Comput Optim Appl (2017) 68:775–797
DOI 10.1007/s10589-017-9926-5

Variable neighborhood scatter search for the incremental graph drawing problem

Jesús Sánchez-Oro[1] · Anna Martínez-Gavara[2] · Manuel Laguna[3] · Rafael Martí[2] · Abraham Duarte[1]

Received: 19 October 2016 / Published online: 27 July 2017

Abstract Automated graph-drawing systems utilize procedures to place vertices and arcs in order to produce graphs with desired properties. Incremental or dynamic procedures are those that preserve key characteristics when updating an existing drawing. These methods are particularly useful in areas such as planning and logistics, where updates are frequent. We propose a procedure based on the scatter search methodology that is adapted to the incremental drawing problem in hierarchical graphs. These drawings can be used to represent any acyclic graph. Comprehensive computational experiments are used to test the efficiency and effectiveness of the proposed procedure.

Keywords Graph drawing · Scatter search · Incremental graph · Dynamic graph drawing · Metaheuristics

✉ Rafael Martí
rafael.marti@uv.es

Jesús Sánchez-Oro
jesus.sanchezoro@urjc.es

Anna Martínez-Gavara
gavara@uv.es

Manuel Laguna
laguna@colorado.edu

Abraham Duarte
abraham.duarte@urjc.es

1 Departamento de Ciencias de la Computación, Universidad Rey Juan Carlos, Madrid, Spain

2 Departamento de Estadística e Investigación Operativa, Universidad de Valencia, Valencia, Spain

3 Leeds School of Business, University of Colorado Boulder, Boulder, CO, USA

1 Introduction

Most complex information systems include visual representations for interpretation and analysis. Graphs have become a fundamental modeling tool in areas such as project management, production planning, line balancing, and data and software visualization. Graph drawing is an established area of research that includes many articles and books [4,11]. As a matter of fact, the Information Visualization community counts with numerous conferences, academic events, and software companies.[1] A key element within graph-drawing relates to the criteria used to judge the quality of a drawing. Arc-crossing minimization is considered one of the most common ways of creating good graphs [3,20,21]. We adopted this criterion in the search procedures that we developed in this work. Figure 1a and b show two representations of the same graph. Using the arc-crossing minimization criterion, Fig. 1b may be considered better than Fig. 1a. Figure 1b is clearer and enables an easier interpretation and extraction of information.

We focus on hierarchical directed acyclic graphs (HDAG) which are also known as layered graphs. The HDAG representation is done by arranging the vertices on a series of equidistant vertical lines called layers in such a way that all arcs point in the same direction. The problem of minimizing the number of arc-crossings between any two layers is $\mathcal{NP}$-Complete, even when the graph consists of only two layers [7]. Figure 2 shows the representation of a graph with 10 vertices and three layers (L_1 to L_3), where the arc directions are implicitly assumed to go from left to right between each pair of layers.

Working with HDAGs is not a limitation, since there exists a number of procedures to transform a directed acyclic graph (DAG) into a HDAG [17]. A standard transformation method, introduced by Sugiyama et al. [23], consists of arranging the vertices in layers so that all the arcs point in the same direction. Some arcs, however, might connect vertices that are in layers that are not contiguous. Then, artificial vertices are added in order to create a graph where arcs exist only within contiguous layers (i.e., a hierarchical graph). Even though the procedures that we developed are for HDAs, they also apply to DAGs thanks to these transformations.

Figure 3 illustrates the transformation from DAG to HDAG following the so-called Sugiyama's et al. method [23]. The DAG is represented in Fig. 3a. In the first step, the vertices are arranged in layers so that all the arcs go from one layer to another and there are no arcs connecting vertices in the same layer. This results in the graph in Fig. 3b. This graph contains two arcs (see dotted lines) that go from the first layer to the third, skipping over the second layer. These are arcs (3, 1) and (3, 6). To avoid these arcs, we create two artificial (or fictitious) vertices F_1 and F_2 and add them to layer 2. Finally, the new vertices are used to connect vertex 1 to 3 through F_1 and 3 to 6 through F_2. Figure 3c shows the HDAG obtained through the transformation of the DAG in Fig. 3a.

Given a HDAG and a list of new vertices and arcs, the incremental graph drawing problem consists of adding the new vertices and arcs to the HDAG without altering the relative position of the vertices in the existing HDAG while minimizing the number

[1] See http://www.infovis-wiki.net for resources on visual analytics.

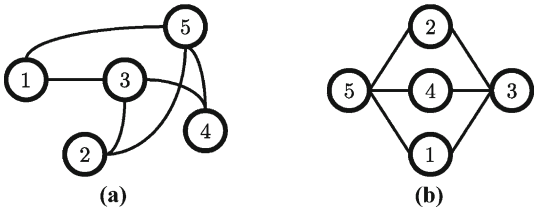

Fig. 1 Two alternative representations of the same graph

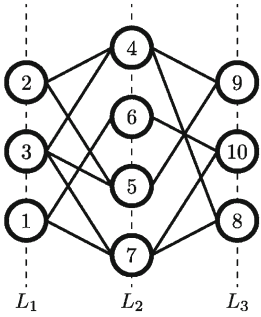

Fig. 2 Multilayered hierarchical directed acyclic graphs (HDAG)

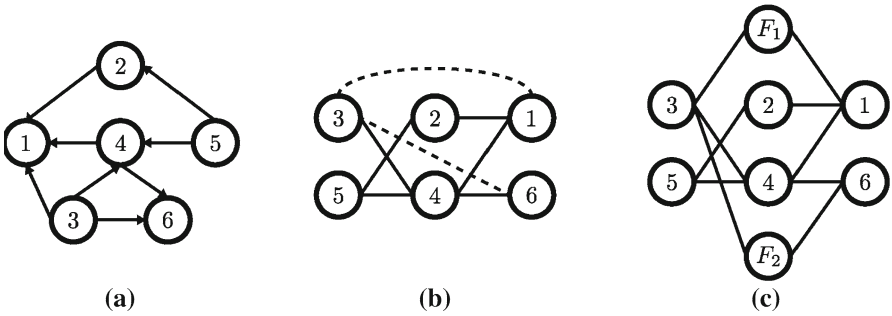

Fig. 3 Transformation from directed acyclic grap (DAG) to HDAG

of arc crossings. Figure 4 illustrates the process of an incremental graph drawing. The original HDAG is shown in Fig. 4a. The new arcs are (2, 10), (3, 11), (5, 11), (8, 4), (8, 5), (9, 10), and (10, 7). Figure 4b shows the HDAG with the new arcs, where the number of arc crossings is 14. An optimized drawing of the new HDAG, with a minimum number of arc crossing of 9, is shown in Fig. 4c. Note that the optimized drawing preserves the relative ordering of the original vertices in each layer.

Although the general problem of minimizing arc crossing on HDAGs has been studied extensively, the incremental graph-drawing version has received very little attention. In fact, we are only aware of one article that is limited to bipartite graphs [15]. The article describes a branch-and-bound procedure that is tested on relatively small graphs and a metaheuristic based on GRASP (greedy randomized adaptive search procedure) [6] applied to medium- and large-size instances.

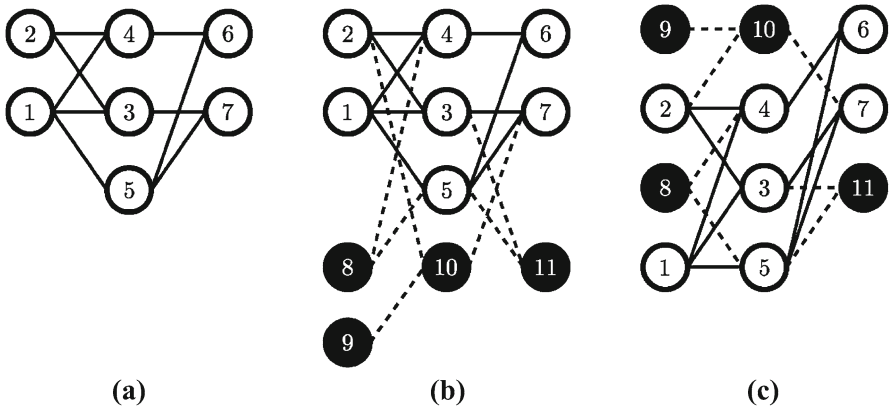

Fig. 4 Illustration of the incremental graph-drawing process

Our first goal was to extend the application of GRASP to multi-layered HDAGs. This gave us a base case for finding solutions to the general incremental graph-drawing problem. We then approached the problem from a different angle by adapting scatter search. Our computational testing includes comparisons of the performance of these two approaches.

The main contributions of this work are: (1) Implementation and testing of a procedure that integrates variable neighborhood [16] and scatter search [13], (2) Adaptation of an existing procedure for the bi-partite incremental graph-drawing problem to the multilayered case, and (3) Improvement of the state of the art in solving the incremental graph-drawing problem.

2 Notation and definitions

A hierarchical graph $H = (V, E, k, L)$ is defined as a graph $G = (V, E)$, where V and E represent the set of vertices and arcs, respectively, and the function $L(v) : V \rightarrow \{1, 2, \ldots, k\}$ indicates the index of the layer where v resides. Hence, $L(v) - L(u) = 1 \quad \forall (u, v) \in E$. The L function implicitly defines the sets of vertices $L_i = \{v \in V : L(v) = i\}$ for $i = 1, 2, \ldots, k$ which we refer to as layers. The set of vertices V is the union of all the layers, i.e., $V = \bigcup_{i=1}^{k} L_i$. Since the arcs in a HDAG are straight lines that join the vertices in two contiguous layers, a drawing of a HDAG is given by the ordering of the vertices in each layer. Therefore, a drawing of H is defined as $D = (H, \Phi)$, where $\Phi = \{\varphi_1, \varphi_2, \ldots, \varphi_k\}$ and φ_i is the ordering (permutation) of the vertices in layer L_i. That is, $\varphi_i(j)$ is the vertex in position j in layer L_i. The position of vertex v is defined as $\pi(v)$ in such a way that if $v = \varphi_i(j)$ then $\pi(v) = j$.

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 $(u'v')$, where $u, u' \in L_i$ and $v, v' \in L_{i+1}$ when:

$$(\pi(u) < \pi(u') \wedge \pi(v) > \pi(v')) \mathrm{V} (\pi(u) > \pi(u') \wedge \pi(v) < \pi(v'))$$

For vertices $u, u' \in L_i$ 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,

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

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

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

For more than two decades, the barycenter has been the standard method to minimize arc crossings in a HDAG. This iterative method, when sweeping H from left to right, fixes the ordering φ_{i-1} while rearranging the ordering of the vertices in L_i by applying the barycenter calculation. This consists of calculating the position of $u \in L_i$ as the average position of the vertices in L_{i-1} that are adjacent to u. That is, the barycenter bc for vertex $u \in L_i$, when sweeping H from left to right is calculated as:

$$bc(u, i-1) = \frac{\Sigma_{v \in \Lambda_{i-1}(u)} \pi(v)}{|\Lambda_{i-1}(u)|}.$$

With a single sweep of H from left to right, the barycenter calculations produce an ordering of the vertices in each layer. The procedure is then repeated from right to left. In this case, the position for $u \in L_i$ is calculated as the average position of the vertices in L_{i+1} that are adjacent to u. That is, the barycenter calculation when sweeping from right to left is $bc(u, i+1)$. The process ends when the positions of the vertices do not change after a sweep.

From an original hierarchical graph $H = (V, E, k, L)$, we define an incremental graph $IH = (IV, IE, k, L)$ that results from adding a set of vertices $\hat{V}$, with their corresponding arcs, without altering the number of layers k. The incremental graph is such that $IV = V \cup \hat{V}$, $E \subseteq IE$ and $L\left(v\right) : IV \rightarrow \{1, 2, \ldots, k\}$, with the value of $L\left(v\right)$ not changing for $v \in V$. Given a graph IH, originating from a graph H, and a drawing D of H, the incremental graph-drawing problem consists of finding a drawing $ID = (IH, \Phi)$ that minimizes the number of arc crossings while keeping the same relative ordering of V in D. This means that if $\pi(u) < \pi\left(u'\right)$ in H then the same should be true in IH for all u and u' such that $L\left(u\right) = L\left(u'\right)$.

We also define a partial incremental drawing ID_t as one with a set of vertices IV_t containing all vertices in V and t of the vertices in $\hat{V}$. Therefore, a partial incremental drawing with $t = 0$ is the original drawing, i.e., $ID_0 = D$. And, a partial drawing with

$t = \left|\hat{V}\right|$ is a completed incremental drawing, i.e., $ID_{\left|\hat{V}\right|} = ID$. Table 1 summarizes all the symbols and definitions introduced above.

3 Extension of GRASP to multilayered graphs

As mentioned in Pinaud et al. [19], "in comparison with static drawings, the literature on dynamic drawings is still in its infancy". Early works [1] introduced the concept of stability across consecutive drawings in a dynamical process. North [18] presented a graph drawing system to achieve this stability. Branke [2] adapted Sugiyama's heuristic to include the stability conditions in the drawing method to preserve the user's mental map. In all these cases, the proposed methods try to obtain a drawing similar to an existing drawing based on a distance function. For example, Pinaud et al. [19] consider the number of pairs of vertices that are inverted in the new drawing with respect to their relative position in the previous drawing. They are in essence multi-objective approaches which optimize both the aesthetic criteria and the stability distance function.

A different approach was proposed in Martí and Estruch [15], in which the relative position between vertices in the original drawing is considered as a constraint to create a new drawing. These authors did not consider a measure or distance between drawings but instead, they included the constraints induced by the previous ordering of vertices. They proposed a greedy randomized adaptive search procedure (GRASP) to solve this incremental graph drawing problem. Their implementation is limited to bipartite graphs. In this paper, we consider their approach to tackle the dynamic or incremental graph drawing problem. We extended this GRASP with the goal of finding baseline benchmarks for the multilayered incremental graph-drawing problem. GRASP consists of two phases: construction and improvement. The construction is an iterative process in which a solution is built by selecting one element at a time. The improvement typically consists of a local search.

3.1 Construction

The process starts with the random selection of a vertex from all those with maximal degree. The vertex is placed in a random position within its layer. In subsequent steps, the candidate list (CL) of vertices to be added to the drawing consists of all the unassigned vertices. The degree $\rho(v)$ of each unassigned vertex v is calculated with respect to the partial drawing. The next vertex to be added to the drawing is selected from a restricted candidate list RCL that contains all the unassigned vertices whose degree is at least as high as $\alpha\%$ of the maximal degree $\rho_{max} = \max_{v \in CL} \rho(v)$. That is, $RCL = \{v \in CL : \rho(v) \geq \alpha\rho_{max}\}$. Let v^* be the selected vertex.

Then, v^* is placed in the position prescribed by $bc\,(v^*, L\,(v^*) + 1)$, except when $L(v^*) = k$, in which case the barycenter is calculated as $bc\,(v^*, k - 1)$. We simplify the notation to refer to the barycenter of a vertex v as $bc\,(v)$, with the understanding that the actual calculation depends on whether $L\,(v)$ is equal to k or not. If $v^* \in V$, then the vertex is placed in the feasible position that is closest to its barycen-

Table 1 Definitions and symbols

Symbol	Definition
H	Hierarchical graph: $H = (V, E, k, L)$
G	Graph: $G = (V, E)$
V	Set of vertices of G
E	Set of arcs of G
k	Number of layers
IH	Incremental graph: $IH = (IV, IE, k, L)$
$\hat{V}$	Set of new vertices
IV	Set of vertices in the incremental graph : $IH : IV = V \cup \hat{V}$
IE	Set of arcs in the incremental graph IH
ID	Incremental graph drawing: $ID = (IH, \Phi)$
ID_t	Partial incremental graph drawing
L	Function that indicates the index of the layer where a vertex v resides
L_i	Set of vertices in layer i : $L_i = \{v \in IV : L(v) = i\}$
D	Drawing: $D = (H, \Phi)$
Φ	Set of the permutation: $\Phi - \{\varphi_1, ...\varphi_k\}$ in D
φ_i	Permutation of the vertices in layer L_i : $\varphi_i(j)$ is the vertex in position j in layer L_i
π	Position of vertex v. If $v = \varphi_i(j)$ then $\pi(v) = j$
$C_{i+1}(u, u')$	Number of arcs crossings between layers L_i and L_{i+1} incidents with u and u'
$\Lambda(u)$	Set of all vertices adjacent to u: $\Lambda = \{v \in IV : (u, v) \in IE\}$
$\Lambda_{i+1}(u)$	Set of vertices in L_{i+1} that are adjacent to u: $\Lambda_{i+1} = \Lambda \cap L_{i+1}$
$C(D)$	Total number of crossings in a drawing D
$\rho(v)$	Degree of (assigned or unassigned) vertex v

ter $bc(v^*)$. Otherwise, it is placed in either $\lfloor bc(v^*) \rfloor$. or $\lceil bc(v^*) \rceil$, whichever is better. If both positions are taken, then v is placed as close as possible. Once v^*. is placed, it is eliminated from CL and the degree values $\rho(v)$. for $v \in CL$ are updated.

3.2 Improvement

The neighborhood search during this phase is based on a probabilistic selection without replacement. All vertices are considered for repositioning but the order in which they are considered is probabilistically determined by. the vertex degrees. The probability that a vertex v is selected is:

$$Pr(v)\frac{\rho(v)}{\Sigma_{u \in IV}\rho(u)}$$

Hence, the higher the degree, the higher the probability of being selected. Let v^* be the selected vertex. Then, three moves are considered:

1. Place v^* in $\lfloor bc\,(v^*) \rfloor - 1$.
2. Plac v^* in $\lfloor bc\,(v^*) \rfloor$ or $\lceil bc\,(v^*) \rceil$.
3. Place v^* in $\lfloor bc\,(v^*) \rfloor + 1$.

If $v^* \in V$, then the repositioning of v^* must be to a feasible position. Feasibility is checked with respect to the relative position of v^* and all v for which $L\,(v^*) = L\,(v)$. The feasible positions are calculated in reference to the closest positions specified by the three moves above. Only improvement moves are executed. If after the exploration of the entire neighborhood, that is, if after all vertices $v \in IV$ are considered, no improvement move is found, the improvement phase terminates.

4 Scatter search

Scatter search (SS) is a so-called population-based metaheuristic that consists of five elements [13]: diversification generation, improvement, reference set update, subset generation, and combination. Scatter search maintains a reference set of solutions (*RefSet*) of size b. The diversification generator is executed first to create a set of solutions P and this is followed by the improvement method. The resulting improved set of solutions P^* is used to initialize the *RefSet* by selecting the best $b/2$ in P^* and then the most diverse $b/2$ from the remaining solutions in P^*. Quality is evaluated with the objective function, while diversity is measured against the reference solutions. In particular, we use a distance measure between two solutions that counts the number of vertices that occupy a different position. The distance between a candidate solution and a set of reference solutions is given by the minimum distance between the candidate solution and all of the reference solutions in the set. The subset generation consists of all pairs of reference solutions that have not been examined in previous iterations. Combinations are generated from each pair of reference solutions. The trial solutions are subjected to the improvement procedure. The reference set is then updated with the best solutions from the existing reference set and all the improved solutions generated by the combinations. The search terminates when no new solutions are admitted to the reference set, that is, when the *RefSet* does not change in the current iteration. Algorithm 1 summarizes the SS steps.

Of the five SS elements, we have implemented two in their standard form. The *RefSet* is updated by solution quality and the subset generation is limited to all pairs for which at least one of the reference solution in the pair is new to the current iteration. The implementation of the remaining three elements is described below.

$P \leftarrow$ Diversification generation
$P^* \leftarrow$ Improvement(P)
$RefSet \leftarrow$ Reference set update(P^*)
do
  $Pairs \leftarrow$ Subset generation($RefSet$)
  $TrialSolutions \leftarrow$ Combinations($Pairs$)
  $ImprovedSolutions \leftarrow$ Improvement($TrialSolutions$)
  $RefSet \leftarrow$ Reference set update($ImprovedSolutions$)
  if $RefSet$ changed **then**
    $NewSolutions \leftarrow TRUE$
  else
    $NewSolutions \leftarrow FALSE$
  end if
while $NewSolutions$

Algorithm 1 Basic scatter search

4.1 Diversification generation and improvement

We propose two diversification generation methods based on GRASP constructions. Solutions are built one element at a time and the selection is guided by a greedy function. Diversification is induced by a random selection from a restricted candidate list (RCL) that contains the most promising set of elements [6].

The construction starts with ID_0 and it performs $|\hat{V}|$ iterations, adding one vertex $v \in \hat{V}$ in each iteration. Limiting the insertions to the vertices $v \in \hat{V}$ is one of the main differences between our constructions and the constructions in the existing GRASP [15]. Initially, the candidate list CL contains all the incremental vertices in $\hat{V}$, that is, $CL = \{v \in \hat{V}\}$. The RCL consists of all the vertices in CL with degrees that are at least as high as a threshold τ. That is, $RCL = \{v \in CL : \rho(v) \geq \tau\}$. The degree $\rho(v)$ of a vertex $v \in CL$ is calculated considering the vertices in the partial solution. That is, $\rho(v) = |u \in IV_t : u \in \Lambda(v)|$. The threshold represents a degree that is in the top α percentil of the degree range for the vertices in CL. That is, $\tau = \min_{v \in IV_t} \rho(v) + \alpha \left(\max_{v \in IV_t} \rho(v) - \min_{v \in IV_t} \rho(v)\right)$. The parameter α controls the balance between diversity (as represented by the randomness proxy) and solution quality (as represented by the greedy function). A value of $\alpha = 0$ induces a total random selection and a value $\alpha = 1$ turns the construction into a deterministic process.

Let $v \in CL$ be the vertex selected for insertion in layer $L(v)$. The procedure calculates the barycenter of the selected vertex by taking into consideration both layer $L(v) - 1$ and layer $L(v) + 1$ in the partial drawing ID_t:

$$bc(v, IV_t) = \frac{\sum_{u \in \Lambda_{L(v)-1}(v) \cap IV_t} \pi(u) + \sum_{u \in \Lambda_{L(v)+1}(v) \cap IV_t} \pi(u)}{|\Lambda(v) \cap IV_t|}$$

We refer to this construction procedure as C1. We also tested a procedure labeled C2 that operates in the same way as C1 but uses a different barycenter calculation. Instead of considering both layers around $L(v)$, the barycenter is calculated with layer $L(v)+1$ if $L(v) < k$ and $L(v)-1$ if $L(v) = k$:

$$bc(v, IV_t) = \begin{cases} \frac{\sum_{u \in A_{L(v)-1}(v) \cap IV_t} \pi(u)}{|A_{L(v)-1}(v) \cap IV_t|} & si \quad i = k \\ \frac{\sum_{u \in A_{L(v)+1}(v) \cap IV_t} \pi(u)}{|A_{L(v)+1}(v) \cap IV_t|} & si \quad i < k \end{cases}$$

The construction process for both C1 and C2 terminates after $\left|\hat{V}\right|$ iterations. Note that both procedures are equivalent in the case of bipartite graphs. Algorithm 2 summarizes the procedural steps.

$ID_0 \leftarrow D; CL \leftarrow \hat{V}; t = 0$
for $t = 1, \ldots, \left|\hat{V}\right|$ **do**
  $RCL \leftarrow \{v \in CL : \rho(v) \geq \tau\}$
  $v \leftarrow Random(RCL)$
  $ID_t \leftarrow Insert(v, ID_{t-1})$
  $CL \leftarrow CL \setminus \{v\}$
end for

Algorithm 2 Diversification generation

The *Random* function in Algorithm 2 selects a vertex at random from the RCL. The *Insert* function calculates the barycenter (for C1 or C2) and then inserts v in the previous drawing ID_{t-1}. The results of the insertion is the current drawing ID_t.

The improving procedure is a simple local search based on a neighborhood that attempts to displace a vertex a single position. The method sweeps the entire solution, one vertex at a time, and probes the movement of vertices to their adjacent positions. Thus, a vertex v currently in position $\pi(v)$ is evaluated for a possible move to position $\pi(v) - 1$ and to $\pi(v) + 1$. The move must be feasible with respect to the relative position of v and the positions of other original vertices in the same layer. The procedure acts as a true local search in the sense that it only executes improving moves and it stops when no improving is possible. The best-move strategy is used, meaning that, in each iteration, all vertices are scanned to detect the most improving move.

4.2 Combinations

Path relinking (PR) is used as the main mechanism to combine solutions [8]. PR was originally proposed as a strategy to integrate search diversification and intensification in the context of tabu search [9]. The technique was later adapted, in the context of graph drawing [12], as an intensification strategy for GRASP, resulting in a family of procedures known as GRASP with PR [22]. It is based on the exploration of paths

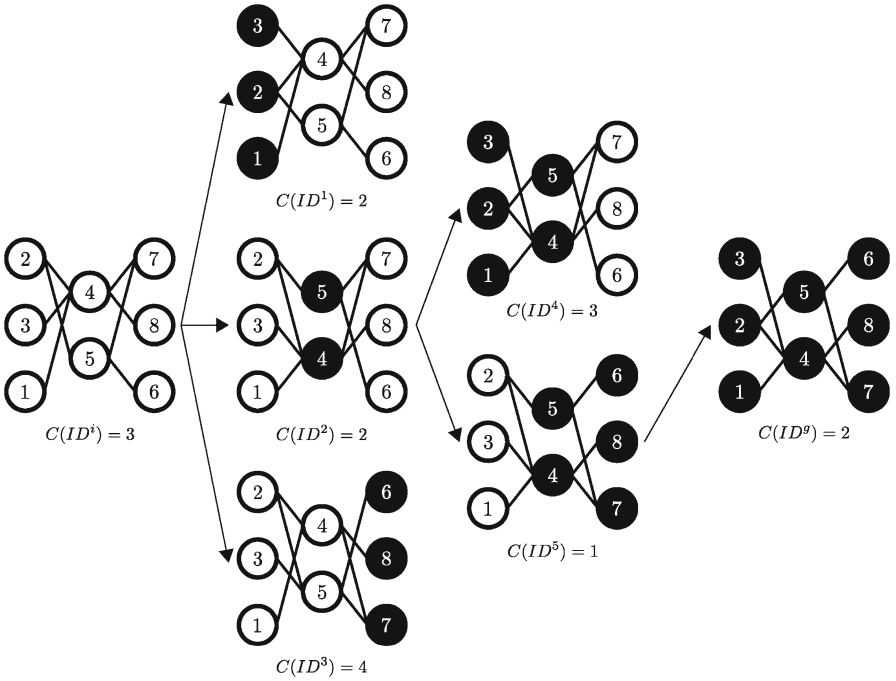

Fig. 5 Illustration of path relinking

that connect high-quality solutions, by generating intermediate solutions that could be better or more diverse than the solutions being connected. These solutions are known as the initiating solution and the guiding solution. In our context, we refer to the initiating solution as the incremental drawing ID^i and to the guiding solution as the incremental drawing ID^g.

The path relinking process consists of replacing an entire layer from a current solution with a layer from ID^g. The relinking is completed in k steps and it generates $\frac{k(k+1)}{2} - 1$ intermediate solutions. Figure 5 illustrates the path relining process for a drawing with 8 vertices and 3 layers.

The white vertices correspond to the ordering in ID^i and the black vertices correspond to the ordering in ID^g. The intermediate solution ID^1 is built by replacing the first layer in ID^i with the first layer in ID^g. Similarly, intermediate solutions ID^2 and ID^3 are constructed by replacing layers 2 and 3. The process continues from the best solution found, breaking ties arbitrarily. Since $C\left(ID^1\right) = C\left(ID^2\right) = 2$, then the process could continue from either one of these solutions. In our illustration, we select ID^2. Solutions ID^4 and ID^5 are generated by replacing layer 1 and 3, respectively. The process terminates when layer 1 is replaced in solution ID^5. In this case, PR is able to find a solution (ID^5) that is better than both ID^i and ID^g. The best intermediate solution found during the PR process is subjected to the improvement method. This is standard procedure in SS implementations.

5 Variable-neighborhood scatter search

In the previous section, we described the elements of a SS implementation that uses a local search with a simple neighborhood as the improvement method. In order to enhance the exploration around solutions that the SS creates with the diversification generator and the combination method, we expand the improvement method to include multiple nested neighborhoods. The expansion follows the variable neighborhood search (VNS) framework. VNS is a general metaheuristic methodology whose search trajectories are dictated by systematic changes of neighborhood structures [16]. Recently, a number of variants have been suggested, including variable neighborhood descent [10], or even parallel variants [5].

Variable neighborhood descent (VND) is based on the notion that the exploration from a given solution should start in the smallest neighborhood (N_s). If no improvement is found, then the search should be expanded to a larger neighborhood. This continues until the search reaches the largest neighborhood available (N_l). If at any point, for example during the exploration of N_q, an improved solution is found, then the process is reinitiated in the smallest neighborhood (N_s). On the other hand, the process stops if all the neighborhoods (from N_s to N_l) have been explored and no improvement is found. In nested neighborhoods, such as the ones used in this work, $N_q \subseteq N_{q+1}$ for $q = s, \ldots, l-1$.

We define N_q as the neighborhood obtained by inserting vertex v, currently in $\pi(v)$, in positions $\pi(v) - q$ and $\pi(v) + q$. The nested nature of the neighborhoods is such that at most $2|IV|$ solutions need to be considered in each of them since the procedure discards infeasible moves. Recall that infeasibility is produced by the desire to preserve the relative order of the vertices in V. In addition, moves that result in positions outside the range of 1 and $|L_i|$ in any given layer i are also discarded. Algorithm 3 summarizes the steps in the VND improvement method.

```
q ← s
do
    ID* ← arg min_{ID*∈N_q} C(ID)
    if C(ID*) < C(ID) then
        ID ← ID*
        q ← s
    else
        q ← q + 1
    end if
while q < l
```

Algorithm 3 VND improvement method

The move value calculations can be performed very efficiently because the change in the number of arc crossings produced from moving vertex v from its current position $\pi(v)$ to position $\pi(v) + q$ (or $\pi(v) - q$) depends on the changes produced from moving vertex v to positions $\pi(v) + j$ (or $\pi(v) - j$) for $j = 1, \ldots, q-1$. Therefore,

the move value calculations for N_q are the basis for the move-value calculations for N_{q+1}.

Our variable-neighborhood scatter search (VNSS) is the result of embedding VND in SS. Instead of simply replacing the local search with VND, we create a fuller integration of the two methodologies. In a typical SS implementation, the search stops when no new solutions are admitted in the reference set. The decision at that point is either to stop completely or to restart the search by introducing new (diverse) solutions in the reference set. Instead of restarting after failing to update the reference set, VNSS increases the size of the neighborhood. Upon success, the neighborhood goes back to the smallest size. If a SS iteration with the largest neighborhood N_l does not produce new reference solutions, the procedure stops. Algorithm 4 summarizes the VNSS logic.

```
q ← s
P ← Diversification generation
P* ← VND(P, q)
RefSet ← Reference set update(P*)
do
    do
        Pairs ← Subset generation(RefSet)
        TrialSolutions ← Combinations(Pairs)
        ImprovedSolutions ← VND(TrialSolutions, q)
        RefSet ←Reference set update(ImprovedSolutions)
        if RefSet changed then
            NewSolutions ← TRUE
            q ← s
        else
            NewSolutions ← FALSE
        end if
    while NewSolutions
    q ← q + 1
while q < l
```

Algorithm 4 Variable-neighborhood scatter search

There are two important differences between Algorithm 1 and 4. The first one is that the improvement method has been replaced with the VND method. The second one is that in Algorithm 1 the search terminates when no new solutions are admitted in the reference set. In Algorithm 4 the search continues at this stage, as long as parameter q is lower than l. Table 2 summarizes the search parameters of the entire VNSS method.

6 Computational experiments

This section describes the computational experiments that we performed to test the effectiveness and efficiency of the procedures discussed above. The GRASP (Sect. 3)

Table 2 Parameters in the VNSS method

	Search parameters
α	Used in construction methods (C1, C2) to trade off randomization and greediness
τ	Threshold that represents a degree that is in the top α percentile of the degree range for the vertices in the Candidate List (CL)
s	Smallest neighborhood in the VNSS method
l	Largest neighborhood in the VNSS method

was implemented in C++. The scatter search (Sect. 4) and the variable neighborhood scatter search (Sect. 5) were implemented in Java 8. All experiments were conducted on a 2.8 Ghz Intel Core i7 processor with 8 GB RAM.

For each experiment, we report the following performance measures: Average number of crossings ($\bar{C}$), computing time in seconds (Time), average deviation with respect to the best solution found in the experiment (Dev), and number of best solutions found in the experiment (Best). Note that both Dev and Best refer to the solutions found within the experiment and not the best solutions known for these problems.

6.1 Problem instances

We employed 240 instances in our experimentation. This set of instances, referred to as IGDPLIB, is available at http://www.optsicom.es/igdp. The hierarchical graphs were generated following the guidelines in the literature [14]. The number of layers is an input to the graph generator and the number of vertices in each layer is randomly chosen between 5 and 30. 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_{k-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_k is found with a degree of zero, an arc is added to a randomly chosen vertex in layer L_{k-1}. Next, the generator compares the current number of arcs with the number of arcs that are required to meet the desired density. The generator then adds enough arcs to cover the difference between the current number and the number that results from the desired density. The additional arcs are added by randomly choosing two vertices in consecutive layers. We used the generator to create 240 instances with the following characteristics. For each combination of 2, 6, 13, and 20 layers and 0.065, 0.175 and 0.300 graph densities, 20 instances were generated. We then applied the barycenter algorithm described in Sect. 2 to obtain a drawing D for each graph.

The set $\hat{V}$ of incremental vertices is created according to a parameter δ that establishes the percentage of additional vertices to be added to each layer. Thus, each layer in the augmented problem has $(1+\delta)L_i$ vertices and $|\hat{V}| = \delta|V|$. For each vertex in $\hat{V} \cap L_i (i = 1, \ldots k-1)$, that is, for each incremental vertex in all but the last layer, an arc is added to a randomly selected vertex in L_{i+1}. Similarly, for each vertex in

$\hat{V} \cap L_k$ an arc is added to a randomly chosen vertex in L_{k-1}. This guarantees that each new vertex has a degree of at least one. Additional arcs are added by randomly choosing two vertices in consecutive layers, up to the desired number dictated by δ. Of the 20 instances generated for each combination of number of layers and density, 10 are augmented by 20 % ($\delta = 1.2$) and 10 are augmented by 60% ($\delta = 1.6$).

6.2 Algorithm configuration and fine-tuning

The goal of our preliminary experimentation is to find effective configurations for SS and VNSS and to fine tune their algorithmic parameters. Our training set consists of 24 representative instances form the set of 240. We first test the constructive methods C1 and C2, described in Sect. 3.1. The performance of these procedures depends on the parameter α, which balances greediness and randomness. We tested three values of α on each procedure (0.25, 0.50, and 0.75) and determined that the best performance for C1 is achieved with $\alpha = 0.5$ and the best performance for C2 is achieved with $\alpha = 0.75$. Computational effort does not play a role because both procedures build solutions in a negligible amount of time. So, our conclusions are based on average quality and number of best solutions found when generating 100 constructions for each instances in the training set.

Using these parameter values, we tested C1 and C2 against the construction procedure of the published GRASP [15]. We refer to this procedure as M&E 2001. We use the training set and generate 100 solutions, with each procedure, for each of the 24 instances in the training set. Table 3 shows the summary of the results. This table reveals that both C1 and C2 outperform M&E 2001 in average solution quality and average deviation with respect to the best solution. In terms of number of best solutions found, C1 is clearly superior to the other two alternatives. Finally, notice that as we mentioned above, the associated computing time is almost negligible. These results support the selection of C1 with $\alpha = 0.5$ as the construction to use to generate solutions in the initial step of our scatter search.

The next fine-tuning experiment is devoted to the VNSS parameters, s and l. These parameters indicate the smallest and the largest neighborhoods, respectively, to be used in the VND improvement method. The goal is to find values for s and l that result in an effective tradeoff between solution quality and computational effort. The experiment consists of, for each of the 24 instances in the training set, generating 100 solutions with C1(0.5), improving these solutions with VND, and then recording the best solution found for each instance. This is done for several values of q. In particular, we run VND(100, q) for q ranging from 1 to 10. Figure 6 presents the scatter plot of

Table 3 Comparison of the proposed constructive methods and the best in the literature

Procedure	$\bar{C}$	Dev (%)	Best	Time (s)
C1(0.50)	14234.29	1.58	17	0.14
C2(0.75)	14391.96	4.84	4	0.15
M&E 2001	15238.71	24.52	4	0.31

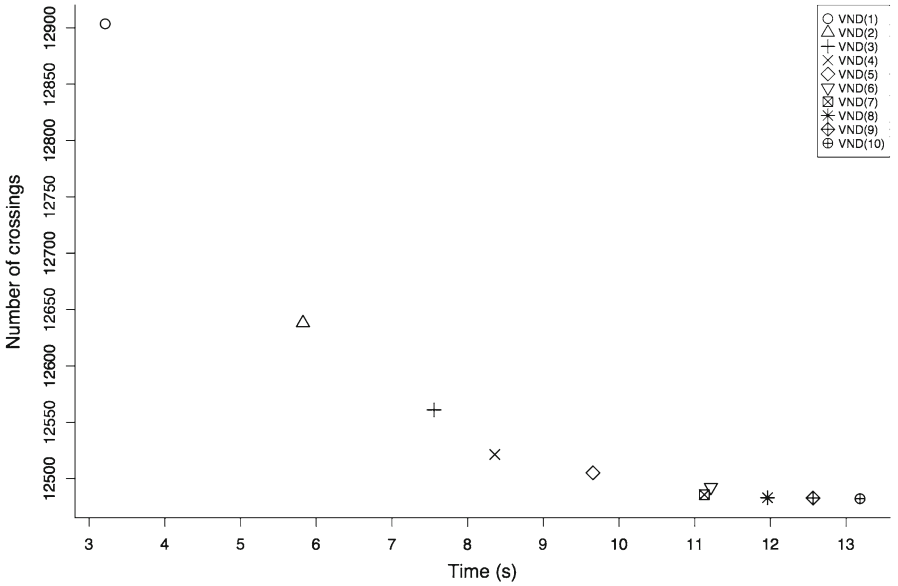

Fig. 6 Solution quality versus computational time for VND(q)

the results, reporting the average number of crossings of the best solutions found (y-axis) and the total computing time, which includes both construction and improvement (x-axis).

As expected, Fig. 6 shows that the quality of the solutions can be improved by using larger neighborhoods and therefore more computational time. In terms of speed, not surprisingly $q = 1$, results in the fastest method, but it also yields the lowest solution quality. Improvement in solution quality is achieved by increasing q. The largest improvements are from $q = 1$ to $q = 2$ and from $q = 2$ to $q = 3$. The graph shows diminishing quality improvement for larger q values and almost no improvement after $q = 7$. Therefore, we choose $s = 3$ and $l = 7$.

6.3 Competitive testing

For the competitive testing, we compare the procedures that we developed (SS and VNSS) and the adaptation of GRASP [15] to the multilayered problem. This experiment consists of executing the three procedures on the entire set of 240 instances. We report the results in 4 tables, two tables for the instances generated with $\delta = 1.2$ and two tables for instances generated with $\delta = 1.6$. The tables focus on solution quality, as measured by average deviation from the best and number of best solutions found. There is one table for each of these metrics and for each δ. Each cell in the table reports the results obtained by the three procedures. The columns in the tables correspond to the graph densities and the rows to the number of layers. Therefore, each cell in a table represents 10 problem instances. Figures 7 and 8 summarize these results. Figure 7 shows for each density value the average deviation that each method exhibits in the

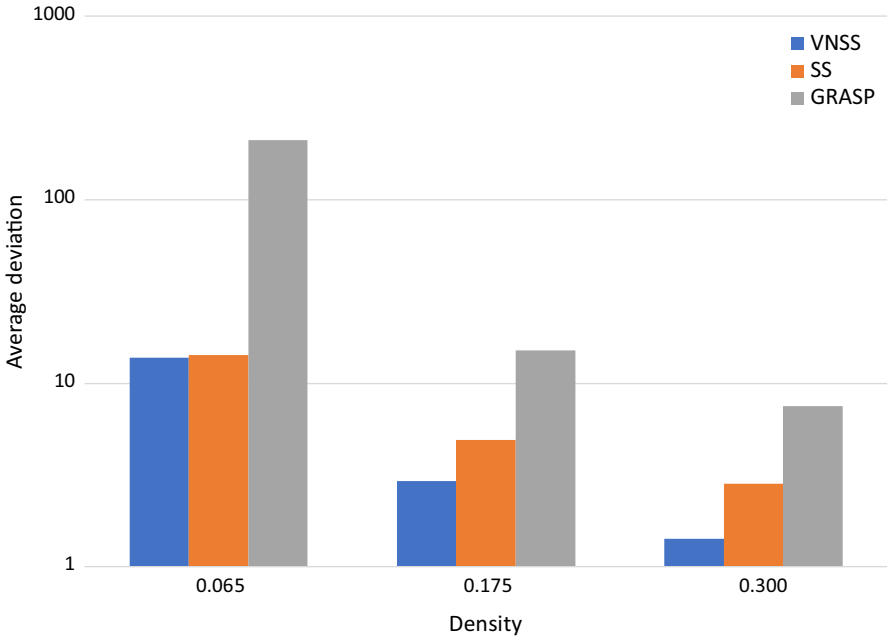

Fig. 7 Average deviation w.r.t. best known solutions

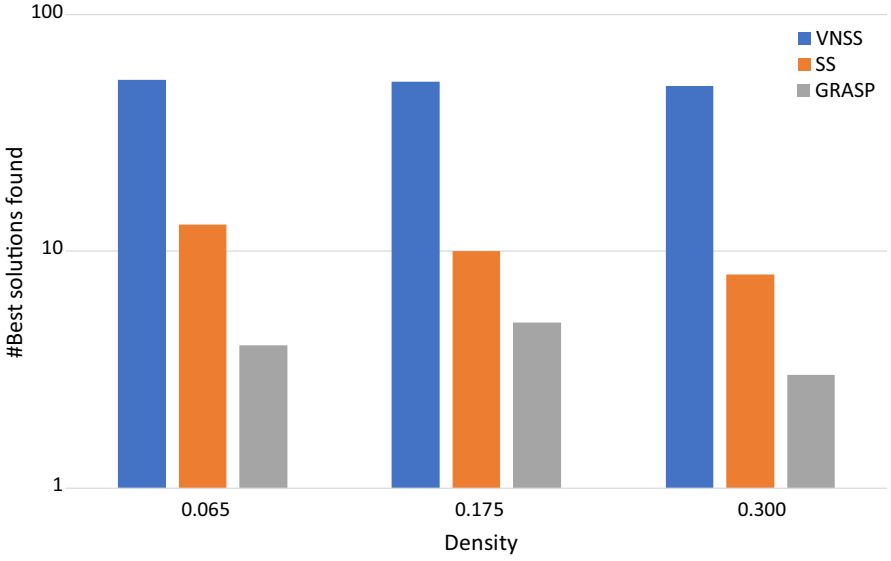

Fig. 8 Number of best solutions found with each method

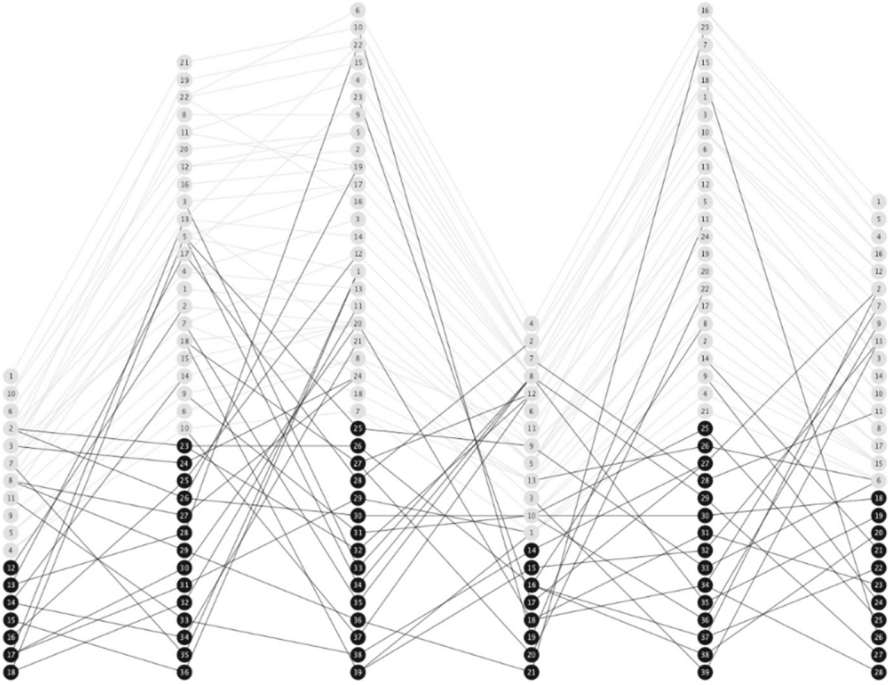

Fig. 9 Original drawing

set of instances, while Fig. 8 shows the number of instances in which each method is able to match the best solution known. Since the differences among the three methods are large, we use a logarithmic scale in the y-axis. Both diagrams clearly show the superiority of the results obtained with the VNSS method.

The results in the Tables 4, 5, 6, and 7 show the advantage of VNSS over SS and GRASP. Specifically, GRASP is only able to produce better results than VNSS in 2-layer graphs with $\delta = 1.2$ and densities of 0.175 and 0.300 (see Table 4). There are 12 cases in each table and a total of 4 tables. Out of these 48 cases, VNSS produces the best results in 41. VNSS is particularly effective in graphs with $\delta = 1.6$.

We applied Friedman's test using the results obtained for all 240 instances. This test computes, for each instance, the rank value of each method according to solution quality. Then, it calculates the average rank values of each method across all the instances solved. If the averages differ greatly, the associated p value will be small. The p value of less than 0.001 indicates significant differences among the average rank of the methods, which were 1.20, 1.93, and 2.68 for VNSS, SS, and GRASP, respectively. We also applied Wilcoxon's test to VNSS and SS, which in general terms answers the question: Do the two samples (solutions obtained with the methods) represent two different populations? The obtained p value less than 0.001 indicates a significance difference between the quality of the results of VNSS and SS. Table 8 summarizes these tests.

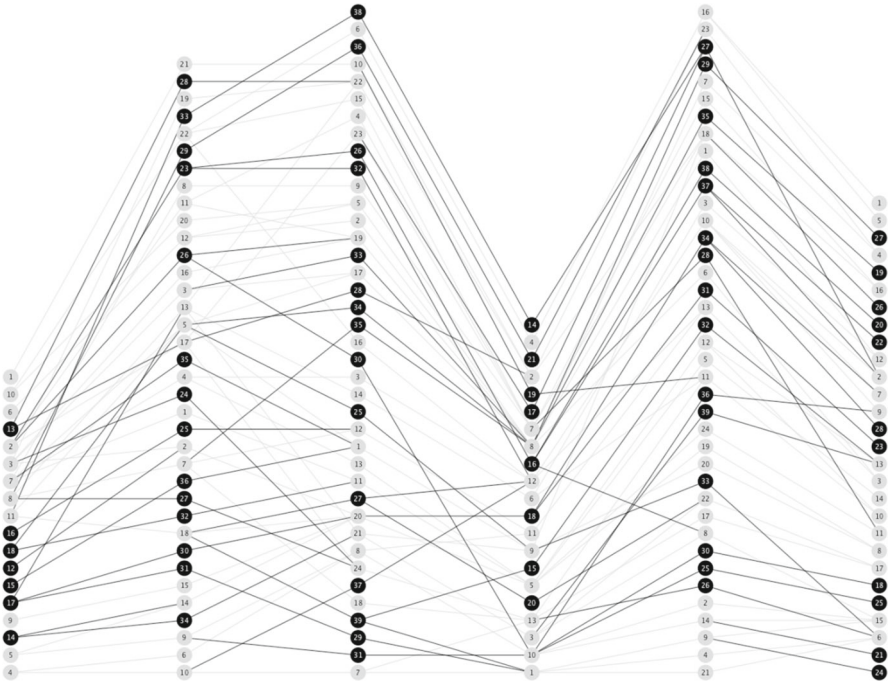

Fig. 10 Drawing obtained with VNSS

Table 4 Average deviation from best for instances generated with $\delta = 1.2$

Layers	Graph density					
	0.065		0.175		0.300	
2	VNSS	63.05	VNSS	23.39	VNSS	10.79
	SS	51.45	SS	18.29	SS	8.17
	GRASP	198.25	GRASP	2.21	GRASP	1.31
6	VNSS	0.00	VNSS	0.00	SS	0.62
	SS	5.48	SS	1.32	VNSS	2.19
	GRASP	69.60	GRASP	7.63	GRASP	2.66
13	VNSS	0.00	VNSS	0.00	VNSS	0.00
	SS	3.97	SS	1.91	SS	0.80
	GRASP	34.50	GRASP	7.16	GRASP	3.46
20	VNSS	0.00	VNSS	0.00	VNSS	0.00
	SS	4.68	SS	1.26	SS	0.82
	GRASP	34.94	GRASP	8.07	GRASP	3.52

Table 5 Number of best solutions (out of 10) for instances generated with $\delta = 1.2$

Layers	Graph density					
	0.065		0.175		0.300	
2	VNSS	4	VNSS	2	VNSS	3
	SS	5	SS	5	SS	5
	GRASP	4	GRASP	5	GRASP	3
6	VNSS	10	VNSS	10	VNSS	7
	SS	1	SS	1	SS	3
	GRASP	0	GRASP	0	GRASP	0
13	VNSS	10	VNSS	10	VNSS	10
	SS	0	SS	0	SS	0
	GRASP	0	GRASP	0	GRASP	0
20	VNSS	10	VNSS	10	VNSS	10
	SS	0	SS	0	SS	0
	GRASP	0	GRASP	0	GRASP	0

Table 6 Average deviation from best for instances generated with $\delta = 1.6$

Layers	Graph density					
	0.065		0.175		0.300	
2	VNSS	47.50	VNSS	0.00	VNSS	0.00
	SS	17.52	SS	2.15	SS	1.87
	GRASP	1076.26	GRASP	32.75	GRASP	15.13
6	VNSS	0.00	VNSS	0.00	VNSS	0.00
	SS	9.44	SS	4.67	SS	2.80
	GRASP	122.48	GRASP	23.14	GRASP	11.04
13	VNSS	0.00	VNSS	0.00	VNSS	0.00
	SS	11.03	SS	4.61	SS	3.11
	GRASP	79.97	GRASP	19.03	GRASP	11.07
20	VNSS	0.00	VNSS	0.00	VNSS	0.00
	SS	9.99	SS	4.95	SS	3.00
	GRASP	70.54	GRASP	20.59	GRASP	11.99

To complement our empirical analysis, we illustrate now the performance or the VNSS method when solving an instance with 6 layers, 181 vertices, and 242 arcs. Figure 9 represents the initial drawing in which we can identify the original vertices (in grey), and the added vertices (in black), located in the bottom part of the diagram. After applying the VNSS method, vertices are rearranged to minimize the number of arc crossings, as shown in Fig. 10. It is clear that the output of our method with 634 crossings provides a much readable diagram than the original graph with 1531 crossings.

Table 7 Number of best solutions (out of 10) for instances generated with $\delta = 1.6$

Layers	Graph density					
	0.065		0.175		0.300	
2	VNSS	9	VNSS	10	VNSS	10
	SS	7	SS	4	SS	0
	GRASP	0	GRASP	0	GRASP	0
6, 13, and 20	VNSS	10	VNSS	10	VNSS	10
	SS	0	SS	0	SS	0
	GRASP	0	GRASP	0	GRASP	0

Table 8 Statistical tests summary

Statistical test	p value	Statistical differences?
Friedman's test to VNSS, SS and GRASP	<0.001	YES
Wilcoxon's test to VNSS and SS	<0.001	YES

The test instances considered in this section range from 25 to 676 vertices, so an interesting question is how would perform our algorithm on larger instances. It is always interesting to investigate the scalability of a heuristic method with respect to its running time. To do so, we performed a linear regression between the running time (y variable) and the number of vertices (x variable). We obtained a correlation coefficient of 0.75 and a regression line of $y = -13.3 + 0.1x$. The statistical test on the slope indicates that the line fits well on the data, so we can expect a linear growth of the running time with respect to the instance size.

7 Conclusions

We had a twofold goal for this work, to experiment with the hybridization of VNS and SS and, in the process, to develop a state-of-the-art procedure for the multilayered incremental graph-drawing problem. We believe that we have achieved the first goal with the VNSS design. The merit of this hybrid method is that it preserves the main characteristics of both VNS and SS. The design is simple but effective. SS implementations reported in the literature use restarting to induce diversity when the search fails to update the reference set. Instead of restarting, diversity in VNSS is achieved by following the VNS strategy of expanding the neighborhood. Upon success (i.e., when new solutions are admitted in the reference set), the neighborhood search is contracted. This also follows the VNS philosophy and provides an excellent balance between intensification and diversification within the framework of SS.

In terms of our second goal, the results reported in Tables 4, 5, 6 and 7 are very strong in favor of VNSS. We have established benchmarks for the instances that we created and we believe that they will help other researchers test additional search strategies on this interesting combinatorial optimization problem.

Acknowledgements This work has been partially supported by the Spanish "Ministerio de Economía y Competitividad" and by "Comunidad de Madrid," Grants Refs. TIN2015-65460-C02 and S2013/ICE-2894, respectively. Additionally, Prof. Martinez-Gavara and Sánchez-Oro thank "Programa de Ayudas para Estancias Cortas en otras Universidades y Centros de Investigación," Universidad de Valencia (Ref. UV-INV_EPDI16-384465) and "Ayudas a la Movilidad Predoctoral para Estancias Breves," Ministerio de Economía y Competitividad (Ref. EEBB-I-16-11312) for supporting their visits to the University of Colorado, Boulder.

References

1. Böhringer, K.F., Paulisch, F.N.: Using constraints to achieve stability in automatic graph layout algorithms. In: Proceedings of CHI'90, pp. 43–51. ACM (1990)
2. Branke, J.: Dynamic graph drawing. In: Wagner, D., Kaufmann, M. (eds.) Drawing Graphs: Methods and Models, pp. 228–246. Springer, Berlin (2001)
3. Carpano, M.: Automatic display of hierarchized graphs for computer-aided decision analysis. IEEE Trans. Syst. Man Cybern. **10**(11), 705–715 (1980)
4. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.: Graph Drawing: Algorithms for the Visualization of Graphs, 1st edn. Prentice Hall PTR, Upper Saddle River (1998)
5. Duarte, A., Pantrigo, J.J., Pardo, E.G., Sánchez-Oro, J.: Parallel variable neighbourhood search strategies for the cutwidth minimization problem. IMA J. Manag. Math. **27**(1), 55–73 (2016)
6. Feo, T.A., Resende, M.G.: Greedy randomized adaptive search procedures. J. Glob. Optim. **6**(2), 109–133 (1995)
7. Garey, M., Johnson, D.: Crossing number is NP-complete. SIAM J. Algebraic Discrete Methods **4**(3), 312–316 (1983)
8. Glover, F.: Tabu search and adaptive memory programming—advances, applications and challenges. In: Barr, R.S., Helgason, R.V., Kennington, J.L. (eds.) Interfaces in Computer Science and Operations Research: Advances in Metaheuristics, Optimization, and Stochastic Modeling Technologies, pp. 1–75. Springer, Boston (1997)
9. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers, Norwell (1997)
10. Hansen, P., Mladenović, N., Moreno-Pérez, J.: Variable neighborhood search: methods and applications. 4OR **6**(4), 319–360 (2008)
11. Kaufmann, M., Wagner, D.: Drawing Graphs: Methods and Models. Lecture Notes in Computer Science. Springer, Berlin (2001)
12. Laguna, M., Martí, R.: GRASP and path relinking for 2-layer straight line crossing minimization. INFORMS J. Comput. **11**, 44–52 (1999)
13. Laguna, M., Martí, R.: Scatter Search: Methodology and Implementations in C. Kluwer Academic Publisher, Norwell (2003)
14. Laguna, M., Marti, R., Valls, V.: Arc crossing minimization in hierarchical digraphs with tabu search. Comput. Oper. Res. **24**(12), 1175–1186 (1997)
15. Martí, R., Estruch, V.: Incremental bipartite drawing problem. Comput. Oper. Res. **28**(13), 1287–1298 (2001)
16. Mladenović, N., Hansen, P.: Variable neighborhood search. Comput. Oper. Res. **24**(11), 1097–1100 (1997)
17. Mutzel, P., Jünger, M., Leipert, S.: How to layer a directed acyclic graph. In: Graph Drawing: 9th International Symposium, GD 2001 Vienna, Austria, September 23–26, 2001 Revised Papers, pp. 16–30. Springer, Berlin (2002)
18. North, S.C.: Incremental layout in DynaDAG. In: Proceedings of of Graph Drawing'95, volume 1027 of Lecture Notes in Computer Science, pp. 409–418. Springer, Berlin (1996)
19. Pinaud, B., Kuntz, P., Lehn, R.: Dynamic graph drawing with a hybridized genetic algorithm. In: Parmee, I.C. (ed.) Adaptive Computing in Design and Manufacture VI, 2004, pp. 365–375. Springer, Bristol (2004)
20. Purchase, H.: Which aesthetic has the greatest effect on human understanding? In: Proceedings of Graph Drawing'97, vol 1353 de Lecture Notes in Computer Science, pp. 248–261. Springer, Berlin (1997)
21. Purchase, H.: Effective information visualization: a study of graph drawing aesthetics and algorithms. Interact. Comput. **13**(2), 147–162 (2000)

22. Resende, M.G., Ribeiro, C.C.: GRASP with path-relinking: recent advances and applications. In: Ibaraki, T., Nonobe, K., Yagiura, M. (eds.) Metaheuristics: Progress as Real Problem Solvers, pp. 29–63. Springer, New York (2005)
23. Sugiyama, K., Tagawa, S., Toda, M.: Methods for visual understanding of hierarchical system structures. IEEE Trans. Syst. Man Cybern. (SMC) **11**(2), 109–125 (1981)