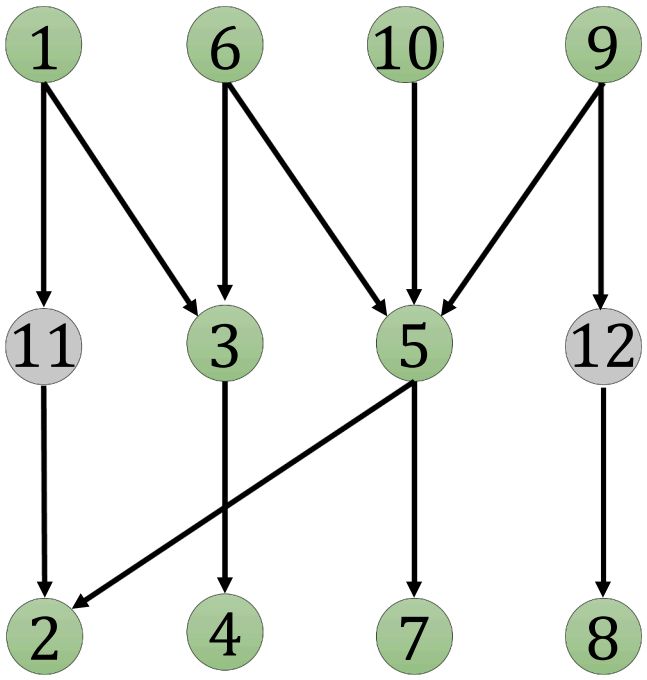


Fig. 3. Valid drawing of the proper hierarchy graph *PH*.

without bends in long arcs, we need to introduce Proper Hierarchy Graphs (Sugiyama et al., 1981; Warfield, 1977). Particularly, given the previous definitions, a hierarchy graph could be represented as a Proper Hierarchy Graph, replacing long arcs in the hierarchy, $A_L \subseteq A$, with a series of dummy nodes in a chain, where the length of the arcs in the chain is 1, obtaining what is called a proper hierarchy $PH = (V \cup V', A', nl, L)$, in which the set of nodes V' contains all the dummy nodes added in this process. The set of arcs A', on the other hand, contains the arcs of length 1 in the original H, and the new arcs obtained from the chains of arcs which replace the long arcs in A. Therefore, all the arcs in A' are of length 1.

We apply this method to the graph H in Fig. 1(b). The resultant proper hierarchy is depicted in Fig. 2, where we identify two chains of dummy nodes colored with grey. The arc $(1,2)$ in Fig. 1(b), is replaced with the chain $\{(1,11),(11,2)\}$ in this figure. Similarly, the arc $(9,8)$ in Fig. 1(b) is replaced with the chain $\{(9,12),(12,8)\}$. Therefore, nodes 11 and 12, colored in grey, are dummy nodes.

Given a proper hierarchy graph, like the one in Fig. 2 our objective is to simultaneously organize the vertices in such a way that crossings are minimized, subject to the constraint that there are no bends in the long arcs, producing what we call an *aligned hierarchy*. Fig. 3 shows an aligned hierarchy of the graph in our example. Note that the chains of arcs that replace the two long arcs $(1,2)$ and $(9,8)$, are now aligned.

Now we introduce some basic notation to formalize this problem. Given the proper hierarchy $PH = (V \cup V', A', nl, L)$, we denote the coordinate of a node u by $x(u)$. For example, in Fig. 3 we can see that node 5 is in layer 2 (numbering the horizontal layers from top to bottom) and has a coordinate of 3 (numbering the columns from left to right). In mathematical terms $L(5) = 2$ and $x(5) = 3$. Then, an **aligned drawing** of *PH* has each vertex in the chain of vertices which replace a long arc with the same x-coordinate. Mathematically, for each long arc $(u,v) \in A_L \in A$ modeled as the chain of arcs of length 1, $AC(u,v) = \{(u,u_1),(u_1,u_2),\ldots,(u_s,v)\}$, and including the arcs in the chain into A', we have $x(u_i) = x(u_{i+1})$ for $i = 1,\ldots,s-1$.

Given a graph $G = (V, A)$, when we generate a proper hierarchy denoted by $PH = (V \cup V', A', nl, L)$, the layering function L is provided by the y-coordinate of each node in V. Therefore, a solution to the drawing problem is determined in *PH* with the x-coordinate of each node. We therefore denote a drawing, or a solution to our problem, by the notation (x,L). For the sake of brevity, considering that L is fixed in the hierarchy, a solution or drawing can be simply denoted as x.

To state the problem formally, we define the alignment of a long arc (u,v), denoted as $LA(u,v)$, as:

$$LA(u,v) = \sum_{(w,z)\in AC(u,v)} |x(w) - x(z)|, \tag{1}$$

We can then compute the alignment of a solution x, as follows:

$$LA(x) = \sum_{(u,v)\in A_L} LA(u,v) \tag{2}$$

In this paper, we consider that to obtain a valid drawing of the graph, any feasible solution must satisfy $LA(x) = 0$.

For example, to calculate the alignment of the solution depicted in Fig. 2 we compute:

$$LA(x) = LA(1,2) + LA(9,8) = 0 + 1 = 1$$

where,

$$LA(1,2) = |x(1) - x(11)| + |x(11) - x(2)| = |1-1| + |1-1| = 0 + 0 = 0$$

$$LA(9,8) = |x(9) - x(12)| + |x(12) - x(8)| = |2-3| + |3-3| = 0 + 1 = 1$$

Note that in this example $LA(x) > 0$, and hence this solution is infeasible and would not represent a valid drawing.

Similarly, we can compute the alignment of the solution depicted in Fig. 3 as follows:

$$LA(x) = LA(1,2) + LA(9,8) = 0 + 0 = 0$$

where,

$$LA(1,2) = |x(1) - x(11)| + |x(11) - x(2)| = |1-1| + |1-1| = 0 + 0 = 0$$

$$LA(9,8) = |x(9) - x(12)| + |x(12) - x(8)| = |3-3| + |3-3| = 0 + 0 = 0$$

In this case, $LA(x) = 0$, and hence this solution represents a feasible solution, and therefore a valid drawing.

For the formal mathematical representation, given a solution x, the objective function expressing the number of crossings is denoted as $C(x)$, and the constraint term expressing the alignment of long arcs is denoted as $LA(x)$. $C(x)$ is calculated as the sum of all the crossings between any pair of arcs $(u,v),(w,z) \in A'$ as follows:

$$C(x) = \sum_{(u,v),\ (w,z)\in A'} C((u,v),(w,z)) \tag{3}$$

where, $C((u,v),(w,z))$ takes the value 1 if there is a crossing between both arcs. More formally:

$$C((u,v),(w,z)) = \begin{cases} 1, x(u) < x(v) \wedge x(z) < x(w) \\ 1, x(v) < x(u) \wedge x(w) < x(z) \\ 0, otherwise. \end{cases} \tag{4}$$

Finally, the aim of this optimization problem is to find a solution $x^\star$, among the set of all possible solutions X that minimizes Equation (3), subject to the alignment constraint for long arcs. Specifically, it is defined as:

$$x^\star \leftarrow \min_{x\in X} C(PH,x), \text{s.t. } LA(x) = 0 \tag{5}$$

For example, the solution shown in Fig. 3 has 1 crossing ($C(x) = 1$) and it is aligned ($LA(x) = 0$), therefore identifying a feasible solution of the problem. Our objective is to identify an optimal solution that minimizes $C(x)$. In this example the solution shown is optimal, and we cannot obtain a feasible solution with no crossings. As we could observe, the dummy nodes were necessary to determine if the long arcs were aligned

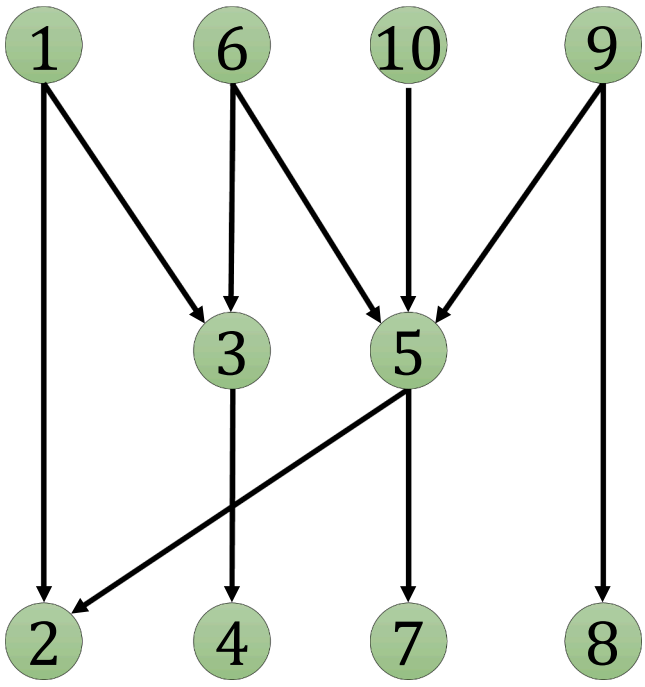


Fig. 4. Optimal graph drawing for input graph G.

or not. Once this constraint is guaranteed in the provided solution, we can replace the dummy nodes with the original long arcs, obtaining the final drawing, which is shown in Fig. 4.

3. Mathematical model

In this section we present the mathematical model of the arc crossing minimization problem in graph drawing. As mentioned, the most popular method is the Sugiyama's framework. In 1981, Sugiyama, Tagawa, and Toda (Sugiyama et al., 1981) revolutionized the graph drawing field by introducing a three-step algorithm that can be applied to drawing any graph (Sugiyama, 2002; Tantau, 2013). Specifically, the framework obtains a drawing of a general graph by first transforming it into a hierarchy graph, in which vertices are arranged into parallel lines, and then applying several algorithms to optimize certain aesthetic criteria to increase the readability of the resulting diagram. This framework has become a graph drawing standard that has been applied in many different contexts (e.g., (Kaufmann & Wagner, 2001; Napoletano et al., 2019; Pastore et al., 2020). The **first step** in Sugiyama's framework is called ***layer assignment***, and basically consists of assigning the nodes of the input graph to layers. The **second step** targets ***arc crossing minimization***. Finally, the **third step** of the framework ***relocates the nodes*** within a layer without changing the order obtained in the second step.

In this paper, we focus on the second step, consisting of minimizing the arc crossing in a graph representation. This problem has been considered relevant by the scientific community given the attention received. In the first implementations simple ordering rules were applied (Carpano, 1980). However, the field of optimization has benefited in the last two decades from the introduction of complex metaheuristics that are able to operate with more advanced rules. For example, Martí (1998) and Laguna & Marti (1999) proposed advanced solution strategies based on the tabu search methodology which were extended in Sánchez-Oro et al. (2017) and Martí et al. (2018) to target some key applications for arc crossing minimization.

The general problem can be formulated in mathematical terms (Jünger & Mutzel, 2002) based on binary variables c_{ijkl} which are used to compute the objective function of the problem. Particularly, each c_{ijkl} takes the value 1 when a crossing between arcs $(i,j),(k,l)$ occurs. In this model, variables x_{ij}^h take the value 1 when node i precedes node j in layer h; and 0 otherwise. The entire model follows

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

$$x_{ik}^h + x_{lj}^{h+1} - c_{ijkl} \leq 1 (i,j),(k,l) \in A', i < k, j \neq l, h = 1, \ldots, nl-1 \quad (6)$$

$$x_{ki}^h + x_{jl}^{h+1} - c_{ijkl} \leq 1 (i,j),(k,l) \in A', i < k, j \neq l, h = 1, \ldots, nl-1 \quad (7)$$

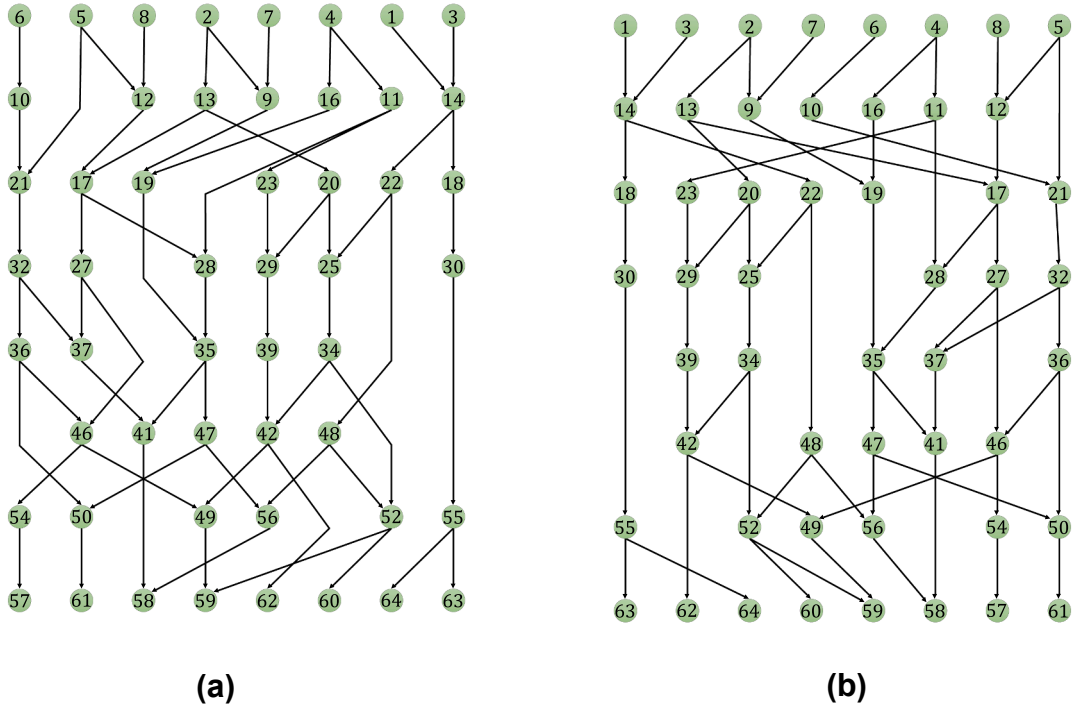


Fig. 5. Optimal solution drawings for crossing minimization.

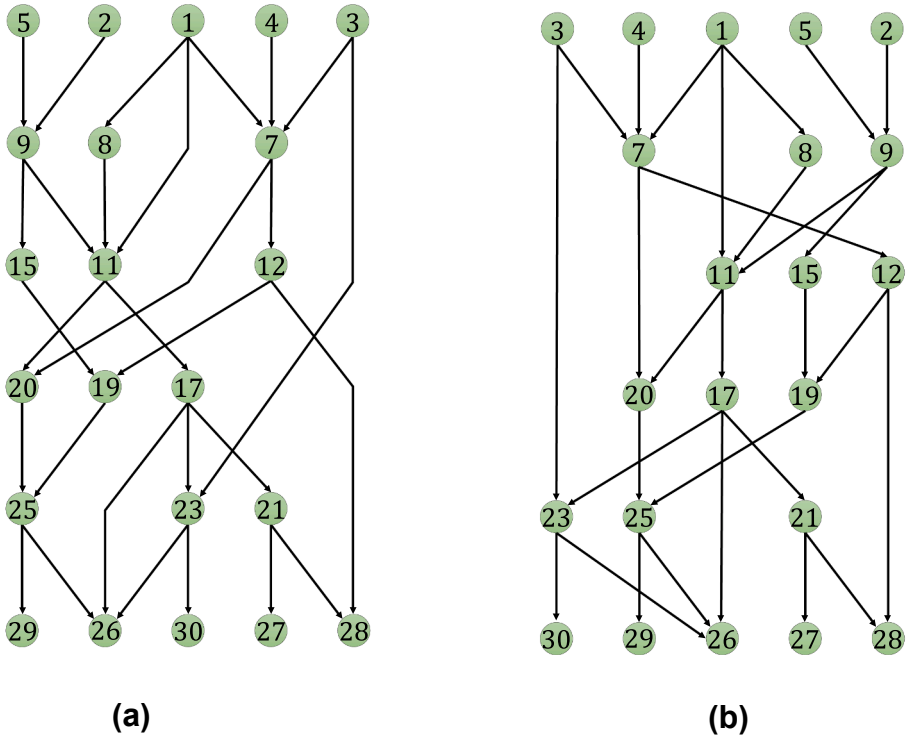


Fig. 6. Optimal drawings for crossing minimization, subject to the alignment constraints.

$$x^h_{ij} + x^h_{jk} + x^h_{ki} \leq 21 \leq i < j < k \leq n_h, h = 1, \ldots, nl \tag{8}$$

$$x^h_{ij} + x^h_{ji} = 11 \leq i < j \leq n_h, h = 1, \ldots, nl \tag{9}$$

$$x^h_{ij}, c_{ijkl} \in \{0, 1\}$$

This model has been deeply studied in the optimization literature, giving its connections with the well-known linear ordering problem, LOP (Martí & Reinelt, 2022) which has been established for its importance to obtain a readable drawing. Constraints (6) and (7) above force $c_{ijkl} = 1$ when there is a crossing. Constraints (8) are the so-called 3-dicycle constraints, originally proposed for the LOP (Jünger & Mutzel, 2002) and guarantee that, together with constraint (9), the variables model a consistent ordering.

The integer linear formulation proposed in Jünger & Mutzel, (2002) for the minimization of arc crossing, with constraints (6)–(9), do not consider the alignment constraints for long arcs. (Glover et al., 2021) adapted that formulation to include these constraints. Particularly, for each long arc (u, v), from layer h to layer $h + s$, and intermediate nodes$u_1, u_2, \ldots, u_s$, the alignment constraints take the form:

$$\sum_{1 \leq i \leq n_h} x^h_{iu} - \sum_{1 \leq i \leq n_{h+1}} x^{h+1}_{iu_1} = 0$$

$$\sum_{1 \leq i \leq n_h} x^h_{iu} - \sum_{1 \leq i \leq n_{h+2}} x^{h+2}_{iu_2} = 0$$

$$\ldots$$

$$\sum_{1 \leq i \leq n_h} x^h_{iu} - \sum_{1 \leq i \leq n_{h+s-1}} x^{h+s-1}_{iu_s} = 0$$

$$\sum_{1 \leq i \leq n_h} x^h_{iu} - \sum_{1 \leq i \leq n_{h+s-1}} x^{h+s}_{iv} = 0$$

where the expression,

$$\sum_{1 \leq i \leq n_h} x^h_{iu} \tag{10}$$

identifies the position of node u in layer h, and corresponds to the variable $x(u)$ introduced in the previous section to represent the x-coordinate of a node u.

We illustrate now how the inclusion of the alignment constraint creates a more readable final drawing. Fig. 5(a) shows the optimal solution of the original model by (Jünger & Mutzel, 2002) with 15 crossings, and Fig. 5(b) shows the optimal solution with the model considering the alignment constraints with 23 crossings. We observe that Fig. 5(b) provides a better drawing than Fig. 5(a), where the long arcs are difficult to trace. For example, arcs $(11, 28)$ and $(27, 46)$ have bends in Fig. 5(a), and are represented as straight lines in Fig. 5(b), resulting in a cleaner and less convoluted drawing. To quantify this in mathematical terms, we defined in Equation (2) the LA-value (Long arcs alignment value) of a drawing. Then, we see that the LA-value of Fig. 5 (a) is 12, while the LA-value in Fig. 5(b) is 0.

Another illustration of the relevance of including alignment constraints is given in Fig. 6. Fig. 6(a) shows the optimal solution of the original model (Jünger & Mutzel, 2002) and Fig. 6(b) shows the optimal solution of our improved model of the same input graph.

The drawing in Fig. 6(b) with $C = 8$ crossings and alignment $LA = 0$ is evidently more readable than the drawing in Fig. 6(a) with $C = 6$ crossings and alignment $LA = 8$. Arcs $(7, 20)$ and $(17, 26)$ which appear with bends in Fig. 6(a) are better depicted in terms of their readability in Fig. 6(b) where the bends are eliminated.

The use of exact MIP solution methods can only solve small instances with our mathematical model to optimality, requiring the introduction of a metaheuristic approach to target large instances. The MIP proposal introduced in Glover et al. (2021) succeeded in efficiently solving instances with up to 10 layers and around 10 nodes per layer, and to find feasible good solutions for larger instances the authors introduced a metaheuristic approach, MS-TS. It consists of a Tabu Search algorithm that has three steps. The first step reduces the number of crossings without considering the alignment, allowing the procedure to move to unfeasible solutions. Then, a short-term tabu search improves the alignment of the solution, handling the number of crossings as a secondary objective. Finally, a local search focuses on the reduction of crossings without deteriorating the alignment. This three-phase

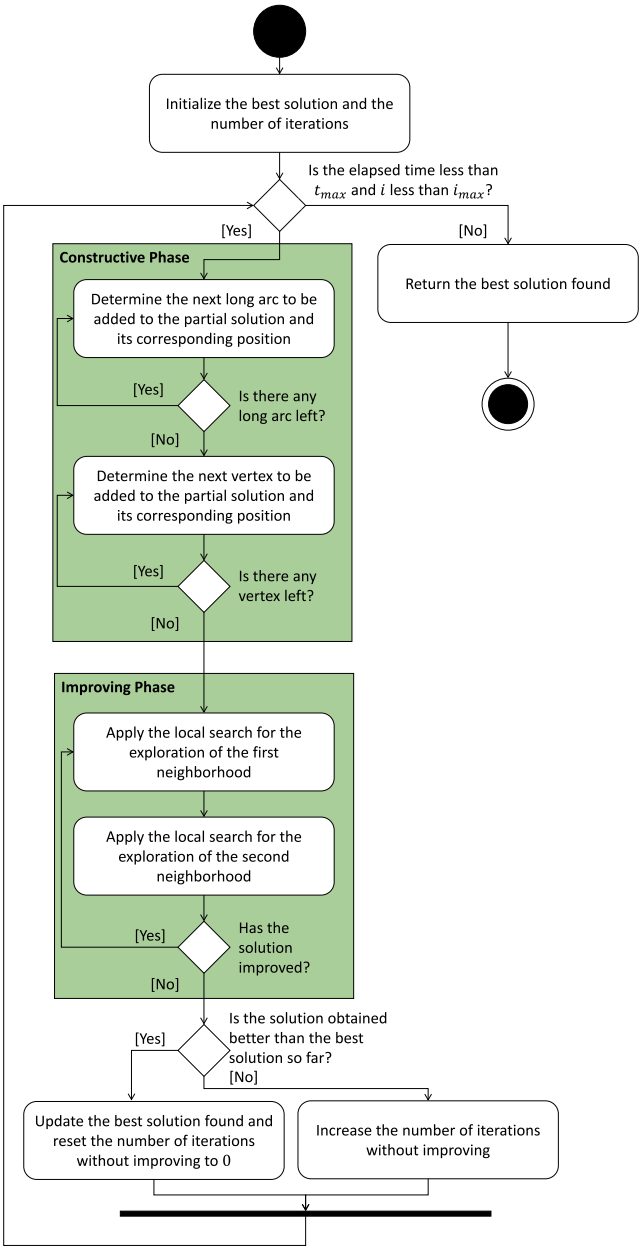


Fig. 7. Activity diagram of the MS-SOS procedure.

procedure, MS-TS, is embedded in a multi-start framework that implements a tabu long-term memory search. As the reader may notice, this procedure could eventually produce unfeasible solutions.

In this paper, we propose an advanced metaheuristic to provide high quality solutions in the short computational times required by drawing systems. From a theorical perspective, our main contribution is to merge the two steps, crossing and alignment, into a single one. From a practical point of view, we compare our method with two previous approaches. The standard Sugiyama's framework implemented in commercial graph drawing systems, and the previous heuristic proposal in the state of the art, specifically tailored for crossing minimization subject to long arc constraints. The experimental comparison reported in Section 5 shows the benefits of our proposal.

4. Multi-start memory-based approach

We propose the combination of three well-known methodologies for solving optimization problems: tabu search (TS), strategic oscillation (SOS), and multi-start procedures (MS). The TS is devoted to generating an initial solution for the method. The SOS is responsible of the improvement phase in each iteration. Finally, the MS approach lets the method to diversify the search using multiple iterations. Therefore, further than introducing a new methodology in the literature, we are using the know-how captured by previously well-known optimization methods and combine it into a powerful algorithm. The overall procedure, MS-SOS, is applied for as long as it improves the best solution found so far, subject to stopping upon reaching a maximum number of iterations.

Over the past decades, researchers have applied tabu search (TS) to many difficult optimization problems, as it is the case of graph drawing problems. Standard TS implementations typically incorporate non-improving moves to escape from local optimality which are implemented by memory structures called tabu lists to record attributes of previously encountered solutions and moves. In this paper, we consider the application of such strategies in the context of a constructive method, since the constraints of graph drawing problems make the

construction phase of the optimization procedure a key part of the method. Although proposed in constructive settings in early TS papers, these types of implementations have been largely overlooked in the scientific literature and one of the discoveries of the present research is that a properly designed constructive approach using strategic oscillation can be competitive with standard TS designs.

The general scheme of our MS-SOS method is outlined in the activity UML diagram (Booch, 2005) depicted in Fig. 7. Each rectangle describes a task or activity, while each diamond expresses a condition. The method starts by initializing the general parameters of the procedure, the best solution found and the number of iterations. Then, the method is divided in two main phases: construction phase and improving phase. If neither the maximum time nor the maximum number of solutions have been reached, the construction phase is executed, followed by the improvement phase. The construction phase produces a feasible solution by first adding the long arcs to the solution and then the rest of the vertices. This solution is provided to the improving phase, which explores several neighborhoods in search for a better solution. These two phases are deeply described in Section 4.1 and Section 4.2, respectively. When both phases finish, the procedure checks whether the solution obtained is better than the best one found so far. If so, the best solution found replaces the previous best solution. In addition, the number of iterations without an improvement is reset to 0. On the contrary, the number of iterations without improvement increases. When either the time limit or the maximum number of iterations without improvement are reached, the procedure ends and returns the best solution found.

To complement the previous figure, in Algorithm 1, we present the pseudocode of the aforementioned method. Specifically, we observe that the MS-SOS procedure receives three input parameters: a proper hierarchy graph (PH), the maximum running time (t_{max}), and the maximum number of consecutive iterations without improvement (i_{max}). The termination criterion of the algorithm (step 3) is determined by the parameters t_{max} and i_{max}. In each iteration, the two previously introduced phases are iteratively repeated: the constructive phase and the improving phase. The constructive phase (step 4) is described in Section 4.1, and it is based on two greedy criteria, compiled in Equations (15) and (17). The improving phase (step 5) is described in Section 4.2, and it is based on the exploration of two different neighborhoods, compiled in Equations (18) and (20). The solution obtained after both phases is compared with the best solution found in previous iterations (step 6). When the procedure stops, the best solution found among all iterations is returned as the output of the method (step 13).

Algorithm 1. General scheme of the multi-start procedure.

Procedure MS-SOS (PH, t_{max}, i_{max})
1. $x^{\star} \leftarrow \varnothing$
2. $i \leftarrow 0$
3. While *elapsedTime* $< t_{max}$ and $i < i_{max}$
4. $x \leftarrow$ *ConstructivePhase*(PH)
5. $x' \leftarrow$ *ImprovingPhase*(x)
6. If x' is better than $x^{\star}$
7. $x^{\star} \leftarrow x'$
8. $i \leftarrow 0$
9. Else
10. $i \leftarrow i + 1$
11. End If
12. End While
13. Return $x^{\star}$

4.1. Constructive phase

Constructive heuristics seek to generate a feasible solution in a short computational time utilizing problem-specific knowledge to exploit the characteristics of the problem at hand. These heuristics are often used to generate initial solutions for other procedures. Our constructive method consists of two main stages: (1) generating an initial partial solution by selecting long arcs (represented by a chain of dummy nodes) of the input PH graph, and (2) introducing the rest of the nodes (non-dummy nodes that are not incident with long-arcs) one by one into the partial solution. Both stages entail the following two tasks: i) Determine the next long arc (or node) to be added to the solution. ii) Determine the best position for the selected long arc (or node) in the partial solution.

The **first stage** of our constructive procedure starts with an empty initial solution and has the objective of allocating long arcs in the solution. We introduce four different criteria for the initial task of this first stage, whose outcomes are compared to select the most suitable one:

- Randomly selecting long arcs.
- Sorting the long arcs according to their cardinality (number of dummy nodes), so that those with a higher cardinality are added first to the solution.
- Sorting the long arcs in reverse order of cardinality so that those with a lower cardinality are added first to the solution.
- Employing a greedy function g_L that measures the urgency of adding a long arc to the solution. This criterion is inspired by previous works related to similar graph layout problems (Cavero et al., 2021; Cavero, Pardo, & Duarte, 2022) and it is formally stated next.

The greedy function g_L identifies the set of nodes incident with a long arc by defining $V_L = \{ u \in V : (u,v) \in A_L \vee (v,u) \in A_L \}$. When assigning nodes and long arcs to the partial solution under construction, we consider the set of nodes already placed in the solution S, and the subset of candidates $U_L \subset V_L$ of nodes incident with long arcs not yet placed in the solution. As a basis for selecting a node from the candidate set U_L to place it and its incident long arc in the partial solution, we study its adjacent nodes already placed in the solution.

Given a node $u \in U_L$, let $\delta(u)$ be the number of nodes adjacent to u already placed in the solution and let $\gamma_L(u)$ be the number of nodes adjacent to u not yet placed in the solution. Clearly, the degree of node u in U_L satisfies $(u) = \delta(u) + \gamma_L(u)$. The cardinalities are formally given as follows:

$$\delta(u) = |\{v \in S : (u,v) \in A \vee (v,u) \in A\}|, \tag{13}$$

$$\gamma_L(u) = |\{v \in U_L : (u,v) \in A \vee (v,u) \in A\}| \tag{14}$$

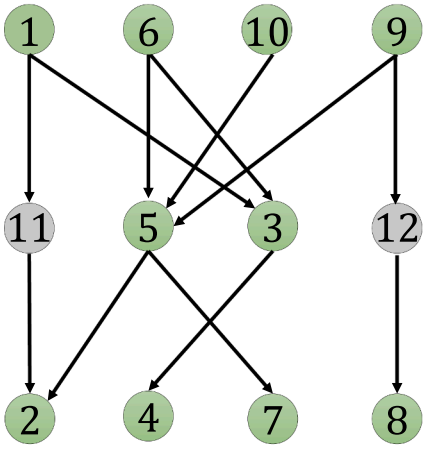
The greedy function $g_L(u,v)$ that computes the attractiveness of selecting the long arc (u,v) to be placed next in the partial solution is defined as:

$$g_L(u,v) = w_L \cdot (\delta(u) + \delta(v)) - (1 - w_L) \cdot (\gamma_L(u) + \gamma_L(v)) \tag{15}$$

where w_L is a parameter to be experimentally tuned and satisfies $0 \leq w_L \leq 1$. The w_L parameter balances the relevance of having a large number of adjacent nodes in the solution ($w_L > 0.5$) or a reduced number of adjacent nodes that remain to be added ($w_L < 0.5$). All unassigned long arcs from the input graph are then evaluated with equation (15) and the one with the largest g_L value is chosen to be added next.

Once a long arc has been selected, the objective of the second task of the first stage is to determine the position in which the selected long arc will be placed in the solution. Three criteria are proposed for this task:

- Select randomly from among the possible positions for a long arc.
- Try to place the long arc (u,v) in the median position among the nodes adjacent to u and v that are part of the partial solution (i.e., that belong to S). If it is not possible to place the long arc in the median position, place it in the closest position to the median one that is available.
- Augment the preceding placement criterion by applying tabu search strategies to penalize assigning a long arc to a position occupied by that long arc in previous constructions. The effect may be viewed as a diversification strategy that encourages the exploration of different solutions by avoiding the repetition of previously generated drawings.



C_1	1	6	9	10
1		1	1	1
6	2		1	1
9	3	0		1
10	1	0	0	

C_2	3	5	11	12
3		1	2	0
5	6		2	0
11	0	0		0
12	3	4	2	

C_3	2	4	7	8
2		0	0	0
4	2		1	0
7	1	0		0
8	2	1	1	

(a) (b)

Fig. 8. Solution x of the input graph depicted in Fig. 3.

The first stage ends with a partial solution in which all long arcs (real and dummy nodes) are part of the solution.

The **second stage** of the constructive method to add the remaining nodes to the solution proceeds as follows. Two different criteria are proposed for the first task of this stage (i.e., selecting a node to be added next to the solution) choosing the one that yields the best final configuration:

- Select a remaining node at random.
- Employ a greedy function g_S, that is a modification of function g_L described in Equation (15), to evaluate the remaining nodes. In this case, the function g_s considers nodes instead of long arcs.

To define the greedy function g_S, first we introduce the set of nodes V_S, not incident to any long arc, by defining $V_S = \{u \in V : u \notin V_L\}$. Let S denote the partial solution obtained with the application of the first stage of the constructive procedure (so, initially $S = V_L$) and let $U_S \subset V_S$ denote the subset of candidates of nodes not yet placed in the solution.

Given a node $u \in U_S$, $\delta(u)$ now identifies the number of nodes already placed in the solution adjacent to u (see Equation (13)). Associated with $\delta(u)$, we define $\gamma_S(u)$ to be the number of nodes not yet placed in the solution which are adjacent to u ($d(u) = \delta(u) + \gamma_S(u)$). Hence:

$$\gamma_S(u) = |\{v \notin S : (u,v) \in A \vee (v,u) \in A\}| \quad (16)$$

The greedy function $g_S(u)$ that computes the attractiveness of selecting a node $u \in V$ to be placed in the partial solution is then defined as:

$$g_S(u) = w_S \cdot \delta(u) - (1 - w_S) \cdot \gamma_S(u) \quad (17)$$

where w_S is a search parameter ($0 \leq w_S \leq 1$) analogous to the parameter w_L.

The first task of stage two ends when the node $u \in U_S$ with maximum g_S-value is selected. The second task of this stage then determines the position of node u in the partial solution. We propose three different criteria for this task:

- A random selection among the available positions.
- The median position of the nodes adjacent to node u. Again, if the desired position is occupied, the procedure looks for the closest available position.
- A position that minimizes the crossings of the selected node u by considering all the available positions in its layer and placing u in the position that produces the smallest number of crossings.

In summary, the first stage places the long arcs and their incident nodes in the solution (drawing), and in the second stage places the remaining nodes in the solution. The different proposals identified for executing these stages are discussed in our computational tests. The construction of solutions is crucial in this problem due to the long-arc constraints that limit the solution space customarily available to local search methods. As subsequently noted, we find empirically that the starting solution has an important influence in the final solution obtained.

4.2. *Improvement by strategic oscillation*

Strategic oscillation (SOS) is a search strategy originally proposed in the context of tabu search as a long-term technique (Glover, 1977, 2000). It tries to find solutions of interest in a *critical boundary* of the search space, and it is widely considered within the context of *adaptive memory* programming methods, since it has produced good results for many different problems such as maximally diverse problems (Gallego et al., 2013) or the linear ordering problem (Duarte et al., 2011), among others. We note that strategic oscillation goes beyond using constructive and destructive phases and involves oscillation over any type of element that is relevant to a search process. This broader notion has been the foundation for research in the past that has interesting links to the widely cited Variable Neighborhood Search (VNS) approach (Cavero, Pardo, & Duarte, 2022; Cavero, Pardo, Duarte, et al., 2022).

The proposal of strategic oscillation for multiple neighborhoods can be tracked back to (Glover et al., 1984). It consists of applying different types of moves – for instance, organized increasingly according to the size of the associated neighborhood – in the terminology VNS has made popular. Simple moves were employed at first until they no longer produced gains, and then steadily more advanced moves of several types were employed, in a process that repeatedly cycled through these progressive stages. We make use of this form of strategic oscillation that

Table 1
Structural properties of the instances generated.

Set name	$\lvert V \cup \dot{V} \rvert$	$\lvert \dot{A} \rvert$	$\lvert nl \rvert$	$\lvert L \rvert$	# instances
Small	[9, 100]	[8, 175]	[3, 10]	[1, 20]	200
Medium	[100, 250]	[125, 350]	[10, 20]	[10, 45]	100
Large	[250, 600]	[300, 1200]	[10, 55]	[35, 225]	50

Table 2
Configuration definition of the proposed algorithm for *irace*.

Phase	Stage	Parameter	Alternatives
Constructive	First: Long arcs	Task 1	Random Longest arc first Shortest arc first $g_L(w_L = 0.78)$
		Task 2	Random Median position **Median and tabu memory ($t_s = 0.21$)**
	Second: Nodes	Task 1	Random $g_S(w_S = 0.43)$
		Task 2	Random **Median** Crossing
	General configuration	Iterations	**1817**
Improving	Strategic Oscillation	Neighborhood exploration order	N_A / N_N N_N / N_A
		N_A exploration strategy	**First Improvement** Best Improvement
		N_N exploration strategy	**First Improvement** Best Improvement

Table 3
Constructive procedure without tabu memory (Cons.) and with tabu memory (Cons. + TS).

Algorithm	C	Dev. (%)	# Best	CPU T. (s)
Cons.	242.53	4.86	5	0.71
Cons. + TS	239.16	0.73	15	0.57

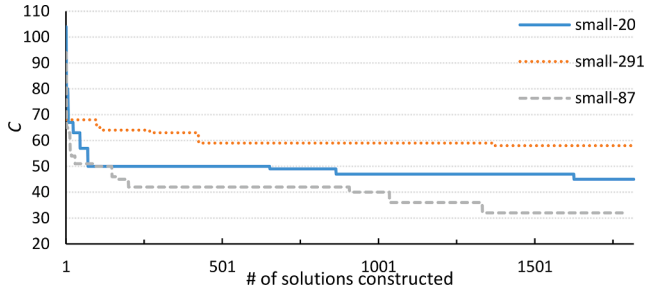


Fig. 9. Average objective value C evolution of the constructive procedure on three small instances.

Table 4
Contribution of advanced strategy to the local search.

Algorithm	C	Dev. (%)	# Best	CPU T. (s)
LS	288.16	0.00	19	1.229
ELS	288.16	0.00	19	0.005

involves the idea of oscillating among alternative choice rules and neighborhoods. These facts make the selection of strategic oscillation a suitable procedure to explore the search space of graph drawing problems.

Table 5
Performance differences between the procedure components and the full procedure.

Algorithm	C	Dev. (%)	# Best	CPU T. (s)
LS-N_N	249.53	53.28	1	60.06
LS-N_A	204.53	9.07	8	60.00
SOS	173.63	0.00	19	60.02

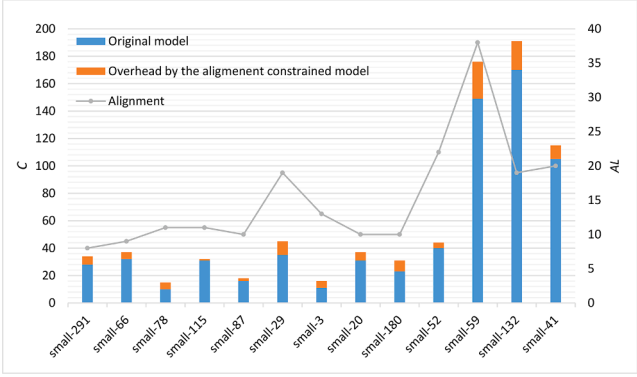


Fig. 10. Influence of the alignment constraint in the cross minimization objective function.

In this paper, we propose two different neighborhoods based on the classic swap move for the graph drawing problem: (1) move of a node to another position of the layer, and (2) move of all nodes of a long arc to another position of the layer. Beginning with (1), our SOS approach applies moves from the selected neighborhood iteratively, as in a standard local search method, and when no further improvement is possible, we invoke the alternative neighborhood to further apply local search with (2). Similarly, the search returns to (1) if at least a promising solution is found during (2). This process is repeated and thus introduces a simple oscillation between the two neighborhoods. This design, which is a special case of the SOS approach of (Glover et al., 1984), has more recently been popularized under the term Variable Neighborhood Descent (VND).

The first neighborhood we propose, denoted N_N, is defined as the set of solutions that can be obtained by applying the SwNodes move, which consists of swapping the position of two nodes in the same layer. Formally, we define the move SwNodes for any two nodes $u, v \in V_S$, such that originally $x(u) = w$, $x(v) = z$, as the one which results in $x(u) = z$, and $x(v) = w$ after the move. Recall that a node in V_S does not belong to any chain of dummy nodes that replaces a long arc (including the beginning and ending nodes of the chain). Considering the SwNodes move, we now define the associated neighborhood N_N as follows:

$$N_N = \{\text{SwNodes}(u, v) \forall u, v \in V_S : u \neq v\}. \tag{18}$$

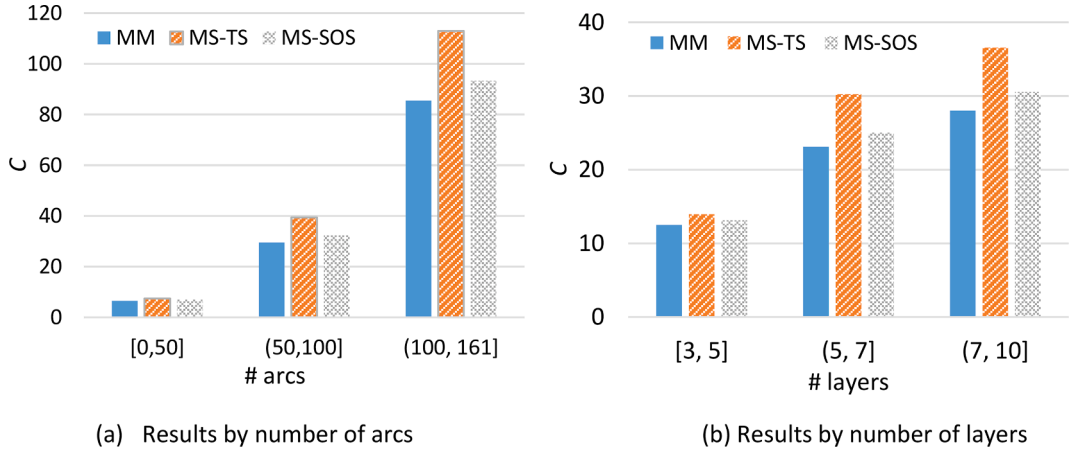
The second neighborhood, denoted N_A, is defined as the set of solutions that can be obtained by applying the SwArc move, which consists of moving a long arc. In this instance, long arcs are replaced by a chain of dummy nodes and, therefore, moving a long arc is equivalent to moving all nodes which compose that chain. More precisely, the move consists of swapping each node in the chain with the node placed in the same layer at a predefined x-coordinate. For the sake of simplicity we denote this x-coordinate as position p. Formally, given a long arc (u, v) (together with its associated chain of dummy nodes $AC(u,v) = \{(u, u_1), (u_1, u_2), \cdots, (u_s, v)\}$) and a position p, we define the move SwArc as follows:

$$\text{SwArc}((u,v), p) = \{\text{SwNodes}(u, w_1), \text{SwNodes}(u_1, w_2), \cdots, \text{SwNodes}(v, w_{s+1}) : x(w_1), x(w_2), \cdots, x(w_{s+1}) = p \wedge w_1, w_2, \cdots, w_{s+1} \in V_S\}. \tag{19}$$

The neighborhood N_A associated with the SwArc move is defined as follows:

Table 6
Comparison of MS-SOS with the best methods on small instances.

	MM		MS-TS				MS-SOS			
Instances	*C*	# Opt.	*C*	Dev. (%)	# Opt.	CPU T. (s)	*C*	Dev. (%)	# Opt.	CPU T. (s)
[0,25]	2.23	35	2.31	6.29	33	0.10	2.40	3.17	31	1.47
(25, 50]	14.92	90	17.81	23.54	26	2.02	16.28	9.38	45	5.38
(50, 75]	36.51	45	49.82	42.45	0	10.08	39.62	9.57	9	14.46
(75, 100]	85.57	30	114.43	39.75	0	30.50	93.63	10.45	0	34.29
Total	28.16	200	36.80	27.71	59	7.77	30.71	8.50	85	11.07



(a) Results by number of arcs (b) Results by number of layers

Fig. 11. Comparison of MS-SOS with the best methods on small instances.

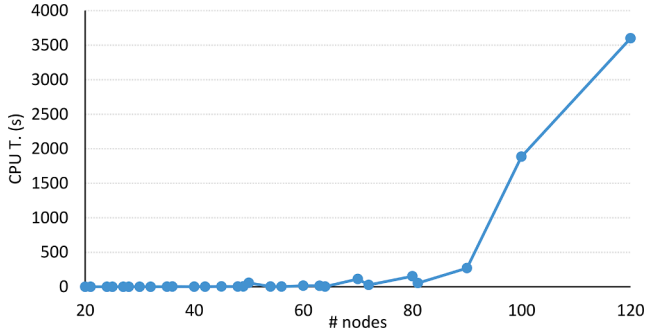


Fig. 12. CPU time consumed by Gurobi w.r.t size instance.

Table 7
Comparison of MS-SOS with the best methods on medium instances.

Algorithm	*C*	Dev. (%)	# Best	CPU T. (s)
MS-TS	392.18	65.17	0	498.29
MS-SOS	245.43	0.00	100	29.50

Table 8
Comparison of MS-SOS with the best methods on large instances.

Algorithm	*C*	Dev. (%)	# Best	CPU T. (s)
MS-TS	2868.50	40.12	0	10694.74
MS-SOS	2392.42	0.00	50	63.84

$$N_A = \{\text{SwArc}((u, v), p) \forall (u, v) \in A_L\}. \quad (20)$$

The exploration of either N_N or N_A is performed with a local search procedure which we implement with two well-known exploration strategies: first improvement and best improvement. In the present case, the first improvement strategy consists of performing the first move found of a node or arc that improves the current solution quality, while the best improvement strategy selects the best move of each node or arc, among all possible moves. Section 5 reports the outcomes of applying these strategies within our proposed search procedure and examines the order in which the considered neighborhoods (N_A and N_N) should be explored.

Additionally, we propose an advanced strategy to update the value of the objective function of a solution obtained after applying $\text{SwNodes}(u, v)$ which considerably reduces the time needed to explore the neighborhoods N_A and N_N.

We start by defining a square matrix C_l for each layer l, to store the number of arc crossings produced by any two nodes u and v of the layer, depending on the relative position between them within the layer. Therefore, the number of rows and columns of the matrix C_l is equal to the number of nodes in the layer l. Then, given any two nodes u and v belonging to the same layer l, let c_{uv} be the number of crossings between arcs incident to u and arcs incident to v, under the condition where u precedes v in its layer (i.e., $x(u) < x(v)$). (An arc incident to a node refers to any incoming or outgoing arc.) We then store the value of c_{uv} in the matrix in the position defined by the row-index u and column-index v.

Fig. 8(a) provides an example of a solution x of the *PH* depicted in Fig. 3. Since *PH* has three layers, it is necessary to calculate three matrices: C_1, C_2 and C_3, which are represented in Fig. 8(b). Focusing on layer 1, which is composed of the nodes 1, 6, 9, and 10, we see that C_1 contains four rows and four columns. In greater detail, observing the number of crossings generated between nodes 1 and 6, we obtain $c_{1,6} = 1$ for $x(1) < x(6)$, as in the diagram of 8(a), and would obtain $c_{6,1} = 2$ if $x(6) < x(1)$. Similarly, nodes 9 and 10 produce 0 crossings for $x(10) < x(9)$ (to yield $c_{10,9} = 0$) as in the diagram of 8(a), and would produce 1 crossing if $x(9) < x(10)$ (to yield $c_{9,10} = 1$), and so on. In Fig. 8(b), we highlight with a green background the arc crossing values corresponding to the current solution depicted in Fig. 8(a) and use a white background to show the arc crossing values if the order within the arc pairs were to be reversed.

Using the information stored in this matrix, the following conclusion can be drawn: if c_{uv} is greater than c_{vu}, and $x(u) < x(v)$, then swapping the position of u and v will result in a better-quality solution.

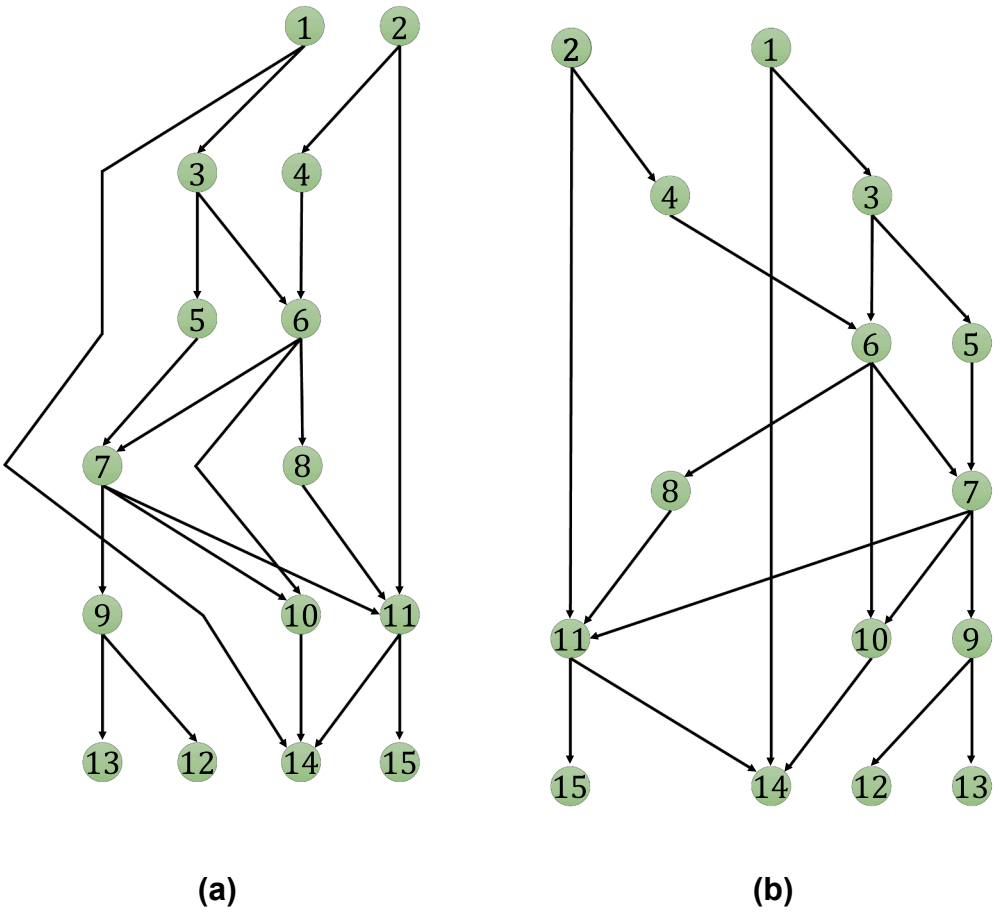


Fig. 13. Optimal drawings for crossing minimization, subject to the alignment constraints.

Given a solution x and the solution x' (obtained as a result of a SwNodes move between node u and node v) the difference in solution quality between the value of $C(x)$ and the value of $C(x')$ is denoted as ΔSwNodes. Therefore, the value of the objective function of x' can also be expressed as follows:

$$C(x') = C(x) + \Delta\text{SwNodes}(u, v) \tag{21}$$

Furthermore, $\Delta\text{SwNodes}(u, v)$ can be easily updated after the operation $\text{SwNodes}(u, v)$ as follows:

$$\Delta\text{SwNodes}(u, v) = c_{vu} - c_{uv} + \sum c_{wu} - c_{wu} + c_{vw} - c_{wv},\ \forall w \in V : x(w) \in (x(u),\ x(v)) \tag{22}$$

Therefore, in sum, $\Delta SwNodes(u, v) < 0$ indicates that x' is better than x and it is an improving move since the number of crossings has been reduced. A key aspect in this computation is that if u and v are consecutive nodes in a layer ($|x(u) - x(v)| = 1$), the change in the total number of crossings only depends on $c_{vu} - c_{uv}$.

To extend the illustration, consider again the solution x and the matrix C_2 in Fig. 8. Since $c_{53} = 6$ is greater than $c_{35} = 1$, and $x(5) < x(3)$, we are interested in swapping nodes 5 and 3. To calculate the objective function of the resultant solution, x' (depicted in Fig. 3), after the move $\text{SwNodes}(5, 3)$ we apply Equations (21) and (22) to obtain:

$$\Delta\text{SwNodes}(5, 3) = c_{35} - c_{53} = 1 - 6 = -5.$$

$$C(x') = C(x) + \Delta\text{SwNodes}(5, 3) = 6 - 5 = 1$$

Finally, we note that factorization also reduces the computation time needed to explore the neighborhood N_A since a move SwArc is defined as a set of consecutive SwNodes.

5. Computational test

In this section, we compile the computational tests carried out in this research. Section 5.1 presents the instances and Section 5.2 describes the tests designed to configure our proposed MS-SOS, together with illustrating the contribution of the components and strategies of the final procedure. Finally, in Section 5.3 we compare our MS-SOS method with the best methods previously identified for the graph drawing problem.

All algorithms were coded in Java 17 and the tests were run on a 16-core vCPU AMD EPYC7282 with a total of 16 GB RAM and Ubuntu 20.04.2 64-bit LTS operating system. We have made the source code used in our implementations available at https://github.com/scaverod/SOS-TS-GraphDrawing.

5.1. Instances

We have considered three sets of drawing problem instances, where an instance is identified by its proper hierarchy graph, $PH = (V \cup V', A', nl, L)$. Table 1 depicts the ranges of the structural properties for each of the generated sets of instances:

The sets of instances used, and the instance generator code are available at https://github.com/scaverod/SOS-TS-GraphDrawing.

Notice that to guarantee the robustness of our algorithms we have used 350 different instances in our final tests. Additionally, to avoid overtraining the algorithms, the configuration of their parameters is performed over reduced data set of 19 out of 350 instances, named preliminary set.

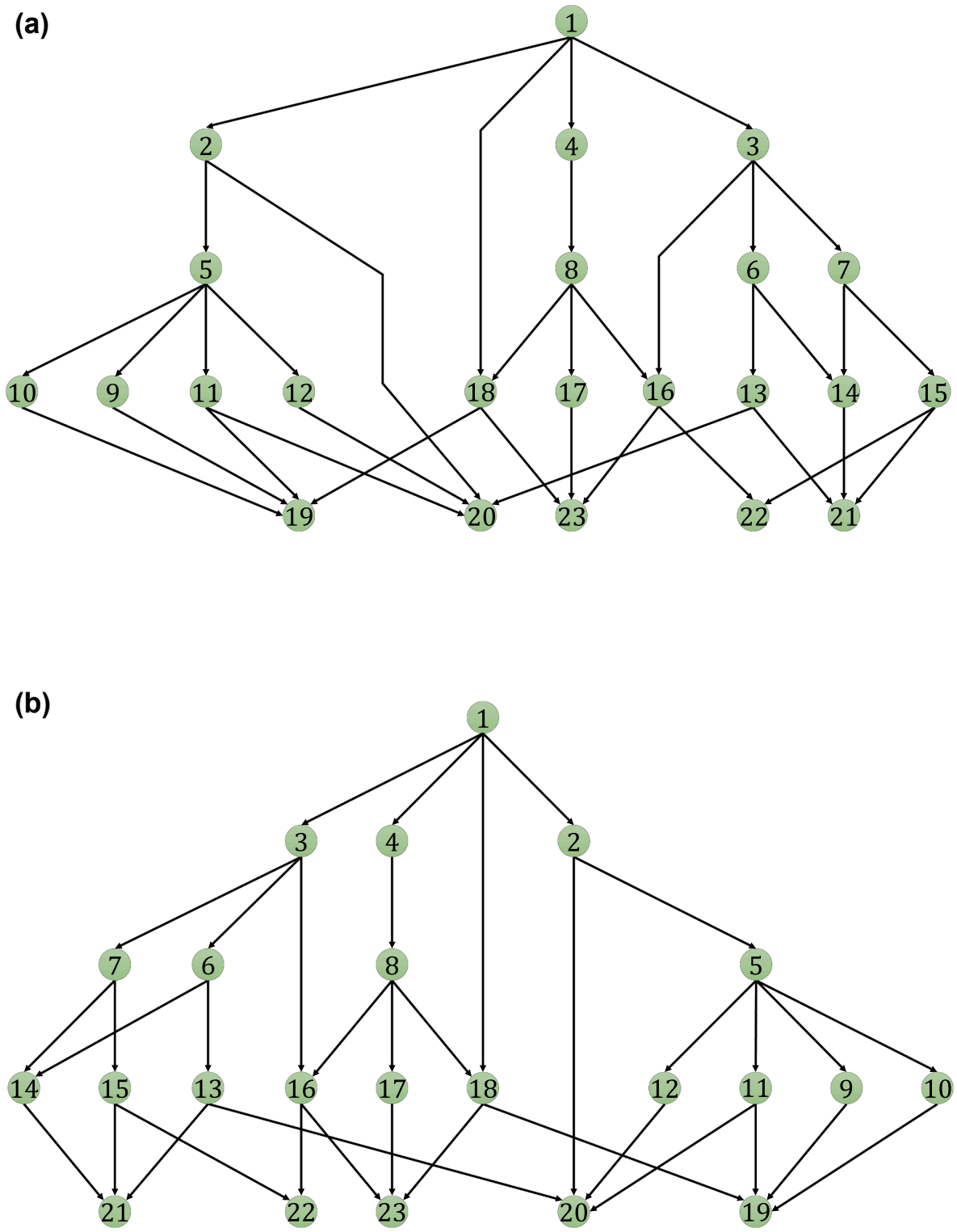


Fig. 14. Optimal drawings for crossing minimization, subject to the alignment constraints.

5.2. *Preliminary test*

To test the different configurations of the components of the MS-SOS procedure to disclose the best variant of our method, we ran each different variant for 60 s, which is a relatively short running time. Notice, that one of the possible real applications of the algorithm proposed in this paper is to be included in a graph drawings software or library. In this sense, the user of the software might expect a quick output and, therefore, a relatively short running time is mandatory.

Our preliminary parameter tuning was done with the *irace* software introduced in López-Ibáñez et al. (2016). Table 2 summarizes the different strategies proposed for each phase of the algorithm and the parameters provided to *irace*. The configuration is divided into two phases: constructive and improving. As previously described, the constructive phase is further split into two stages and each stage is split into two tasks. Different criteria are provided for each task and

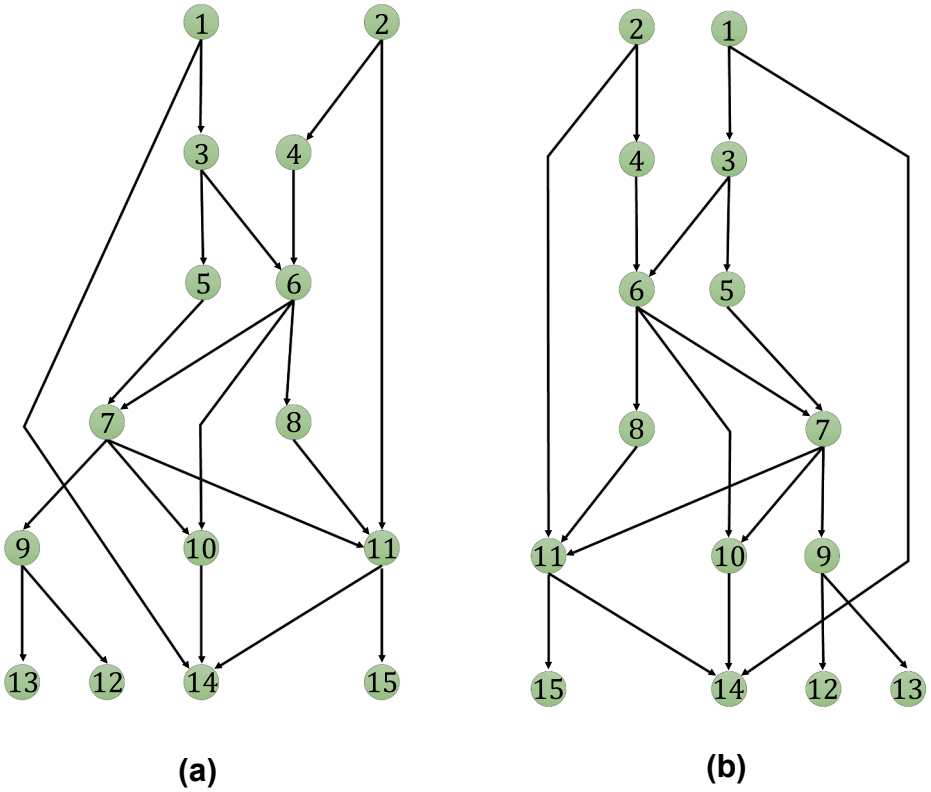


Fig. 15. Drawings obtained with the Dot and yEd software systems.

the number of constructions must be defined.

The strategic oscillation-based improvement phase requires two main decisions: the order in which the neighborhoods will be explored and the exploration strategy for each neighborhood. The best configurations for our proposal, obtained as output of *irace*, are identified in bold in Table 2.

The following tests determine the relevance of the strategies introduced in the final configuration of the method. For each test, we report the average of the value of the objective function (C), and the deviation to the best in the test (Dev. (%)). Also, we report the total execution time in seconds (CPU T. (s)) and the number of best solutions found (# Best).

Our first preliminary test explores the increase in solution quality when the constructive procedure incorporates a tabu memory. For this, we compared the best solution found by the constructive procedure of our final proposal (see the constructive phase in Table 3) without tabu memory (Cons.) and with tabu memory (Cons. + TS). As it can be observed, Cons. + TS shows a notable improvement in the quality of the solutions generated, obtaining 15 best solutions out of 19 instances and an average deviation closer to 0.

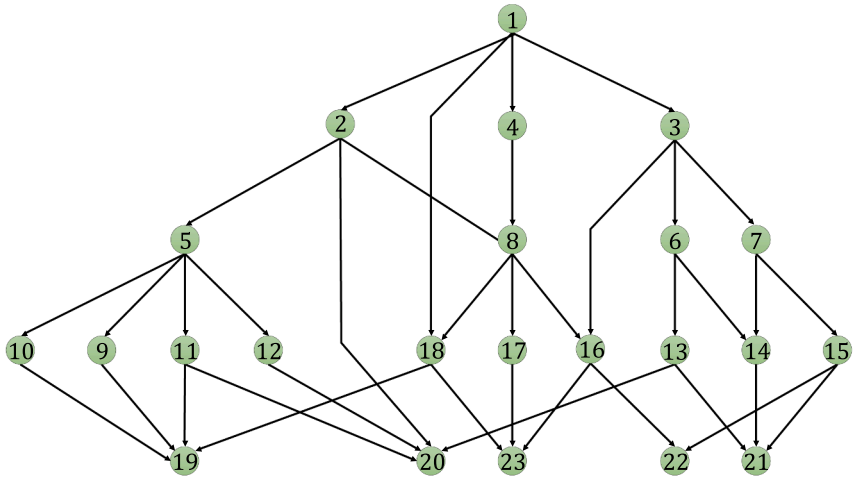
The next preliminary test addresses the evolution of the quality of the best solution found in each iteration of the construction procedure, reporting the best solution found for three preliminary small instances. As shown in Fig. 9, the performance of the procedure dramatically improves in the first 250 executions for each of the instances. A moderate improvement occurs for up to 500 executions and marginal improvements continue for up to 1500 executions. Quite likely there is still room for improvement if the number of constructions increases beyond 1500 executions. However, *irace* determined that 1817 is an appropriate number of iterations as a trade-off between the time spent on the construction phase and the time spent on the improvement phase.

The last preliminary test involving the constructive phase analyzes the number of infeasible solutions generated due to the influence of the alignment constraints. We find that for some large instances of the preliminary set a total of 80 % of the solutions generated are infeasible. For the medium and small instances, the number of infeasible solutions ranges up to 35 % of the solutions generated. This underlines the importance of running the construction procedure multiple times in the search for good starting solutions for the improvement procedure.

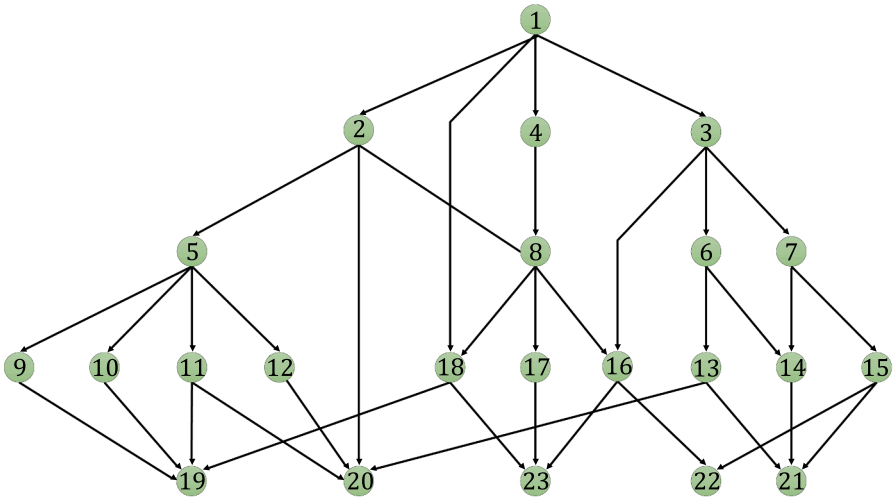
Proceeding beyond the constructive phase, we next study the ability of the advanced strategy proposed in Section 4.2 to efficiently evaluate neighboring solutions. For this, we compared two local search procedures that start from the same random initial solution and explore neighborhood N_N following a best improvement strategy. Then, we analyze the solutions obtained by a simple local search procedure (LS) with a local search that implements the efficient move evaluation (ELS). Table 4 reveals that both variants reach the same solutions quality, but ELS consumes three order of magnitude computational time less than LS to explore the same number of solutions.

The next preliminary test is devoted to identifying the contribution of each of the strategies proposed for the improving phase. Based on the output configuration of *irace* (see Table 2), we compare: (1) a local search that explores N_N using a first improving pattern (LS-N_N), (2) a local search that explores N_A using a first improving pattern (LS-N_A), and (3) the strategic oscillation procedure that combines LS-N_A and LS-N_N (SOS). All tests start from the same random solution. Upon reaching a local optimum, a new random solution is generated which is the same for each until a maximum time of 60 s is reached. The results obtained are reported in Table 5. As expected, SOS is the best method in terms of the average number of crossings, deviation and # Best solutions found. The second-best procedure is LS-N_A and the last is LS-N_N.

Since we are proposing a multistart procedure, in order to evaluate the convergence performance of our algorithm, we conducted an additional test where the best configuration of our algorithm was executed until it performed 100 iterations without finding an improvement. For each iteration, we reported the best solution found and compared them with the rest of the iterations. Our analysis revealed that, on average, the algorithm was able to find the best solution in the iteration 42. Additionally, the latest iteration where a best solution was found was the



(a)



(b)

Fig. 16. Drawings obtained with the Dot and yEd software systems.

iteration 96. The average time needed to find the best solution was 63.23 s. Notably, only 6 out of 19 instances needed more than 60 s. Therefore, a combined stopping criterion which involves a maximum time per instances and a maximum number of iterations without improvement might be desirable. The previous results indicate a fast convergence performance of the algorithm.

The final preliminary test is devoted to identifying the influence of the alignment constraint in the cross minimization objective function. To that aim, we have analyzed the number of crossings produced in the optimal solutions of several instances. Particularly, in Fig. 10 we depict the number of crossings for those instances in the preliminary data set that can be solved to optimality with the mathematical model introduced in Section 3. In blue, we represent the number of crossings obtained by the original model and, in orange, we represent the overhead of crossings produced by the alignment constrained model.

As expected, there is an increase in the number of crossings when the alignment constraint is introduced. Note however that this increase is very moderate, which supports the main point of this paper; it is worth in terms of the final readability to marginally increase the number of crossings in order to align the long arcs.

5.3. Comparison with previous methods

In this section, we compare our MS-SOS procedure with the two best previous state-of-the-art methods in the literature. Particularly, the Mathematical Model (MM) and the Multi-Start Tabu Search (MS-TS), proposed in Glover et al. (2021). For the MS-TS comparison, we have implemented the original source code and configuration provided by the authors in Glover et al. (2021), using the same execution environment as the one used for our code. Although both methods are based on the Tabu Search framework, they are quite different. In particular, the previous method starts from a random solution, while our approach is built from a new greedy constructive procedure based on long term memory structures (see Section 4.1). Additionally, the improvement phase in the previous method was composed by different local search procedures, which we found inefficient. Instead, we introduce a single local search

that combines several neighborhoods by implementing a Strategic Oscillation Search (see Section 4.2). Finally, we propose an advanced search strategy that avoids unfeasible solutions. Therefore, the new tabu search method proposed here is not a modification or enhancement of the previous one, but it constitutes a new design built from scratch. Their comparison, both theoretical and empirical, reveals valuable lessons for researchers on metaheuristic methods.

The mathematical model was implemented in Java 17 and Gurobi and was also run in the same execution environment as the two previous procedures, setting a maximum execution time of 1 h. On the other hand, the stopping criterion for the MS-SOS is either 60 s or 100 iterations without improvement.

Table 6 presents the results of the three procedures for the set of small instances. We report the same indicators previously introduced (Dev. (%), CPU T. (s), and # Opt.). Instances are grouped according to the number of nodes and the bottom row of the table (labelled as "Total") reports the average for all instances.

As seen in Table 6, the MM approach (using the mathematical model solved by the exact Gurobi method) disposes of the small instances relatively easily, finding optimal solutions for these 200 instances within the maximum time limit. (As will be seen, the picture changes radically when we get beyond the small instances.) The two heuristic procedures do not fare as well as the exact algorithm on these instances. MS-SOS has the smallest deviation with respect to the optimum values (8.50 %), finding the optimum for half of the instances under consideration. The difference in computation between the two heuristic procedures is very small. Overall, MS-SOS emerges as an efficient method to solve small instances in a reasonable amount of computational time but does not match the exact method for overall solution quality. We refer the reader to https://github.com/scaverod/SOS-TS-GraphDrawing, to obtain the individual results per instance.

The results obtained for the small instances are complemented by the charts illustrated in Fig. 11. Fig. 11(a) shows the average crossings for the three algorithms, MM (blue), MS-TS (orange) and MS-SOS (gray), grouping the instances according to the number of their arcs. Fig. 11(b) shows the average crossings for the same algorithms grouping the instances by the number of layers. As can be seen in both figures, MM is the best algorithm for dealing with small instances, followed by MS-SOS and finally MS-TS.

Fig. 12, however, discloses information not available in the preceding comparison by showing the computation time needed by Gurobi for these small instances, identifying the average time needed to solve an instance on the y-axis and the number of nodes of the instances on the x-axis. As can be seen, the time abruptly grows exponentially when the graphs exceed 80 nodes, which makes the use of MM for the solution of graphs with more than 100 nodes totally impractical.

In Table 7 and Table 8 we present the results obtained for the sets of medium and large instances respectively. Now the exact solution procedure Gurobi using the MM model deteriorates to the point of dropping out of consideration as a useful alternative. Gurobi was unable to find a single feasible solution within the allotted solution time and has been removed from the comparison, leaving us to compare the MS-TS and MS-SOS procedures. Since optimal solutions are unknown for the instances considered in these tests, in this case we report the number of best values obtained (#Best) and the deviation to this value.

Table 7 shows that the MS-SOS method finds better solutions in an order of magnitude less time than MS-TS, reaching the best solution for all the instances under consideration, while MS-TS is unable to find any of these best solutions.

Similar conclusions can be drawn from the analysis in Table 8 with an even more significant difference in efficiency for MS-SOS, which finds the best results for all instances in three orders of magnitude less time than MS-TS.

We complement the numerical comparisons introduced in this section with a graphical comparison of the drawings obtained by the original mathematical model from Jünger and Mutzel's (2002) and the MS-SOS heuristic method. This comparison is presented in the Appendix A. In addition, in Appendix B, we compare the optimal solution of our proposed model with the drawing obtained with two well-known drawing software: Dot (Gansner et al., 2015) and yEd (yWorks, 2023), that constitute a reference in the field.

6. Conclusions and future work

In this paper, we introduce a modification of Sugiyama's graph drawing framework by proposing the addition of long arcs constraints to increase the readability of the drawings. In particular, our proposed modification constrains the optimization to solutions in which long arcs have no bends. We introduce both a mathematical model to obtain the optimal solution to this problem for small instances, and a metaheuristic to obtain high-quality drawings of larger graphs in a reasonable time.

Our algorithm is based on the tabu search methodology joined with strategic oscillation (SOS) to perform a fast and effective exploration of the search space. The method implements a multi-start framework that integrates a constructive tabu search method with an SOS improvement procedure that oscillates between the exploration of two neighborhoods. The process is augmented by an efficient move value calculation to reduce the time needed to explore these two neighborhoods.

We compare the outcomes from our new approach with a classic design of the tabu search method recently published for this problem. The computational comparison clearly establishes the superiority of the new proposal, calling attention to possibilities of the tabu search methodology that have been largely overlooked.

In the interest of exploring options that invite closer examination, we note that the alternation between moves and evaluations helps SOS approaches to succeed in the search process. This suggests the merit of those forms of strategic oscillation that focus on the choice of moves and evaluation criteria selected from a set of options, according to rules which let the method to transition between solutions. In such applications, there is a considerable opportunity for doing multiple neighborhood studies that go beyond current proposals in the VNS literature, and that may lead to developing useful advances in solving various kinds of problems where multiple neighborhoods are naturally available.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

I have shared a link to the data and source code used in the paper

Acknowledgement

This research has been partially supported by the Ministerio de Ciencia e Innovación of Spain (Grant Refs. PGC2018-095322-B-C21/C22, PID2021-125709OA-C22/OB-C21, FPU19/04098 and EST22/00444) funded by MCIN/AEI /10.13039/501100011033/FEDER, UE. It has been also supported by the Comunidad de Madrid and the European Regional Development Fund (Grant Ref. P2018/TCS-4566) and by the Generalitat Valenciana (CIAICO/2021/224). It has also been supported by Programa Propio de la Universidad Rey Juan Carlos (Grant Ref. M2988) and Cátedra de Innovación y Digitalización Empresarial (Ref. ID MCA06).

Appendix A: Drawing examples

We complement the numerical comparisons of Section 5.3 by making a graphical comparison of the drawings obtained by the two main approaches compared in this paper, on some well-known examples in the graph drawing field. Particularly, we compare the result obtained by the

integer linear formulation proposed by Jünger and Mutzel's (2002) and the MS-SOS heuristic introduced in this paper.

The examples shown next are extracted from figures 13.2 and 13.16 respectively, in chapter "*Hierarchical Drawing Algorithms*" in the "*Handbook of graph drawing and visualization*" by Healy and Nikolov (2014), which are also used in Bachmaier et al. (2012), among others.

The solutions obtained for the first example are depicted in Fig. 13. Specifically, Fig. 13(a) shows the Sugiyama framework solution obtained by the MIP with 2 crossings. Fig. 13(b) shows the best solution found by the MS-SOS procedure considering the alignment constraints, resulting in 4 crossings but producing a clearer drawing.

In Fig. 14, we illustrate a second example of the drawings obtained by the two methods compared. Specifically, the MIP solution is depicted in Fig. 14(a), while the drawing obtained by the MS-SOS procedure is depicted in Fig. 14(b). Both drawings have 9 crossings, so from the perspective of crossing minimization the drawings are equally good. However, in Fig. 14(b) the long arcs are aligned, making the drawing more legible.

Appendix B: Comparison with commercial software

This appendix includes the drawings of the two examples introduced in Appendix A obtained with two well-known commercial software, Dot (Gansner et al., 2015) and yEd (yWorks, 2023). We compare the optimal solution of our proposed model with the drawing obtained with these drawing software that constitute a reference in the field. Note that they are based on the drawing model by Sugiyama et al. and therefore this comparison also illustrates the benefits of our model with respect to that one.

The first example is depicted in Fig. 15. Specifically, Fig. 15(a) shows the drawing obtained by Dot, while Fig. 15(b) shows the drawing generated by yEd. Note that in both cases the drawings present arc bends, and therefore as solutions of our model can be considered unfeasible since the alignment constraint is not satisfied.

Regarding crossing minimization, both software products do a good job since the drawings obtained have 2 and 3 crossings respectively. The drawing obtained with our method, MS-SOS, depicted in Fig. 13(b), presents 4 crossings and no bends. In our view, the graphical inspection of these figures supports the assessment that a marginal increase in the number of crossing due to the long-arcs constraint is worth it in terms of the final readability of the drawing.

Fig. 16 represents the second example in the Appendix. Specifically, the solution generated by Dot is depicted in Fig. 16(a), while the solution generated by yEd is presented in Fig. 16(b). As it can be observed, the two solutions provided by the drawing software present arc bends (i.e., are not feasible for our model), and a similar number of crossings (11 and 10 respectively) than our MS-SOS (10 crossings as shown in Fig. 14 (b), which is able to represent the graph with no bends. This example clearly shows the benefits of our model and solving method that are able to improve the readability of two well established graph drawing systems.

References

Bachmaier, C., et al. (2012). Drawing Recurrent Hierarchies. *Journal of Graph Algorithms and Applications, 16*(2), 151–198.

Battista, G. D., Eades, P., Tamassia, R., & Tollis, I. G. (1998). *Graph drawing: algorithms for the visualization of graphs* (1st ed.). Prentice Hall PTR.

Booch, G. (2005). *The unified modeling language user guide*. Pearson Education India.

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. https://doi.org/10.1109/TSMC.1980.4308390

Cavero, S., Pardo, E. G., & Duarte, A. (2022). A general variable neighborhood search for the cyclic antibandwidth problem. *Computational Optimization and Applications, 81*(2), 657–687. https://doi.org/10.1007/s10589-021-00334-y

Cavero, S., Pardo, E. G., Duarte, A., & Rodriguez-Tello, E. (2022). A variable neighborhood search approach for cyclic bandwidth sum problem. *Knowledge-Based Systems, 246*, Article 108680. https://doi.org/10.1016/j.knosys.2022.108680

Cavero, S., Pardo, E. G., Laguna, M., & Duarte, A. (2021). Multistart search for the Cyclic Cutwidth Minimization Problem. *Computers & Operations Research, 126*, Article 105116. https://doi.org/10.1016/j.cor.2020.105116

Chen, X., Song, X., Peng, G., Feng, S., & Nie, L. (2021). Adversarial-enhanced hybrid graph network for user identity linkage. In *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval* (pp. 1084–1093). https://doi.org/10.1145/3404835.3462946.

Chimani, M., Gutwenger, C., Junger, M., Klau, G. W., Klein, K., & Mutzel, P. (2013). *The Open Graph Drawing Framework (OGDF)*. 28.

Duarte, A., Laguna, M., & Martí, R. (2011). Tabu search for the linear ordering problem with cumulative costs. *Computational Optimization and Applications, 48*, 697–715. https://doi.org/10.1007/s10589-009-9270-5

Gallego, M., Laguna, M., Martí, R., & Duarte, A. (2013). Tabu search with strategic oscillation for the maximally diverse grouping problem. *Journal of the Operational Research Society, 64*(5), 724–734. https://doi.org/10.1057/jors.2012.66

Gansner, E. R., Koutsofios, E., & North, S. (2015). *Drawing graphs with dot*. https://www.graphviz.org/pdf/dotguide.pdf.

Gansner, E. R., & North, S. C. (2000). An open graph visualization system and its applications to software engineering. *Software: Practice and Experience, 30*(11), 1203-1233. https://doi.org/10.1002/1097-024X(200009)30:11<1203::AID-SPE338>3.0.CO;2-N.

Glover, F. (1977). Heuristics for integer programming using surrogate constraints. *Decision Sciences, 8*(1), 156–166. https://doi.org/10.1111/j.1540-5915.1977.tb01074.x

Glover, F. (2000). Multi-start and strategic oscillation methods—principles to exploit adaptive memory. In En M. Laguna, & J. L. G. Velarde (Eds.), *Computing tools for modeling, optimization and simulation: interfaces in computer science and operations research* (pp. 1–23). Springer US. https://doi.org/10.1007/978-1-4615-4567-5_1.

Glover, F., Campos, V., & Martí, R. (2021). Tabu search tutorial. A graph drawing application. *TOP, 29*(2), 319–350. https://doi.org/10.1007/s11750-021-00605-1

Glover, F., & Laguna, M. (1998). Tabu Search. In En D.-Z. Du, & P. M. Pardalos (Eds.), *Handbook of Combinatorial Optimization: Volume1–3* (pp. 2093–2229). Springer US. https://doi.org/10.1007/978-1-4613-0303-9_33.

Glover, F., McMillan, C., & Glover, R. (1984). A heuristic programming approach to the employee scheduling problem and some thoughts on "managerial robots". *Journal of Operations Management, 4*(2), 113–128. https://doi.org/10.1016/0272-6963(84)90027-5

Healy, P., & Nikolov, N. (2014). Hierarchical drawing algorithms. In Roberto Tamassia (Ed.), *Handbook of Graph Drawing and Visualization* (pp. 409–453).

Jünger, M., & Mutzel, P. (2002). 2-Layer Straightline Crossing Minimization: Performance of Exact and Heuristic Algorithms. In *En Graph Algorithms and Applications I* (pp. 3–27). World Scientific. https://doi.org/10.1142/9789812777638_0001.

Kaufmann, M., & Wagner, D. (2001). *Drawing Graphs: Methods and Models*. LNCS Tutorial: Springer.

Laguna, M., & Marti, R. (1999). GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing, 11*(1), 44–52. https://doi.org/10.1287/ijoc.11.1.44

Laguna, M., Martí, R., & Valls, V. (1997). Arc crossing minimization in hierarchical digraphs with tabu search. *Computers & Operations Research, 24*(12), 1175–1186. https://doi.org/10.1016/S0305-0548(96)00083-4

López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., & Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives, 3*, 43–58. https://doi.org/10.1016/j.orp.2016.09.002

Martí, R. (1998). A tabu search algorithm for the bipartite drawing problem. *European Journal of Operational Research, 106*(2), 558–569. https://doi.org/10.1016/S0377-2217(97)00291-9

Martí, R. (2001). Arc crossing minimization in graphs with GRASP. *IIE Transactions, 33*(10), 913–919. https://doi.org/10.1080/07408170108936883

Martí, R., Martínez-Gavara, A., Sánchez-Oro, J., & Duarte, A. (2018). Tabu search for the dynamic Bipartite Drawing Problem. *Computers & Operations Research, 91*, 1–12. https://doi.org/10.1016/j.cor.2017.10.011

Martí, R., & Reinelt, G. (2022). *Exact and Heuristic Methods in Combinatorial Optimization: A Study on the Linear Ordering and the Maximum Diversity Problem*. Springer.

Napoletano, A., Martínez-Gavara, A., Festa, P., Pastore, T., & Martí, R. (2019). Heuristics for the constrained incremental graph drawing problem. *European Journal of Operational Research, 274*(2), 710–729. https://doi.org/10.1016/j.ejor.2018.10.017

Pastore, T., Martínez-Gavara, A., Napoletano, A., Festa, P., & Martí, R. (2020). Tabu search for min-max edge crossing in graphs. *Computers & Operations Research, 114*, Article 104830. https://doi.org/10.1016/j.cor.2019.104830

Paulisch, F. N., & Tichy, W. F. (1990). Edge: An extendible graph editor. *Software: Practice and Experience, 20*(S1), S63–S88. https://doi.org/10.1002/spe.4380201307

Preitl, Z., Precup, R. E., Tar, J. K., & Takács, M. (2006). Use of multi-parametric quadratic programming in fuzzy control systems. *Acta Polytechnica Hungarica, 3*(3), 29–43.

Rigatos, G., Siano, P., Selisteanu, D., & Precup, R. E. (2017). Nonlinear optimal control of oxygen and carbon dioxide levels in blood. *Intelligent Industrial Systems, 3*, 61–75.

Sánchez-Oro, J., Martínez-Gavara, A., Laguna, M., Martí, R., & Duarte, A. (2017). Variable neighborhood scatter search for the incremental graph drawing problem. *Computational Optimization and Applications, 68*(3), 775–797. https://doi.org/10.1007/s10589-017-9926-5

Sugiyama, K. (2002). *Graph Drawing and Applications for Software and Knowledge Engineers*. World Scientific.

Sugiyama, K., Tagawa, S., & Toda, M. (1981). Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics, 11*(2), 109–125. https://doi.org/10.1109/TSMC.1981.4308636

Tan, G. W. H., Ooi, K. B., Leong, L. Y., & Lin, B. (2014). Predicting the drivers of behavioral intention to use mobile learning: A hybrid SEM-Neural Networks approach. *Computers in Human Behavior, 36*, 198–213.

Tantau, T. (2013). Graph drawing in TikZ. In W. Didimo, & M. Patrignani (Eds.), *Graph drawing* (pp. 517–528). Springer, 10.1007/978-3-642-36763-2_46.

Warfield, J. N. (1977). Crossing theory and hierarchy mapping. *IEEE Transactions on Systems, Man, and Cybernetics, 7*(7), 505–523. https://doi.org/10.1109/TSMC.1977.4309760

West, D. B. (2001). *Introduction to graph theory* (Vol. 2). Prentice Hall Upper Saddle River.

yWorks, *YEd Graph Editor (2023)*. YWorks, the Diagramming Experts. Online: https://www.yworks.com/products/yed.

Zamfirache, I. A., Precup, R. E., Roman, R. C., & Petriu, E. M. (2023). Neural network-based control using actor-critic reinforcement learning and grey wolf optimizer with experimental servo system validation. *Expert Systems with Applications, 225*, Article 120112.