TOP (2021) 29:319–350
https://doi.org/10.1007/s11750-021-00605-1

INVITED PAPER

Tabu search tutorial. A Graph Drawing Application

Fred Glover[1] · Vicente Campos[2] · Rafael Martí[2]

Accepted: 10 May 2021 / Published online: 14 June 2021

Abstract

Tabu search is an optimization methodology that guides a local heuristic search procedure to explore the solution space beyond local optimality. It is substantiated by the hypothesis that an intelligent solving algorithm must incorporate memory to base its decisions on information collected during the search. The method creates in this way a learning pattern to explore the solution space economically and effectively. Tabu search is a metaheuristic that has proved its effectiveness in a wide variety of problems, especially in combinatorial optimization. We provide here a practical description of the methodology and apply it to a novel graph drawing problem. The most popular method of drawing graphs is the Sugiyama's framework, which obtains a drawing of a general graph by transforming it into a proper hierarchy. In this way, the number of edge crossing is minimized in the first stage of the procedure. Many metaheuristics have been proposed to solve the crossing minimization problem within this drawing convention. The second stage of this procedure minimizes the number of bends of long arcs without increasing the number of crossings, thus obtaining a readable drawing. In this paper, we propose an alternative approach to simultaneously minimize the two criteria: crossing and long arc bends. We apply tabu search to solve this problem and compare its solutions with the optimal values obtained with CPLEX in small and medium-size instances.

Keywords Tabu search · Metaheuristic · Graph drawing · Crossing minimization · Long edges

This invited paper is discussed in the comments available at: https://doi.org/10.1007/s11750-021-00606-0, https://doi.org/10.1007/s11750-021-00607-z, https://doi.org/10.1007/s11750-021-00608-y.

✉ Rafael Martí
Rafael.Marti@uv.es

Fred Glover
fredwglover@yahoo.com

Vicente Campos
Vicente.Campos@uv.es

[1] Meta-Analytics, Inc, Boulder, CO, USA

[2] Universitat de València, Valencia, Spain

1 Introduction

Tabu search (TS) is a metaheuristic methodology (Glover and Laguna 1997) based on the premise that problem-solving, to qualify as intelligent, must incorporate *adaptive memory* and *responsive exploration.* The adaptive memory feature of TS allows the implementation of procedures capable of searching the solution space economically and effectively. Since local choices are guided by information collected during the search, TS contrasts with memoryless designs that heavily rely on semi-random processes that implement a form of sampling. The emphasis on responsive exploration in tabu search derives from the supposition that a bad strategic choice can often yield more information than a good random choice. It is therefore a good instance of how Artificial Intelligence can be applied to solve optimization problems.

The foundations of tabu search (Glover 1963) reflect the benefits of merging elements from the domains of artificial intelligence (AI) and operations research (OR). Both fields started out with the goal of developing methods to solve challenging problems, although OR focused more strongly on mathematical results and AI gave more attention to qualitative analyses. They provided the foundations for the ideas that in the eighties became the source of TS.

TS can be applied to any kind of optimization problem defined as optimize $f(x)$, subject to $x \in X$. The function $f(x)$ may be linear, nonlinear or even stochastic, and the set X summarizes constraints on the vector of decision variables x. The constraints may similarly include linear, nonlinear or stochastic inequalities, and may compel all or some components of x to receive discrete values. What is even more important, is that TS can be applied to optimization problems where we do not have a mathematical model, as is the case of many practical situations. In this paper, we first describe the TS methodology and then apply it to solve a combinatorial optimization problem that arises in many applications, the graph drawing.

Geometric representations have been the subject of study since ancient times. The famous words by Archimedes “Do not disturb my circles!” ("Noli turbare circulos meos!") indicate the early connection between graph drawing and geometrics. Today, automated graph-drawing systems utilize procedures to position nodes and arcs to produce graphs with desired properties. The literature contains many different standards to represent a graph (Di Battista et al. 1998). All of them have readability as the main objective, although there are different ways to achieve this goal. Research studies of graph drawing have produced a prolific range of applications from trees to orthogonal graphs. In this paper, we focus on the realm of graph drawing called hierarchical representations.

General graphs can be drawn as hierarchical maps following Sugiyama’s framework, which obtains a good drawing by representing the arcs according to certain aesthetics that induce readability: straight lines, uniform direction, and low number of crossings (Sugiyama et al. 1981). This framework has been applied in many settings (e.g., Kaufmann and Wagner 2001; Napoletano et al. 2019; or Pastore et al. 2020), and consists of three steps: assign nodes to layers, reduce edge crossings, and assign coordinates to nodes. We apply this method to the graph in Fig. 1 to illustrate its operation.

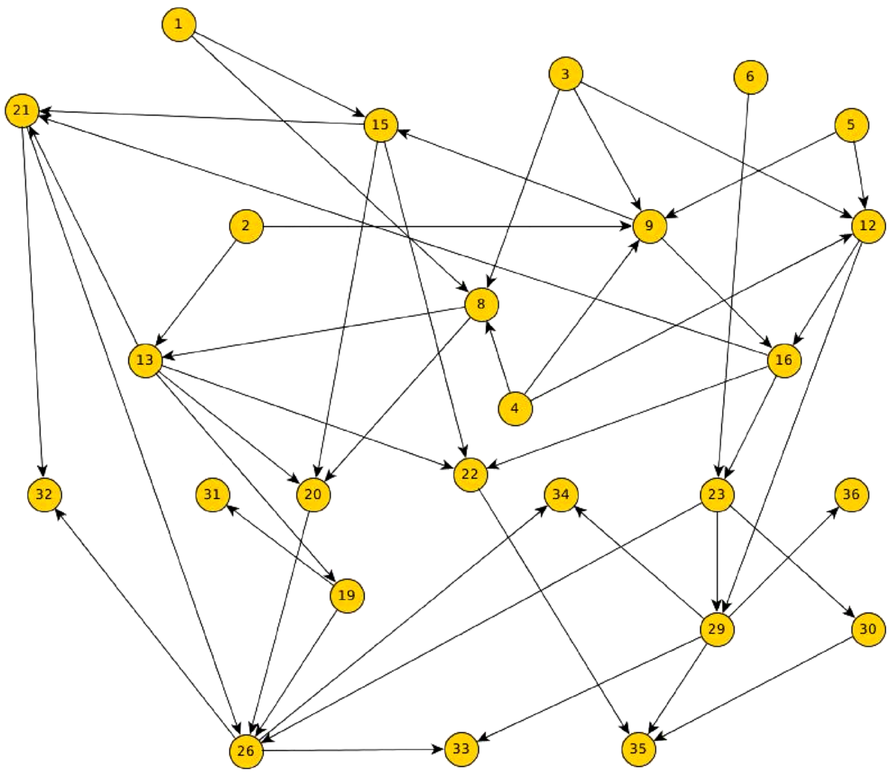

Fig. 1 Original drawing not optimized

The first step in Sugiyama's framework starts by reversing some arcs as appropriate. In particular, if the graph has cycles, pre-processing is applied to reverse the direction of some arcs to make it acyclic (Di Battista et al. 1998). The method identifies a set of arcs with minimum cardinality to be reversed (the so-called "minimum feedback arc set problem"). Note that the graph in Fig. 1 does not contain any cycles and so this step is not required. After pre-processing, the method assigns nodes to layers in the step called *layer assignment*. Arcs joining nodes in noncontiguous layers are replaced with chains of dummy nodes and arcs (between consecutive layers), thus obtaining what is called a proper hierarchy.

Figure 2 shows the proper hierarchy obtained from the example in Fig. 1. We can easily identify chains of dummy nodes, with arcs in red, replacing long arcs. For example, the arc (6, 23) in Fig. 1, is replaced with the chain {(6, 11), (11,18), (18, 23)}. Therefore, nodes 11 and 18 are dummy and the entire chain will be replaced back by the original arc in the final stage of Sugiyama's framework.

In step 2, the framework targets arc crossing minimization. This is a very important problem that has received extensive attention in the scientific literature. In the first implementations of Sugiyama's framework, this step was based on simple ordering rules, reflecting the goal of researchers and practitioners of

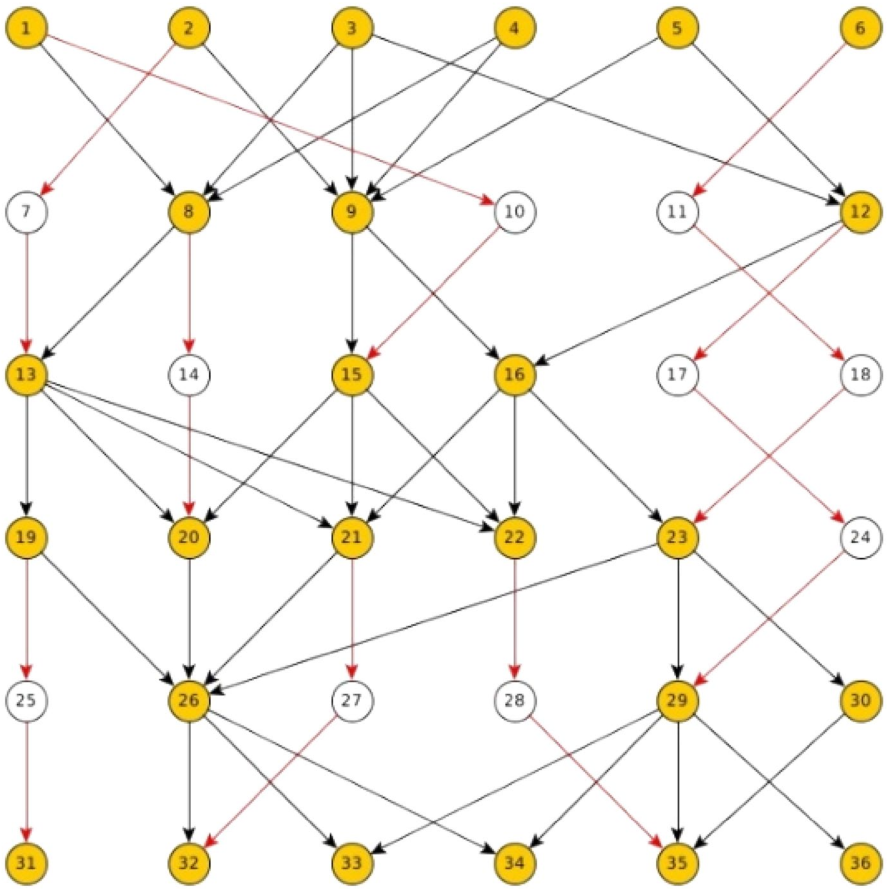

Fig. 2 Layer assignment (step 1) in Sugiyama's framework

quickly obtaining solutions of reasonable quality (Carpano 1980). However, the field of optimization has recently introduced complex metaheuristic methods and, starting with tabu search (Martí 1998; Laguna and Martí 1999), advanced solution strategies have been proposed in the last 20 years to solve graph drawing applications (see for example Sánchez-Oro et al. (2017) and Martí et al. (2018)). In Sect. 2, we provide a mathematical programming model of this NP-hard problem. Figure 3 shows the solution obtained with CPLEX with 22 arc crossings, which compares favorably with the 36 crossings in Fig. 2. In fact, 22 is the optimum number of crossings for this problem.

In the third step of Sugiyama's framework the method assigns x-coordinates to the nodes to obtain straight arcs and to center the nodes among their neighbors. This is done without reordering the nodes, which implies that the number of crossings remains unchanged. Several heuristics have been proposed to perform this coordinate assignment (see for example Brandes and Köpf 2002) in a way

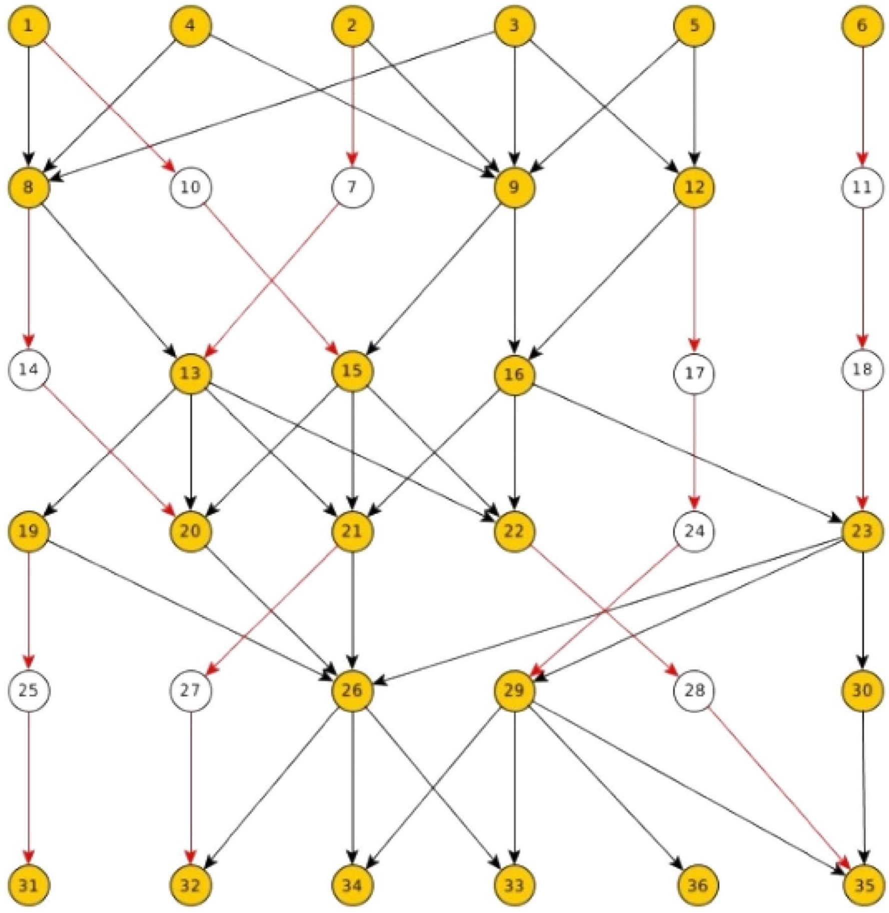

Fig. 3 Crossing minimization (step 2) in Sugiyama's framework

that produces mostly straight long arcs. Specifically, given the original arc (u, v), replaced with the chain with dummy nodes $(u, u_1), (u_1, u_2) \dots, (u_s, v)$ in step 1 of the method, the heuristic in step 3 tries to obtain a drawing with $x\left(u_i\right) = x(u_{i+1})$ for $i = 1, \dots s - 1$, where $x(u_i)$ is the coordinate of node u_i. Long arcs ideally have no bends, although this goal is normally not achieved.

Figure 4 shows the final output of Sugiyama's method in our example with 22 arc crossings and 5 long arcs bends. To describe the entire process in simple terms, we can say that it seeks to minimize crossings in step 2 and to minimize bends in step 3. Many heuristics and metaheuristics have been proposed to solve these two problems and yield a multiplicity of different drawings. Crossing minimization is a primary criterion in these methods, and bend minimization is a secondary one, considering that step 3 does not allow the nodes in a layer to be reordered, thus keeping the number of crossings fixed.

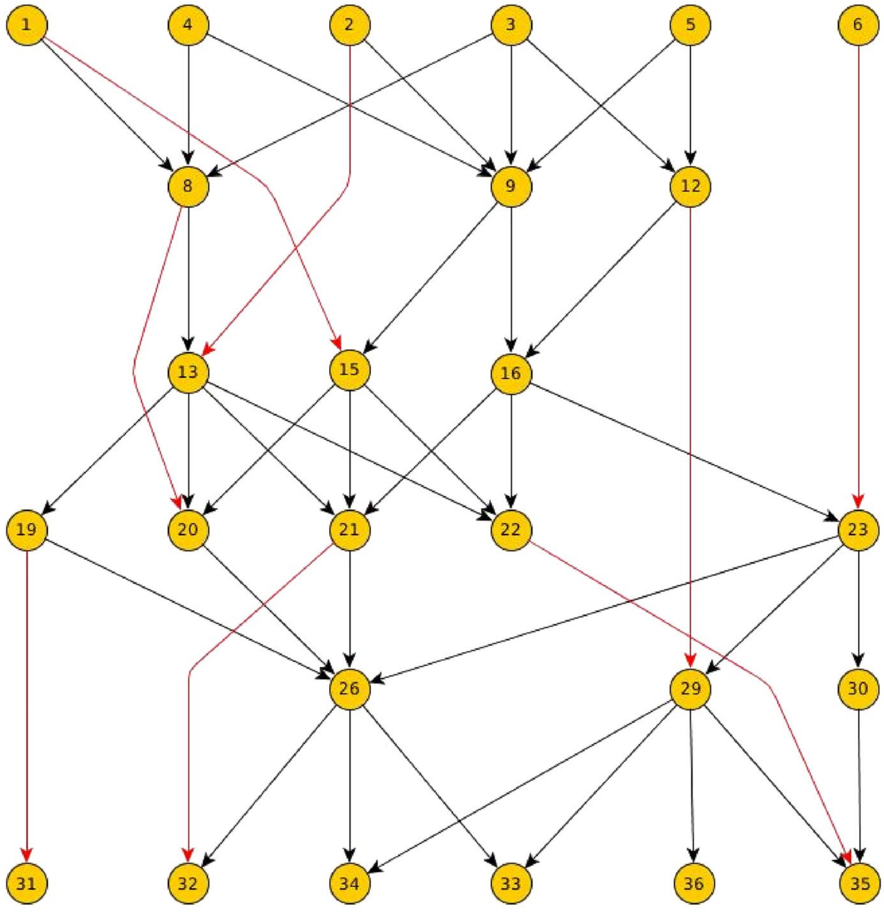

Fig. 4 Coordinate assignment (step 3) in Sugiyama's framework

In this paper, we propose an alternative to this 3-step method, where we seek to minimize the number of crossings subject to the constraint that long arcs have no bends. In our experience, bends of long arcs present an important difficulty when analyzing a drawing, and should not be made entirely subordinate to the goal of minimizing the number of crossings. A small increase in the number of crossings is more than acceptable in terms of readability if long arcs are properly displayed.

We show that our alternative framework for hierarchical graph drawing creates more readable maps than the customary sequential approach. We provide a mathematical formulation to solve small and medium-size instances with this framework, and a tabu search metaheuristic to target large instances. An empirical comparison shows that our tabu search method is able to obtain high-quality solutions in short computational times, an essential feature for automatic graph drawing systems.

2 Tabu search

In this section, we describe the tabu search methodology in general terms; i.e., as a tutorial to be applied to any combinatorial optimization problem. The term *tabu search* was coined in the same paper that introduced the term *metaheuristic* (Glover 1986), and it is based on four early developments:

- Strategies that combine decision rules based on logical restructuring and non-monotonic search,
- Systematic violation and restoration of feasibility,
- Flexible memory based on recency and frequency,
- Selective processes for combining solutions, applied to a systematically maintained population.

Traditional approaches to solving job shop scheduling problems evolved in the 60 s to decision criteria for generating schedules, and embedded these criteria in local decision rules that were previously applied in isolation from each other. This approach of seeking to benefit from multiple rules by intelligently alternating among them motivated consideration of a contrasting strategy (Glover 1963) that combines decision rules to create new ones. The method then generated a series of trial solutions by varying the parameters determining the integrated rules. Instead of stopping at a local optimum of the series, the process generated a non-monotonic search pattern by systematically varying the underlying parameters to produce additional trial solutions. From a historical perspective, this can be considered the direct ancestor of tabu search. Two consecutive papers in *ORSA Journal on Computing* (Glover 1989, 1990) established the methodology in practical terms that permitted researchers and practitioners to create implementations for many different optimization problems.

To introduce the methodology, we compare a simple version of TS with a standard descent method for minimizing an objective function. Such a method only permits moves to neighbor solutions that improve (reduce) the current objective function value and ends when no further improvement is possible. The final solution is a local optimum, since it is at least as good or better than all solutions in its neighborhood. TS may start in the same way as a descent method, but instead of stopping at the local optimum, it continues the search. To do that, it has to select moves that cause the objective function value to deteriorate, and therefore it incorporates a mechanism to prevent cycling when alternating between improving and non-improving moves. This is implemented in an efficient way by means of a dynamic neighborhood definition.

2.1 Short term memory

Let $N(x)$ be the neighborhood of solution $x \in X$, where each solution $y \in N(x)$ is reached from x applying a move. In a simple implementation of tabu search, the method utilizes a *short-term memory* function by drawing on the search history to modify the neighborhood. Specifically, a subset $N^*(x)$ of $N(x)$ is created that employs a *tabu* classification to exclude elements of $N(x)$ added to x by moves of previous iterations. Let $T \subseteq X$ be the set of solutions visited, within a specified number of previous iterations, that we want to exclude from examination (and thus selection) in the following iterations. We label these solutions as tabu by defining $N^*(x) = N(x) \setminus T$.

One of the TS innovations comes from the fact that instead of directly labeling some solutions as tabu, the methodology first identifies properties or attributes that are important for characterizing a solution or a move, and then labels some of them as tabu according to the search history. In problems where the moves are defined by adding and deleting elements, such as nodes, arcs, or edges in a graph, these elements can be used as attributes. Solutions or moves containing such tabu attributes are excluded from exploration, but important benefits are derived from recording, selecting, and updating such attributes, generally giving preference to solution attributes over move attributes. Attributes can be coarse-grained, shared by many different solutions, or fine-grained, shared by few solutions. An extreme instance of fine-grained attributes that proves useful in certain settings occurs in so-called solution-based tabu search, which uses hash function values as attributes to differentiate among solutions.

The most commonly used short-term memory keeps track of solutions attributes that have changed during the recent past, and is called **recency-based memory**. To exploit this memory, selected attributes that occur in solutions recently visited are labeled tabu-active, and solutions subsequently examined that contain tabu-active elements are those that are tabu. This memory is usually implemented by creating one or several tabu lists, which record the tabu-active attributes. A simple and effective implementation of the **tabu list** is to record the iteration number that identifies when the tabu-active status of an attribute starts or ends. This permits the tabu status of a move to be tested in constant time.

To illustrate the principles above, consider a problem in which each solution x is a permutation of n elements, where x_i is the element in position i. The first step to create a short-term memory tabu search is to define a move to modify solutions. Say for instance that we consider an insertion move, $move(x, x_i, j)$, in which the element x_i in position i is inserted in position j. Then, the neighborhood $N(x)$ consists of all the solutions y that can be obtained when an element in x is inserted in a different position. We can denote it in symbolic terms as $y = x \oplus move(x, x_i, j)$. Given a move mechanism, such as the insert mechanism we have selected for our example, the next step is to choose the attribute or attributes that will be used for the tabu classification. A straightforward move attribute is the element x_i inserted, and we may record in $tabu(x_i)$ the iteration in which we move the attribute, thus preventing the attribute from being moved again for a certain number of iterations, called **tabu tenure**. Clearly, we may define many other attributes, such as the position j in which we insert the element, or both element and position together. Figure 5 shows a simple algorithm to implement the approach, where $CL(x)$ refers to the candidate list of non-tabu moves.

Figure 5 shows in Steps 5 and 6 that the main difference between TS and a descent method is that the former always perform a move, the best available one

```
1.  Generate an initial solution $x$ and initialize $x_{best} \leftarrow x$
2.  Initialize $tabu(x_i) = 0 \quad i = 1, \dots, n$
For $IterNum = 1$ to $MaxIter$
    3.  Compute $CL(x) = \{x_i : IterNum - tabu(x_i) > TabuTenure\}$
    4.  Compute $N^*(x) = \{\ y \in X :\ y = x \oplus move(x, x_i, j) \quad x_i \in CL(x)\ \}$
    5.  Select the best solution $y^* \in N^*(x) \quad (y^* = x \oplus move(x, x_{i^*}, j^*))$
    6.  Perform the move: $x \leftarrow y^*$
    7.  If $x$ is better than $x_{best}$ then $x_{best} \leftarrow x$
    8.  Update tabu status: $tabu(x_{i^*})= IterNum$
9.  Return $x_{best}$
```

Fig. 5 A simple short term tabu search

from the candidate list, even if it causes the objective function to deteriorate. This outline of the method also makes explicit that the neighborhood composition, $N^*(x)$, only considers solutions obtained by populating the candidate list $CL(x)$ with elements that are non-tabu. The parameter *TabuTenure* is typically adjusted via experimentation and depends on the size of the problem instance. No single rule has been designed to yield an effective tenure for all classes of problems. This is partly because an appropriate tabu tenure depends on the strength of the tabu activation rule employed.

The version of the short term tabu search depicted in Fig. 5 scans the entire neighborhood $N^*(x)$ of x in search for the best neighbor solution y^*. It is well documented in the heuristic literature that this can be computationally too expensive or even impractical in some problems. It is therefore customary in TS to apply **candidate list strategies** to perform a selective or simply efficient examination of the neighborhood, thereby additionally limiting the candidate list $CL(x)$ to narrow the examination of elements of $N^*(x)$ and achieve an effective tradeoff between the quality of y^* and the effort expended to find it.

A well-known way to speed up the neighborhood examination in a descent method is the **first improving strategy** that chooses the first move found that improves the objective function and disregards the rest of the neighborhood. This strategy may be reinforced by examining solutions in an appropriate ordering, where the most promising ones come first. Problem-specific knowledge can clearly facilitate this approach by evaluating the moves' influence in the objective function to anticipate which ones to try first in future evaluations. The following three procedures are among the candidate list strategies proposed in Glover and Laguna (1997):

- The **Aspiration Plus** strategy establishes a threshold for the quality of a move, based on the history of the search pattern. The procedure operates by examining moves until finding one that satisfies this threshold.
- The **Elite Candidate List** approach first builds a Master List by examining a large number of moves and saving some of the best moves encountered. Then at each subsequent iteration, the current best move from the Master List is chosen to be executed (without re-evaluation the list and move values), continuing until such a move falls below a given quality threshold and the Master List is rebuilt.

- In the **Successive Filter Strategy**, moves are broken into component operations, and the set of moves examined can be reduced by restricting consideration to those that yield high-quality outcomes for each operation separately.

Short-term memory designs are usually accompanied by an **Aspiration Criterion**, which in its simplest form consists of removing a tabu classification from a trial move when the move yields a solution better than the best obtained so far. This criterion is widely used, although other aspiration criteria can prove effective for improving the search.

2.2 Long term memory

Many tabu search implementations limit themselves to the short-term memory strategies described above. However, it can often be advantageous to include additional elements. In those cases, TS becomes significantly stronger by including longer term memory and its associated strategies that exploit tradeoffs between intensification and diversification.

Intensification strategies are based on modifying choice rules to encourage move combinations and solution features historically found good. They may also initiate a return to attractive regions to search them more thoroughly. **Diversification** strategies, on the other hand, seek to incorporate new attributes and attribute combinations that were not included within solutions previously generated. In one form, these strategies undertake to drive the search into regions dissimilar to those already examined. It is important to keep in mind that intensification and diversification are mutually reinforcing.

Frequency-based memory provides a type of information that complements the information provided by recency-based memory. Frequencies typically consist of ratios, whose numerators represent counts expressed in two different measures: a transition measure (the number of iterations where an attribute changes), or a residence measure (the number of iterations where an attribute belongs to solutions visited on a particular trajectory). The denominators generally represent the total number of occurrences of all events represented by the numerators (sum or maximum depending on the case). The ratios produce transition frequencies that keep track of how often attributes change and residence frequencies that keep track of how often attributes are members of solutions generated.

A high residence frequency may indicate that an attribute is highly attractive if the domain consists of high-quality solutions. On the other hand, a residence frequency that is high when the domain is chosen to include both high and low-quality solutions may point to an attribute that causes the search space to be restricted, and that needs to be relaxed to allow increased diversity. Attributes that have greater frequency measures, just as those that have greater recency measures, may trigger a tabu activation if they are based on consecutive solutions that end with the current solution. However, frequency-based memory often finds its most productive use as part of a longer term strategy, which employs incentives as well as restrictions to determine which moves are selected.

We do not have space here to describe the advanced designs, such as **strategic oscillation** or **path relinking**, that may complement long-term strategies to create complex tabu search implementations. We refer the reader to the tabu search monograph (Glover and Laguna, 1997) for a comprehensive description of the methodology. Now we return to considering the graph drawing problem to provide an improved model and examine its solution by CPLEX and tabu search.

3 Arc crossing and long arcs

Figures 1, 2, 3 and 4 in the introduction illustrate Sugiyama's framework to represent graphs. Roughly speaking, the framework provides a readable drawing of a graph in which arc crossings are minimized and other aesthetic criteria are considered secondarily. In this section we propose a modification of this framework to improve the graph readability. We first motivate our modification and then propose a formulation as a mathematical programming model.

As previously noted, we can readily compute the number of arc crossings and identify the long arcs with bends, as illustrated in the example of Fig. 3. Here the number of arc crossings is 22 and the long arcs with bends are identified to be (1, 15), (8,20), (2, 13), (21, 32), (22, 35) which have one bend each one, totalizing 5 bends. Note however, that not all the bends are equally important in terms of readability. In particular, the long arc (1, 15) appears more difficult to read than the long arc (8, 20), although both have one bend. We propose to consider the **alignment of the nodes** in a long arc as a way to capture its readability and generalize in this way the concept of bend. For example, in the case of long arc (1, 15), we can consider that the position (x-coordinate) of node 1 is $x = 1$, and the position of node 15 is $x = 3$. Therefore, we observe a difference of 2 in the x-coordinates of this long arc. Moreover, if we also compute the x-coordinate of the intermediate dummy node in this arc (node 10 in Fig. 3), $x = 2$, we can include this value in the computation of the alignment of the long arc. In general, we propose to subtract the x-coordinate of each intermediate dummy node and the final node from the coordinate of the initial node, and add the absolute values of these differences. Thus, for arc (1, 15) we obtain an alignment value of $1+2=3$.

The alignment value of the long arc (8, 20) in Fig. 4 is 1, reflecting that this arc is more readable than arc (1,15) with an alignment value of 3 computed above. The alignment value of (8, 20) is derived by observing that the x-coordinate of node 8 is $x = 2$, the dummy node 14 in this arc has a position of $x = 1$, and the final node in the arc is 20 with a position of $x = 2$. Subtract these latter two x-coordinates from the coordinate of the initial node 8 and summing the absolute values of these differences yields

$$|1 - 2| + |2 - 2| = 1$$

Therefore, instead of considering the number of bends in Fig. 2b, which is 5, as a measure of the long arcs readability, we propose to evaluate the alignment on the nodes in these long arcs, which results in a value of 10 in this figure.

We have empirically found that drawings with an alignment of 0, where all the nodes in a long arc are in the same x-position, are very easy to read. However, the standard 3-step method in Sugiyama's model limits the reduction of the alignment in step 3 to those positions of the nodes in which the ordering obtained in step 2 (when arc crossings are minimized) is not changed. Hence, by contrast, we consider a model in which we incorporate the alignment simultaneously with the minimization of arc crossings. In this way, we merge steps 2 and 3 of Sugiyama's method in a single step. In particular, we propose to minimize the number of crossings subject to a constraint to keep the alignment of the long arcs equal to 0.

Figure 6 shows an alternative way to draw the example shown in Fig. 4 obtained with our new formulation with CPLEX given below, which includes the constraint that each long arc has an alignment of 0, and therefore no bends.

Figure 6 has 29 arc crossings and no bends. In fact, it has an alignment value of 0, meaning that all long arcs are drawn with a straight vertical line. Although this drawing has a larger number of crossings than the one in Fig. 4 (29 versus 22 crossings), we believe that it is easier to read due to the alignment of the long arcs.

We have made several tests similar to this one and have shown both drawings, the one with minimum crossings, and the one minimizing crossings but restricted to

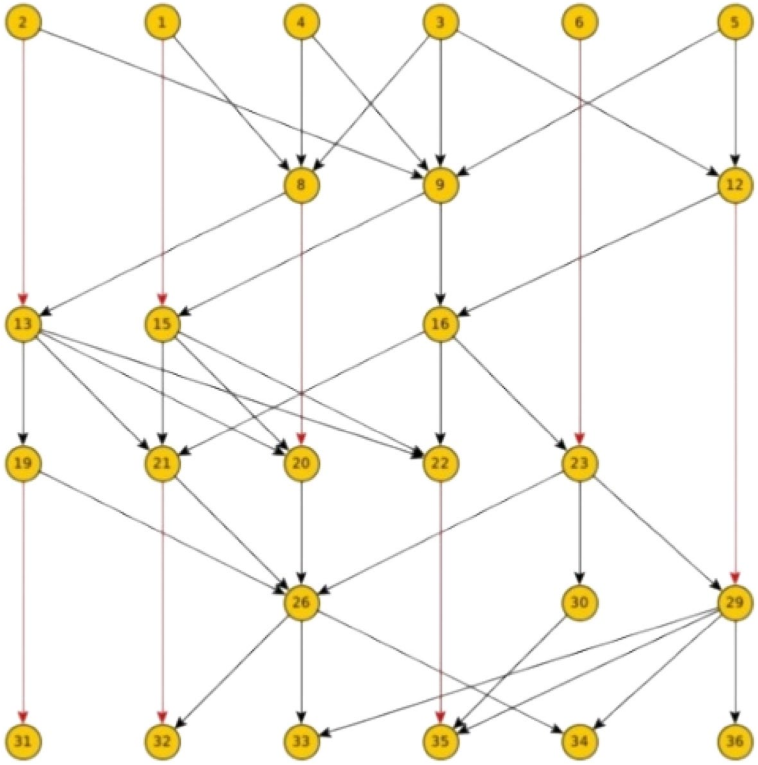

Fig. 6 Crossings minimized subject to no bends

long arcs aligned, to operations research practitioners. Figures 7 and 8 illustrate one of these tests in which we depict the long arcs in red to make them easy to identify. In all the cases, the OR practitioners gave us the same answer that the new (constrained) model obtains better drawings than the previous (sequential) model that first minimizes crossings and then aligns long arcs.

3.1 Mathematical model

A hierarchical graph $H = (V, A, nl, L)$ is defined as a graph$G = (V, A)$, where V and A represent the set of nodes and arcs, respectively, and the layering function $L(v) : V \rightarrow \{1,2, \ldots, nl\}$ indicates the number of the layer (index) where v resides. Hence, $L(v) - L(u)$ is the length of the arc $(u, v) \in A$. The L function implicitly defines the sets of nodes $L_h = \{v \in V : L(v) = h\}$ for $h = 1,2, \ldots, nl$ which we refer to as layers. Let n_h be the number of nodes in layer h. For example, in Fig. 9

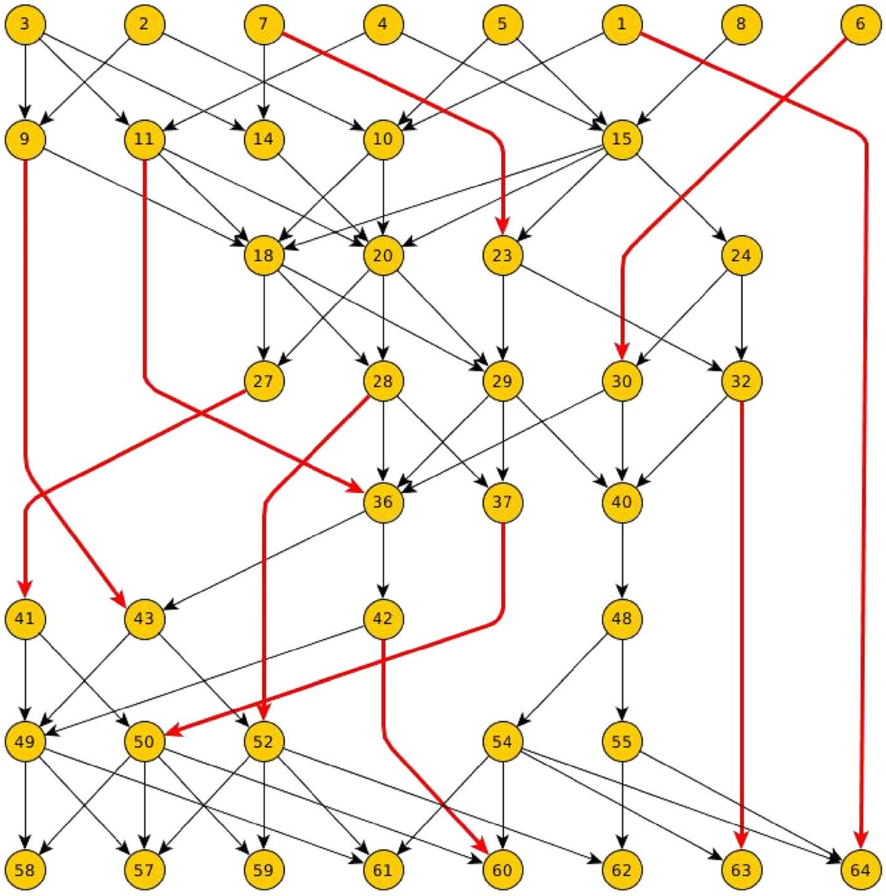

Fig. 7 Example of Sugiyama's output

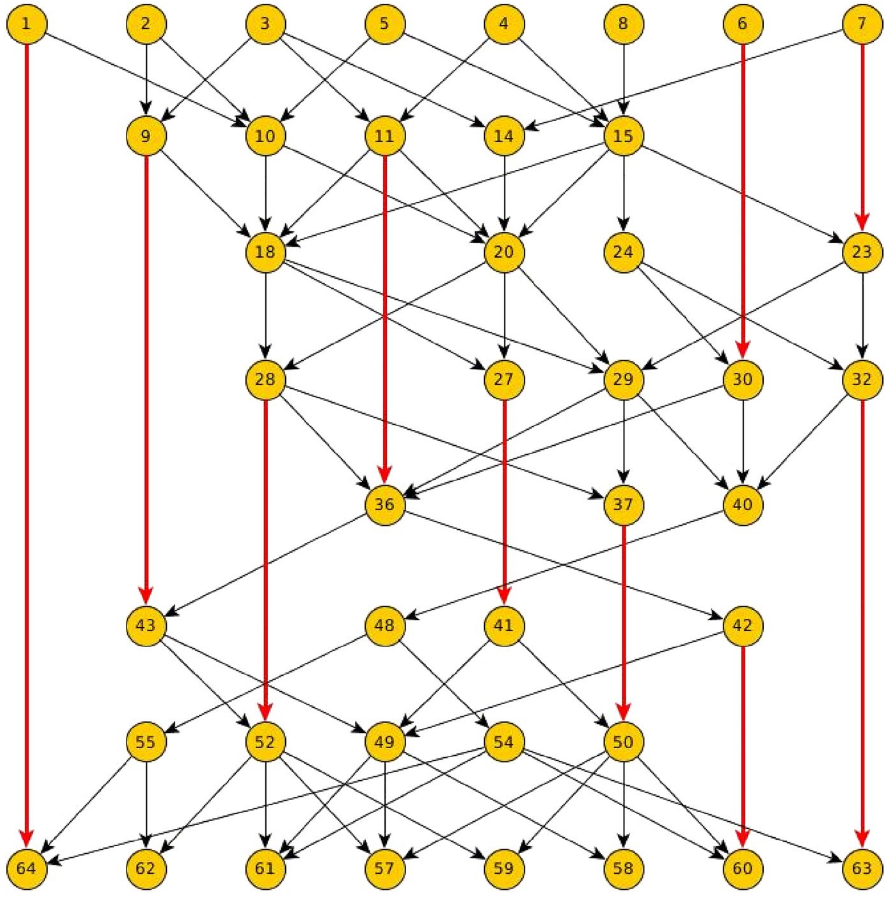

Fig. 8 New model output of the example in Fig. 7

we can identify four layers, depicted as horizontal lines, where node 8 is in layer 2 ($L(8) = 2$), and arc (12, 19) has a length of $L(19) - L(12) = 4 - 3 = 1$. An arc (u, v) is long if $L(v) > L(u) + 1$.

A proper hierarchy is obtained from a hierarchy by replacing long arcs $A_L \subseteq A$ with a chain of dummy nodes and arcs of length 1. Given a long arc $(u, v) \in A_L$, we replace it with a chain of arcs $(u, u_1), (u_1, u_2) \dots, (u_s, v) \in A'$, where $u_1, u_{2,} \dots, u_s \in V'$ are intermediate (dummy) nodes. In this way, we obtain the proper hierarchy $PH = (V \cup V', A', nl, L)$, in which the set of nodes V' contains all the nodes in the original hierarchy V and the dummy ones added in this process. The set of arcs A' contains all the original arcs of length 1, and the chains of arcs replacing the long arcs in A. In this way, all the arcs in A' are of length 1.

Jünger and Mutzel (1997) proposed an integer linear formulation for arc crossing minimization in proper hierarchical graphs (see also Jünger et al. 1997) based on binary variables c_{ijkl} that take the value 1 when a crossing between arcs $(i, j), (k, l)$

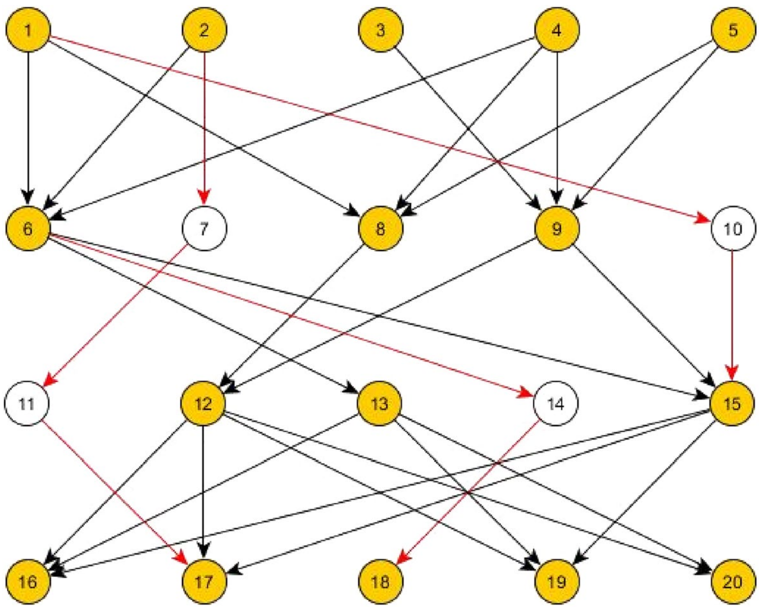

Fig. 9 Example with 3 long arcs and 4 dummy nodes

occurs. Additionally, variable $x_{ij}^h = 1$ when node i precedes node j in layer h, and 0 otherwise. We adapt this formulation to include the alignment constraints for long arcs.

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

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

$$x_{ki}^h + x_{jl}^{h+1} - c_{ijkl} \leq 1 \ (i,j),(k,l) \in A',\ i < k,\ j \neq l,\ h = 1, \dots, nl - 1 \tag{2}$$

$$x_{ij}^h + x_{jk}^h + x_{ki}^h \leq 2 \ \ 1 \ \leq i < j < k \leq n_h,\ h = 1, \dots, nl \tag{3}$$

$$x_{ij}^h + x_{ji}^h = 1 \ \ 1 \leq i < j \leq n_h, h = 1, \dots, nl \tag{4}$$

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

Constraints (1) and (2) above force c_{ijkl} to take the value 1 when the x-variables indicate a crossing. Constraints (3) are the so-called 3-dicycle constraints, originally proposed for the Linear Ordering Problem by Grötschel et al. (1984), and adapted by Jünger and Mutzel (1997) to the drawing problem.

For each long arc (u, v), from layer h to layer $h + s$, and intermediate nodes$u_1, u_2, \ldots, u_s$, we add the following alignment constraints:

$$\sum_{1 \le i \le n_h} x_{iu}^{h} - \sum_{1 \le i \le n_{h+1}} x_{iu_1}^{h+1} = 0$$

$$\sum_{1 \le i \le n_h} x_{iu}^{h} - \sum_{1 \le i \le n_{h+2}} x_{iu_2}^{h+2} = 0$$

$$\ldots$$

$$\sum_{1 \le i \le n_h} x_{iu}^{h} - \sum_{1 \le i \le n_{h+s-1}} x_{iu_s}^{h+s-1} = 0$$

$$\sum_{1 \le i \le n_h} x_{iu}^{h} - \sum_{1 \le i \le n_{h+s-1}} x_{iv}^{h+s} = 0$$

Note that the expression

$$\sum_{1 \le i \le n_h} x_{iu}^{h}$$

provides the position of node u in layer h. Then, with the additional alignment constraints, we are adding to the classic formulation, we are indicating that all dummy nodes $u_1, \ldots, u_s$ have the same position in the ordering of their layer as the original node u. This also applies to the final node v of the original arc (u, v). In this way, we are constraining the model to provide solutions in which the long arcs are aligned, i.e., with no bends.

We illustrate now the additional constraints on a small example. Figure 9 shows a graph with 20 nodes, in which four of them are dummy nodes modeling long arcs (7, 10, 11, and 14).

The graph in Fig. 9 has 4 layers with 5 nodes in each one. The three long arcs are: (2, 17), modeled with the dummy nodes 7 and 11, (1, 15), modeled with the dummy node 10, and finally (6, 18), modeled with dummy node 14. Note that, by design, dummy nodes always have a degree of 2. Therefore, the *long-arc constraints* that force the model to produce no-bends solutions in the case of the (2, 17) are:

$$\sum_{1 \le i \le 5} x_{i2}^{1} - \sum\nolimits_{1 \le i \le 5} x_{i7}^{2} = 0$$

$$\sum_{1 \le i \le 5} x_{i2}^{1} - \sum\nolimits_{1 \le i \le 5} x_{i11}^{3} = 0$$

$$\sum_{1 \le i \le 5} x_{i2}^{1} - \sum\nolimits_{1 \le i \le 5} x_{i17}^{4} = 0$$

Solving the mathematical model with the long arcs constraints for the three arcs described above, we obtain the solution depicted in Fig. 10. It has 19 crossings and no bends.

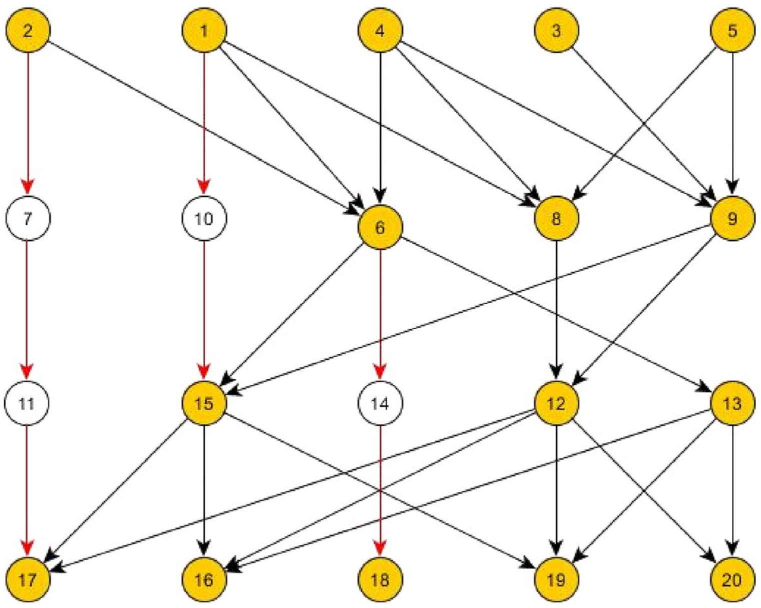

Fig. 10 Optimal solution of the Fig. 6 example

The small example in Fig. 10 with 4 layers and 5 nodes in each layer, can be solved with CPLEX in a few seconds. Unfortunately, if we increase the size of the problem, CPLEX rapidly encounters difficulty in solving it optimally. It must be noted that the original formulation for the crossing minimization problem, is easily transformed to the well-known linear ordering model, for which efficient branch and cut methods have been proposed (Martí and Reinelt 2011). However, the inclusion of the long-arc constraints modifies the polyhedron of solutions, resulting in a more complex problem. We found that CPLEX requires significantly longer running times to solve the model when we include these constraints, and therefore we are compelled to use a different method if we want to target medium and large size instances. In particular, CPLEX is not able to solve instances with 10 layers and 10 nodes per layer in 1 h of CPU time. We propose a method based on the tabu search methodology to solve the problem with our new formulation.

4 Tabu search for graph drawing

Our tabu search algorithm is designed in a flexible way for solving the arc crossing minimization problem subject to the long arcs alignment constraints. We have empirically found that if we restrict the search to the orderings in which all long arcs are completely aligned (with no bends), the search only enables a fraction of the solutions to be explored, leading to low-quality final solutions. We, therefore, permit our algorithm to explore solutions represented by any ordering of the nodes in each layer. We then evaluate both the alignment and the number of crossings to guide the search to solutions with long arcs aligned as much as possible and with a minimum number of crossings.

4.1 Solution evaluation

Since the arcs in a HDAG are straight lines that join the nodes in two contiguous layers, a drawing of a hierarchical graph $H = (V, A, nl, L)$ defined on graph $G = (V, A)$ is given by the ordering of the nodes in each layer. Therefore, a drawing of H is defined as$D = (H, \Phi)$, where $\Phi = \{\varphi_1, \varphi_2, \dots, \varphi_{nl}\}$ and φ_i is the ordering (permutation) of the nodes in the layerL_i. That is, $\varphi_i(j)$ is the node in position j in layerL_i. The position of node v is defined as $\pi(v)$ in such a way that if $v = \varphi_i(j)$ then $\pi(v) = j$.

As described in the previous section, each long arc $(u, v) \in A_L$ is replaced with a chain of arcs $(u, u_1), (u_1, u_2) \dots, (u_s, v) \in A'$, where $u_1, u_{2,} \dots, u_s \in V'$ are intermediate (dummy) nodes. We compute the alignment of long arc (u, v) as:

$$AL(u, v) = \sum_{i=1}^{s} \left|\pi(u) - \pi\left(u_i\right)\right|$$

Note that $AL(u, v)$, as a mathematical function, also depends on the hierarchical graph H and the ordering Φ, but for the sake of simplicity we do not include them in the AL definition. The total **alignment value** in a drawing $D = (H, \Phi)$ is computed as:

$$AL(D) = \sum_{(u,v)\in A_L} AL(u, v)$$

Although we want to obtain a drawing D with all the long arcs aligned, yielding $AL(D) = 0$, as indicated above, we permit drawings to be generated with positive alignment values to give more freedom to the search process to minimize the number of crossings, which is by itself a hard optimization problem.

The problem of **minimizing the arc crossings** in a HDAG may be formulated as the problem of finding the optimal ordering in each layer. The optimal drawing $D^\star$ is such that no other D has fewer arc crossings. An arc crossing is produced between arcs (u, v) and $\left(u', v'\right)$, where $u, u' \in L_i$ and $v, v' \in L_{i+1}$ when:

$$\left(\pi(u) < \pi\left(u'\right) \wedge \pi(v) > \pi\left(v'\right)\right) \vee \left(\pi(u) > \pi\left(u'\right) \wedge \pi(v) < \pi\left(v'\right)\right)$$

For nodes $u, u' \in L_i$ where $\pi(u) < \pi(u')$, we define $C_{i+1}(u, u')$ as the number of arc crossings between layers L_i and L_{i+1} which are due to all the arcs incident to u and u'. Formally,

$$C_{i+1}\left(u, u'\right) = \sum_{v\in\Lambda_{i+1}(u)} \left|v' \in \Lambda_{i+1}\left(u'\right) : \pi(v) > \pi\left(v'\right)\right|$$

where $\Lambda_{i+1}(u) = \Lambda(u) \cap L_{i+1}$ is the set of nodes in L_{i+1} that are adjacent to u and $\Lambda(u) = \{v \in V : (u, v) \in E\}$ is the set of all nodes adjacent to$u$. Similarly, we define $C_{i-1}\left(u, u'\right)$ as the number of crossings between layers L_{i-1} and L_i produced by arcs incident to u and u' inL_i. We also define $\Lambda_{i-1}(u) = \Lambda(u) \cap L_{i-1}$ as the set of nodes in L_{i-1} that are adjacent to$u$.

With these definitions, we can calculate the total number of arc crossings in a drawing $D = (H, \Phi)$ as:

$$C(D) = \sum_{i=1}^{k-1} \sum_{\substack{u, u' \in L_i \\ \pi(u) < \pi(u')}} C_{i+1}(u, u') = \sum_{i=2}^{k} \sum_{\substack{u, u' \in L_i \\ \pi(u) < \pi(u')}} C_{i-1}(u, u')$$

In the previous section, we describe our problem with a mathematical model, which is equivalent to find an optimal drawing $D^\star$ with minimum $C(D)$ value, overall possible drawings D with $AL(D) = 0$. However, such a description does not consider the running time to obtain the solution, just the quality or properties of the optimal solution. To obtain relatively good solutions in short computational times, we adopt the goal of obtaining a drawing D close to the optimal minimum value $C(D^\star)$ with a low $AL(D)$ value. Note that this can be easily formulated with a model in which the alignment constraints have a positive value on the right hand side, instead of the 0 in our model in Sect. 3.1, reflecting the desired $AL(D)$ value. However, we made some tests with such a model with CPLEX and conclude that the proposed one with 0 alignment, requires a similar CPU time and produces better solutions.

4.2 Neighborhood definition

The neighborhood in our tabu search method is based on a deterministic selection without replacement. All vertices are considered for repositioning in their corresponding layer. The method sweeps the entire solution, one vertex at a time, and probes the movement of vertices to all positions in their layer. Thus, a vertex v currently in position $\pi(v)$ in layer L_i, is evaluated for a possible move to positions $p = 1, 2, \ldots, \pi(v) - 1, \pi(v) + 1, \ldots |L_i|$, all of them in the same layer L_i.

The move of inserting vertex v in position p has two associated evaluations: the change in the number of crossings, and the change in the alignment value. In mathematical terms, if D is the current drawing (a feasible solution of our problem), and $D_{v,p}$ is the drawing obtained when removing v from its current position $\pi(v)$ and inserting it in position $p \neq \pi(v)$, then we define

$$Cval\_move(D, v, p) = C(D) - C(D_{v,p}),$$
$$ALval\_move(D, v, p) = AL(D) - AL(D_{v,p}).$$

It is clear that if a move is able to improve both the number of crossings ($Cval\_move(D, v, p) > 0$) and the alignment ($ALval\_move(D, v, p) > 0$), we can say that it is an improving move and perform it. However, the tabu search procedure acts beyond a standard local search in the sense that it always executes a move and it only stops when the time limit is reached. Therefore, we define a more elaborated move selection applying candidate list strategies, as it is customary in tabu search.

In this problem, we want to minimize the number of crossings $C(D)$ subject to the constraint that long arcs have to be aligned ($AL(D) = 0$). However, since we treat it as a soft constraint, exploring solutions with positive (but low) AL-values, we propose a move classification based on their two evaluations above.

Given a drawing D and a layer L_i, let $moves\left(D, L_i\right)$ be the set of all feasible moves in this layer, containing all the moves that insert a vertex $v \in L_i$ in a position $p \neq \pi(v)$. We define two sets of moves contained in this sets: $Gmoves\left(L_i\right)$ and $Amoves\left(L_i\right)$. The set $Gmoves\left(L_i\right)$ contains the **good moves** associated with layer L_i, meaning that they reduce both criteria, number of crossings, and alignment. In mathematical terms, if $move(v,p)$ consists of inserting v in position p, then:

$$Gmoves\left(D, L_i\right) = \left\{ \begin{array}{c} move(v,p) \in moves\left(D, L_i\right) \ : \ Cval\_move(D,v,p) > 0, \\ ALval\_move(D,v,p) > 0 \end{array} \right\}$$

Note, however, that if the current drawing has an alignment of 0 (i.e., all long arcs are straight lines), then we consider that a good move reduces the number of crossings and keeps the alignment 0, since it cannot be further reduced. Symmetrically if the current drawing has 0 crossings, we consider that a good move reduces the alignment while keeping the number of crossings in 0.

The set $Amoves\left(D, L_i\right) \subseteq moves\left(D, L_i\right)$ contains the **acceptable moves** in layer L_i. Acceptable means in this context that the move reduces the alignment but marginally increases the number of crossings. Considering that the alignment is a soft constraint and that crossing reduction is an objective, we give priority to the alignment, and if there is no move that improves both objectives ($Gmoves\left(D, L_i\right) = \emptyset$), we permit the objective to slightly deteriorate. In particular, we introduce the search parameter $\alpha \in [0,1]$ that indicates the maximum relative increase allowed in the number of crossings. For example, a value of $\alpha = 0.2$ indicates that we consider moves that increase the number of crossings by 20% of its current value. In mathematical terms:

$$Amoves\left(D, L_i\right) = \left\{ move(v,p) \in moves\left(D, L_i\right) \ : \ \Delta Cmove(D,v,p) \langle \alpha, ALval\_move(D,v,p) \rangle 0 \right\}$$

where $\Delta Cmove(D,v,p) = \frac{C(D_{v,p}) - C(D)}{C(D)}$.

4.3 Short term tabu search

Figure 11 depicts the pseudocode of a sweep of the algorithm, which scans all the layers trying to perform good moves. The complete tabu search method performs many consecutive sweeps, until the time limit is reached.

It must be noted that "perform the best move" refers to the move that leads to the minimum alignment, and if there is more than one with the same alignment evaluation, then we select the one that reduces the number of crossings among them. Additionally, note that in Step 7 of Fig. 11, the term "best" has to be taken as the "least bad" since this step is only performed when there is no improving move in the entire sweep. As described in the previous section, one of the main differences between tabu search and the standard local search is that tabu search always performs a

```
For $i = 1$ to $k$ (number of layers)
    For $j = 1$ to $|L_i|$ (size of layer $L_i$)
        1. $v = \varphi_i(j)$ (i.e., $\pi(v) = j$)
        2. Compute the move values $Cval\_move(D, v, p)$
           and $ALval\_move(D, v, p)$ for $p = 1$ to $|L_i|$.
        3. Include the moves in the corresponding sets,
           $moves(D, L_i)$, $Gmoves(D, L_i)$, and $Amoves(D, L_i)$.
    Next $j$
    If $Gmoves(D, L_i) \neq \emptyset$
        4. Perform the best move in $Gmoves(D, L_i)$.
    Else If $Amoves(D, L_i) \neq \emptyset$
        5. Perform the best move in $Amoves(D, L_i)$.
    End If
Next $i$
If no move has been performed in the entire sweep from $L_1$ to $L_k$
    6. Compute $moves(D) = \cup_{i=1,\ldots,k} moves(D, L_i)$
    7. Perform the best move in $moves(D)$
End If
```

Fig. 11 A sweep of the algorithm

move, and if none of them improves the solution, then it resorts to the least bad one (or one that is best by a supporting criterion such as enhancing the diversity of solutions visited).

We include a memory structure in the algorithm outlined in Fig. 11 to create a short-term tabu search method. In particular, when we select a new vertex v and move it, we record in $tabu(v)$ the number of the current iteration (sweep), in order to prohibit moving it in subsequent sweeps. In this way, in a given iteration $iter$, we only permit a vertex u to be moved if:

$$iter - tabu(u) > tenure,$$

where *tenure* is a search parameter specifying the number of iterations that a *tabu* element cannot be selected. After this number of iterations, the tabu status of u changes to not-active, and u may be moved if it qualifies.

4.4 Overall method

Our algorithm has three phases. In the first one, we implement a **pre-processing** step to reduce the number of crossings, ignoring the alignment. Specifically, the method applies a local search based on insertions in which, at each iteration, the best move with respect to the number of crossings is selected in $moves(D, L_i)$. The move is only performed if it reduces arc crossings. The pre-processing finishes when crossings cannot be further minimized. Then, the main phase, consisting of the **short-term tabu search**, is applied. As described in the previous section, this phase gives priority to the alignment, handling the number of crossings as a secondary objective. Finally, the third phase implements a **post-processing** step to further reduce the number of crossings without deteriorating the alignment. To this end, this phase applies a local search based on switching between original nodes (i.e., it does not move the intermediate nodes in long arcs). The method performs sweeps across layers, performing in each one the best available exchange of vertices for the goal of reducing crossings. It finishes when no further crossing reduction is possible with node exchanges.

As will be shown in our experimentation, the three-phase method described above is able to solve medium size problems in about 1 or 2 s of CPU time. When extra time is available, we consider an extended algorithm in which we embed this method in a multi-start framework. In particular, we implement a **long-term memory tabu search** that applies iteratively the three-phase method from different initial solutions. These initial solutions are built for diversification purposes by inserting each vertex in its least frequently occupied position of the layer. To this end, during the entire search, we record in *freq* (i, p) the number of times that vertex i has been in position p, and then, we use this frequency value to build an initial solution in each global iteration. Frequency memory is one of the most commonly used types of memory in tabu search, both for intensification and diversification purposes. A more extensive discussion of different forms of frequency memory appears in Glover and Laguna (1993).

It must be noted that the flexibility introduced in the search by relaxing the alignment constraint may eventually produce final solutions with positive alignment (i.e., with some bends in the long arcs). However, we have empirically found that in this way, the tabu search method usually produces solutions with a small number of bends and a small number of crossings as well, and if we restrict the search to visit only solutions with no bends, it obtains solutions with a significantly larger number of crossings. In our view, a partial relaxation of the model, in terms of permitting a small number of bends, constitutes a good trade-off between the two criteria.

5 Computational experiments

In this section, we evaluate the performance of both the mathematical model and the tabu search algorithm. Additionally, some drawings will show the final appearance of the solution obtained with our method, thus illustrating in a graphical way its effectiveness. In particular, we generated 10 instances (hierarchical graphs) given the number of vertices n, number of arcs m, number of layers nl, and number of long arcs, m_l.

The hierarchical graphs were generated following the guidelines in the literature (Napoletano et al. 2019). The number of layers nl is an input to the graph generator. For each vertex u in layer L_i, an arc to a randomly chosen vertex v in layer L_{i+1} is included. This guarantees that all vertices in layers L_1 to L_{nl-1} have a degree of at least one. In addition, the generator checks that all vertices in the last layer have a degree of at least one. If a vertex in layer L_{nl} is found with a degree of zero, an arc is added to a randomly chosen vertex in layer L_{nl-1}. The generator then adds m_l long arcs. Additional arcs are added by randomly choosing two vertices in consecutive layers to meet the required density of 0.2 considered in previous works (see also Pastore et al., 2020).

5.1 Numerical results

Table 1 reports the parameter values for our 10 instances (number of vertices, n, number of arcs, m, number of layers, nl, and number of long arcs, m_l). We generated small and medium size instances, to be able to obtain the optimal solution, or good upper bounds, with CPLEX implementing our formulation.

In our first experiment, we compare the two mathematical models described in Sect. 3.1: the original model due to Jünger and Mutzel (1997) for crossing minimization, called Original model, and our adaptation that includes additional alignment constraints, called Aligned model. Both models are described in Sect. 3.1. We run CPLEX with a time limit of 1 h for each instance and each model. Within this time limit, it is able to obtain the optimal solution in the first 8 instances. However, in the last two instances in our benchmark, 9 and 10, with 144 and 140 elements respectively, it does not reach the optimal solution, and we are therefore considering the best lower bound found in each model. Table 2 reports the number of crossings, $C(D)$, obtained with each model, and the relative increment on the number of crossings of the aligned model with respect to the original model.

Results in Table 2 confirm that, as expected, the number of crossings usually increases when we add alignment constraints to the original model. As a matter of fact, this table shows an important variability in terms of the relative increment when using the aligned model. The extreme cases are instance 5, in which we obtain the same number of crossings with both models, equal to 6, and instance 10, in which the number of crossings increases in an 77.3% when the alignment constraints are included in the model (although as mentioned above, in this instance we are comparing lower bounds, not optimal solutions). We can therefore conclude that, since it depends on each particular instance, it is recommended to have both solutions, with and without alignment constraints, when solving this problem.

Table 1 Instances characteristics

Instance id	n	m	nl	m_l
1	36	51	6	7
2	49	69	7	6
3	48	72	6	9
4	64	110	8	10
5	50	62	10	10
6	50	65	5	10
7	120	154	10	17
8	120	181	12	21
9	144	192	12	19
10	140	239	10	25

Table 2 Number of crossings obtained with CPLEX within 1 h of CPU

Instance id	Original model $C(D)$	Aligned model $C(D)$	Rel. increment
1	27	30	11.1%
2	29	38	31.0%
3	30	34	13.3%
4	52	56	7.6%
5	6	6	0.0%
6	38	42	10.5%
7	52	69	32.7%
8	144	157	9.0%
9	107*	168*	57.0%
10	233*	413*	77.3%
Average	71.9	101.4	25.0%

*Not certified as optimal

We can draw another important conclusion from Table 2. We need to resort to metaheuristic methods to solve this problem, since CPLEX requires long running times even for the small and medium size instances considered in our benchmark. It is well documented (see for example North 1996), that graph drawing systems require algorithms that are able to generate graphs in a few seconds. The largest graphs in our benchmark have around 150 vertices, which can be considered of medium size, and CPLEX requires more than one hour of CPU time to solve them, making CPLEX impractical for graph drawing systems.

We consider two versions of our tabu search algorithm, the short term tabu search, to provide a solution in few seconds (in a fraction of a second for small problems), and the long term tabu search, to explore the solution space more thoroughly (requiring longer running times). As described in Sect. 4, the long-term tabu search approach implements a frequency-based multi-start method, applying multiple times the short-term tabu search from different starting solutions. Table 3 reports the solutions obtained with the short term tabu search, reporting the number of crossings, Tabu $C(D)$, the alignment value, tabu $AL(D)$, and the running time, CPU Time. This table also reports the number of crossings obtained with the CPLEX solution, CPLEX $C(D)$.

Table 3 shows that the short-term tabu search is able to obtain good solutions in extremely short CPU times (less than 1 s on average). The number of crossings is however around 30% larger on average than the optimal solution obtained with CPLEX. Instances 5 and 9 are especially difficult to solve by our short term approach since it obtains much larger numbers of crossings than CPLEX, and an alignment value $AL(D)$ relatively far from 0 (which would correspond to a perfect alignment of long arcs). On the other hand, in many cases, the short term tabu search is able to obtain a solution with $AL(D) = 0$, indicating the effectiveness of the flexible strategy that permits it to visit solutions with positive alignment values. Especially interesting is the result of the largest example in our test bed, number 10 with 25 long arcs. In this example, CPLEX was not able to obtain the optimal

Table 3 Solutions obtained with the Short Term Tabu Search

Instance	CPLEX $C(D)$	Tabu $C(D)$	Tabu $AL(D)$	CPU time
1	30	42	0	0.03
2	38	71	0	0.08
3	34	45	0	0.09
4	56	80	0	0.18
5	6	26	9	0.02
6	42	60	0	0.14
7	69	175	5	1.31
8	157	313	9	1.14
9	168	319	17	2.39
10	413	394	4	4.49
Average	101.3	152.5	4.4	0.99

solution in 1 h of CPU time, and the short-term tabu search obtains a better solution than CPLEX in only 4.49 s of CPU time.

Table 4 shows the results obtained with the long-term tabu search, reporting the same statistics shown in Table 3 but now with the more complete version of our method which takes more time to execute, as reflected in the CPU Time column in this table.

As expected, the results obtained with the long-term tabu search improve those of the short term version, although they come with larger running times. In particular, in all the instances, with the exception of number 9, our metaheuristic is able to obtain a solution with all the long arcs aligned ($AL(D) = 0$), and the number of crossings is on average $C(D) = 140.6$. They are larger than those obtained with CPLEX (on average $C(D) = 101.3$), but running times are moderate (19.73 s on average), while CPLEX requires running times close to 1 h. It indicates that tabu search is a suitable method for graph drawing systems.

Table 4 Solutions obtained with the long term Tabu search

Instance	Cplex $C(D)$	Tabu $C(D)$	Tabu $AL(D)$	CPU time
1	30	30	0	0.55
2	38	47	0	1.56
3	34	36	0	2.27
4	56	69	0	4.73
5	6	8	0	0.50
6	42	44	0	2.89
7	69	158	0	24.67
8	157	259	0	24.39
9	168	283	2	42.34
10	413	472	0	93.39
Average	101.3	140.6	0.2	19.73

5.2 Drawings

The conventional wisdom that a picture is worth a thousand words applies here to evaluate our proposals. In this subsection we show some drawings obtained with both CPLEX and the tabu search methods, to illustrate their quality.

Figure 12 shows the solution of our mathematical formulation to minimize the number of crossings on instance 4 with the model to align long arcs. As shown in Table 2, it has $C(D) = 56$ arc crossings, and all long arcs aligned ($AL(D) = 0$).

As shown in the numerical experiments above, the CPLEX method is too time-consuming and we find it valuable to make use of our metaheuristic approaches to solve this problem in graph drawing systems. Figures 13 and 14 show the drawings of two instances in our testbed, namely 4 and 5, obtained with our tabu search method in short running times.

Although this admittedly involves a subjective evaluation, we believe that Fig. 13, obtained with tabu search in 4.7 s of CPU time, is very similar to Fig. 12, obtained with CPLEX in about 1 h of CPU time, in terms of their readability (i.e., how easy is to capture information from them). As a matter of fact, both drawings have all long

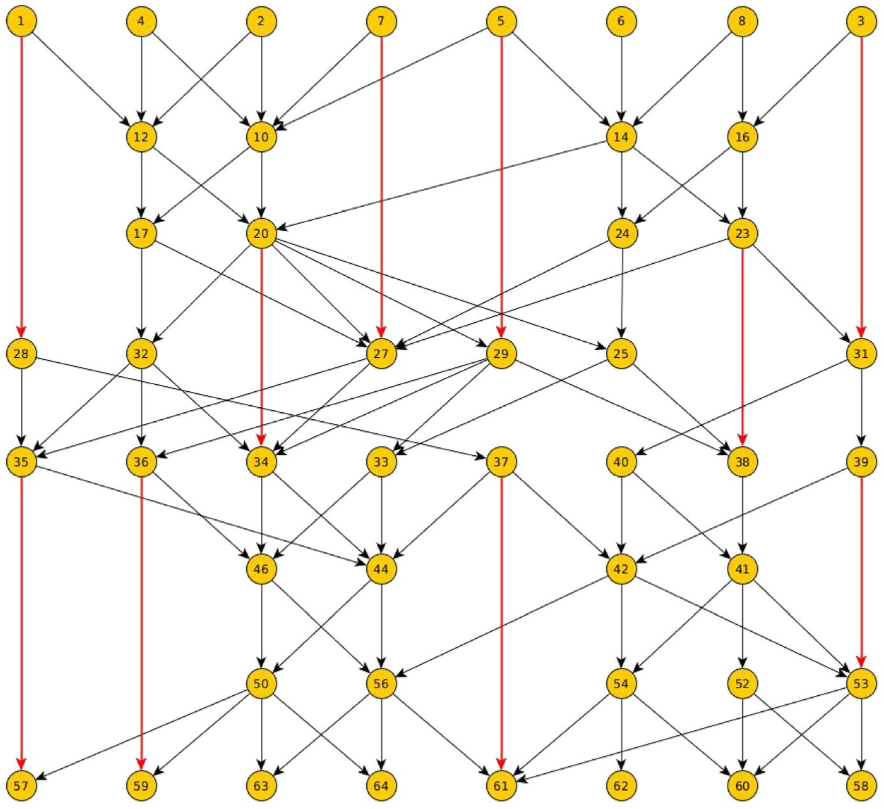

Fig. 12 Cplex solution of instance 4

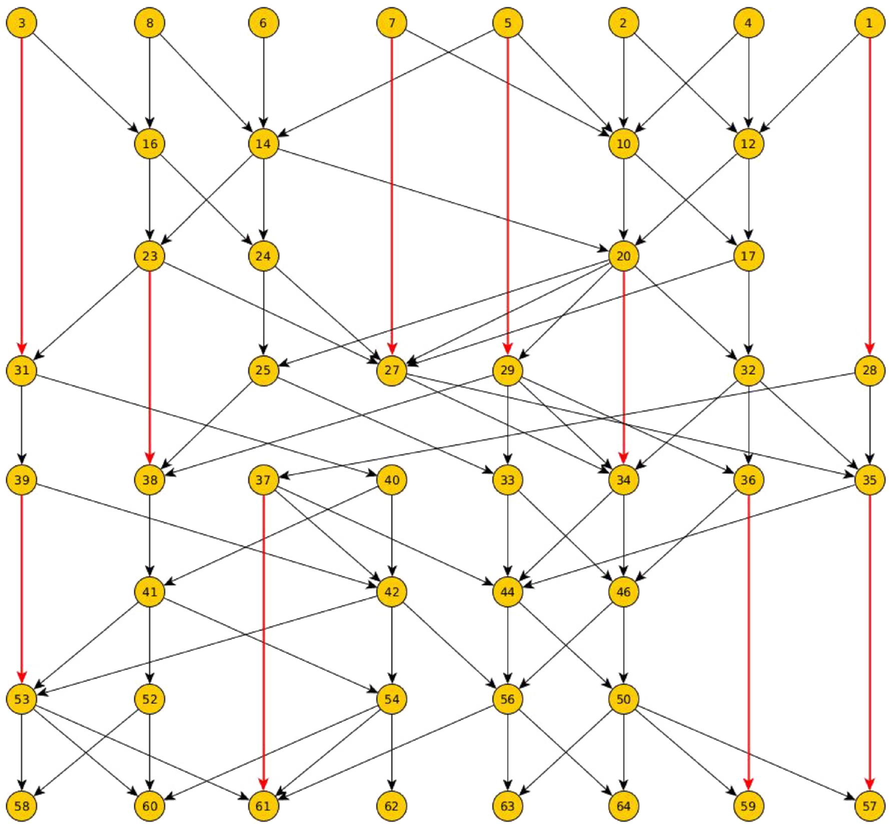

Fig. 13 Tabu search solution of instance 4

arcs aligned, and the optimal one in Fig. 12 has 56 arc crossings, while the heuristic one in Fig. 13 has 69 crossings.

In our opinion, these drawings confirm that tabu search is able to obtain readable drawings automatically (without any user intervention) in very short running times, as required by software tools.

5.3 Two objectives discussion

As noted, we elect to minimize the number of crossings, $C(D)$, subject to seeking an alignment of long arcs, $AL(D)$, that is close to 0 (a perfect alignment). In other words, we implement this as a soft constraint, considering solutions where $AL(D)$ is relatively low without compelling it to be 0. This opens the door to considering $AL(D)$ as a second objective, and therefore approaching this problem within a multi-objective optimization framework that simultaneously optimizes $C(D)$ and $AL(D)$. This can be the topic for future research, but we believe that keeping $AL(D)$ as a soft

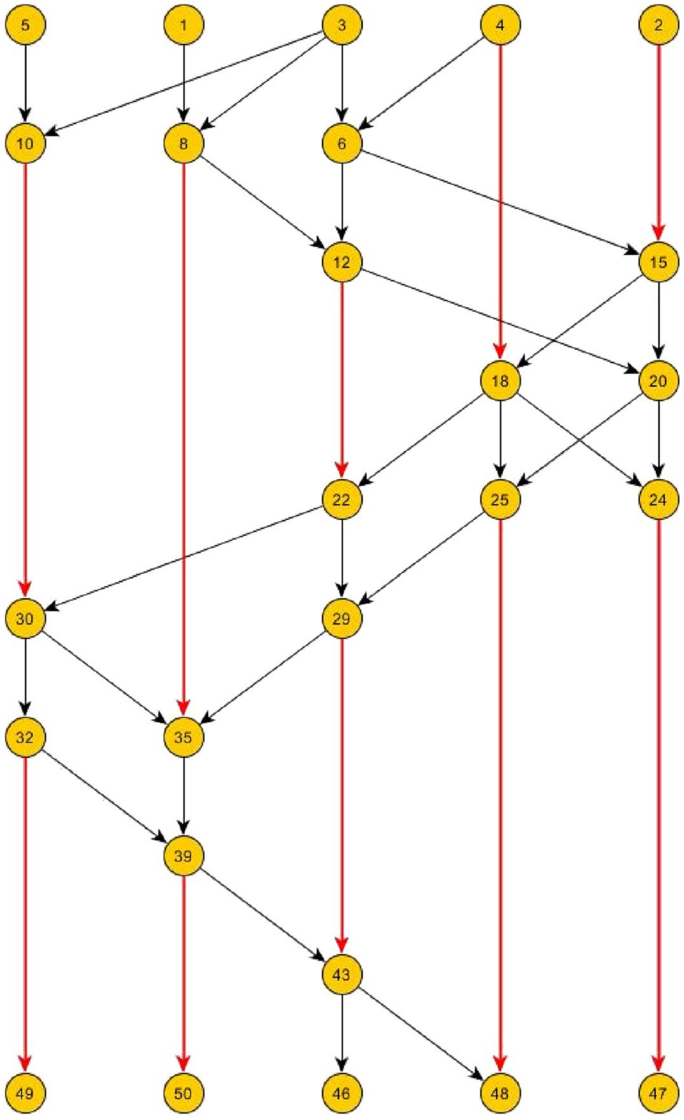

Fig. 14 Tabu search solution of instance 5

constraint provides a good trade-off between the well-established criterion in graph drawing of crossing minimization, and our proposal of favoring long arcs alignment.

Our short-term tabu search applies three phases to handle these two criteria. Figure 15 illustrates the performance of the method in Instance 1 in terms of these two values, $C(D)$ and $AL(D)$. This figure shows that the initial solution has $C(D) = 95$ crossings and $AL(D) = 34$. In the first phase, called pre-processing, a local search method reduces the number of crossings, ignoring the alignment value. Specifically,

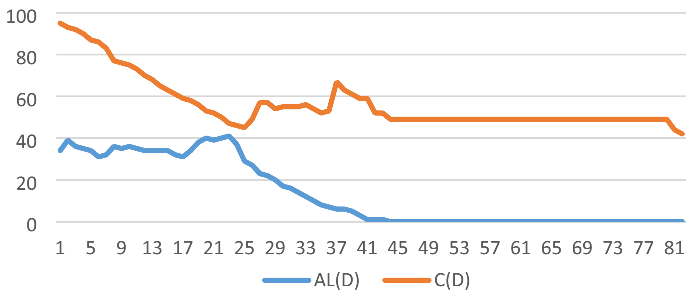

Fig. 15 Search profile in instance #1

from iteration 1 to 25 (depicted in the x-axis), the number of crossings drops from 95 to 45, and the alignment value slightly oscillates in the interval [30, 40]. Then, the tabu search phase is applied, minimizing the alignment value $AL(D)$ from 40 to 0. The tabu search is able to reduce this value while keeping the number of crossings relatively low (in the interval [50, 70]). In the next iterations, from 45 to 80, the tabu search is not able to improve any of the two values, and it ends. Finally, the local search post-processing based on exchanges of the original vertices is able to further reduce the number of crossings to $C(D) = 42$ keeping $AL(D) = 0$. This figure is representative of the method performance, as confirmed by Fig. 16, where the search profile of Instance 2 exhibits a similar pattern.

6 Conclusions

This paper has a twofold objective. In one hand, we provide a short tutorial on a metaheuristic methodology, tabu search, and on the other hand we propose a new model for hierarchical graph drawing. Our goal has been to show how tabu search provides a robust and effective alternative to target optimization problems in the context of graph drawing and, in general, in graphical computer science.

To amplify the relevance of these connections, we observe that the overarching principle of tabu search is to identify strategies for exploiting adaptive memory,

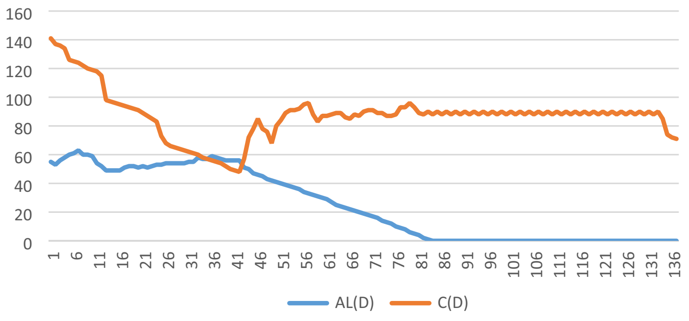

Fig. 16 Search profile in instance #2

rather than following the policy of relegating decisions to random choices, as often is fashionable in a variety of other metaheuristic methods. The highly attractive results provided by the adaptive memory strategies underlying tabu search have had an evident impact on the design of metaheuristic methods in general and are giving rise to extensions that are proving valuable across a wide range of applications.

Adaptive memory in TS has the dual purpose of exploiting *structure* and *behavior.* As we have emphasized, this memory is linked to differentiating various attributes of search processes that may be broadly classified as *solution attributes* and *move attributes.* While solution attributes take precedence for most purposes, move attributes, which concern the differentiation and exploitation of neighborhoods, lead to multi-neighborhood search strategies as in Glover et al. (1984), Xu et al. (1996, 1997a), Yagiura et al. (2004), Wu, et al. (2013) that may be compared and contrasted to those used in Variable Neighborhood Search (VNS). (See, e.g., Sánchez-Oro et al. (2017).) The connection between attributes and frequency and recency memory also underlies the emphasis on probabilistic choices in TS in contrast to random choices, defining probabilities in relation to frequency and quality as in Lokketangen and Glover (1996), Xu et al. (1997b) and Guermi et al. (2019).

It is not possible within the brief space of this tutorial to fully cover these aspects of tabu search, and for readers who want to explore TS adaptive memory strategies more deeply, a selection of "Additional Suggested References" is included at the end of the References section. These references are divided into subsections consisting of *Fundamental Papers* and *A Sampling of Studies Establishing New Records*, together with links for downloading papers of interest. Both groups, but particularly the Fundamental Papers, include proposals that have yet to be examined in detail, and therefore provide a source for future research: Glover et al. (2018a), Shang et al. (2021), Glover (1997), Glover and Laguna (2013), Barr et al. (2020), Glover et al. (2018b), Lai et al. (2016), Lai et al. (2018a), Lai et al. (2019), Lai et al. (2020), Lai et al. (2018b), Wang et al. (2014), Yagiura et al. (2006), and Yagiura et al. (2007).

Acknowledgements This research has been partially supported by the Spanish Government, *Ministerio de Ciencia, Innovación y Universidades*, with Grant Ref. PGC2018-0953322-B-C21/MCIU/AEI/FEDER-UE.

References

Brandes U, Köpf BA (2002) Fast and simple horizontal coordinate assignment. In: Mutzel P, Jünger M, Leipert S (eds) Graph drawing. Lecture Notes in Computer Science. Springer, Berlin, vol 2265, pp 31–44

Carpano M (1980) Automatic display of hierarchized graphs for computer-aided decision analysis. IEEE Trans Syst Man Cybern 10(11):705–715

Di Battista G, Eades P, Tamassia R, Tollis I (1998) Graph drawing: algorithms for the visualization of graphs, 1st edn. Prentice Hall PTR, Upper Saddle River

Glover F (1963) Parametric combinations of local job shop rules. Chapter IV, ONR Research Memorandum No. 117, GSIA, Carnegie-Mellon University, Pittsburgh

Glover F (1986) Future paths for integer programming and links to artificial intelligence. Comput Oper Res 13(533):549

Glover F (1989) Tabu search, Part I. ORSA J Comput 1(3):190–206

Glover F (1990) Tabu search, Part II. ORSA J Comput 2(1):4–32

Glover F, Laguna M (1993) Tabu search. In: Reeves C (ed) Chapter in modern heuristic techniques for combinatorial problems. Blackwell Scientific Publishing, Oxford, pp 71–140

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

Glover F, Glover R, McMillan C (1984) A heuristic programming approach to the employee scheduling problem and some thoughts on managerial robots. J Oper Manag 4(2):113–128

Glover F, Hanafi S, Guermi O, Crevits I (2018a) A simple multi-wave algorithm for the uncapacitated facility location problem. Front Eng Manag 5:451–465

Grötschel M, Michael Jünger J, Reinelt G (1984) A cutting plane algorithm for the linear ordering problem. Oper Res 32(6):1195–1384

Guermi O, Nduwayoa P, Todosijevic R, Hanafi S, Glover F (2019) Probabilistic Tabu search for the cross-docking assignment problem. Eur J Oper Res 277(3):875–885

Jünger M, Lee EK, Mutzel P, Odenthal T (1997) A polyhedral approach to the multi-layer crossing minimization problem. In: International symposium on graph drawing. Springer, pp 13–24

Jünger M, Mutzel P (1997) 2-layer straightline crossing minimization: performance of exact and heuristic algorithms. J Gr Algorithms Appl. https://doi.org/10.1142/9789812777638_0001

Kaufmann M, Wagner D (2001) Drawing graphs: methods and models. Lecture Notes in Computer Science. Springer, Berlin

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

Løkketangen A, Glover F (1996) Probabilistic move selection in Tabu search for zero-one mixed integer programming problems. In: Osman IH, Kelly JP (eds) Meta-heuristics: theory & applications. Springer, Berlin, pp 467–487

Martí R (1998) A tabu search algorithm for the bipartite drawing problem. Eur J Oper Res 106:558–569

Martí R, Reinelt G (2011) The linear ordering problem. Exact and heuristic methods in combinatorial optimization. Springer, Heidelberg

Martí R, Campos V, Hoff A, Peiró J (2018) Heuristics for the min-max arc crossing problem in graphs. Expert Syst Appl 109:100–113

Napoletano A, Martínez-Gavara A, Festa P, Pastore T, Martí R (2019) Tabu search for min-max edge crossings in graphs. Eur J Oper Res 274:710–729

North SC (1996) Incremental layout in DynaDAG. In: Proceedings of Graph Drawing’95. Lecture Notes in Computer Science, vol 1027, pp 409–418. Springer, Berlin

Pastore T, Martínez-Gavara A, Napoletano A, Festa P, Martí R (2020) Heuristics for the constrained incremental graph drawing problem. Comput Oper Res 114:104830

Sánchez-Oro J, Martínez-Gavara A, Laguna M, Martí R, Duarte A (2017) Variable neighborhood scatter search for the incremental graph drawing problem. Comput Optim Appl 68:775–797

Shang Z, Hao J-K, Zhao S, Wang Y (2021) Multi-wave tabu search for the boolean quadratic programming problem with generalized upper bound constraints. In: IFORS 2021, the 22nd conference of the International Federation of Operational Research Societies

Sugiyama K, Tagawa S, Toda M (1981) Methods for visual understanding of hierarchical system structures. IEEE Trans Syst Man Cybern SMC 11(2):109–125

Wu Q, Hao J-K, Glover F (2013) Multi-neighborhood tabu search for the maximum weight clique problem. Ann Oper Res 196(1):611–634

Xu J, Chiu S, Glover F (1996) Using Tabu search to solve the Steiner tree-star problem in telecommunications network design. Telecommunications Systems 6:117–125

Xu J, Chiu S, Glover F (1997a) Tabu search for dynamic routing communications network design. Telecommun Syst 8:1–23

Xu J, Chiu S, Glover F (1997b) Probabilistic Tabu search for telecommunications network design. Comb Optim: Theory Pract 1(1):69–94

Yagiura M, Iwasaki S, Ibaraki T, Glover F (2004) A very large-scale neighborhood search algorithm for the multi-resource generalized assignment problem. Discret Optim 1:87–98

Additional selected references: fundamental papers

Glover F (1997) Tabu search and adaptive memory programming—advances, applications and challenges. In: Barr RS, Helgason RV, Kennington JL (eds) Interfaces in computer science and operations research. Kluwer Academic Publishers, Boston, pp 1–75

Glover F, Laguna M (2013) Tabu Search in Analytics and Computational Science. In: Pardalos PM, Du D-Z, Graham RL (eds) Handbook of Combinatorial Optimization, vol XXI, 2nd edn. Kluwer Academic Publishers, Norwell, pp 3261–3362

A sampling of studies establishing new records

Barr R, Glover F, Huskinson T, Kochenberger G (2020) An extreme-point Tabu-search algorithm for fixed charge network problems. Netw Int J: Spec Issue Celebr 50 Years Netw: Part 2 77(2):322–340
Glover F, Hanafi S, Guermi O, Crevits I (2018b) A simple multi-wave algorithm for the uncapacitated facility location problem. Front Eng Manag 5(4):451–465
Lai X, Hao J-K, Lü Z, Glover F (2016) A learning-based path relinking algorithm for the bandwidth coloring problem. Eng Appl Artif Intell 52:81–91
Lai X, Hao J-K, Glover F, Lü Z (2018a) A two-phase tabu-evolutionary algorithm for the 0–1 multidimensional knapsack problem. Inf Sci 436–437:282–301
Lai X, Hao J-K, Glover F, Yue D (2019) Intensification-driven tabu search for the minimum differential dispersion problem. Knowl-Based Syst 167(1):168–186
Lai X, Hao J-K, Glover F (2020) A study of two evolutionary/tabu search approaches for the generalized max-mean dispersion problem. Expert Syst Appl 139:112856
Lai X, Yuea D, Hao J-K, Glover F (2018b) Solution-based tabu search for the maximum min-sum dispersion problem. Inf Sci 441:79–94
Wang Y, Hao J-K, Glover F, Lu Z (2014) A Tabu search based memetic algorithm for the maximum diversity problem. Eng Appl Artif Intell 27:103–114
Yagiura M, Ibaraki T, Glover F (2006) A path relinking approach with ejection chains for the generalized assignment problem. Eur J Oper Res 169:548–569
Yagiura M, Komiya A, Kojima K, Nonobe K, Nagamochi H, Ibaraki T, Glover F (2007) A path relinking approach for the multi-resource generalized quadratic assignment problem. Lecture Notes in Computer Science. Springer, Berlin, vol 4638, pp 121–135