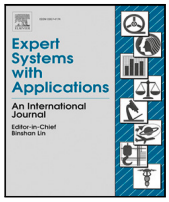
Expert Systems With Applications 237 (2024) 121477

Contents lists available at ScienceDirect

Expert Systems With Applications

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



Solving the incremental graph drawing problem by multiple neighborhood solution-based tabu search algorithm

Bo Peng [a], Songge Wang [a], Donghao Liu [b], Zhouxing Su [b,*], Zhipeng Lü [b], Fred Glover [c]

[a] School of Business Administration, Faculty of Business Administration, Southwestern University of Finance and Economics, Chengdu, 610074, PR China
[b] SMART, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074, PR China
[c] Entanglement, Inc., Boulder, CO 80305, USA

ARTICLE INFO

Keywords:
Incremental graph drawing problem
Multiple neighborhood search
Solution-based tabu search
Dynamic diversification mechanism

ABSTRACT

With the advent of the era of big data, more and more complex networks and knowledge based systems require data visualization for analysis and presentation, where graphs have become a standard representation model. Graph drawing addresses the problems of constructing geometric representations of graphs to make them easy to analyze. In this paper we aim to reduce the edge crossing number in the context of the incremental graph drawing problem (IGDP), in which we want to preserve the layout of a graph over successive drawings to present complex networks or knowledge-based systems. Specifically, we propose a metaheuristic algorithm called multiple neighborhood solution-based tabu search (MNSB-TS) by combining a solution-based tabu strategy and a multiple neighborhood structure for solving the incremental graph drawing problem. Our MNSB-TS approach introduces four neighborhood operators to accompany the solution-based tabu strategy for determining the tabu status of each neighbor solution. Extensive computational experiments on public benchmark instances demonstrate that the proposed MNSB-TS is highly competitive in comparison with the state-of-the-art heuristics and the exact optimization solver Gurobi. Key components of the approach are analyzed to evaluate their impact on algorithm performance and learn which search mechanisms are better suited for incremental graph drawing.

1. Introduction

The enormous growth in the amount of data confronted in business and engineering analysis has brought the problem of representing large graphs into the spotlight. For most graphs, the only information available concerns which pairs of vertices are connected by arcs and there are numerous ways to represent the same graph with different arrangements of these vertices. There is accordingly a major need for automatic layout algorithms for graph drawing to create graphs that are more understandable, useable, and aesthetically pleasing.

The primary challenge of graph drawing is to construct geometric representations of graphs that make them easy to analyze. Although the criteria used to judge the quality of a drawing is quite subjective, it is widely considered preferable to minimize the number of arc crossings to get a clearer layout. This minimization problem has received special attention in layered digraphs, since any graph can be transformed into a layered graph using Sugiyama's framework (Sugiyama et al., 1981). Therefore, methods developed to tackle the problem of minimizing arc crossings in layered graphs can be applied to any general graph (Fulek & Tóth, 2020; Tiezzi et al., 2022). The problem, which is NP-Complete, even when the graph consists of only two layers (Garey & Johnson, 1983), is one of the most difficult ones for optimization algorithms.

Palubeckis et al. (2019) proposed an approach for solving this problem, which combines a simulated annealing (SA) method with a variable neighborhood search (VNS) scheme. The concept of dynamic or incremental graph drawing can be traced back to 1991, when Eades et al. (1991) pointed out that the user usually builds up a mental map when reading a drawing, so he or she expects that if new vertices or arcs are added, the new graph should be represented in a way (i.e., using a layout) that resembles the original one. In a wide variety of practical situations, it is considered helpful to maintain a so-called 'mental picture' of the layout from a graph over the continuous drawings. When the vertices are deleted from or added to a graph, users must adjust their mental map to grasp the new modified graph. Research for dynamic graph drawing aims to minimize this effort. As one approach to this, Martí and Estruch (2001) proposed the idea of stability across drawing by maintaining the relative ordering invariant among the common vertices in the original graph and new modified graph.

* Corresponding author.
E-mail addresses: pengbo@swufe.edu.cn (B. Peng), suzhouxing@hust.edu.cn (Z. Su), zhipeng.lv@hust.edu.cn (Z. Lü).

https://doi.org/10.1016/j.eswa.2023.121477
Received 15 March 2023; Received in revised form 3 September 2023; Accepted 3 September 2023
Available online 7 September 2023

Although the general problem of minimizing arc crossing in graph drawing problems has been studied extensively, the corresponding problem on incremental layered graphs, called the incremental graph drawing problem (IGDP), has received very little attention. In one of the earlier papers devoted to this, Martí and Estruch (2001) examined the instance of the IGDP problem containing two layers, proposing a greedy randomized adaptive search procedure (GRASP) and an exact algorithm based on branch and bound, that explores the set of solutions (permutations of the vertices in each layer) using a search tree. Subsequently, Martí et al. (2018b) adapted the mathematical programming formulation originally proposed for the 2-layer incremental graph drawing problem, and reported experiments with Gurobi, showing they can solve small and medium-size instances to optimality. Sánchez-Oro et al. (2017) addressed the incremental graph drawing problem in hierarchical graphs not limited to bipartite graphs, and proposed several solution strategies. Their approaches first adapted the GRASP method to solve these problems and then introduced a procedure based on scatter search joined with variable neighborhood search to tackle problems with up to 20 layers. Napoletano et al. (2019) minimized the number of edge crossings while satisfying some constraints required to preserve the position of vertices with respect to previous drawings. They proposed heuristic methods to obtain high-quality solutions to this optimization problem in short computational times and presented a mathematical programming formulation for the goal of obtaining optimal solutions for small and medium instances.

Martí et al. (2018a) focused on a particular variant called the min–max graph drawing problem, which minimizes the maximum crossing among all edges — a challenging variant of the original min-sum graph drawing problem that arises in the optimization of VLSI circuits and the design of interactive graph drawing tools. New heuristic methods based on strategic oscillation methodology were proposed to tackle it. In a refinement, Pastore et al. (2020) proposed a tabu search method to obtain high-quality solutions. After that, Wu et al. (2021) presented a variable depth neighborhood search algorithm for solving this problem.

The solution strategies used in the present paper incorporate tabu search, as proposed for graph drawing in Glover et al. (2021). In the form most commonly adopted, tabu search uses a tabu list to prevent revisiting recently encountered solutions as a means to avoid cycling and being trapped in local optimality. Multiple neighborhood search as used here builds on the use of multiple neighborhoods introduced in Glover et al. (1984) and elaborated in the context of tabu search in Glover (1986, 1997) and Xu et al. (1996, 1998). A popular form of multiple neighborhood search called variable neighborhood search (VNS) introduced by Mladenović and Hansen (1997) uses a different protocol for visiting neighborhoods than employed here and recently has been studied by joining it with strategies associated with scatter search in Sánchez-Oro et al. (2017).

Unlike the popular attribute-based tabu search approaches which use memory that records only a small amount of information about solutions generated (such as the identity and values of one or two variables that are chosen to transition from one solution to another) solution-based tabu search, whose idea were first introduced in Woodruff and Zemel (1993), has begun to draw attention only very recently (Chang et al., 2021; Lai et al., 2018a; Wang et al., 2017). In general, solution-based tabu search (also called "fine grain" tabu search in Glover (1997)), uses hash methods to (approximately) record complete solutions instead of the partial solution information recorded in attribute-based tabu search. Recently, Peng et al. (2020) proposed a tabu search method that incorporates adaptive memory to search the solution space efficiently for a bipartite graph (two-layered hierarchy).

Although solution-based tabu strategy has been successfully applied to various combinatorial optimization problems, e.g., the minimum differential dispersion problem (Wang et al., 2017), the maximum diversity problem (Liu et al., 2020), and the obnoxious p-median problem (Chang et al., 2021), research on using solution-based tabu search to solve graph drawing problems is rare. Therefore in this study, we attempt to explore combining this method with a multiple neighborhood mechanism to tackle the general case (i.e., incremental graph drawing problem) by extending the previous two-layered hierarchy to multiple-layered hierarchy.

The main contributions of this paper are as follows:

- A novel hybrid algorithm that combines solution-based tabu search and multiple neighborhood search for solving IGDP.
- Four neighborhood operators with corresponding fast neighborhood evaluation methods that employ dedicated matrices to record information for search intensification.
- A dynamic diversification mechanism to complement the search intensification provided by the solution-based TS procedure.
- A performance evaluation of our proposed algorithm that compares it with the best-performing heuristics and the exact solver Gurobi for the 240 benchmark problem instances. Our computational study further examines key components and parameters of the algorithm and identifies their impact on its performance, disclosing that our algorithm outperforms the best known algorithms from the literature in a statistically significant way.

Our proposed combination of solution-based multiple neighborhood tabu search and hash functions suitable for permutation problems is quite general, and can be applied to solve other related optimization problems, especially multi-layer permutation optimization problems. The rest of the paper is organized as follows. Section 2 presents the IGDP problem description and mathematic model. Section 3 describes our proposed multiple neighborhood solution-based tabu search algorithm. Section 4 presents the experimental protocols and parameter tuning, and Section 5 reports experimental results and comparisons with state-of-the-art algorithms from the literature. The effectiveness of several key ingredients in the proposed algorithm are evaluated in Section 6 and concluding remarks are given in Section 7.

2. Problem description and mathematic model

We represent a hierarchical graph by $G = (V, E, K, L)$, where V, E and K identify the set of vertices, edges, and the number of layers, respectively. The function $L(u)$ denotes the index of the layer where each vertex $u \in V$ resides by a mapping: $V \rightarrow \{1, \ldots, K\}$. We are concerned with the instance of incremental graph drawing defined on an incremental graph $IG = (IV, IE, K, L)$ derived from an original hierarchical graph $G = (V, E, K, L)$ as proposed by Sánchez-Oro et al. (2017), where the incremental graph arises by introducing an additional set of vertices AV and their corresponding edges AE.

Formally, the incremental graph is given by $IV = V \cup AV, IE = E \cup AE$ and $L(u) : IV \rightarrow \{1, 2, \ldots, K\}$, with the value of $L(u)$ unchanged for vertex $u \in V$. We call the vertices in V original vertices and call the other vertices in AV incremental vertices. The goal of IGDP is to find a drawing that yields the minimum number of edge crossings while preserving the ordering of the original vertices in each layer of the incremental graph.

Figs. 1 and 2 show two drawings (i.e., solutions) of a six-layer incremental graph for the IGDP. The black vertices denote the original elements whose relative positions must be preserved while the white vertices denote the new (incremental) ones added which can be arbitrarily placed.

The first drawing in Fig. 1 that contains a lot of edge crossings (381) gives the readers a messy feeling. The second drawing, in Fig. 2, is optimized for the IGDP depicted in Fig. 1 by relocating the new white nodes in positions that minimize the number of crossings while preserving the relative position of the original nodes to yield 172 edge crossings. It is apparent that the drawing in Fig. 2 with fewer edges crossings is neater and more readable.

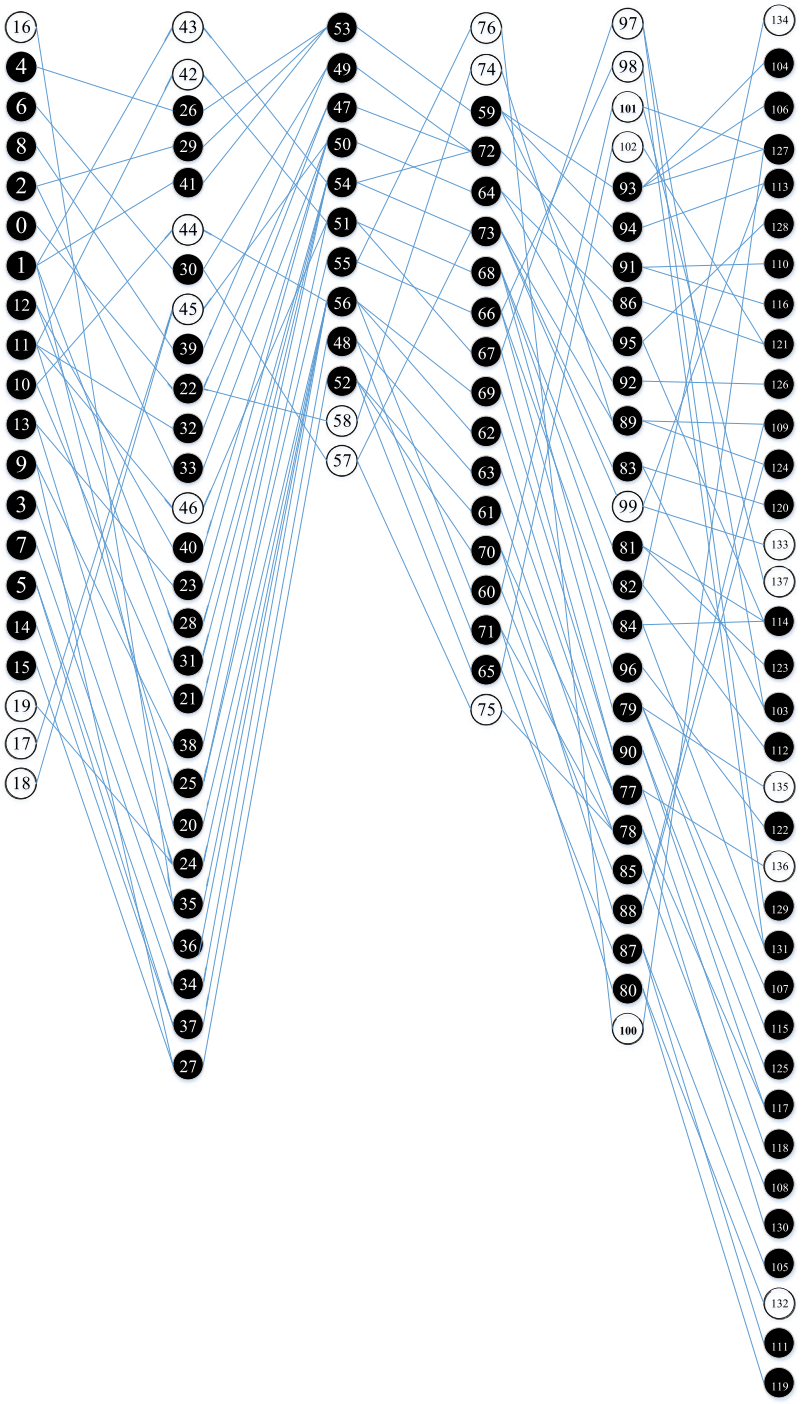


Fig. 1. A drawing for an example (incgraph_6_0.06_5_30_1.20_1 instance) with a six-layer graph. The number of edge crossings for this drawing is 381.

To formulate the IGDP problem more precisely, we use the mathematic model given by Napoletano et al. (2019), which is presented as follows:

$$\min \sum_{k=1}^{K-1} \sum_{(i,j),(p,q)\in IE_k} C_{ijpq}^k \tag{1}$$

$$1 - C_{ijpq}^k \le x_{qj}^{k+1} + x_{ip}^k \le 1 + C_{ijpq}^k; \quad \forall (i,j),(p,q) \in IE_k, j > q, \forall k \tag{2}$$

$$-C_{ijpq}^k \le x_{jq}^{k+1} - x_{ip}^k \le C_{ijpq}^k; \quad \forall (i,j),(p,q) \in IE_k, j < q, \forall k \tag{3}$$

$$0 \le x_{ij}^k + x_{jp}^k - x_{ip}^k \le 1; \quad \forall 1 \le i < j < p \le n_k, \forall k \tag{4}$$

$$x_{ip}^k + x_{pi}^k = 1; \quad \forall 1 \le i < p \le n_k, \forall k \tag{5}$$

$$x_{ij}^k = 1; \quad \forall i,j \in V_k, \pi_k(i) < \pi_k(j), \forall k \tag{6}$$

$$x_{ij}^k, C_{ijpq}^k \in \{0,1\}; \quad \forall (i,j),(q,p) \in IE_k, \forall k \tag{7}$$

where C_{ijpq}^k denotes the crossing variable that takes the value 1 if edges (i,j) and (p,q) cross. Additionally, i and p are vertices that reside in the same layer k, and j, q are vertices that reside in the next layer $k+1$ ($k = 1, \ldots, K-1$), where x_{ip}^k takes the value 1 if vertex i precedes vertex p

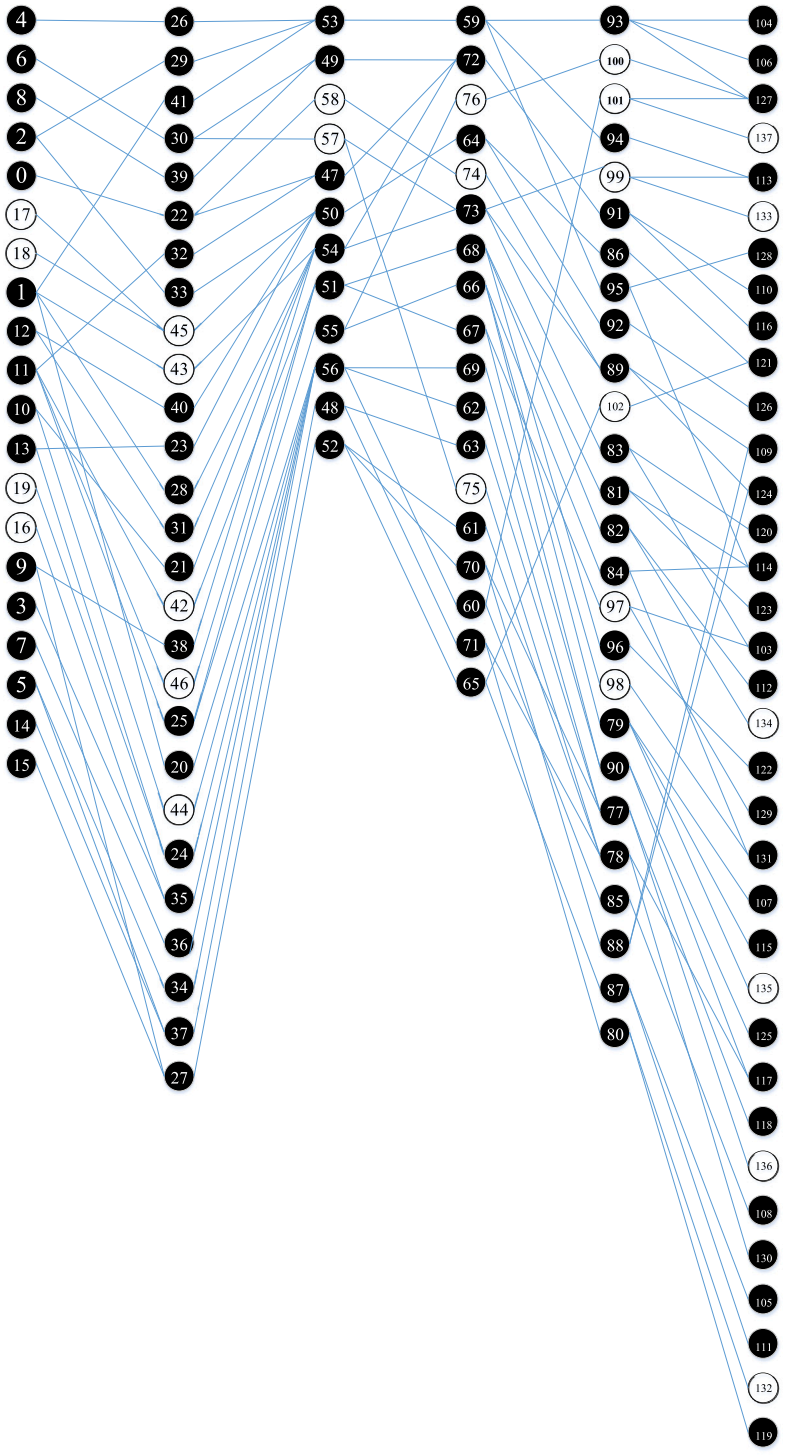


Fig. 2. The optimal drawing for an example (incgraph_6_0.06_5_30_1.20_1 instance) with a six-layer graph. The number of edge crossings for this drawing is 172.

and 0 otherwise. IE_k denotes the set of edges from the vertices in layer k to the vertices in layer $k+1$. V_k denotes the set of all the original vertices in layer k. Constraints (2) and (3) above force C^k_{ijpq} to take the value of 1 if the variables x^k_{ip} and x^k_{jq} indicate an edge crossing. Constraints (4) are the 3-dicycle inequalities, which guarantee that the ordering variables in fact represent an ordering. Constraints (5) and (6) preserve the ordering of the original vertices. Constraints (7) impose the binary requirements on the problem variables. Table 1 summarizes all the symbols and definitions introduced in this study.

3. Multiple neighborhood solution-based tabu search for the incremental graph drawing problem

The Iterated Local Search (ILS) algorithm is a well-known metaheuristic algorithm used to solve combinatorial optimization problems. It follows an iterative process that combines local search procedure to optimize it locally and perturbation to escape the local optima. Although the proposed MNSB-TS algorithm follows the similar idea by combining the main search process and the dynamic diversification to

Table 1
Symbols and definitions for incremental graph drawing problem.

Symbol	Definition
G	Original graph $G = (V, E, K, L)$
V	Set of original vertices in G
E	Set of original edges in G
$L(u)$	Label of the layer containing vertex u
$\pi_k(u)$	Position of vertex u in its layer k ($k = 1,\ldots,$K) of G
IG	Incremental graph $IG = (IV, IE, K, L)$
V_k	Set of all the original vertices in layer k of IG
IV_k	Set of all the vertices in layer k of IG, and $IV = \bigcup_{k=1}^{K} IV_k$
AV_k	Set of incremental vertices in layer k of IG, and $AV = \bigcup_{k=1}^{K} AV_k$
AE	Set of incremental edges in IG
n_k	Number of vertices in layer k ($k = 1, \ldots,$ K) for IG, and $n = \sum_{k=1}^{K} n_k$
S	Incremental drawing (i.e., solution) $S = (\omega_1, \ldots, \omega_K)$ for IG
$\Pi_k(u)$	Position of vertex u in its layer k of IG
$\omega_k(i)$	Vertex locating ith position in layer k ($k = 1, \ldots,$ K) for IG
$A(u)$	Set of all vertices adjacent to u, i.e., $A(u) = \{v: (u, v) \in IE\}$
$IM(u, v)$	All the intermediate vertices between vertices u and v in the same layer i.e., $IM(u, v) = \{ u' : \Pi(u) < \Pi(u') < \Pi(v)$ or $\Pi(u) > \Pi(u') > \Pi(v);\ L(u) = L(v) = L(u')\}$

explore and improve solutions, the main search process of the MNSB-TS employs various strategies, including multiple neighborhood moves, solution-based tabu strategy based on a hash mechanism. Compared to the simple local search component in ILS, it has the ability to perform global search.

Specifically, the main search process of the proposed multiple neighborhood solution-based tabu search provides a stronger form of intensification than attribute-based tabu search, which has been shown to be valuable for locating good local optima in certain settings. Our solution-based tabu strategy utilizes hash functions and hash vectors to store the information encoding a solution to make it tabu, instead of recording a simpler solution attribute such as values assigned to one or two variables which can prevent a range of solutions from being re-visited. By joining the solution-based TS approach with a multiple neighborhood strategy, we are additionally able to take advantage of a stronger diversification ability, thereby searching a broader region of the search space in quest of improved solutions. The key components of our MNSB-TS algorithm are presented in the following subsections.

3.1. Form of the multiple neighborhood solution-based tabu search algorithm

Algorithm 1: Framework of the multiple neighborhood solution-based tabu search algorithm for solving incremental graph drawing problem

Input: An incremental graph (G); The maximum computing time (T_{max})
Output: Best-found solution (S^{best})
1 $S \leftarrow GenerateInitialSolution(G)$;
2 $S^{best} \leftarrow S$, $S^* \leftarrow S$;
3 $\xi \leftarrow 0$;
4 **while** *The maximum computing time T_{max} is not reached* **do**
5 $S_p \leftarrow S^*$;
6 $S^* \leftarrow MainSearch(S)$;
// Update the best solution S^{best} found so far, and the counter ξ of consecutive non-improving best found solution
7 **if** S^* *is better than* S^{best} **then**
8 $S^{best} \leftarrow S^*$;
9 $\xi \leftarrow 0$;
10 **else**
11 $\xi \leftarrow \xi + 1$;
12 **end**
13 $S \leftarrow DynamicDiversification(S^*, S_p, \xi)$;
14 **end**
15 **return** S^{best}

The general scheme of MNSB-TS presented in Algorithm 1 can be described as follows: Starting from an initial solution created by a dedicated constructive procedure (line 1 and Section 3.2), MNSB-TS initializes the parameters consisting of the best-found solution S^{best}, the locally optimal solution S^* and the counter ξ for consecutive non-improving best found solutions (lines 2–3). After the initial phase, the method records the previous locally optimal solution S_p and then employs a $MainSearch$ phase to reach a new local optimum S^* (lines 5–6, Section 3.3). Then, the method updates the record of the best solution S^{best} found so far and the counter ξ of consecutive non-improving local optima (lines 7–12). Following this, our diversification operator is employed to enter a new search space, determining the diversification strength by reference to the current local optimum S^* and the previous local optimum S_p. The main search phase and the dynamic diversification procedure iteratively alternate until the stopping criterion (i.e., the maximum computing time T_{max}) is met, returning the best found solution S^{best} as the final result (line 15).

3.2. Initial solution phase

Our initial solution procedure is adapted from the construction procedure for bipartite graphs proposed by Martí et al. (2018b) in their tabu search algorithm for the dynamic bipartite drawing problem. Starting from a partial solution only containing the original vertices, the procedure iteratively inserts each incremental vertex into the initial partial solution to make it complete, as presented in Algorithm 2. Specifically, we first generate the initial partial solution S_0 consisting of the permutations π_k ($k = 1, \ldots, K$) of original vertices in each layer, and assign incremental vertices AV_k of each layer to the incremental vertex set AV (lines 1–3). After that, we iteratively insert each incremental vertex v from AV into the position p of the partial solution S_0 until S_0 becomes complete (lines 4–10).

At each iteration, the set CL records all the pairs of remaining vertices in AV and their corresponding feasible positions (line 5). The restricted candidate list RCL is then created, which contains the $\max\{1, \alpha * |CL|\}$ vertices with the minimum incremental objective value δ in CL (line 6). We next randomly choose a vertex v_c and insert it into the determined position p_c (line 7). The partial solution S_0 and the remaining vertex set AV are updated in lines 8–9. When the incremental vertex set AV becomes empty, the initial solution phase terminates and the complete solution S_0 is returned as the final solution generated in the initial solution phase (line 11).

3.3. Main search phase of the multiple neighborhood solution-based tabu search

Our MNSB-TS algorithm employs a hybrid search procedure by combining a multiple neighborhood structure with the solution-based tabu search strategy. The details of the solution-based tabu search phase are depicted in the pseudo-code given in Algorithm 3.

The hash lists (i.e., hash vectors HV_k ($k = 1, \ldots, K$)) are initialized only once in the algorithm (lines 1–8). Then, we initialize two counters of iterations (i.e., $NoImproveIter$ and $NeighborAlterIter$) to 0 (line 9).

Algorithm 2: Initial solution phase $GenerateInitialSolution(B)$

Input: An incremental graph (G);
Output: Initial solution (S_0)
1 Generate the permutations π_k of original vertices and incremental vertices AV_k in each layer in an incremental graph (G), where $k = \{1, \dots, K\}$;
2 $S_0 = (\pi_1, \dots, \pi_K)$;
3 $AV \leftarrow \bigcup_{k=\{1,\dots,K\}} AV_k$;
4 **while** *Solution* S_0 *is not a complete solution i.e.,* $|AV| > 0$ **do**
5 $CL \leftarrow$ $\{(v,p) : v \in AV, p$ *denotes each feasible position of incremental vertex* $v\}$;
6 $RCL \leftarrow \{(v,p) \in CL : |\{(v',p') : \delta(v',p') \le \delta(v,p)\}| \le max\{1, \alpha * |CL|\}\}, \delta(v,p)$ *denotes the incremental objective value when vertice* v *locates position* p;
7 Randomly choose a pair (v_c,p_c) and insert vertex v_c into the position p_c (i.e., (v_c, p_c) $\in RCL$);
8 $S_0 \leftarrow S_0 \oplus (v_c, p_c)$;
9 $AV \leftarrow AV \setminus \{v_c\}$;
10 **end**
11 **return** S_0

Algorithm 3: The main search phase of MNSB-TS

Input: Initial solution S, hash vectors HV_k $(k = 1, \dots, K)$ of length λ, depth Θ of tabu search
Output: The local optima S^* found so far
// Initialize the hash vectors once in the MNSB-TS algorithm.
1 **if** *Initial_hash_bool is False* **then**
2 **for** $k \leftarrow 1$ *to* K **do**
3 **for** $i \leftarrow 0$ *to* $\lambda - 1$ **do**
4 $HV_k[i] \leftarrow 0$;
5 **end**
6 **end**
7 *Initial_hash_bool* $\leftarrow$ True;
8 **end**
9 $NoImproveIter \leftarrow 0$, $NeighborAlterIter \leftarrow 0$;
10 $S^* \leftarrow S$, t $\leftarrow 1$;
// Main search procedure
11 **while** *The maximum number of consecutive non-improving local optima is not reached (i.e.,* $NoImproveIter \le \Theta$*)* **do**
// Find a best neighborhood solution S' that is not tabu. The objective value of the neighborhood solutions based on four neighborhood moves (i.e., $N_1 - N_4$) can be calculated according to Equations (13), (14), (15) and (16).
12 $S' = \{\omega'_1, \dots, \omega'_K\} \leftarrow$ $\arg\min_{f(s)}\{s = \{\omega^s_1, \dots, \omega^s_K\} \in N_t(S) : \prod_{1 \le k \le K} HV_k(hf_k(s)) = 0\}$;
13 $S \leftarrow S'$;
14 **if** $f(S') < f(S^*)$ **then**
15 $f(S^*) = f(S')$;
16 $NoImproveIter \leftarrow 0$, $NeighborAlterIter \leftarrow 0$;
17 **else**
18 $NoImproveIter \leftarrow NoImproveIter + 1$;
19 $NeighborAlterIter \leftarrow NeighborAlterIter + 1$;
20 **end**
21 **if** $NeighborAlterIter > \theta_t$ **then**
22 t $\leftarrow$ t + 1;
23 $NeighborAlterIter \leftarrow 0$;
24 **end**
25 **if** $t > t_{max}$ **then**
26 t $\leftarrow 1$;
27 **end**
// The hash function of the chosen neighborhood solution S' can be quickly calculated according to Equations (19) and (20), assuming that Q denotes the layer of the moving vertex (or vertices).
28 $HV_Q(hf_Q(S')) = 1$;
29 **end**
30 **return** S^*

The consecutive non-improving local optima counter $NoImproveIter$ in the current main search determines when to end the main search and start the next diversification procedure in Section 3.4, while the consecutive non-improving local optima counter $NeighborAlterIter$ for each neighborhood move determines when to end the search based on the current neighborhood operator (such as N_1 (or N_2)) and start another type of neighborhood move (such as N_2 (or N_3)). The local optimum S^* is initialized to the current solution, and the neighborhood move counter t, which is initialized to 1 identifies the N_1 operator as the first neighborhood operator (line 10). After that, the procedure performs a series of iterations to improve the current solution until the consecutive non-improving local optima value $NoImproveIter$, reaches the maximum threshold Θ (lines 11–29). At each iteration, the algorithm replaces the current solution S by a best non-tabu neighbor solution S' identified by the chosen neighborhood operator $N_t(S)$ (for $t \in \{1, 2, \dots, t_{max}\}$, where t_{max} denotes the number of neighborhood move operators) according to the proposed tabu rule in Section 3.3.3 (lines 12–13). During the search, the values of the counters $NeighborAlterIter$ and $NoImproveIter$ are reset to 0, respectively, when a new local optimum is encountered, and S^* is updated each time a better solution is found (lines 14–16). Otherwise, both the counters $NeighborAlterIter$ and $NoImproveIter$ increase by 1. The neighborhoods alternate by the token-ring pattern (i.e., $N_1 - N_2 - N_3 - N_4 - N_1 \dots$) when the counter $NeighborAlterIter$ reaches the threshold θ_t of each neighborhood operator N_t (lines 21–27). The objective values of the neighborhood solutions based on the neighborhood moves are calculated according to Eqs. (13), (14), (15) and (16). Accordingly, the hash vectors are updated by the new solution where Q denotes the layer where the moving vertex resides (lines 28). Finally, the algorithm terminates if the threshold Θ is reached, and then returns the best local optimum S^* found during this search process (line 30).

3.3.1. Neighborhood structure

To move from one solution to another, the previous work proposed by Sánchez-Oro et al. (2017) only considerers moving one vertex from its current position, the so-called barycenter, to its antecedent position or posterior position (Battista et al., 1998). We consider this neighborhood structure to be too limited to search efficiently and employ four neighborhood operators based on an incremental graph $IG = (IV, IE, K, L)$, as follows:

- Neighborhood operator N_1: Insert an incremental vertex v_a upwards to occupy the position of another vertex u if vertex u is located above v_a, or insert v_a downwards to occupy the position of u if vertex u is located below v_a. Let v_a be an incremental vertex and u be any other vertex in the same layer $L(v_a)$. Then N_1 is defined as follows:

$$N_1(S) = \{S \oplus Insert(v_a, u) : v_a \in AV, u \in IV, v_a \neq u, L(v_a) = L(u)\} \tag{8}$$

- Neighborhood operator N_2: Swap the positions of two incremental vertices denoted v_a and u_a, that are located in the same layer by the operation:

$$N_2(S) = \{S \oplus Swap(v_a, u_a) : v_a \in AV, u_a \in AV, v_a \neq u_a, L(v_a) = L(u_a)\} \tag{9}$$

- Neighborhood operator N_3: Insert an original vertex v_o upwards to occupy the position of another incremental vertex u_a if vertex u_a is located above v_o or insert v_o downwards to occupy the position of u_a if vertex u_a is located below v_o by keeping the relative positions of the original vertices unchanged (i.e., satisfying the Constraints (6)). Denoting the two vertices located in the same layer that are to be moved by v_o and u_a, then the corresponding neighborhood operator can be written as follows:

$$N_3(S) = \{S \oplus Insert(v_o, u_a) : v_o \in V, u_a \in AV, L(v_o) = L(u_a); IM(v_o, u_a) \cap V = \emptyset\} \tag{10}$$

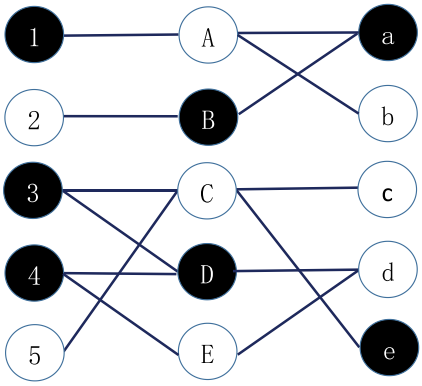


Fig. 3. An example for one drawing for a three-layered graph with 15 vertices.

- Neighborhood operator N_4: Swap the positions of one original vertex and any chosen incremental vertex by keeping the relative positions of original vertices unchanged (i.e., satisfying the Constraints (6)). Let v_o be an original vertex and u_a be any selected incremental vertex in the same layer $L(v_o)$, then N_4 is defined by:

$$N_4(S) = \{S \oplus Swap(v_o, u_a) : v_o \in V, u_a \in AV, L(v_o) = L(u_a); \\ IM(v_o, u_a) \cap V = \emptyset\} \tag{11}$$

It may be noted that when two vertices are adjacent, the swap move (N_2 or N_4) is the same as an insert move (N_1 or N_3).

3.3.2. Incremental evaluation mechanism for the neighborhood structure

As stated above, the four proposed neighborhood moves give rise to several candidate neighborhood solutions to be evaluated. After making the neighborhood moves, the crossing states among most edges in the graph remain unchanged and it is not necessary to recalculate the sum of the crossing numbers from scratch. To efficiently evaluate the changed objective values produced by the candidate neighborhood moves, we utilize dedicated matrices to record the relevant information associated with the number of edge crossings for two vertices.

Given two vertices u and v in the same layer of the graph, let $NC_k(u, v)$ denote the number of crossings for the edges incident to vertex u and the edges incident to vertex v when u precedes v, where only the vertices incident to u and v located in the layer k will be considered (i.e., $\Pi(u) < \Pi(v)$, $k = L(A(v)) = L(A(u))$) (and $NC_k(v, u)$ when v precedes u). The matrices MX_k in the kth layer thus can be given by:

$$MX_k(u, v) = \begin{cases} NC_{k+1}(u, v), & \text{if } k = 1; \\ NC_{k-1}(u, v) + NC_{k+1}(u, v), & \text{if } 1 < k < K; \\ NC_{k-1}(u, v), & \text{if } k = K; \end{cases} \tag{12}$$

Fig. 4 depicts the matrices of the example in Fig. 3 to illustrate how they save the information of the number of edge crossing for each pair of vertices. The first matrix MX_1 records the number of crossings for the edges incident to the vertices in the first layer. We observe that the number of crossings for the edge incident to vertex 1 (i.e., (1, A)) and the edge incident to vertex 2 (i.e., (2, B)) is 0, since edge (1, A) does not cross edge (2, B). Therefore, $NC_2(1, 2) = 0$ and $MX_1(1, 2) = 0$, as shown in Fig. 4(a) (row 1, column 2 in matrix MX_1). On the other hand, edge (1, A) would cross edge (2, B) if we insert vertex 2 upwards to lie above vertex 1, (i.e., after vertex 2 precedes vertex 1). That is why, in Fig. 4(a), the value of row 2 and column 1 in matrix MX_1 is 1, which means the number of crossings for the edge incident to 1 and the edge incident to 2 is 1, when vertex 2 precedes vertex 1 in the first layer.

Similarly, the second matrix MX_2 records the number of crossings for the edges incident to the vertices in the second layer. $NC_1(A, B) = 0$ since the edge incident to vertex A (i.e., (1, A)) does not cross the edge incident to vertex B (i.e., (2, B)) in the first layer, while the number of crossings for two edges incident to vertex A (i.e., (A, a) and (A, b)) and the edge incident to vertex B (i.e., (B, a)) in the third layer is 1. Therefore, $NC_3(A, B) = 1$ and $MX_2(1, 2) = NC_1(1, 2) + NC_3(1, 2) = 1$, as shown in Fig. 4(b) (row 1, column 2 in matrix MX_2). On the other hand, edge (B, a) would not cross edges (A, a) and (A, b) if we insert vertex B above (before) vertex A. Edge (2, B) would cross edge (1, A) when vertex B precedes vertex A. Therefore, the number of crossings for the edges incident to A and the edges incident to B is 1. That is why, in Fig. 4(b), the value of row 2 and column 1 in matrix MX_2 is 1. The third matrix is constructed in a similar way. Note that the precedence relationships among the original vertices must be kept and so $MX(u, v), \forall u, v \in V$ is denoted by the symbol ' -', which stands for "don't care".

Making use of these matrices, we propose a fast neighborhood evaluation mechanism to efficiently obtain the objective value of each move. Next, we take an example (i.e., insert vertex v immediately before (above) u_{j-1} or u_j) as shown in Fig. 5 to illustrate the evaluation procedure. Comparing these two moves, the positions of the vertices marked in the two black boxes are the same, hence we only need to consider the difference between u_{j-1} and v to give the changed objective value with the two neighborhood moves. The current solution can be denoted by S and the objective value of its neighboring solution can be denoted by $f(S \oplus Insert(v, u_{j-1}))$ when vertex v is inserted immediately before u_{j-1}. The latter becomes $f(S \oplus Insert(v, u_j))$ after vertex v is inserted immediately before u_j, and we calculate the objective value $f(S \oplus Insert(v, u_j))$ of the resulting solution based on the neighborhood move $Insert(v, u_j)$ by the following formula:

$$f(S \oplus Insert(v, u_j)) = f(S \oplus Insert(v, u_{j-1})) \\ - MX_{L(v)}(u_j, v) + MX_{L(v)}(v, u_j) \tag{13}$$

where $MX_{L(v)}$ denotes the corresponding evaluation matrix involving vertex v.

To efficiently evaluate all the neighboring solutions, we first calculate the objective value (i.e., $f(S \oplus Insert(v, u_1))$) of the neighboring solution when vertex v is moved to the previous position of vertex u_1 from its current position, according to the following Equation:

$$f(S \oplus Insert(v, u_1)) = f(S) - MX_{L(v)}(u_1, v) + MX_{L(v)}(v, u_1) \tag{14}$$

Afterwards, by making use of Eq. (13) iteratively, we can quickly calculate other objective values when vertex v is inserted immediately before other vertices. Likewise, the evaluation of the objective value produced by downward moves of vertex v (placing it immediately after other vertices) can be iteratively calculated in a similar manner.

As for the swap move, the move of swapping any two vertices clearly is equivalent to two insert operations. Without loss of generality, we assume that node u is located above vertex v, and vertex z_1 is the adjacent vertex above vertex v, as shown in Fig. 6. The move $swap(v, u)$ can be seen to result by first inserting vertex v upwards to lie above vertex u (i.e., $insert(v, u)$), and then inserting vertex u downwards to lie below vertex z_1 (i.e., $Insert(u, z_1)$). The objective value of the neighboring solution generated by the move $swap(v, u)$ can be calculated by:

$$f(S \oplus Swap(v, u)) = f(S \oplus Insert(v, u) \oplus Insert(u, z_1)) \\ = f(S) + \Delta_{insert}(v, u) + \Delta_{insert}(u, z_1) \tag{15}$$

where nodes v, u and z_1 are located in the same layer (i.e., $L(v) = L(u) = L(z_1)$), vertex z_1 precedes v (i.e., $\Pi(z_1) + 1 = \Pi(v)$), and $\Delta_{insert}(v, u)$ denotes the incremental objective function value which can be calculated by:

$$\Delta_{insert}(v, u) = f(S \oplus insert(v, u)) - f(S) \tag{16}$$

$$MX_1 = \begin{bmatrix} - & 0 & - & - & 0 \\ 1 & - & 0 & 0 & 0 \\ - & 2 & - & - & 1 \\ - & 2 & - & - & 2 \\ 1 & 1 & 0 & 0 & - \end{bmatrix} \quad MX_2 = \begin{bmatrix} - & 1 & 0 & 0 & 0 \\ 1 & - & 0 & - & 0 \\ 6 & 4 & - & 3 & 2 \\ 4 & - & 2 & - & - \\ 3 & 2 & 2 & 1 & - \end{bmatrix} \quad MX_3 = \begin{bmatrix} - & 1 & 0 & 0 & - \\ 0 & - & 0 & 0 & 0 \\ 2 & 1 & - & 0 & 0 \\ 4 & 2 & 2 & - & 2 \\ - & 1 & 0 & 0 & - \end{bmatrix}$$

(a) The first matrix for the first layer. (b) The second matrix for the second layer. (c) The third matrix for the third layer.

Fig. 4. The number of edge crossing for pairs of vertices in matrices.

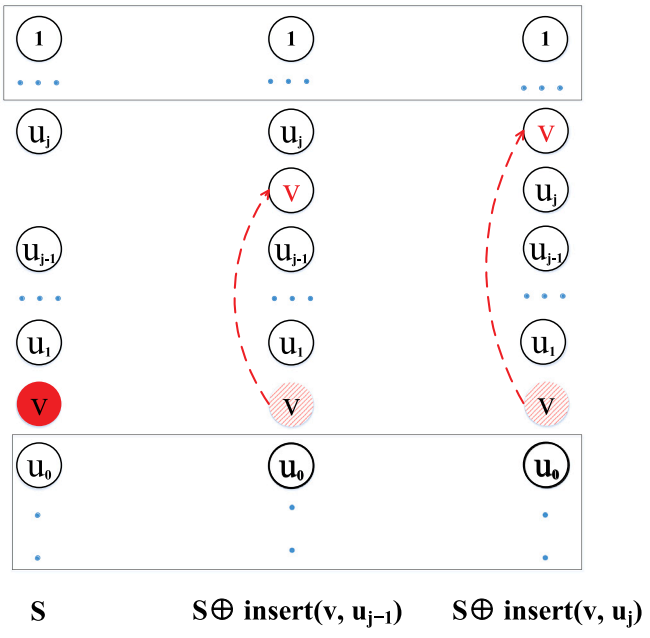


Fig. 5. Insert vertex v immediately before (above) vertex u_{j-1} or u_j.

Therefore, we can quickly evaluate the objective function value of the neighbor solutions obtained by the insert and swap moves. Any vertex v inserted upwards (or downwards) relative to vertex u (depending on the position relationship between vertices v and u) generates at most n_i^2 neighboring solutions, where i denotes the layer label of vertices u and v, because only incremental vertices can be placed in all possible positions. Each candidate solution can be evaluated by the incremental evaluation mechanism, utilizing the result of overlapping sub-problems. The time complexity of the incremental evaluation mechanism for the neighborhood structure thus is $O(\sum_{1\le k\le K} n_k^2)$.

After evaluating all the neighboring solutions based on the matrices, only one best neighborhood move is chosen to be executed. Then, the proposed matrix mechanism only needs to update at most three matrices corresponding to the layer of the chosen moved vertex and its adjacent layers, respectively. Therefore, the cost to update the matrices is not large. Meanwhile, the computational advantage of the proposed mechanism increases as the number of layers in the graph increases.

3.3.3. Solution-based tabu strategy

To utilize the solution-based tabu strategy to determine the tabu status of neighborhood solutions, we employ several hash vectors HV_k of the same length of λ as the tabu list, where each position represents a binary variable, and each hash vector is associated with a hash function hf_k. The hash functions map a candidate solution S of the search space Ω to a K-dimensional index of hash vectors as follows:

$$hf : S \in \Omega \to \{0, 1, 2, \dots, \lambda - 1\}^K \tag{17}$$

Based on the hash vectors and the corresponding hash functions, we determine the tabu status of candidate solutions by the following rule. Given a candidate solution $S = (\omega_1, \dots, \omega_K)$, the hash vector HV_k and the associated hash functions hf_k, S is identified as a tabu solution if $\prod_{1\le k\le K} HV_k(hf_k(S)) = 1$. Otherwise, S is identified as a non-tabu solution.

Hash functions previously proposed for binary decision variables in the literature (Lai et al., 2018a, 2018b, 2018c; Wang et al., 2017) are quite suitable for the problems they address. Unlike these binary optimization problems, however, permutation problems constitute a special class that preferably should be treated somewhat differently in order to distinguish different solutions with a high probability while permitting the hash values of the neighbor solutions to be easily evaluated. In this study, we propose a simple but effective hash function for IGDP, which is also applicable to other multi-layer permutation problems. Formally, given a candidate solution $S = \{\omega_1, \dots, \omega_K\}$, our hash function hf_k for each layer of solution S can be represented by:

$$hf_k(S) = \left(\sum_{i=2}^{n_k} (\omega_k(i) - \omega_k(i-1))^{\Gamma}\right) \mod \lambda \tag{18}$$

where $\omega_k(i)$ denotes the vertex located in the ith position in layer k, parameter Γ is used to define the hash function and λ is the length of hash vector that is empirically set to 10^7 in this work.

For a specific solution S and its hash values $hf_k(S)$ for all layers, it is intuitively clear that we only need to evaluate the hash value of the layer where the neighbor solutions are located, while the hash values for the other layers remain unchanged. For the example depicted in Fig. 5 (i.e., inserting vertex v upwards above vertex u_{j-1}), a hash value of solution S for the layer where the *insert* move (N_1 or N_3 move) locates vertex v can be quickly calculated according to the following Equation:

$$\begin{aligned} &hf_{L(v)}(S \oplus insert(v, u_{j-1})) \\ &= (hf_{L(v)}(S) - (v - u_1)^{\Gamma} - (u_0 - v)^{\Gamma} + (u_0 - u_1)^{\Gamma} \\ &\quad + (v - u_j)^{\Gamma} + (u_{j-1} - v)^{\Gamma} - (u_{j-1} - u_j)^{\Gamma}) \mod \lambda \end{aligned} \tag{19}$$

For the example illustrated in Fig. 6 (i.e., swapping vertex v with vertex u), a hash value of solution S for the layer where the *swap* move (N_2 or N_4 move) locating can be quickly calculated by:

$$\begin{aligned} hf_{L(v)}(S \oplus swap(v,u)) =& (hf_{L(v)}(S) - (z_0 - v)^{\Gamma} - (v - z_1)^{\Gamma} - (z_2 - u)^{\Gamma} \\ &- (u - z_3)^{\Gamma} + (z_0 - u)^{\Gamma} + (u - z_1)^{\Gamma} \\ &+ (z_2 - v)^{\Gamma} + (v - z_3)^{\Gamma}) \\ &\mod \lambda \end{aligned} \tag{20}$$

Thus, the time complexity of determining the tabu status of a neighbor solution is $O(K)$.

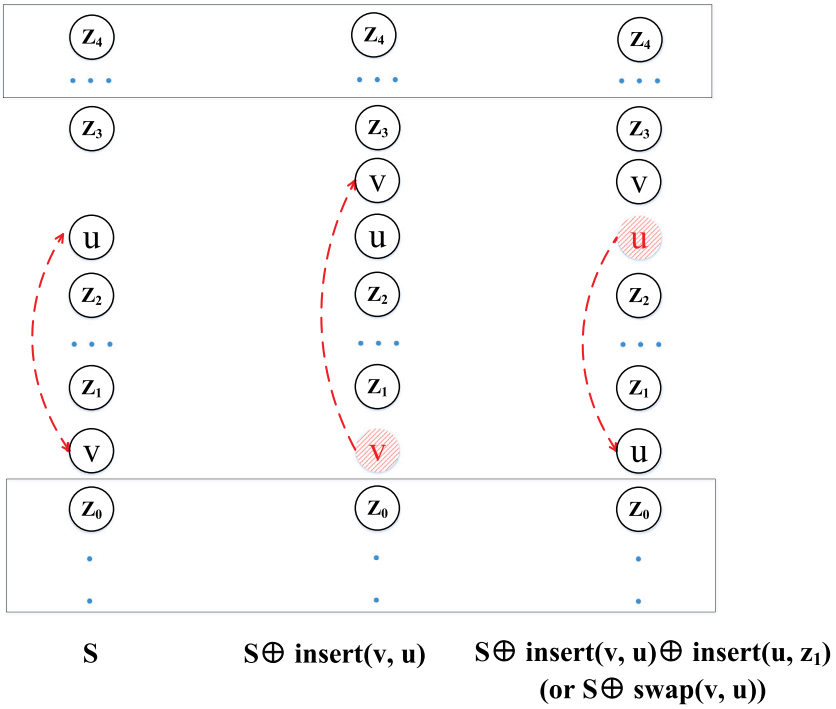


Fig. 6. Swap vertex v with vertex u.

3.4. Dynamic diversification mechanism

The diversification procedure is called whenever no improvement in the current solution has been made after a preset number of iterations. The purpose of the diversification mechanism thus is to allow MNSB-TS to escape from the current local optimum in order to discover other local optima of better solution quality. For this, we utilize a dynamic diversification mechanism which varies the diversification intensity, depending on the search status.

We present the diversification phase in Algorithm 4, which first determines the diversification strength l according to the current search state (lines 1–8), and then modify the current local optimum S by reconstructing it using the selected diversification strength l (lines 9–16). More precisely, the dynamic diversification phase re-initializes the diversification strength with a weak strength l_{min} if the search has escaped from the previous local optimum S_p (lines 1–2). However, if the diversification phase did not succeed in escaping from the previous optimum, the diversification strength l is incremented by 1 and the diversification is applied again to the current attractor (lines 3–4). A strong diversification with a large strength value l_{max} is carried out only after visiting a certain number ξ_{max} of local optima without any improvement in the quality of the best solution found (lines 5–8). After the determination of the diversification strength, we first generate a partial solution S_u by randomly removing a number l of incremental vertices RV from solution S, and then iteratively insert the selected vertex into the solution S_u to make it complete by using a rule similar to the one presented in Section 3.2 (lines 10–16). The value of β can be set at l/l_{max} to balance the tradeoff between greedy and random strategies according to the diversification strength. When l is increased, the procedure tends to be more random, and when decreased, it tends to be greedier. We randomly choose a vertex v_c and insert it into the chosen position p_c (lines 13 and 14). The partial solution S_u and the remaining vertex set RV are updated in line 15. When the vertex set RV becomes empty, the diversification procedure terminates and the complete solution S_u is returned as the final solution generated in the diversification phase (line 17).

Algorithm 4: Dynamic diversification operator $DynamicDiversification(S, S_p, \xi)$

Input: The current solution S, the previous local optimum S_p, the counter ξ for consecutive non-improving best found solutions
Output: A new solution S_u after diversification
// Determine the diversification strength l adaptively
1 **if** S *is better than* S_p **then**
  // Search escaped from the previous local optimum, re-initialize diversification strength
2 $l \leftarrow l_{min}$;
3 **else if** S *is not better than* S_p *and* $\xi < \xi_{max}$ **then**
  // Search returns to the previous local optimum, increment diversification strength
4 $l \leftarrow Min(l+1, l_{max})$;
5 **else if** $\xi \geq \xi_{max}$ **then**
  // Search seems to be stagnating, strong diversification required
6 $l \leftarrow l_{max}$;
7 $\xi \leftarrow 0$;
8 **end**
// Modify the current local optimum S using the selected diversification strength l
9 Generate a partial solution S_u by randomly removing a number l of incremental vertices RV from solution S;
10 **while** *the solution* S_u *is not a complete solution, i.e.,* $|RV| > 0$ **do**
11 $CL \leftarrow \{(v,p) : v \in RV, p \text{ denotes each feasible position of incremental vertex } v\}$;
12 $RCL \leftarrow \{(v,p) \in CL : |\{(v',p') : \delta(v',p') \leq \delta(v,p)\}| \leq max\{1, \beta * |CL|\}\}$;
13 Randomly choose a pair (v_c, p_c) and insert the vertex v_c into the position p_c (i.e., $(v_c, p_c) \in RCL$);
14 $S_u \leftarrow S_u \oplus (v_c, p_c)$;
15 $RV \leftarrow RV \setminus \{v_c\}$;
16 **end**
17 **return** S_u

In general, the proposed dynamic diversification phase is controlled by the parameter l of the jump magnitude by determining the number of removed vertices and the balance ratio between random and greedy strategies in the modification procedure, i.e., the diversification becomes stronger as the magnitude l becomes larger.

Table 2
Settings of the parameters used in MNSB-TS.

Parameter	Description	Candidate values	Final value
α	The size of the restricted candidate list in initial solution phase	1/2, 1/3, 1/5	1/3
l_{min}	Minimum diversification strength	($\|AV\|/3$, $\|AV\|/4$, $\|AV\|/5$)	$\|AV\|/3$
l_{max}	Maximum diversification strength	($\|AV\|$, $\|AV\|/1.3$, $\|AV\|/2$)	$\|AV\|/1.3$
Θ	The maximum threshold of iterations without improving the local optimum in the main search phase	(30×n, 45×n,60×n)	45*n
$\theta_1 - \theta_4$	The maximum threshold of iterations without improving the local optima for four neighborhoods (i.e., $N_1 - N_4$), respectively	(0,1)×Θ	[$\theta_1 = 0.3846 \times \Theta$, $\theta_2 = 0.51282 \times \Theta$, $\theta_3 = 0.0513 \times \Theta$, $\theta_4 = 0.0513 \times \Theta$]
ξ_{max}	The maximum threshold of iterations without improving the best solution found	(2000,3000,4000)	3000
Γ	The parameter in the hash function	(1.2,2,4)	4

Table 3
Results for the performance of MNSB-TS compared with the reference heuristics (GRASP, VNSS and SS) and the general solver Gurobi for the first instance set with 2 layers where the number of vertices for each instance is between 47 and 54.

Instance name	Gurobi		GRASP		SS		VNSS		MNSB-TS(1 run)		MNSB-TS(10 runs)				
	Cross	Time	Cross	Time	Cross	Time	Cross	Time	Cross	Time	f_{best}	f_{avg}	T_{avg}	ARD (%)	SR(10)
incgraph_2.0.06.5_30_1.20_1	**7**	5.13	**7**	0.02	**7**	0.02	**7**	0.02	**7**	0	**7**	7	0	0	10
incgraph_2.0.06.5_30_1.20_2	**11**	7.36	69	0.52	69	0.03	69	0.05	**11**	0	**11**	11	0	0	10
incgraph_2.0.06.5_30_1.20_3	**31**	8.61	116	0.21	36	0.02	116	0.05	**31**	0	**31**	31	0	0	10
incgraph_2.0.06.5_30_1.20_4	**16**	7.05	20	0.1	**16**	0.02	19	0.02	**16**	0	**16**	16	0	0	10
incgraph_2.0.06.5_30_1.20_5	**1**	0.93	3	0	3	0	3	0	**1**	0	**1**	1	0	0	10
incgraph_2.0.06.5_30_1.20_6	**9**	8.11	36	0.68	36	0.05	16	0.05	**9**	0	**9**	9	0	0	10
incgraph_2.0.06.5_30_1.20_7	**14**	8.33	28	0.4	28	0.05	28	0.07	**14**	0	**14**	14	0	0	10
incgraph_2.0.06.5_30_1.20_8	**0**	2	2	0.01	2	0.02	2	0	**0**	0	**0**	0	0	0	10
incgraph_2.0.06.5_30_1.20_9	**0**	4.52	51	0.27	51	0.04	51	0.05	**0**	0	**0**	0	0	0	10
incgraph_2.0.06.5_30_1.20_10	**3**	3.23	11	0.01	**3**	0.01	**3**	0.02	**3**	0	**3**	3	0	0	10
incgraph_2.0.06.5_30_1.60_1	**7**	7.52	**7**	0.88	**7**	0.05	**7**	0.06	**7**	0	**7**	7	0	0	10
incgraph_2.0.06.5_30_1.60_2	**11**	10.09	27	0.62	27	0.18	16	0.3	**11**	0	**11**	11	0	0	10
incgraph_2.0.06.5_30_1.60_3	**31**	12.18	33	0.48	33	0.1	**31**	0.12	**31**	0.01	**31**	31	0.01	0	10
incgraph_2.0.06.5_30_1.60_4	**19**	9.98	21	0.83	21	0.07	21	0.07	**19**	0.02	**19**	19	0.01	0	10
incgraph_2.0.06.5_30_1.60_5	**2**	1.1	6	0.44	6	0.02	3	0.03	**2**	0	**2**	2	0	0	10
incgraph_2.0.06.5_30_1.60_6	**9**	13.3	10	0.13	10	0.17	10	0.32	**9**	0.01	**9**	9	0.01	0	10
incgraph_2.0.06.5_30_1.60_7	**15**	14.33	115	0.52	20	0.23	115	0.33	**15**	0.00	**15**	15	0.01	0	10
incgraph_2.0.06.5_30_1.60_8	**1**	2.75	**1**	0.65	**1**	0.02	**1**	0.02	**1**	0	**1**	1	0	0	10
incgraph_2.0.06.5_30_1.60_9	**0**	6.91	90	0.23	90	0.15	90	0.15	**0**	0	**0**	0	0.00	0	10
incgraph_2.0.06.5_30_1.60_10	**4**	4.91	5	0.18	5	0.04	5	0.07	**4**	0.02	**4**	4	0.01	0	10
incgraph_2.0.17.5_30_1.20_1	**371**	3.71	**371**	0.54	**371**	0.02	**371**	0.02	**371**	0	**371**	371	0	0	10
incgraph_2.0.17.5_30_1.20_2	**396**	4.77	508	0.19	508	0.06	508	0.08	**396**	0	**396**	396	0	0	10
incgraph_2.0.17.5_30_1.20_3	**404**	5.72	603	0.83	409	0.03	586	0.04	**404**	0	**404**	404	0	0	10
incgraph_2.0.17.5_30_1.20_4	**993**	6.07	1022	0.72	1003	0.03	1022	0.05	**993**	0	**993**	993	0	0	10
incgraph_2.0.17.5_30_1.20_5	**12**	0.49	17	0.79	17	0	17	0	**12**	0	**12**	12	0	0	10
incgraph_2.0.17.5_30_1.20_6	**1316**	8.78	1398	0.23	1398	0.02	1398	0.06	**1316**	0	**1316**	1316	0	0	10
incgraph_2.0.17.5_30_1.20_7	**1539**	10.05	1701	0.84	1562	0.1	1653	0.09	**1539**	0	**1539**	1539	0	0	10
incgraph_2.0.17.5_30_1.20_8	**74**	1.47	79	0.93	79	0.02	79	0.02	**74**	0	**74**	74	0	0	10
incgraph_2.0.17.5_30_1.20_9	**0**	2.98	51	0.78	51	0.03	51	0.04	**0**	0	**0**	0	0	0	10
incgraph_2.0.17.5_30_1.20_10	**102**	2.63	132	0.27	132	0.02	132	0.02	**102**	0	**102**	102	0	0	10
incgraph_2.0.17.5_30_1.60_1	**727**	9.23	750	0.79	744	0.12	**727**	0.23	**727**	0.05	**727**	727	0.02	0	10
incgraph_2.0.17.5_30_1.60_2	**737**	11.36	792	0.15	783	0.32	**737**	0.54	**737**	0.02	**737**	737	0.03	0	10
incgraph_2.0.17.5_30_1.60_3	**740**	13.62	802	0.47	793	0.2	**740**	0.29	**740**	0.07	**740**	740	0.05	0	10
incgraph_2.0.17.5_30_1.60_4	**1852**	13.22	1880	0.81	1875	0.2	1870	0.54	**1852**	0.03	**1852**	1852	0.21	0	10
incgraph_2.0.17.5_30_1.60_5	**11**	1.17	12	0.83	12	0.02	12	0.01	**11**	0	**11**	11	0	0	10
incgraph_2.0.17.5_30_1.60_6	**2429**	18.16	2489	0.74	2478	0.4	2435	0.49	**2429**	0.05	**2429**	2429	0.12	0	10
incgraph_2.0.17.5_30_1.60_7	**2742**	20.21	2916	0.88	2844	0.62	**2742**	1.1	**2742**	0.1	**2742**	2742	0.20	0	10
incgraph_2.0.17.5_30_1.60_8	**187**	3.29	**187**	0.39	**187**	0.03	**187**	0.04	**187**	0	**187**	187	0.01	0	10
incgraph_2.0.17.5_30_1.60_9	**0**	6.68	90	0.41	90	0.13	90	0.16	**0**	0	**0**	0	0	0	10
incgraph_2.0.17.5_30_1.60_10	**198**	5.58	200	0.06	200	0.09	200	0.15	**198**	0.88	**198**	198	0.48	0	10
incgraph_2.0.30.5_30_1.20_1	**1965**	6.48	1980	0.3	1970	0.05	2062	0.06	**1965**	0	**1965**	1965	0	0	10
incgraph_2.0.30.5_30_1.20_2	**1863**	7.47	2141	0.72	1873	0.07	2038	0.11	**1863**	0	**1863**	1863	0	0	10
incgraph_2.0.30.5_30_1.20_3	**2172**	8.55	2362	0.75	2362	0.03	2362	0.06	**2172**	0	**2172**	2172	0	0	10
incgraph_2.0.30.5_30_1.20_4	**4180**	11.22	4205	0.91	4188	0.27	**4180**	0.36	**4180**	0	**4180**	4180	0	0	10
incgraph_2.0.30.5_30_1.20_5	**59**	0.66	70	0.02	70	0.01	70	0	**59**	0	**59**	59	0	0	10
incgraph_2.0.30.5_30_1.20_6	**6249**	15.07	6341	0.69	6285	0.22	6266	0.13	**6249**	0.01	**6249**	6249	0	0	10
incgraph_2.0.30.5_30_1.20_7	**6417**	17.45	6750	0.29	6420	0.09	6649	0.14	**6417**	0	**6417**	6417	0	0	10
incgraph_2.0.30.5_30_1.20_8	**571**	2.34	**571**	0.65	**571**	0.03	**571**	0.02	**571**	0	**571**	571	0	0	10
incgraph_2.0.30.5_30_1.20_9	**105**	3.21	185	0.62	185	0.05	185	0.05	**105**	0	**105**	105	0	0	10
incgraph_2.0.30.5_30_1.20_10	**826**	3.77	910	0.87	**826**	0.03	906	0.04	**826**	0	**826**	826	0	0	10
incgraph_2.0.30.5_30_1.60_1	**3635**	13.09	3680	0.2	3671	0.27	3636	0.3	**3635**	0.00	**3635**	3635	0.01	0	10
incgraph_2.0.30.5_30_1.60_2	**3349**	16.25	3521	0.52	3446	0.66	**3349**	1.09	**3349**	0.08	**3349**	3349	0.05	0	10
incgraph_2.0.30.5_30_1.60_3	**3740**	18.33	3822	0.53	3795	0.32	**3740**	0.52	**3740**	0.01	**3740**	3740	0.02	0	10
incgraph_2.0.30.5_30_1.60_4	**7341**	26.03	7482	0.04	7440	0.52	7348	1.1	**7341**	0.5	**7341**	7341	0.27	0	10
incgraph_2.0.30.5_30_1.60_5	**132**	1.39	135	0.24	134	0.02	133	0.03	**132**	0	**132**	132	0	0	10
incgraph_2.0.30.5_30_1.60_6	**10114**	32.93	10398	0.69	10214	0.75	10126	0.85	**10114**	0.04	**10114**	10114	0.05	0	10
incgraph_2.0.30.5_30_1.60_7	**12281**	41.39	12884	0.79	12639	1.6	12305	1.62	**12281**	0.07	**12281**	12281	0.37	0	10
incgraph_2.0.30.5_30_1.60_8	**1169**	5.14	1250	0.7	1221	0.14	1186	0.22	**1169**	0.01	**1169**	1169	0.02	0	10
incgraph_2.0.30.5_30_1.60_9	**208**	7.19	230	0.6	216	0.15	209	0.29	**208**	0.00	**208**	208	0.01	0	10
incgraph_2.0.30.5_30_1.60_10	**1604**	8.75	1719	0.6	1685	0.17	1660	0.29	**1604**	0.08	**1604**	1604	0.11	0	10
#*Avg*	**1383.35**	8.90	1454.9	0.49	1420.3	0.15	1419.18	0.22	**1383.35**	0.04	**1383.35**	1383.35	0.04	0.00	10
#*Best*	**60**		6		9		15		**60**		**60**				

4. Computational studies

In this section, we report extensive computational experiments conducted to assess the performance of MNSB-TS by comparing our algorithm with the best performing heuristics and the exact optimization solver Gurobi on solving public benchmark instances of IGDP.

4.1. Benchmark instances and experimental protocols

We employ the benchmark instances in our experimentation used in the studies Martí and Estruch (2001) and Sánchez-Oro et al. (2017). The hierarchical graphs were generated following the guidelines proposed by Laguna et al. (1997). For each combination of 2, 6, 13, and 20 layers and 0.065, 0.175 and 0.3 graph densities, 20 instances were generated.

Table 4
Results for the performance of MNSB-TS compared with the reference heuristics (GRASP, VNSS and SS) and the general solver Gurobi for the second instance set with 6 layers where the number of vertices for each instance is between 138 and 181.

Instance name	Gurobi		GRASP		SS		VNSS		MNSB-TS(1 run)		MNSB-TS(10 runs)				
	Cross	Time	Cross	Time	Cross	Time	Cross	Time	Cross	Time	f_{best}	f_{avg}	T_{avg}	ARD (%)	SR(10)
incgraph_6_0.06_5_30_1.20_1	**172**	9.23	196	4.78	196	0.25	174	0.42	**172**	0.03	**172**	172	0.02	0.00	10
incgraph_6_0.06_5_30_1.20_2	**264**	11.59	278	3.62	277	0.42	269	0.83	**264**	0	**264**	264	0.01	0.00	10
incgraph_6_0.06_5_30_1.20_3	**446**	15.76	482	3.6	482	0.55	454	0.79	**446**	0.04	**446**	446	0.03	0.00	10
incgraph_6_0.06_5_30_1.20_4	**172**	10.08	204	0.87	204	0.19	189	0.25	**172**	0.03	**172**	172	0.03	0.00	10
incgraph_6_0.06_5_30_1.20_5	**65**	3.79	67	2.44	67	0.07	**65**	0.12	**65**	0	**65**	65	0	0.00	10
incgraph_6_0.06_5_30_1.20_6	**326**	10.32	350	2.78	348	0.26	330	0.37	**326**	0	**326**	326	0	0.00	10
incgraph_6_0.06_5_30_1.20_7	**257**	13.1	303	3.62	296	0.35	279	0.47	**257**	0.02	**257**	257	0.02	0.00	10
incgraph_6_0.06_5_30_1.20_8	**51**	3.81	54	4.72	54	0.1	54	0.12	**51**	0.14	**51**	51	0.14	0.00	10
incgraph_6_0.06_5_30_1.20_9	**153**	8.68	182	2.2	179	0.25	164	0.42	**153**	0.01	**153**	153	0	0.00	10
incgraph_6_0.06_5_30_1.20_10	**296**	8.57	321	4.42	310	0.27	306	0.32	**296**	0.01	**296**	296	0.01	0.00	10
incgraph_6_0.06_5_30_1.60_1	**312**	21.53	384	1.08	367	1.03	342	1.69	314	1.5	**312**	318.2	1.06	1.99	3
incgraph_6_0.06_5_30_1.60_2	**367**	26.61	407	0.46	407	2.06	394	2.55	368	2.8	**367**	368.5	1.24	0.41	2
incgraph_6_0.06_5_30_1.60_3	**913**	36.7	1067	1.33	1067	2.34	990	4.1	**913**	4.07	**913**	921.1	3.06	0.89	2
incgraph_6_0.06_5_30_1.60_4	**424**	23.4	534	4.9	517	1.64	498	1.92	**424**	2.13	**424**	424	1.17	0.00	10
incgraph_6_0.06_5_30_1.60_5	**97**	8.92	140	3.49	138	0.32	129	0.56	**97**	0.69	**97**	97.5	0.63	0.52	8
incgraph_6_0.06_5_30_1.60_6	**527**	24.15	698	2.57	684	2.25	575	5.31	**527**	1.91	**527**	527.6	1.67	0.11	5
incgraph_6_0.06_5_30_1.60_7	**519**	31.82	666	3.25	654	1.84	595	4.06	524	4.28	**519**	523.6	1.47	0.89	1
incgraph_6_0.06_5_30_1.60_8	**151**	8.74	190	0.11	187	0.36	170	1.1	**151**	0.14	**151**	151.6	0.42	0.40	8
incgraph_6_0.06_5_30_1.60_9	**324**	20.29	409	1.8	399	1.65	362	2.75	**324**	1.74	**324**	324.1	0.5	0.03	9
incgraph_6_0.06_5_30_1.60_10	**601**	20.22	768	1.78	742	1.48	639	3.76	**601**	0.98	**601**	601.9	1.04	0.15	3
incgraph_6_0.17_5_30_1.20_1	**2689**	14.13	2768	4.81	2747	0.27	2723	0.47	**2689**	0.01	**2689**	2689	0.01	0.00	10
incgraph_6_0.17_5_30_1.20_2	**3263**	17.14	3337	0.33	3322	0.55	**3263**	1.18	**3263**	0.01	**3263**	3263	0.01	0.00	10
incgraph_6_0.17_5_30_1.20_3	**7984**	28.77	8096	1.08	8061	0.77	8010	0.92	**7984**	0.03	**7984**	7984	0.03	0.00	10
incgraph_6_0.17_5_30_1.20_4	**3784**	16.49	3869	0.99	3851	0.49	3808	1.05	**3784**	0.02	**3784**	3784	0.01	0.00	10
incgraph_6_0.17_5_30_1.20_5	**427**	4.53	437	4.71	437	0.11	428	0.17	**427**	0	**427**	427	0.01	0.00	10
incgraph_6_0.17_5_30_1.20_6	**3786**	16.08	3921	2.33	3884	0.68	3843	0.72	**3786**	0.06	**3786**	3786	0.03	0.00	10
incgraph_6_0.17_5_30_1.20_7	**4098**	19.7	4201	0.24	4140	0.47	**4098**	0.85	**4098**	0.01	**4098**	4098	0.01	0.00	10
incgraph_6_0.17_5_30_1.20_8	**894**	5.53	897	2.44	895	0.17	895	0.15	**894**	0.01	**894**	894	0.02	0.00	10
incgraph_6_0.17_5_30_1.20_9	**501**	9.32	549	3.16	532	0.19	513	0.55	**501**	0	**501**	501	0	0.00	10
incgraph_6_0.17_5_30_1.20_10	**3530**	14.42	3599	0.05	3586	0.43	3557	0.5	**3530**	0.02	**3530**	3530	0.02	0.00	10
incgraph_6_0.17_5_30_1.60_1	**4936**	30.72	5233	3.09	5075	1.87	4961	2.56	**4936**	0.38	**4936**	4936	0.59	0.00	10
incgraph_6_0.17_5_30_1.60_2	**5579**	37.09	6297	3.33	6071	2.22	5757	4.63	**5579**	0.4	**5579**	5579	0.62	0.00	10
incgraph_6_0.17_5_30_1.60_3	**14404**	76.76	15782	3.87	15175	4.36	14592	16.91	**14404**	0.97	**14404**	14404.2	1.02	0.00	8
incgraph_6_0.17_5_30_1.60_4	**6392**	35.29	6810	1.78	6710	2.63	6446	4.89	**6392**	0.83	**6392**	6392.6	0.61	0.01	8
incgraph_6_0.17_5_30_1.60_5	**785**	10.64	836	3.26	825	0.6	817	0.7	**785**	0.18	**785**	785	0.46	0.00	10
incgraph_6_0.17_5_30_1.60_6	**6484**	35.53	6910	0.64	6826	2.35	6575	6.1	**6484**	1.16	**6484**	6484	0.81	0.00	10
incgraph_6_0.17_5_30_1.60_7	**7281**	50.08	7814	3.78	7639	2.85	7405	6.8	7282	0.17	**7281**	7282.1	1.12	0.02	2
incgraph_6_0.17_5_30_1.60_8	**1519**	12.22	1602	3.21	1581	0.59	1532	0.87	**1519**	1.27	**1519**	1519.2	0.5	0.01	8
incgraph_6_0.17_5_30_1.60_9	**921**	21.36	1341	3.9	1235	1.33	1068	3.23	**921**	0.45	**921**	921	0.22	0.00	10
incgraph_6_0.17_5_30_1.60_10	**6479**	33.03	7032	4.72	6933	1.66	6660	4.77	**6479**	0.86	**6479**	6479.4	0.63	0.01	8
incgraph_6_0.30_5_30_1.20_1	**10963**	26	11108	3.51	10999	0.8	10976	0.7	**10963**	0.05	**10963**	10963	0.02	0.00	10
incgraph_6_0.30_5_30_1.20_2	**12601**	31.21	12895	2.77	12700	1.16	12609	1.33	**12601**	0.03	**12601**	12601	0.02	0.00	10
incgraph_6_0.30_5_30_1.20_3	**28283**	58.38	28671	1.05	28466	1.92	28289	2	**28283**	0.02	**28283**	28283	0.02	0.00	10
incgraph_6_0.30_5_30_1.20_4	**14286**	32	14518	3.12	14344	0.71	14968	0.99	**14286**	0.11	**14286**	14286	0.07	0.00	10
incgraph_6_0.30_5_30_1.20_5	**2179**	7.37	2240	4.11	2240	0.16	2211	0.41	**2179**	0.01	**2179**	2179	0	0.00	10
incgraph_6_0.30_5_30_1.20_6	**13652**	30.96	14012	0.34	13754	0.92	13659	1.2	**13652**	0.04	**13652**	13652	0.02	0.00	10
incgraph_6_0.30_5_30_1.20_7	**14918**	36.01	15510	1.33	15259	1.19	14929	2.18	**14918**	0.05	**14918**	14918	0.07	0.00	10
incgraph_6_0.30_5_30_1.20_8	**4073**	10.35	4099	0.82	4098	0.28	4081	0.25	**4073**	0.03	**4073**	4073	0.02	0.00	10
incgraph_6_0.30_5_30_1.20_9	**2487**	12.54	2594	3.14	2554	0.49	2946	0.9	**2487**	0.01	**2487**	2487	0.01	0.00	10
incgraph_6_0.30_5_30_1.20_10	**12893**	29.74	13241	1.6	12934	0.61	13219	0.65	**12893**	0.09	**12893**	12893	0.03	0.00	10
incgraph_6_0.30_5_30_1.60_1	20435	601.69	20561	4.71	20133	3.67	19647	7.36	**19416**	1.65	**19416**	19416	0.78	0.00	10
incgraph_6_0.30_5_30_1.60_2	23994	602.43	24012	1.55	23837	3.31	23053	8.1	**22810**	1	**22810**	22810	1.03	0.00	10
incgraph_6_0.30_5_30_1.60_3	53944	603.5	53596	3.71	51367	5.92	50637	10.25	50496	1.99	**50493**	50493.7	1.67	0.00	7
incgraph_6_0.30_5_30_1.60_4	**24822**	91.1	25896	0.71	25739	6.15	24840	8.2	**24822**	0.78	**24822**	24822	0.35	0.00	10
incgraph_6_0.30_5_30_1.60_5	**3689**	15.94	3987	2.3	3944	0.65	3834	1.2	**3689**	0.02	**3689**	3689	0.07	0.00	10
incgraph_6_0.30_5_30_1.60_6	25213	602	26013	1.33	25714	4.39	24815	10.2	**24658**	3.87	**24658**	24709	1.71	0.21	7
incgraph_6_0.30_5_30_1.60_7	**26836**	245.46	28002	2.88	27774	6.53	26869	8.22	**26836**	1.14	**26836**	26836	0.44	0.00	10
incgraph_6_0.30_5_30_1.60_8	**6853**	22.37	7210	2.14	7083	0.95	6920	2.22	**6853**	0.37	**6853**	6853	0.17	0.00	10
incgraph_6_0.30_5_30_1.60_9	**4320**	29.16	4689	3.7	4552	2.24	4407	4.77	**4320**	0.04	**4320**	4320	0.16	0.00	10
incgraph_6_0.30_5_30_1.60_10	23998	601.72	24869	2.65	24016	2.68	23653	8.3	**23173**	3.02	**23173**	23173	1.43	0.00	10
#Avg	7043.53	74.93	7279.23	2.55	7143.45	1.45	7008.18	2.82	6926.5	0.69	**6926.3**	6927.59	0.46	0.09	8.82
#Best	55		0		0		3		55		**60**				

The instances are constructed by adding vertices and edges up to pre-established numbers. These numbers are calculated as a percentage γ of the quantities in the original graph, where $|IV_i| = \gamma\ |V_i|$ and $|IE_i| = \gamma\ |E_i|$ for each i = (1, …, K). Out of the 20 instances generated for each combination of number of layers and density, 10 are augmented by 20% ($\gamma = 1.2$) and 10 are augmented by 60% ($\gamma = 1.6$).

We coded the MNSB-TS algorithm in C++ and ran it on a PC with a 2.40 GHz Intel Core i5-9300HF processor with the Windows 10 operating system. We perform comparisons with the following implementations in the literature:

- The greedy random adaptive search procedure (GRASP) method proposed by Martí and Estruch (2001).
- Two heuristics, i.e., scatter search (SS) and variable neighborhood scatter search (VNSS) proposed by Sánchez-Oro et al. (2017). Both of these algorithms were implemented in Java and were run on a 2.8 GHz Intel Core i7 processor with 8 GB RAM.
- The (attribute-based) tabu search (ATS) method proposed by Martí et al. (2018b), and the similar tabu strategy is also used in Pastore et al. (2020) for another graph drawing problem (i.e., min–max edge crossing problem). We have incorporated the key component (attribute-based tabu strategy) into our algorithm framework for the comparison, since the ATS method can only solve the bipartite graph case. The experimental results are presented in Section 6.2.
- The exact optimization solver Gurobi.[1]

For a fair comparison, we assume that the CPU speed of the algorithms tested is approximately linearly proportional to the CPU frequency. The CPU speed in Sánchez-Oro et al. (2017) is faster than ours due to a larger CPU frequency (2.7 GHz vs. 2.4 GHz). We utilize the reported results of the corresponding results of SS and VNSS methods presented in Sánchez-Oro et al. (2017).

We perform 10 independent runs of MNSB-TS for each problem instance, with a 200-s time limit per run, while the time limit for the Gurobi solver was set to 600 s in this study. Note however, that when we compare with a previous method that was run only once, we report the results of our method on a single run for a fair comparison.

4.2. Parameter tuning

Table 2 presents the settings of the MNSB-TS parameters used in this study. To avoid over-fitting of the parameters, we adopt a subset of 36 representative instances as the training instances to configure the best values of key parameters of our method. The parameters of (α, l_{min}, l_{max}, $\theta_1 - \theta_4$, Θ, ξ_{max}, and Γ) were tuned with Iterated F-race (IRACE) (Birattari et al., 2010), and an automated configure

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

Table 5
Results for the performance of MNSB-TS compared with the reference heuristics (GRASP, VNSS and SS) and the general solver Gurobi for the third instance set with 13 layers where the number of vertices for each instance is between 311 and 382.

Instance name	Gurobi		GRASP		SS		VNSS		MNSB-TS(1 run)		MNSB-TS(10 runs)				
	Cross	Time	Cross	Time	Cross	Time	Cross	Time	Cross	Time	f_{best}	f_{avg}	T_{avg}	ARD (%)	SR(10)
incgraph_13_0.06_5_30_1.20_1	**795**	24.01	863	10.36	828	1.72	817	2.22	**795**	0.49	**795**	795	0.64	0.00	10
incgraph_13_0.06_5_30_1.20_2	**1082**	29.48	1212	3.5	1123	1.82	1091	3.48	**1082**	0.43	**1082**	1082.2	0.74	0.02	8
incgraph_13_0.06_5_30_1.20_3	**1052**	24.17	1263	9.68	1169	1.8	1088	2.96	**1052**	0.16	**1052**	1052	0.20	0.00	10
incgraph_13_0.06_5_30_1.20_4	**841**	20.9	909	12.46	891	1.19	854	2.12	**841**	0.02	**841**	841	0.32	0.00	10
incgraph_13_0.06_5_30_1.20_5	**532**	11.59	600	3.99	584	0.71	563	1.05	**532**	0.18	**532**	532.4	0.17	0.08	7
incgraph_13_0.06_5_30_1.20_6	**853**	20.38	919	5.57	884	1	866	2.14	856	0.06	**853**	853.9	0.40	0.11	5
incgraph_13_0.06_5_30_1.20_7	**826**	27.1	943	12.94	890	1.33	869	2.03	832	0.17	**826**	828.1	0.56	0.25	5
incgraph_13_0.06_5_30_1.20_8	**552**	9.85	608	2.03	575	0.59	555	0.86	**552**	0.07	**552**	552	0.06	0.00	10
incgraph_13_0.06_5_30_1.20_9	**534**	19.65	734	0.58	684	1.41	638	1.96	**534**	0.97	**534**	534.5	0.41	0.09	7
incgraph_13_0.06_5_30_1.20_10	**870**	19.3	978	11.54	922	1.04	881	1.72	**870**	0.05	**870**	870	0.11	0.00	10
incgraph_13_0.06_5_30_1.60_1	**1372**	57.13	1874	12.27	1726	7.29	1571	20.32	1380	5.36	1373	1379.4	4.30	0.54	1
incgraph_13_0.06_5_30_1.60_2	**1760**	71.54	2125	6.09	1993	10.85	1865	21.52	1766	4.08	**1760**	1763.6	3.32	0.20	1
incgraph_13_0.06_5_30_1.60_3	**1825**	55.19	2560	0.95	2377	6.37	2089	18.45	1827	5.62	**1825**	1829.4	4.67	0.24	1
incgraph_13_0.06_5_30_1.60_4	**1444**	46.76	1863	2.94	1863	5.35	1652	20.96	1449	4.22	**1444**	1446.5	5.86	0.17	5
incgraph_13_0.06_5_30_1.60_5	**984**	26.4	1263	12.44	1223	3.1	1110	6.13	992	3.36	**984**	992	2.98	0.81	2
incgraph_13_0.06_5_30_1.60_6	**1525**	46.09	2203	4.01	2004	5.85	1739	21	1534	2.70	**1525**	1529	4.86	0.26	1
incgraph_13_0.06_5_30_1.60_7	**1525**	62.25	2043	8.62	1944	9.71	1747	20.35	1527	1.94	1527	1533.3	3.68	0.54	1
incgraph_13_0.06_5_30_1.60_8	**980**	21.21	1125	10.08	1088	2.84	1033	7.44	981	1.86	**980**	980.8	1.44	0.08	6
incgraph_13_0.06_5_30_1.60_9	**1129**	46.18	1486	1.58	1450	6.95	1224	14.99	1142	1.21	1131	1137.2	2.13	0.73	1
incgraph_13_0.06_5_30_1.60_10	**1527**	44.03	1840	3.88	1800	5.27	1690	13.89	1531	4.23	1529	1531.3	4.78	0.28	1
incgraph_13_0.17_5_30_1.20_1	**11036**	41.13	11885	10	11181	1.61	11056	2.62	**11036**	0.55	**11036**	11036	0.25	0.00	10
incgraph_13_0.17_5_30_1.20_2	**15376**	53.96	15833	2.44	15615	2.99	15506	5.52	**15376**	0.76	**15376**	15376.1	0.63	0.00	9
incgraph_13_0.17_5_30_1.20_3	**7625**	35.7	7897	5.84	7857	1.97	7671	3.15	**7625**	0.14	**7625**	7625	0.20	0.00	10
incgraph_13_0.17_5_30_1.20_4	**6055**	28.75	6332	1.03	6191	2.28	6136	2.03	**6055**	0.65	**6055**	6055	0.33	0.00	10
incgraph_13_0.17_5_30_1.20_5	**2904**	15.13	3161	8.71	3042	0.8	2973	1.61	**2904**	0.59	**2904**	2904	0.34	0.00	10
incgraph_13_0.17_5_30_1.20_6	**7925**	31.78	8572	10.22	8198	1.93	7950	4.83	**7925**	0.13	**7925**	7925	0.10	0.00	10
incgraph_13_0.17_5_30_1.20_7	**9201**	40	9756	3.88	9446	2.83	9218	4.68	**9201**	0.17	**9201**	9201	0.56	0.00	10
incgraph_13_0.17_5_30_1.20_8	**2323**	12.46	2550	4.2	2366	0.8	2329	1.2	**2323**	0.03	**2323**	2323	0.18	0.00	10
incgraph_13_0.17_5_30_1.20_9	**2622**	21.98	2937	9.8	2753	1.86	2680	3.01	**2622**	2.01	**2622**	2622.1	0.59	0.00	9
incgraph_13_0.17_5_30_1.20_10	**8964**	32.52	9402	5.33	9200	1.6	9045	1.74	**8964**	0.15	**8964**	8964	0.27	0.00	10
incgraph_13_0.17_5_30_1.60_1	**19546**	171.42	21302	14.31	20748	10.87	19862	26.69	19549	2.62	**19546**	19548.3	3.06	0.01	1
incgraph_13_0.17_5_30_1.60_2	28237	610.55	30979	4.65	29067	16.81	27879	46.73	27455	4.94	**27413**	27437.3	6.48	0.09	5
incgraph_13_0.17_5_30_1.60_3	**13626**	213.86	14826	2.42	14542	8.74	13902	21.11	13627	4.22	**13626**	13627.6	2.91	0.01	2
incgraph_13_0.17_5_30_1.60_4	**10497**	78.17	11890	12.73	11110	8.06	10703	19.91	10513	6.58	**10497**	10504.5	3.70	0.07	5
incgraph_13_0.17_5_30_1.60_5	**5007**	50.64	5733	4.59	5417	4.33	5167	10.23	**5007**	1.40	**5007**	5016.6	2.23	0.19	3
incgraph_13_0.17_5_30_1.60_6	**13776**	80.69	16142	10.32	14745	9.86	14158	23.86	13788	1.50	**13776**	13793.1	3.51	0.12	4
incgraph_13_0.17_5_30_1.60_7	16818	603.87	18778	5.47	17244	12.04	16398	41.13	16053	1.34	**16021**	16028.8	3.12	0.05	2
incgraph_13_0.17_5_30_1.60_8	**4170**	27.36	4949	1.95	4641	4.85	4430	7.18	**4170**	2.69	**4170**	4170	1.46	0.00	10
incgraph_13_0.17_5_30_1.60_9	**4613**	58.02	5259	3.3	5144	7.4	4822	18.23	**4613**	2.88	**4613**	4651.2	1.93	0.83	5
incgraph_13_0.17_5_30_1.60_10	16627	607.28	16934	12.4	16495	9.43	15948	19.41	15641	1.95	**15613**	15647.9	3.39	0.22	1
incgraph_13_0.30_5_30_1.20_1	**38957**	82.86	41728	9.9	39183	2.66	38990	3.88	**38957**	0.06	**38957**	38957	0.07	0.00	10
incgraph_13_0.30_5_30_1.20_2	**54434**	112.68	59914	5.67	54792	4.51	54457	7.14	**54434**	0.34	**54434**	54434.1	0.36	0.00	9
incgraph_13_0.30_5_30_1.20_3	**28655**	66.41	31695	6.31	29085	2.88	28913	4.49	**28655**	0.10	**28655**	28655	0.08	0.00	10
incgraph_13_0.30_5_30_1.20_4	**22250**	53.43	22736	1.89	22487	2.83	22286	4.45	**22250**	0.16	**22250**	22250	0.15	0.00	10
incgraph_13_0.30_5_30_1.20_5	**10435**	28.11	11176	11.26	10608	1.14	10540	1.49	**10435**	0.15	**10435**	10435	0.06	0.00	10
incgraph_13_0.30_5_30_1.20_6	**28619**	62.46	29588	6.71	28811	2.91	28668	2.27	**28619**	0.72	**28619**	28619	0.64	0.00	10
incgraph_13_0.30_5_30_1.20_7	**33093**	76.05	33460	13.01	33293	4.44	33142	4.13	**33093**	0.10	**33093**	33093	0.48	0.00	10
incgraph_13_0.30_5_30_1.20_8	**9289**	23.47	10309	10.88	9384	0.92	9307	1.99	**9289**	0.04	**9289**	9289	0.32	0.00	10
incgraph_13_0.30_5_30_1.20_9	**11232**	34.52	12618	7.2	11586	1.03	11283	6.39	**11232**	0.13	**11232**	11232	0.36	0.00	10
incgraph_13_0.30_5_30_1.20_10	**31925**	66.99	32901	13.95	32021	2.37	31936	2.47	**31925**	0.07	**31925**	31925	0.08	0.00	10
incgraph_13_0.30_5_30_1.60_1	71361	600.93	76030	10.66	72107	15.71	70239	39.22	69799	6.40	**69484**	69618.7	6.08	0.19	1
incgraph_13_0.30_5_30_1.60_2	98326	603.57	102688	1.18	96583	24.99	94082	66.45	**93995**	5.89	**93995**	94005.9	5.09	0.01	5
incgraph_13_0.30_5_30_1.60_3	52742	607.83	59026	4.54	53920	13.48	50960	43.68	50751	6.79	**50750**	50750.5	3.13	0.00	5
incgraph_13_0.30_5_30_1.60_4	41453	604.16	40858	7.29	40808	12.54	39236	40.37	39153	6.19	**39139**	39173.1	4.34	0.09	2
incgraph_13_0.30_5_30_1.60_5	**18900**	285.71	20284	14.17	19600	4.91	19054	11.43	18903	0.21	**18900**	18900.9	1.34	0.00	6
incgraph_13_0.30_5_30_1.60_6	53579	604.3	56744	5.49	53885	14.62	52045	42.65	51451	3.13	**51423**	51468.9	5.26	0.09	7
incgraph_13_0.30_5_30_1.60_7	61240	600.29	64138	10.73	60141	14.86	58623	63.38	**57785**	4.49	**57785**	57785.7	4.84	0.00	4
incgraph_13_0.30_5_30_1.60_8	17344	606.37	17393	12.18	17051	5.36	16721	10.73	16606	1.04	**16535**	16563	1.52	0.17	6
incgraph_13_0.30_5_30_1.60_9	**19409**	605.73	20568	12.5	20317	9.16	19650	19.63	**19409**	5.56	**19409**	19409	4.37	0.00	10
incgraph_13_0.30_5_30_1.60_10	59263	600.44	61645	14.33	58266	15.9	57348	22.73	56911	7.15	**56909**	56912.3	5.50	0.01	4
#Avg	16524.37	163.75	17633.78	7.48	16749.13	5.74	16320.92	14.16	16169.57	2.02	**16159.10**	16166.62	2.01	0.11	6.45
#Best	49		0		0		0		34		**56**				

method that is part of the IRACE package (Lopez-Ibanez et al., 2016). The tuning was performed on all the 36 training instances. For each parameter, IRACE requires a limited set of values as input to choose from the column (*Candidate values*) presented in Table 2. The total time budget for IRACE was set to 100 executions of MNSB-TS, with the maximum time limit being 200 s for each instance. The parameter settings suggested by IRACE are reported in the *Final value* column in Table 2.

5. Comparative testing

Our comparisons of MNSB-TS with the three reference heuristics GRASP, VNSS, and SS, and the general optimization solver Gurobi, are presented in Table 3. The first column gives the instance name and the next ten columns show the minimum crossing number *Cross*, the computing time *Time* in seconds produced by the compared algorithms (i.e., Gurobi, GRASP, SS, VNSS and MNSB-TS(1 run)) in one run. To test the robustness of MNSB-TS, the best and average results obtained by the method are presented in the following four columns (marked as MNSB-TS(10 runs)), including the best and average objective value f_{best} and f_{avg}, the average running time T_{avg}, average relative deviation in percentage ARD (%) and the success times to obtain the best solution of each reference algorithm in ten runs ($SR(10)$). The best results obtained over all the compared methods are indicated in bold.

The row *#Avg* indicates the average value of each measure and the row *#Best* shows the number of instances for which the associated algorithm obtains the best results among all the compared algorithms.

Table 3 shows that our MNSB-TS algorithm significantly outperforms the other three reference algorithms. In particular, for both the single-run track and ten-run track, MNSB-TS can obtain the best results (1383.35 and 0.04 s) in terms of both the minimum crossing number *Cross* and the computing time *Time* among the compared solvers. The experimental results show MNSB-TS is the only heuristic to find the optimal solutions for all 60 instances of two layered graphs, whose optimality is proven by Gurobi within 600 s. The average computing time of MNSB-TS is much lower than Gurobi (0.04 s vs. 8.90 s), indicating that MNSB-TS also has a high computational efficiency on this group of benchmark instances.

Table 4 reports 60 instances in the second test set consisting of problems with 6 layers. Here the MNSB-TS algorithm achieves the best results among the compared algorithms in most instances, obtaining better results in terms of the minimum crossing number than the best reference heuristic VNSS (6926.3 vs. 7008.18) with less computing time (0.46 s vs. 2.82 s). More significantly, MNSB-TS finds the best solutions for all the 60 instances while VNSS finds the best solutions only for 3 out of 60 instances. For this group of problems, Gurobi finds (and verifies) the optimal solutions for all but 5 of the instances, within the allotted 600-s time limit. Compared with the general solver Gurobi,

Table 6
Results for the performance of MNSB-TS compared with the reference heuristics (GRASP, VNSS and SS) and the general solver Gurobi for the fourth instance set with 20 layers where the number of vertices for each instance is between 510 and 559.

Instance name	Gurobi		GRASP		SS		VNSS		MNSB-TS(1 run)		MNSB-TS(10 runs)				
	Cross	Time	Cross	Time	Cross	Time	Cross	Time	Cross	Time	f_{best}	f_{avg}	T_{avg}	ARD (%)	SR(10)
incgraph_20_0.06_5_30_1.20_1	**1952**	42.55	2094	16.81	2050	2.83	1977	7.28	**1952**	0.507	**1952**	1955.2	0.77	0.16	1
incgraph_20_0.06_5_30_1.20_2	**1766**	39.41	1983	27.1	1846	3.72	1791	7.18	**1766**	0.45	**1766**	1766.6	1.56	0.03	7
incgraph_20_0.06_5_30_1.20_3	**1524**	35.05	1803	16.17	1692	3.03	1593	7.21	1526	0.136	**1524**	1525.2	0.91	0.08	6
incgraph_20_0.06_5_30_1.20_4	**1373**	36.27	1556	15.57	1477	3.37	1408	6.64	**1373**	0.756	**1373**	1374.3	0.49	0.09	7
incgraph_20_0.06_5_30_1.20_5	**1126**	21.29	1310	29.98	1213	1.65	1167	2.84	**1126**	0.257	**1126**	1126.1	0.98	0.01	9
incgraph_20_0.06_5_30_1.20_6	**1419**	30.64	1524	22.73	1513	2.65	1457	4.81	**1419**	0.375	**1419**	1419.2	0.68	0.01	8
incgraph_20_0.06_5_30_1.20_7	**1357**	37.62	1625	13.98	1529	3.59	1397	8.03	**1357**	0.276	**1357**	1358.0	0.84	0.07	4
incgraph_20_0.06_5_30_1.20_8	**1310**	25.79	1481	11.41	1364	2.2	1321	3.38	1311	0.205	**1310**	1310.6	0.25	0.05	5
incgraph_20_0.06_5_30_1.20_9	**1429**	30.53	1627	21.63	1519	3.69	1444	5.99	**1429**	1.108	**1429**	1429.1	0.32	0.01	9
incgraph_20_0.06_5_30_1.20_10	**1401**	27.03	1473	23.03	1452	2.36	1406	4.65	**1401**	0.058	**1401**	1401.0	0.04	0.00	10
incgraph_20_0.06_5_30_1.60_1	**3489**	97.17	4070	22.23	3817	23.62	3641	65.14	3498	4.565	**3495**	3503.7	4.90	0.42	1
incgraph_20_0.06_5_30_1.60_2	**3164**	87.9	3734	23.47	3704	22.21	3388	81.17	3169	5.691	**3166**	3182.9	4.33	0.60	1
incgraph_20_0.06_5_30_1.60_3	**2854**	79.64	3630	12.02	3620	12.61	3133	39.53	2918	4.898	**2862**	2885.5	6.32	1.10	1
incgraph_20_0.06_5_30_1.60_4	**2673**	83.31	3233	24.51	3146	12.79	2868	45.38	2720	6.739	**2678**	2693.7	5.96	0.77	2
incgraph_20_0.06_5_30_1.60_5	**1839**	49.59	2471	13.66	2324	9.89	2113	28.62	1859	3.15	**1844**	1854.7	3.78	0.85	1
incgraph_20_0.06_5_30_1.60_6	**2672**	71.81	3483	17.05	3273	12.24	2957	47.86	2688	4.87	**2684**	2691.5	3.32	0.73	1
incgraph_20_0.06_5_30_1.60_7	**2367**	88.39	3260	28.65	3132	22.87	2725	77.61	2382	5.288	**2367**	2375.6	7.89	0.36	2
incgraph_20_0.06_5_30_1.60_8	**2301**	60.3	2685	15.08	2654	11.56	2457	36.88	2319	3.025	**2303**	2319.0	4.71	0.78	1
incgraph_20_0.06_5_30_1.60_9	**2382**	72.13	3058	28.04	2846	18.84	2598	46.46	**2388**	6.861	**2388**	2396.5	4.64	0.61	2
incgraph_20_0.06_5_30_1.60_10	**2374**	62.71	2814	18.69	2761	10.33	2573	31.83	2428	1.445	**2385**	2409.6	3.41	1.50	1
incgraph_20_0.17_5_30_1.20_1	**25355**	79.64	25959	11.68	25715	5.51	25474	10.28	**25355**	0.361	**25355**	25355.4	0.72	0.00	9
incgraph_20_0.17_5_30_1.20_2	**17715**	65.15	18479	25.67	17893	4.37	17752	8.89	**17715**	1.716	**17715**	17715.5	1.31	0.00	8
incgraph_20_0.17_5_30_1.20_3	**9191**	47.72	10253	25.63	9411	4.26	9300	8.14	**9191**	2.317	**9191**	9191.6	0.44	0.01	8
incgraph_20_0.17_5_30_1.20_4	**13018**	55.8	13861	17.42	13229	4.32	13076	7.76	13019	0.605	**13018**	13018.1	2.09	0.00	9
incgraph_20_0.17_5_30_1.20_5	**6295**	30.29	6439	28.43	6408	2.04	6334	3.25	**6295**	0.291	**6295**	6295.0	0.27	0.00	10
incgraph_20_0.17_5_30_1.20_6	**11525**	47.93	12643	23.15	11841	3.51	11656	7.88	**11525**	1.098	**11525**	11525.4	0.56	0.00	9
incgraph_20_0.17_5_30_1.20_7	**12156**	55.89	12487	16.3	12445	5	12236	10.05	**12156**	0.513	**12156**	12156.0	0.88	0.00	10
incgraph_20_0.17_5_30_1.20_8	**9707**	40.68	10056	23	9848	3.32	9720	6.69	**9707**	4.445	**9707**	9707.0	1.03	0.00	10
incgraph_20_0.17_5_30_1.20_9	**6767**	40.04	7610	28.68	7097	3.62	6971	7.65	**6767**	0.191	**6767**	6767.0	0.26	0.00	10
incgraph_20_0.17_5_30_1.20_10	**10205**	42.18	11045	13.59	10302	3.45	**10205**	7.22	**10205**	0.02	**10205**	10205.0	0.04	0.00	10
incgraph_20_0.17_5_30_1.60_1	45878	602.31	49371	24.61	46814	26.75	45315	118.87	44438	14.127	**44398**	44421.8	12.58	0.05	1
incgraph_20_0.17_5_30_1.60_2	**31349**	245.21	35042	12.47	33616	28.42	31749	129.4	31352	7.349	**31349**	31369.6	5.26	0.07	4
incgraph_20_0.17_5_30_1.60_3	**16522**	136.95	17586	14.54	17521	17.78	16794	46.68	**16522**	7.421	**16522**	16530.1	5.87	0.05	1
incgraph_20_0.17_5_30_1.60_4	**22820**	265.99	26361	14.26	24642	22.14	23568	65.06	22831	1.804	**22820**	22822.9	6.45	0.01	3
incgraph_20_0.17_5_30_1.60_5	**11020**	88.38	12097	28.73	11882	11.35	11401	25.09	11047	4.648	**11020**	11031.0	3.59	0.10	1
incgraph_20_0.17_5_30_1.60_6	20514	400.07	22087	22.02	22066	21.34	21018	61.69	20641	6.022	**20482**	20525.1	6.72	0.21	1
incgraph_20_0.17_5_30_1.60_7	**21169**	176.61	24383	14.95	23096	25.05	21620	127.65	21179	12.742	**21169**	21175.2	11.77	0.03	2
incgraph_20_0.17_5_30_1.60_8	**16387**	107.38	17624	14.11	17551	15.41	16793	45.55	16403	4.819	**16387**	16398.7	7.94	0.07	1
incgraph_20_0.17_5_30_1.60_9	**12397**	96.66	14834	15.92	13610	20.85	12908	55.21	12398	9.293	**12397**	12405.8	9.63	0.07	1
incgraph_20_0.17_5_30_1.60_10	**17849**	341.59	20398	13.82	19183	17.23	18194	62.85	17929	16.473	**17849**	17868.0	11.54	0.11	2
incgraph_20_0.30_5_30_1.20_1	**88290**	170.24	91612	19.17	88885	8.5	88391	14.54	**88290**	2.146	**88290**	88290.0	0.51	0.00	10
incgraph_20_0.30_5_30_1.20_2	**63685**	132.96	64777	13.88	64593	8.25	63753	17.46	**63685**	0.971	**63685**	63685.1	1.16	0.00	9
incgraph_20_0.30_5_30_1.20_3	**35728**	86.74	37739	22.77	36190	6.97	35826	12.46	**35728**	1.211	**35728**	35728.1	0.77	0.00	9
incgraph_20_0.30_5_30_1.20_4	**47165**	106.41	50964	26.91	47809	6.56	47322	15.56	**47165**	0.819	**47165**	47165.5	0.65	0.00	9
incgraph_20_0.30_5_30_1.20_5	**23907**	56.46	25548	26.91	24176	3.87	24018	4.43	**23907**	0.253	**23907**	23907.0	0.24	0.00	10
incgraph_20_0.30_5_30_1.20_6	**42705**	94.67	45372	15.1	42990	5.74	42760	7.83	**42705**	0.914	**42705**	42705.0	0.51	0.00	10
incgraph_20_0.30_5_30_1.20_7	**44585**	104.45	48842	17.53	45061	6.09	44711	12.95	44586	0.311	**44585**	44585.4	0.26	0.00	7
incgraph_20_0.30_5_30_1.20_8	**35463**	79.97	38308	23.66	35748	4.45	35598	5.27	**35463**	1.076	**35463**	35463.0	1.66	0.00	10
incgraph_20_0.30_5_30_1.20_9	**27240**	70.26	29359	20.44	27716	5.46	27358	14.16	**27240**	1.476	**27240**	27240.0	0.41	0.00	10
incgraph_20_0.30_5_30_1.20_10	**37395**	81.1	39819	26.56	37674	5.16	37469	7.45	**37395**	0.731	**37395**	37395.1	0.73	0.00	9
incgraph_20_0.30_5_30_1.60_1	159703	611.38	164423	25.34	159779	47.11	155985	130.7	155486	12.067	**155419**	155541.1	9.63	0.08	2
incgraph_20_0.30_5_30_1.60_2	116536	608.78	122963	17.58	115439	36.02	112116	126.89	**111709**	15.185	**111709**	111714.7	9.58	0.01	6
incgraph_20_0.30_5_30_1.60_3	65915	605.1	71878	29.19	65864	26.55	63965	67.62	62827	8.97	**62825**	62834.4	7.52	0.01	2
incgraph_20_0.30_5_30_1.60_4	85849	606.75	93479	24.99	87303	38.57	84894	106.49	83541	14.641	**83502**	83685.4	8.16	0.22	1
incgraph_20_0.30_5_30_1.60_5	44342	603.72	43417	17.38	42898	17.58	42145	34.46	**42083**	4.457	**42083**	42104.0	3.12	0.05	5
incgraph_20_0.30_5_30_1.60_6	77885	606.1	78585	26.34	77759	25.08	75164	75.9	74394	9.085	**74315**	74325.4	8.10	0.01	2
incgraph_20_0.30_5_30_1.60_7	83052	605.49	83829	10.25	83336	31.92	80014	101.16	79520	7.325	**79351**	79460.2	8.47	0.14	1
incgraph_20_0.30_5_30_1.60_8	64949	604.9	70084	18.27	64746	27.75	62803	70.95	62408	9.59	**62368**	62401.4	7.34	0.05	2
incgraph_20_0.30_5_30_1.60_9	49060	604.07	53456	18.53	50077	23.72	48562	59.01	**47506**	3.997	**47506**	47639.2	9.60	0.28	1
incgraph_20_0.30_5_30_1.60_10	66939	604.78	68195	10.52	65886	20.5	63819	59.53	63388	6.989	**63332**	63343.3	6.79	0.02	1
#Avg	27316.73	184.83	29002.97	20.03	27750.52	13.13	27036.18	38.65	26778.83	4.15	**26762.15**	26777.93	3.84	0.17	5.08
#Best	48		0		0		1		30		**51**				

Table 7
Summary for the performance of MNSB-TS compared with the reference heuristics (GRASP, VNSS and SS) and the general solver Gurobi for all the 240 instances.

	Gurobi	GRASP	SS	VNSS	MNSB-TS(1 run)	MNSB-TS(10 runs)
2-layer graph ($47 \le n \le 54$)						f_{best} (f_{avg})
Cross	**1383.35**	1454.9	1420.3	1419.18	1383.35	**1383.35** (1383.35)
Time (s)	8.90	0.49	0.15	0.22	0.04	0.04
6-layer graph ($138 \le n \le 181$)						
Cross	7043.53	7279.23	7143.45	7008.18	6926.5	**6926.3** (6927.59)
Time (s)	74.93	2.55	1.45	2.82	0.69	0.46
13-layer graph ($311 \le n \le 382$)						
Cross	16524.37	17633.78	16749.13	16320.92	16169.57	**16159.10** (16166.62)
Time (s)	163.75	7.48	5.74	14.16	2.02	2.01
20-layer graph ($510 \le n \le 559$)						
Cross	27316.73	29002.97	27750.52	27036.18	26778.83	**26762.15** (26777.93)
Time (s)	184.83	20.03	13.13	38.65	4.15	3.84
#Average						
Cross	13066.99	13842.72	13265.85	12946.12	12814.56	**12807.73** (12813.87)
Time (s)	108.10	7.64	5.17	13.96	1.73	1.59

n denotes the number of all the vertices in the incremental graph (i.e., the augmented graph obtained by adding incremental vertices).

Table 8
Comparison between the MNSB-TS and its variation MNAB-TS with an attribute-based tabu strategy on the 36 representative instances.

Instance name	MNAB-TS			MNSB-TS		
	f_{best}	f_{avg}	SR(10)	f_{best}	f_{avg}	SR(10)
incgraph_2_0.06_5_30_1.60_7	**15**	17.8	4	**15**	15	10
incgraph_2_0.06_5_30_1.60_8	**1**	1.4	7	**1**	1	10
incgraph_2_0.06_5_30_1.60_9	**0**	0.2	8	**0**	0	10
incgraph_2_0.17_5_30_1.60_7	2754	2807.6	1	**2742**	2742	10
incgraph_2_0.17_5_30_1.60_8	**187**	188.6	3	**187**	187	10
incgraph_2_0.17_5_30_1.60_9	**0**	0	10	**0**	0	10
incgraph_2_0.30_5_30_1.60_7	12308	12406.3	1	**12281**	12281	10
incgraph_2_0.30_5_30_1.60_8	**1169**	1179.7	1	**1169**	1169	10
incgraph_2_0.30_5_30_1.60_9	211	213.9	2	**208**	208	10
incgraph_6_0.06_5_30_1.60_7	591	696.5	1	**519**	523.6	1
incgraph_6_0.06_5_30_1.60_8	208	244.6	1	**151**	151.6	8
incgraph_6_0.06_5_30_1.60_9	413	461.9	1	**324**	324.1	9
incgraph_6_0.17_5_30_1.60_7	7446	7592.1	1	**7281**	7282.1	2
incgraph_6_0.17_5_30_1.60_8	1559	1598.1	1	**1519**	1519.2	8
incgraph_6_0.17_5_30_1.60_9	1064	1096.6	1	**921**	921	10
incgraph_6_0.30_5_30_1.60_7	27103	27497.1	1	**26836**	26836	10
incgraph_6_0.30_5_30_1.60_8	7016	7100.2	1	**6853**	6853	10
incgraph_6_0.30_5_30_1.60_9	4432	4533.9	1	**4320**	4320	10
incgraph_13_0.06_5_30_1.60_7	1722	1787.5	1	**1527**	1533.3	1
incgraph_13_0.06_5_30_1.60_8	1086	1118.7	1	**980**	980.8	6
incgraph_13_0.06_5_30_1.60_9	1257	1327.1	1	**1131**	1137.2	1
incgraph_13_0.17_5_30_1.60_7	16192	16375.4	1	**16021**	16028.8	2
incgraph_13_0.17_5_30_1.60_8	4241	4432.7	1	**4170**	4170	10
incgraph_13_0.17_5_30_1.60_9	4911	4997.2	1	**4613**	4651.2	5
incgraph_13_0.30_5_30_1.60_7	58196	58697.7	1	**57785**	57785.7	4
incgraph_13_0.30_5_30_1.60_8	16728	16884.9	1	**16535**	16563	6
incgraph_13_0.30_5_30_1.60_9	19655	19908	1	**19409**	19409	10
incgraph_20_0.06_5_30_1.60_7	2651	2723.4	1	**2367**	2375.6	2
incgraph_20_0.06_5_30_1.60_8	2487	2556.2	1	**2303**	2319	1
incgraph_20_0.06_5_30_1.60_9	2554	2673.8	1	**2388**	2396.5	2
incgraph_20_0.17_5_30_1.60_7	21706	21964.7	1	**21169**	21175.2	2
incgraph_20_0.17_5_30_1.60_8	16576	16841.2	1	**16387**	16398.7	1
incgraph_20_0.17_5_30_1.60_9	12684	12785.2	1	**12397**	12405.8	1
incgraph_20_0.30_5_30_1.60_7	80050	80398.6	1	**79351**	79460.2	1
incgraph_20_0.30_5_30_1.60_8	62558	62945	1	**62368**	62401.4	2
incgraph_20_0.30_5_30_1.60_9	48101	48463.6	1	**47506**	47639.2	1
#*Avg*	12217.56	12347.71	1.78	**12048.17**	12060.12	6.00
#*Best*	6	1		**36**	36	

MNSB-TS obtains better results in terms of both the minimum crossing number and the average computing time (6926.3 vs 7043.53 and 0.46 s vs. 74.93 s).

Table 5 shows the experimental results of all the reference methods (Gurobi, GRASP, SS, VNSS, MNSB-TS) for the 60 instances of the third instance set with 13 layers. As expected, the difficulty of solving the problem instances increases with the increased number of layers. Gurobi fails to obtain verified optimal solutions for a larger number of instances than in the 6-layer graphs (49 vs. 55). For most instances, where Gurobi fails to find solutions that it verifies as optimal, MNSB-TS obtains better results than Gurobi. In general, the MNSB-TS algorithm outperforms Gurobi in terms of the minimum crossing number and computing time over the third instance set (16159.10 vs. 16524.37 and 2.01 s vs. 163.75 s). In addition, MNSB-TS algorithm also dominates the three other competing heuristics (i.e., GRASP, SS and VNSS) in one run in terms of the minimum crossing number for all the instances (16159.10 vs. 17633.78, 16749.13, and 16320.92, respectively) with a less computing time than the other heuristics.

From Table 6, which reports the fourth instance set with 20 layers, the MNSB-TS algorithm again achieves the best results among the compared algorithms in most instances. In the one-run test, MNSB-TS obtains a better average crossing number ($Cross$) than all three heuristics and the exact solver Gurobi (26778.83 vs. 29002.97, 26778.83 vs. 27750.52, 26778.83 vs. 27036.18, and 26778.83 vs. 27316.73). In the ten-run test, MNSB-TS obtains the best results in 51 out of 60 instances. Compared to Gurobi, MNSB-TS obtains a better average crossing number and computing time (26777.93 vs. 27316.73 and 0.17 s vs. 184.83 s). In addition, MNSB-TS obtains the best results for more instances than Gurobi (51 vs. 48).

Table 7 summarizes the total experimental results based on all 240 instances. As shown in this table, the MNSB-TS algorithm ranks at the top of the state-of-the-art algorithms including the exact optimization solver Gurobi in terms of both solution quality and computational efficiency. Compared to Gurobi, MNSB-TS is able to find better results in terms of average solution quality and tens of times faster computational efficiency (1.59 s vs 108.10 s).

6. Analysis and discussions

In this section we analyze several key elements of the proposed algorithm: the incremental evaluation technique, the solution-based tabu strategy, the multiple neighborhood mechanism, the dynamic diversification strategy, and the spatial distribution of high-quality solutions respectively. We perform 10 independent runs of MNSB-TS and the variants in the following subsections for each problem instance, with a 200-s time limit per run.

Table 9
Comparison among the MNSB-TS and its variations $SBTS_i$ with N_i neighborhood operator on the 36 representative instances.

Instance name	$SBTS_1$			$SBTS_2$			$SBTS_3$			$SBTS_4$			MNSB-TS		
	f_{best}	f_{avg}	SR(10)	f_{best}	f_{avg}	SR(10)	f_{best}	f_{avg}	SR(10)	f_{best}	f_{avg}	SR(10)	f_{best}	f_{avg}	SR(10)
incgraph_2_0.06_5_30_1.60_7	**15**	15	10	25	26.8	6	84	88.8	5	89	93.2	3	**15**	15	10
incgraph_2_0.06_5_30_1.60_8	**1**	1	10	4	6.8	2	16	22.8	2	21	29.8	3	**1**	1	10
incgraph_2_0.06_5_30_1.60_9	**0**	0	10	**0**	0.4	8	15	18.6	4	8	18.9	2	**0**	0	10
incgraph_2_0.17_5_30_1.60_7	**2742**	2745.7	4	2774	2782.1	2	2922	3035.7	4	2995	3024.4	1	**2742**	2742	10
incgraph_2_0.17_5_30_1.60_8	**187**	187	10	192	196	2	210	238.1	3	210	231	3	**187**	187	10
incgraph_2_0.17_5_30_1.60_9	**0**	0	10	**0**	0.3	7	18	23	4	10	17.7	3	**0**	0	10
incgraph_2_0.30_5_30_1.60_7	**12281**	12286.6	4	12340	12404	1	12961	13096.8	1	12859	13057.5	1	**12281**	12281	10
incgraph_2_0.30_5_30_1.60_8	**1169**	1172.3	7	1181	1191.6	2	1224	1267.6	1	1226	1255.3	3	**1169**	1169	10
incgraph_2_0.30_5_30_1.60_9	**208**	208	10	**208**	212.3	1	242	259	2	250	270.8	2	**208**	208	10
incgraph_6_0.06_5_30_1.60_7	**519**	524.6	1	608	623	1	582	608.2	1	683	737.3	1	**519**	523.6	1
incgraph_6_0.06_5_30_1.60_8	**151**	151.2	9	227	237	2	184	196.4	2	254	282.7	1	**151**	151.6	8
incgraph_6_0.06_5_30_1.60_9	**324**	324.3	7	365	375.9	1	347	368.4	1	447	459.2	1	**324**	324.1	9
incgraph_6_0.17_5_30_1.60_7	**7281**	7282.1	1	7449	7487.6	1	7356	7450	1	7646	7758.5	1	**7281**	7282.1	2
incgraph_6_0.17_5_30_1.60_8	**1519**	1519.1	9	1619	1627.9	3	1557	1579.4	1	1669	1704.1	2	**1519**	1519.2	8
incgraph_6_0.17_5_30_1.60_9	**921**	921	10	1001	1023.5	1	951	969.8	1	1140	1171.3	1	**921**	921	10
incgraph_6_0.30_5_30_1.60_7	**26836**	26836	10	27285	27403.3	1	27076	27230.8	1	27798	28016.5	1	**26836**	26836	10
incgraph_6_0.30_5_30_1.60_8	**6853**	6853	10	7069	7163.7	1	6918	6980	2	7244	7394.9	1	**6853**	6853	10
incgraph_6_0.30_5_30_1.60_9	**4320**	4320	10	4480	4521.3	1	4374	4421.9	2	4705	4782.3	1	**4320**	4320	10
incgraph_13_0.06_5_30_1.60_7	1530	1542.9	1	1778	1836.4	1	1595	1646.9	1	1911	1970.5	1	**1527**	1533.3	1
incgraph_13_0.06_5_30_1.60_8	982	985.9	2	1128	1144.4	1	1003	1011	2	1151	1169.1	1	**980**	980.8	6
incgraph_13_0.06_5_30_1.60_9	**1131**	1134	2	1340	1367.3	1	1159	1180.9	2	1404	1457.2	1	**1131**	1137.2	1
incgraph_13_0.17_5_30_1.60_7	16023	16140	1	16850	17003.7	1	16165	16327.6	1	17189	17340.8	1	**16021**	16028.8	2
incgraph_13_0.17_5_30_1.60_8	**4170**	4178.7	6	4486	4575.5	1	4212	4256.6	1	4621	4681.3	1	**4170**	4170	10
incgraph_13_0.17_5_30_1.60_9	**4613**	4648.6	4	5176	5224.4	1	4697	4815	1	5267	5317.2	1	**4613**	4651.2	5
incgraph_13_0.30_5_30_1.60_7	57786	57886.9	3	60059	60301.6	1	57963	58280.1	1	60052	60791.2	1	**57785**	57785.7	4
incgraph_13_0.30_5_30_1.60_8	**16535**	16577.3	2	17309	17421.8	1	16632	16691.1	2	17422	17541.1	1	**16535**	16563	6
incgraph_13_0.30_5_30_1.60_9	**19409**	19443.9	2	20233	20445.4	1	19543	19637.6	1	20583	20812.8	1	**19409**	19409	10
incgraph_20_0.06_5_30_1.60_7	2391	2404.6	1	2894	2948.1	1	2496	2536.9	1	3130	3176	1	**2367**	2375.6	2
incgraph_20_0.06_5_30_1.60_8	2327	2337.7	1	2716	2761.6	1	2392	2433.4	1	2814	2859.9	1	**2303**	2319	1
incgraph_20_0.06_5_30_1.60_9	2400	2413.2	1	2757	2872.4	1	2479	2491.4	1	2937	2971.3	1	**2388**	2396.5	2
incgraph_20_0.17_5_30_1.60_7	21176	21242.5	1	22624	22748.7	1	21316	21477.7	1	22903	23038.6	1	**21169**	21175.2	2
incgraph_20_0.17_5_30_1.60_8	**16387**	16401	1	17568	17790.8	1	16492	16556.5	1	17766	17969.8	1	**16387**	16398.7	1
incgraph_20_0.17_5_30_1.60_9	12401	12422.2	1	13269	13390.7	1	12486	12545.1	1	13452	13568	1	**12397**	12405.8	1
incgraph_20_0.30_5_30_1.60_7	79385	79549.2	1	82882	83230.4	1	79647	79819.8	1	83739	84068.3	1	**79351**	79460.2	1
incgraph_20_0.30_5_30_1.60_8	62370	62448.8	1	64671	65043.7	1	62503	62651.5	1	65133	65546.8	1	**62368**	62401.4	2
incgraph_20_0.30_5_30_1.60_9	47649	47822.2	1	50079	50282.4	1	47658	47923.8	1	50217	50761.3	1	**47506**	47639.2	1
#*Avg*	12055.33	12081.29	4.83	12629.06	12713.13	1.69	12152.08	12226.06	1.64	12804.03	12926.29	1.36	**12048.17**	12060.12	6.00
#*Best*	24	15		3	0		0	0		0	0		**36**	32	

6.1. Importance of the incremental evaluation technique

It is particularly important to be able to rapidly evaluate the neighborhoods in a neighborhood search method. In Section 3.3.2, we proposed an incremental evaluation mechanism to evaluate the four neighborhood operators by employing dedicated matrices to record information about the number of edge crossings for pairs of vertices.

To evaluate the effectiveness of this incremental evaluation technique, we carry out computational experiments to compare the performance of the MNSB-TS algorithm with this technique (abbreviated as FE) and its variant (RC) on six representative instances with different layers (where the number of layers is 2, 6, 13 and 20, respectively). The variant RC recalculates the edge crossings of each candidate solution without employing the incremental evaluation mechanism, while keeping other MNSB-TS components unaltered.

The evaluation of the iteration numbers with the CPU time in seconds is shown in Fig. 7. One can observe that the FE procedure using the incremental evaluation technique performs many more iterations in the 10-s time interval than the RC procedure that does not employ this mechanism. FE obviously outperforms RC and the gap between the methods becomes larger as time grows, demonstrating the importance of the proposed fast e- valuation strategy.

6.2. Effectiveness of the solution-based tabu strategy

The solution-based tabu strategy is a pivotal component of our MNSB-TS algorithm. To show its merit for the IGDP problem compared to the popular attribute-based tabu strategy, we produce a variant MNAB-TS of MNSB-TS whose only change is to replace the solution-based tabu strategy with the attribute-based strategy. Other MNSB-TS components are unaltered according to the experimental protocol given in Table 2.

In the attribute-based approach, when we select a new vertex v and move it, vertex v is conceived to be the solution attribute of interest, and we record v in the tabu list with tabu tenure tt, in order to prohibit moving it again during the next tt iterations. Based on empirical testing to identify a good choice for tt, we set $tt = 5$ which is similar to the setting chosen in Martí et al. (2018b). Finally, we use the aspiration criterion that overrides the tabu status of a move (rendering the move non-tabu) if the generated solution leads to a solution better than all previously visited solutions.

To compare the relative performance of MNSB-TS and MNAB-TS, we carried out computational experiments using 36 representative benchmark instances with different scales (the layers varying among 2, 6, 13, 20), where both methods were run in 10 trials.

The experimental results including the best objective value (f_{best}), the average objective value (f_{avg}), and the average computing time t_{av} for each algorithm are summarized in Table 8. The row #*Avg* indicates the average value of each measure in the 36 representative instances. The row #*Best* shows the number of instances for which the associated algorithm obtained the best results in terms of f_{best} or f_{avg} results. Clearly, MNSB-TS proves more effective than the variant MNAB-TS by obtaining better results for f_{best} and f_{avg} (12048.17 vs. 12217.56 and 12060.12 vs. 12347.71). In addition, MNAB-TS can obtain better results regarding robustness and stability than MNAB-TS in terms of the indicator ($SR(10)$), which denotes the success times to obtain their best solutions in ten runs (6 vs. 1.78). The MNSB-TS method dominates the variant MNAB-TS in all the indicators, by obtaining better results for f_{best}, f_{avg} and $SR(10)$. We therefore conclude that the solution-based tabu strategy plays a positive role in the high performance of the MNSB-TS algorithm.

6.3. Impact of the multiple neighborhood mechanism

As described in Algorithm 3, we iteratively employ multiple neighborhood search iteratively in the main search phase. As such, at each iteration, a new neighborhood usually offers a chance to encounter a neighboring solution of higher quality but requires additional computational effort. Hence, we face the challenge of identifying an appropriate multiple neighborhood mechanism to enable the resulting algorithm to reach a good tradeoff between solution quality and computing speed. To study the impact of such a mechanism, we compared our standard algorithm with four variants of MNSB-TS, i.e., $SBTS_i$ by using a single neighborhood move N_i ($i = 1, \ldots, 4$) (as described in Section 3.3.1) without the multiple neighborhood framework.

Table 9 shows that the computational performance of $SBTS_1$ performs best among all the neighborhood moves in each indicator. $SBTS_1$ is close to MNSB-TS in terms of f_{best} and f_{avg}, while the gap becomes

Table 10
Comparison between the MNSB-TS and its variation MNSB-TS$_{fs}$ with a fixed strength perturbation strategy on the 36 representative instances.

Instance name	MNSB-TS$_{fs}$			MNSB-TS		
	f_{best}	f_{avg}	SR(10)	f_{best}	f_{avg}	SR(10)
incgraph_2_0.06_5_30_1.60_7	**15**	15	10	**15**	15	10
incgraph_2_0.06_5_30_1.60_8	**1**	1	10	**1**	1	10
incgraph_2_0.06_5_30_1.60_9	**0**	0	10	**0**	0	10
incgraph_2_0.17_5_30_1.60_7	**2742**	2742	10	**2742**	2742	10
incgraph_2_0.17_5_30_1.60_8	**187**	187	10	**187**	187	10
incgraph_2_0.17_5_30_1.60_9	**0**	0	10	**0**	0	10
incgraph_2_0.30_5_30_1.60_7	**12281**	12281	10	**12281**	12281	10
incgraph_2_0.30_5_30_1.60_8	**1169**	1172.3	7	**1169**	1169	10
incgraph_2_0.30_5_30_1.60_9	**208**	208	10	**208**	208	10
incgraph_6_0.06_5_30_1.60_7	522	524.8	1	**519**	523.6	1
incgraph_6_0.06_5_30_1.60_8	**151**	151.6	7	**151**	151.6	8
incgraph_6_0.06_5_30_1.60_9	**324**	324	10	**324**	324.1	9
incgraph_6_0.17_5_30_1.60_7	**7281**	7288.1	1	**7281**	7282.1	2
incgraph_6_0.17_5_30_1.60_8	**1519**	1519.7	7	**1519**	1519.2	8
incgraph_6_0.17_5_30_1.60_9	**921**	921	10	**921**	921	10
incgraph_6_0.30_5_30_1.60_7	**26836**	26836	10	**26836**	26836	10
incgraph_6_0.30_5_30_1.60_8	**6853**	6853	10	**6853**	6853	10
incgraph_6_0.30_5_30_1.60_9	**4320**	4320	10	**4320**	4320	10
incgraph_13_0.06_5_30_1.60_7	**1527**	1538.8	1	**1527**	1533.3	1
incgraph_13_0.06_5_30_1.60_8	**980**	981.2	3	**980**	980.8	6
incgraph_13_0.06_5_30_1.60_9	**1131**	1136.3	1	**1131**	1137.2	1
incgraph_13_0.17_5_30_1.60_7	**16021**	16064.1	2	**16021**	16028.8	2
incgraph_13_0.17_5_30_1.60_8	**4170**	4172.8	9	**4170**	4170	10
incgraph_13_0.17_5_30_1.60_9	**4613**	4659.9	4	**4613**	4651.2	5
incgraph_13_0.30_5_30_1.60_7	**57785**	57786	2	**57785**	57785.7	4
incgraph_13_0.30_5_30_1.60_8	**16535**	16579.1	3	**16535**	16563	6
incgraph_13_0.30_5_30_1.60_9	**19409**	19413.2	8	**19409**	19409	10
incgraph_20_0.06_5_30_1.60_7	2380	2399.9	1	**2367**	2375.6	2
incgraph_20_0.06_5_30_1.60_8	2320	2335.7	1	**2303**	2319	1
incgraph_20_0.06_5_30_1.60_9	2393	2399.8	2	**2388**	2396.5	2
incgraph_20_0.17_5_30_1.60_7	21175	21190	1	**21169**	21175.2	2
incgraph_20_0.17_5_30_1.60_8	**16387**	16397.3	1	**16387**	16398.7	1
incgraph_20_0.17_5_30_1.60_9	12401	12413.7	1	**12397**	12405.8	1
incgraph_20_0.30_5_30_1.60_7	79358	79550.4	1	**79351**	79460.2	1
incgraph_20_0.30_5_30_1.60_8	**62368**	62417.7	1	**62368**	62401.4	2
incgraph_20_0.30_5_30_1.60_9	**47506**	47627.7	1	**47506**	47639.2	1
#*Avg*	12049.69	12066.89	5.44	**12048.17**	12060.12	6.00
#*Best*	29	17		**36**	32	

larger as the scale of the test instances becomes larger. This outcome shows that N_1 move is the best performing move, and using the four neighborhood operators can obtain better results than only employing each single move N_1 to N_4. Multiple neighborhood moves can delay the convergence to avoid prematurely converging as can happen with the algorithm that uses a single move. Meanwhile, the larger average result in terms of $SR(10)$ in comparison with SBTS$_1$ (6.00 vs 4.83) shows the multiple neighborhood mechanism can make the algorithm performance more robust and stable. This experiment thus confirms the effectiveness of the multiple neighborhood framework in conjunction with solution-based multiple neighborhood tabu search.

6.4. Importance of the dynamic diversification strategy

To check the influence of the dynamic strength strategy in the dynamic diversification phase, we compare our algorithm with a variant of MNSB-TS with a fixed diversification strength (denoted as MNSB-TS$_{fs}$) obtained by removing the dynamic strength mechanism and keeping the other ingredients unchanged. In other words, in contrast with the dynamic strength strategy employed in MNSB-TS, which relies on the historical search information (e.g., the previous local optimum S_p and the counter ξ of consecutive non-improving best solutions found), the variant MNSB-TS$_{fs}$ adopts a fixed diversification strength l_{sp}. The value of l_{sp} is empirically set to $(|AV|)/3$, where $|AV|$ denotes the number of all the incremental vertices. Given the stochastic nature of the two algorithms, we solved each instance 10 times by each reference algorithm, and recorded the best objective value f_{best}, the average objective values f_{avg}, and the times to obtain the best result of each compared algorithm in ten runs $SR(10)$.

The experimental results are summarized in Table 10, which includes the same statistics as in Table 9. Table 10 discloses that the MNSB-TS algorithm (with the dynamic diversification mechanism) performs consistently better than the variant MNSB-TS$_{fs}$ (with the fixed diversification strength mechanism) over all performance indicators considered and on all the tested instances.

6.5. Spatial distribution of high-quality solutions

To understand the structure of the IGDP problem and to motivate the use of our proposed algorithm, we perform an experimental study on spatial distribution of high-quality locally optimal solutions inspired by Porumbel et al. (2010). This study is based on 6 representative instances with different numbers of layers (6, 13, 20). For each instance, we perform one execution of our MNSB-TS algorithm with the prefixed maximum time of MNSB-TS given in Tables 7–9, and record 1600 distinct high-quality solutions encountered during the run.

To visualize the spatial distribution of high-quality solutions, we employ the multidimensional scaling procedure from Kruskal (1964) to generate the distribution in Euclidean R^3 space. The procedure consists of the following two steps:

Table 11
Summary for the key components in MNSB-TS.

Key components	Description	Motivation	Influence and results
Incremental evaluation technique	To efficiently evaluate the changed objective values produced by the candidate neighborhood moves, we utilize dedicated matrices to record the relevant information associated with the number of edge crossings for two vertices.	A similar technique is successfully applied to the DBDP (i.e., two-layer graph drawing problem), and we extend it to IGDP (i.e., multi-layer graph drawing problem).	Experimental results in Section 6.1 show that the FE procedure using the incremental evaluation technique performs many more iterations in the 10-s time interval than the RC procedure that does not employ this mechanism. FE conspicuously outperforms RC and the gap between the methods becomes larger as time grows, demonstrating the importance of the proposed fast evaluation strategy.
Solution-based tabu strategy	The solution-based tabu strategy utilizes hash functions and hash vectors to store the information encoding a solution to make it tabu, instead of recording simpler solution attributes such as values assigned to one or two variables which can prevent a range of solutions from being re-visited.	Solution-based tabu strategy has been successfully applied to various combinatorial optimization problems, e.g., the minimum differential dispersion problem (Wang et al., 2017), the multidemand multidimensional knapsack problem (Lai et al., 2018a), the maximum diversity problem (Liu et al., 2020), and the obnoxious p-median problem (Chang et al., 2021). In this study, we explore combining this method with a multiple neighborhood mechanism for solving the IGDP problem.	Computational results disclose that the solution-based tabu strategy plays an instrumental role in the high performance of the MNSB-TS algorithm (see Section 6.2).
Multiple neighborhood mechanism	MNSB-TS iteratively employs multiple neighborhood search in the main search phase, including four neighborhood moves $N_1 - N_4$.	At each iteration of the Algorithm 3, a new neighborhood move usually offers a chance to encounter a neighboring solution of higher quality.	Experimental results in Section 6.3 show that the multiple neighborhood mechanism can obtain a better tradeoff between the computational efficiency and solution quality in comparison with two other neighborhood frameworks.
Dynamic diversification strategy	The proposed dynamic diversification phase is controlled by a specific parameter of the jump magnitude by determining the number of removed vertices and the balance ratio between random and greedy strategies in the modification procedure in Section 3.4. Specifically, it varies the diversification intensity, depending on the search status.	The purpose of the diversification mechanism is to allow MNSB-TS to escape from the current local optimum in order to discover other local optima of better solution quality.	Experimental results in Sections 6.4 show the effectiveness of the dynamic diversification mechanism.

- Step 1: We first construct the distance matrix $M_{p\times p}$ ($p = 1600$ in this study), where each entry $M[a, b]$ represents the distance between solutions S^a and S^b, and p is the total number of sampled locally optimum solutions. The distance $dist(S^a, S^b)$ between S^a and S^b is defined as follows:

$$dist(S^a, S^b) = \sum_{k=1}^{K} \sum_{i=1}^{n_k} H_k(\omega_k^a(i), \omega_k^b(i)) \tag{21}$$

$$H_k(\omega_k^a(i), \omega_k^b(i)) = \begin{cases} 0, & \text{If } \omega_k^a(i) = \omega_k^b(\text{i}); \\ 1, & \text{If } \omega_k^a(i) \neq \omega_k^b(\text{i}); \end{cases} \tag{22}$$

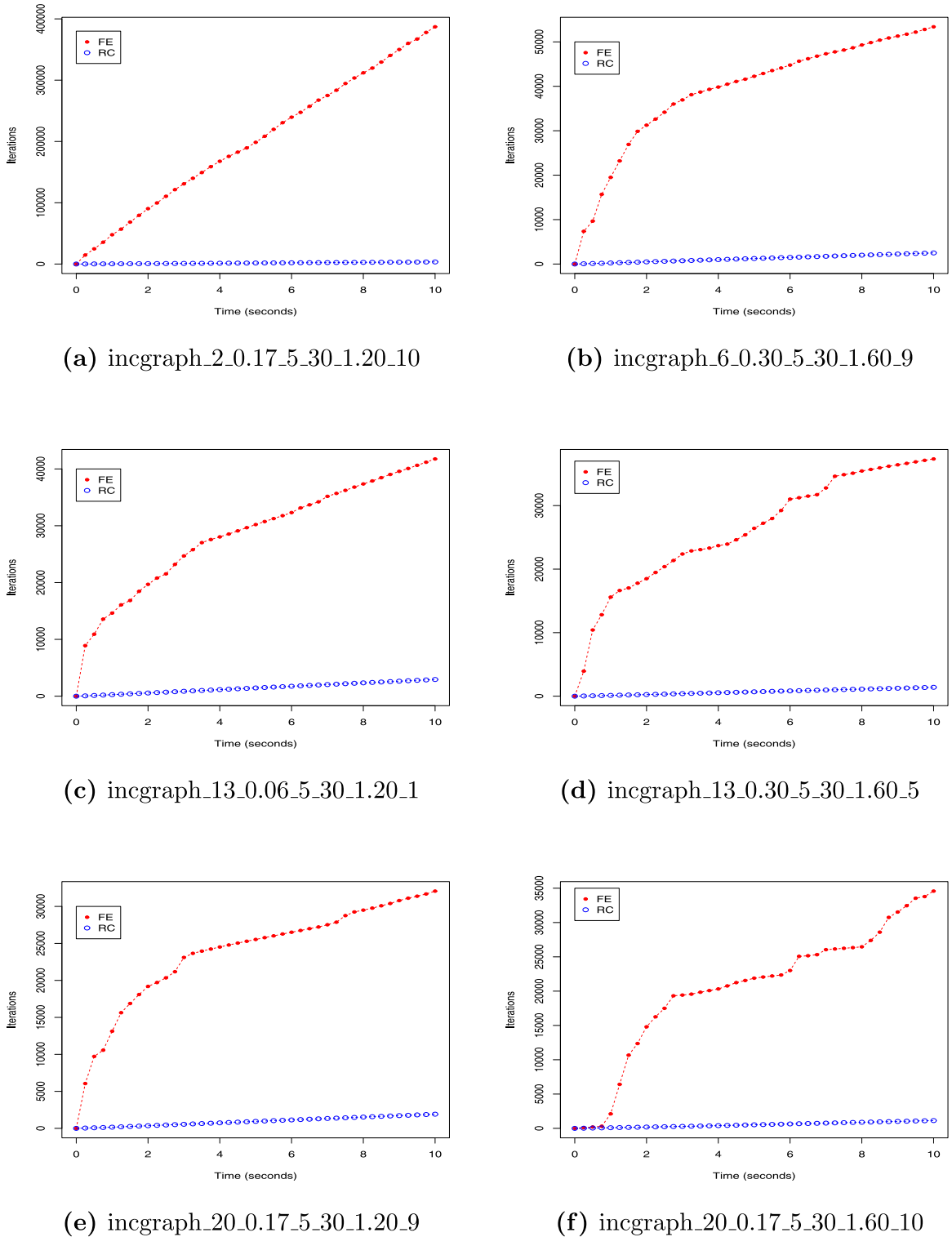
  where $\omega_k^a(i)$ and $\omega_k^b(i)$ denote the vertex locating ith position in layer k for solutions S^a and S^b, respectively. The H_k function value is a binary value set to 0 if both two solutions S^a and S^b has the same vertex in the same position in kth layer, and set to 1 otherwise.
- Step 2: Given the computed distance matrix $M_{p\times p}$, we then use the classic multidimensional scaling procedure from Kruskal (1964) to map the p points in the n dimensional space into the Euclidean space R^3. The distance between two points in R^3 is approximately equal to that in the original n-dimensional space. Finally, we plot a scatter graph of the obtained points in R^3 presented in Fig. 8.

Interestingly, Fig. 8 reveals that high-quality local optima tend to be grouped into clusters. On the one hand, the distance between solutions within a same cluster is small in general, which implies that exploitation should be reinforced to detect neighboring high quality local optima. Indeed, the solution-based tabu strategy of the MNSB-TS algorithm exploits this property by systematically launching a search from the best solution found so far to discover other nearby high-quality solutions. On the other hand, local optima from different clusters are generally separated by a large distance. In the case of the MNSB-TS algorithm, to discover a new cluster (that can contain new high-quality solutions), it is useful to apply strong diversification strategies (i.e., the multiple neighborhood mechanism and dynamic diversification strategy). Hence, the spatial distribution of high-quality solutions reinforces the rationale underlying the proposed strategies in MNSB-TS.

At last, we summarize and explain the key proposed components, as well as highlighting their corresponding motivation and their influence on our MNSB-TS in Table 11.

7. Conclusion and future works

We present a novel algorithm (MNSB-TS) for solving the incremental graph drawing problem (IGDP), whose key features include an



(a) incgraph_2_0.17_5_30_1.20_10

(b) incgraph_6_0.30_5_30_1.60_9

(c) incgraph_13_0.06_5_30_1.20_1

(d) incgraph_13_0.30_5_30_1.60_5

(e) incgraph_20_0.17_5_30_1.20_9

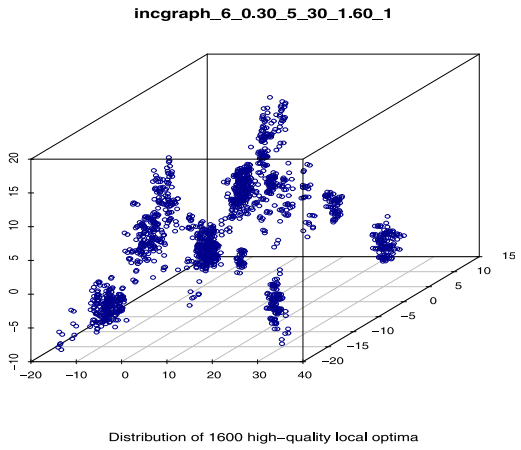
(f) incgraph_20_0.17_5_30_1.60_10

Fig. 7. Computational performance comparison with and without the fast neighborhood evaluation strategy.

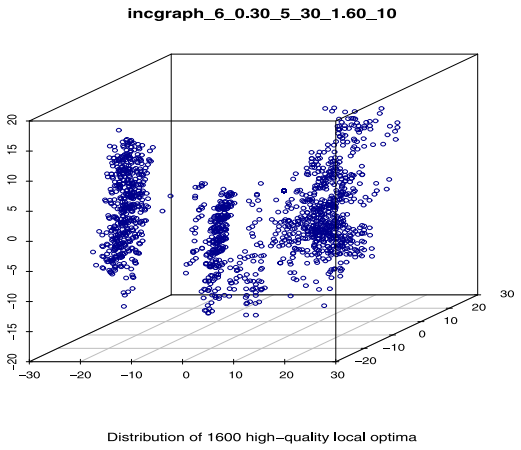
efficient solution-based tabu search that uses a multiple neighborhood structure with four neighborhood moves and a corresponding fast evaluation strategy for solution improvement, together with a dynamic diversification phase to encourage the search to explore new regions in the search space.

Experimental evaluations on extensive benchmarks show that our MNSB-TS algorithm competes very favorably with the current state-of-the-art algorithms, obtaining highly competitive results in terms of both computational efficiency and solution quality in comparison with the best-performing heuristics and the exact solver Gurobi. In addition, we carry out experimental analysis to reveal the effectiveness of the new features incorporated in the MNSB-TS algorithm.

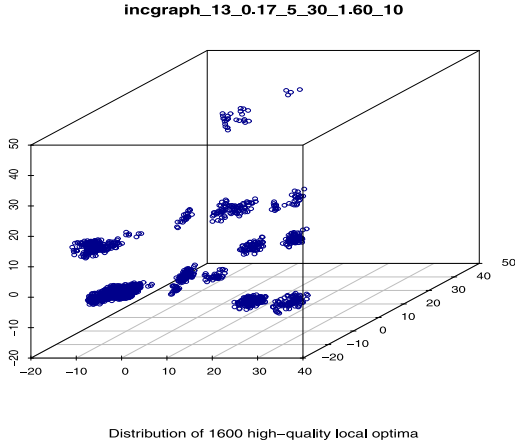
The main advantages of our multiple neighborhood solution-based tabu search can be summarized as follows: First, the solution-based tabu strategy provides a more effective tabu search procedure for the IGDP problem Second, our solution-based tabu strategy features a



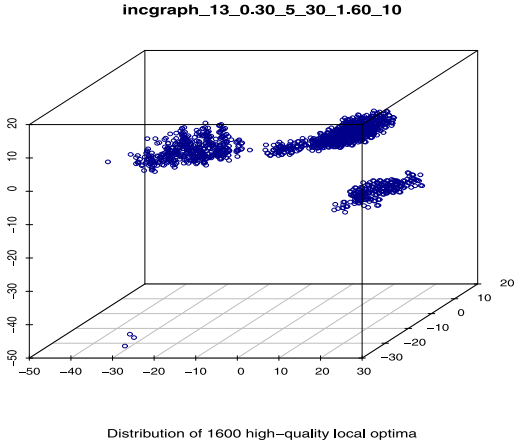
(a) Incgraph_6_0.30_5_30_1.60_1



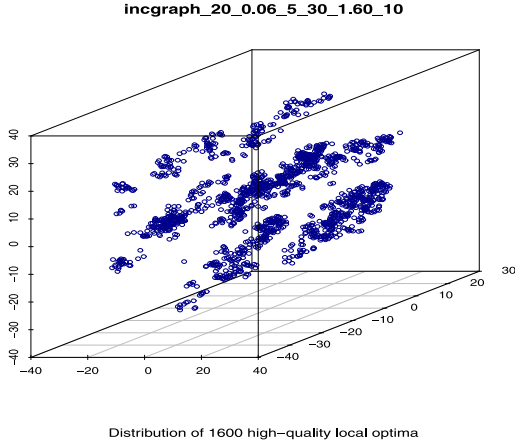
(b) Incgraph_6_0.30_5_30_1.60_10



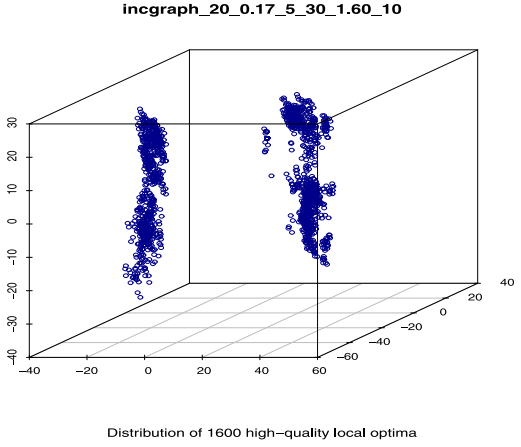
(c) Incgraph_13_0.17_5_30_1.60_10



(d) Incgraph_13_0.30_5_30_1.60_10



(e) Incgraph_20_0.06_5_30_1.60_10



(f) Incgraph_20_0.17_5_30_1.60_10

Fig. 8. Distribution of high-quality solutions produced by the MNSB-TS algorithm on the six representative instances.

simpler implementation structure to combine with the multiple neighborhood moves, since it does not contain tabu tenure and tabu aspiration, which are standard components in attribute-based tabu search. Third, compared with previous solution-based tabu search methods (such as Lai et al., 2018a, 2018b, 2018c; Wang et al., 2017), the MNSB-TS algorithm adopts a multiple neighborhood mechanism to enhance diversification. Extensive computational testing establishes the contribution of each of these advantages.

There are several issues for future work. It would be interesting to examine different protocols for multiple neighborhood search as proposed in the references cited in Section 1. Given the importance of diversification, it would also be interesting to replace randomization in our dynamic diversification approach by adaptive memory designs. Another option of interest is to combine our proposed strategies with population-based frameworks like genetic algorithms, path relinking or memetic algorithms. Finally, the success of our strategies in tackling the IGDP problem suggests that it would be worthwhile to investigate how they perform for solving other graph drawing problems such as the classic two-layer or multiple-layer graph drawing problem.

CRediT authorship contribution statement

Bo Peng: Methodology, Writing – original draft. **Songge Wang:** Software. **Donghao Liu:** Software. **Zhouxing Su:** Supervision, Writing – review & editing. **Zhipeng Lü:** Funding acquisition, Supervision, Project administration. **Fred Glover:** Methodology, Writing – review & editing.

Declaration of competing interest

We declare that we have no financial and personal relationships with other people or organizations that can inappropriately influence our work, there is no professional or other personal interest of any nature or kind in any product, service and or company that could be construed as influencing the position presented in, or the review of, the manuscript entitled.

Data availability

Data will be made available on request.

Acknowledgments

We thank Prof. Rafael Martí for giving us the benchmark instances and experimental results of their algorithms (i.e., SS and VNSS). The research is supported by Guanghua Talent Project of Southwestern University of Finance and Economics, National Natural Science Foundation of China (No. 72201216 and No. 71871184), and Sichuan Sicence and Technology Program (No. 2023NSFSC1019).

References

Battista, G. D., Eades, P., Tamassia, R., & Tollis, I. G. (1998). Graph drawing. In *Algorithms for the visualization of graphs*. Prentice Hall PTR.

Birattari, M., Yuan, Z., Balaprakash, P., & Stützle, T. (2010). F-race and iterated F-race: An overview.

Chang, J., Wang, L., Hao, J.-K., & Wang, Y. (2021). Parallel iterative solution-based tabu search for the obnoxious p-median problem. *Computers & Operations Research*, *127*, Article 105155.

Eades, P., Lai, W., Misue, K., & Sugiyama, K. (1991). *Preserving the mental map of a diagram*: *Tech. rep.*, Technical Report IIAS-RR-91-16E, Fujitsu Laboratories.

Fulek, R., & Tóth, C. D. (2020). Crossing minimization in perturbed drawings. *Journal of Combinatorial Optimization*, *40*(2), 279–302.

Garey, M. R., & Johnson, D. S. (1983). Crossing number is NP-complete. *SIAM Journal on Algebraic Discrete Methods*, *4*(3), 312–316.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, *13*(5), 533–549.

Glover, F. (1997). Tabu search and adaptive memory programming advances, applications and challenges. In *Interfaces in Computer Science and Operations Research* (pp. 1–75). Springer.

Glover, F., Campos, V., & Martí, R. (2021). Tabu search tutorial. A graph drawing application. *TOP*, *29*, 319–350.

Glover, F., McMillan, C., & Glover, R. (1984). A heuristic programming approach to the employee scheduling problem and some thoughts on managerial robots. *Journal of Operations Management*, *4*(2), 113–128.

Kruskal, J. B. (1964). Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, *29*(1), 1–27.

Laguna, M., Martí, R., & Valls, V. (1997). Arc crossing minimization in hierarchical digraphs with tabu search. *Computers & Operations Research*, *24*(12), 1175–1186.

Lai, X., Hao, J. K., & Dong, Y. (2018). Two-stage solution-based tabu search for the multidemand multidimensional knapsack problem. *European Journal of Operational Research*.

Lai, X., Hao, J. K., Glover, F., & L, Z. (2018). A two-phase tabu-evolutionary algorithm for the 0-1 multidimensional knapsack problem. *Information Sciences*, *436–437*.

Lai, X., Yue, D., Hao, J. K., & Glover, F. (2018). Solution-based tabu search for the maximum min-sum dispersion problem. *Information Sciences*, *441*.

Liu, X., Chen, J., Wang, M., Wang, Y., Su, Z., & L, Z. (2020). A two-phase tabu search based evolutionary algorithm for the maximum diversity problem. *Discrete Optimization*, Article 100613.

Lopez-Ibanez, M., Dubois-Lacoste, J., Caceres, L. P., Birattari, M., & Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, *3*, 43–58.

Martí, R., Campos, V., Hoff, A., & Peiró, J. (2018). Heuristics for the min–max arc crossing problem in graphs. *Expert Systems with Applications*, *109*, 100–113.

Martí, R., & Estruch, V. (2001). Incremental bipartite drawing problem. *Computers & Operations Research*, *28*(13), 1287–1298.

Martí, R., Martínez-Gavara, A., Sánchez-Oro, J., & Duarte, A. (2018). Tabu search for the dynamic Bipartite Drawing Problem. *Computers & Operations Research*, *91*, 1–12.

Mladenović, N., & Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, *24*(11), 1097–1100.

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

Palubeckis, G., Tomkevičius, A., & Ostreika, A. (2019). Hybridizing simulated annealing with variable neighborhood search for bipartite graph crossing minimization. *Applied Mathematics and Computation*, *348*, 84–101.

Pastore, T., Martínez-Gavara, A., Napoletano, A., Festa, P., & Martí, R. (2020). Tabu search for min-max edge crossing in graphs. *Computers & Operations Research*, *114*, Article 104830.

Peng, B., Liu, D., L, Z., Martí, R., & Ding, J. (2020). Adaptive memory programming for the dynamic bipartite drawing problem. *Information Sciences*, *517*, 183–197.

Porumbel, D. C., Hao, J.-K., & Kuntz, P. (2010). A search space cartography for guiding graph coloring heuristics. *Computers & Operations Research*, *37*(4), 769–778.

Sánchez-Oro, J., Martínez-Gavara, A., Laguna, M., Martí, R., & Duarte, A. (2017). Variable neighborhood scatter search for the incremental graph drawing problem. *Computational Optimization and Applications*, *68*(3), 775–797.

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

Tiezzi, M., Ciravegna, G., & Gori, M. (2022). Graph neural networks for graph drawing. *IEEE Transactions on Neural Networks and Learning Systems*.

Wang, Y., Wu, Q., & Glover, F. (2017). Effective metaheuristic algorithms for the minimum differential dispersion problem. *European Journal of Operational Research*, *258*(3).

Woodruff, D. L., & Zemel, E. (1993). Hashing vectors for tabu search. *Annals of Operations Research*, *41*(2), 123–137.

Wu, X., Xiong, C., Deng, N., & Xia, D. (2021). A variable depth neighborhood search algorithm for the Min-Max Arc Crossing Problem. *Computers & Operations Research*, *134*, Article 105403.

Xu, J., Chiu, S. Y., & Glover, F. (1996). Probabilistic tabu search for telecommunications network design. *Combinat. Optim. Theory Pract.*, *1*(1), 69–94.

Xu, J., Chiu, S. Y., & Glover, F. (1998). Fine-tuning a tabu search algorithm with statistical tests. *International Transactions in Operational Research*, *5*(3), 233–244.