Adaptive Memory Programming for the Dynamic Bipartite Drawing Problem

Bo Peng[a], Donghao Liu[b], Zhipeng Lü[b,*], Rafael Martí[c,*], Junwen Ding[b]

[a]*School of Business Administration, Southwestern University of Finance and Economics, Chengdu, 610074, P.R. China*
[b]*SMART, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074, P.R. China*
[c]*Departamento de Estadística e Investigación Operativa, Universidad de Valencia, Spain*

Abstract

The bipartite drawing problem is a well-known NP-hard combinatorial optimization problem with numerous applications. Its aim is to minimize the number of edge crossings in a two-layer graph where the edges are drawn as straight lines. In this work, we tackle the dynamic variant of this problem, which is called the dynamic bipartite drawing problem (DBDP), which consists in adding or removing vertices and edges to a given bipartite drawing, and drawing the resulting graph with a similar layout as the original one. To solve this problem, we propose a Tabu Search method (Glover and Laguna, 1998) that incorporates adaptive memory, to search the solution space in an efficient way. In this study, we compare the explicit memory in our solving method (called iterated solution-based tabu search, ISB-TS) with the previous best method based on attributive memory, thus comparing these two memory implementations. Starting from a semi-greedy initial solution, ISB-TS iteratively explores the search space by combining a solution-based tabu search with an adaptive perturbation mechanism to escape from local

*Corresponding author.
Email addresses: pengbo@swufe.edu.cn (Bo Peng), zhipeng.lv@hust.edu.cn (Zhipeng Lü), rafael.marti@uv.es (Rafael Martí)

optima. The extensive computational experiments on two sets of more than 1,000 problem instances indicate that the proposed ISB-TS is highly competitive in comparison with existing methods. Key components of the approach are analyzed to evaluate their impact on the proposed algorithm and learn which search mechanisms are better suited for this type of problems.

Keywords: Dynamic bipartite drawing problem; solution-based tabu search; constrained neighborhood structure; adaptive perturbation mechanism.

1. Introduction

With the advent of the era of big data, more and more complex information systems require data visualization for analysis and presentation. Graphs have become a fundamental modeling tool to represent and analyze data in many fields such as software engineering (Burch et al. (2012)), VLSI circuit design (Wu et al. (2006)), or information visualization (Herman et al. (2000)). Graph drawing addresses the problems of constructing geometric representations of graphs in a way that they are easy to analyze. Although the criteria used to judge the quality of a drawing is quite subjective, the goal of minimizing the number of edge crossings is a widely accepted standard for a good drawing. The problem to minimize the number of edge crossings in a two-layer graph where the edges are drawn as straight lines is called bipartite drawing problem (BDP), which is well-known to be NP-hard (Johnson (1982)).

This research, as many others in the field, is built upon the so-called Sugiyama's framework. Given a directed graph, this framework or drawing standard (Sugiyama et al. (1981)) arranges vertices on a series of equidistant parallel vertical lines called layers in such a way that the drawing has short straight lines, pointing in a uniform direction with low number of crossings. This drawing standard obtains a hierarchical or layered graph in which arc crossing minimization only depends on the ordering of the vertices in each layer.

The crossing minimization problem in hierarchical digraphs has received a lot of attention given that with Sugiyamas framework any graph can be transformed into a hierarchical graph, and therefore methods developed to hierarchies can be applied to any general graph. In line with this, we can find many papers on the particular case of bipartite graphs, since they can be viewed as hierarchical graphs with 2 layers. Solving methods to minimize the number of crossings in a 2-layered graph, can be easily adapted to the general case of a hierarchical graph and thus to directed graphs. Note that the problem of minimizing the number of arc-crossings is NP-Complete, even when the graph consists of only two layers (Garey and Johnson (1983)).

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 has to be represented in a similar way (layout) as the original one.

Erten et al. (2003) considered the dynamic graph drawing problem of a graph evolving over time. A dynamic (or incremental) graph is modified from an original graph by removing and adding edges and vertices. The challenge in this context is to create a mathematical model addressing the idea that the original elements are arranged in the final drawing in a similar way as in the original one. In fact, in a wide variety of practical situations, it is helpful to maintain the *mental picture* of the layout from a graph over successive drawings. When the vertices are deleted from or added to a graph, the users have to adjust their mental map to be familiar with the new modified graph (Napoletano et al., 2019). It is therefore desirable to draw the new drawing as similar as possible to the original one, to help the user to adjust this mental or abstract representation of the graph over successive drawings.

The edge crossing minimization has received a lot of attention. However, the dynamic graph drawing problem has more practical significance by considering the real-time scenarios with removing and adding edges and vertices for the original (or static) graph. In fact, the static bipartite drawing problem can be considered as a special case of dynamic bipartite drawing problem, when all the vertices in the graph are incremental vertices, which can be placed arbitrarily. Hence in this study we focus on the general and representative dynamic graph drawing problem.

Martí and Estruch (2001) introduced the concept of the stability across drawings by keeping the relative ordering among the common vertices in the original graph and the new one. After that, the term dynamic bipartite drawing problem (DBDP) was introduced by Martí et al. (2018) to describe the problem of incremental edge crossing minimization for bipartite graphs, which is also NP-hard according to the crossing number theory by Garey and Johnson (1983).

The literature on incremental bipartite graph drawing is quite scarce. Martí and Estruch (2001) proposed an exact algorithm based on the branch and bound method that explores the set of solutions (permutations of the vertices in each layer) with a combinatorial search tree. Additionally, they developed a heuristic procedure based on the greedy randomized adaptive search procedure (GRASP) to provide high-quality solutions for medium and large size instances, since the exact algorithm can only solve small instances with less than 32 vertices. After that, Martí et al. (2018) adapted the mathematical programming formulation originally developed for the standard bipartite drawing problem (Jünger and Mutzel (1997)) to the incremental case of the DBDP. Their experiments show that Gurobi (a general optimization solver) is able to solve in many case small and medium size instances with 50 and 100 vertices to optimality. In addition, they proposed a heuristic method to solve large instances in short computational times. Their proposed hybrid tabu search and path relinking algorithm is able to generate high-quality solutions for larger instances with up to 471 vertices.

Heuristic methodologies can be classified in terms of their use of memory. This motivates the definition of the area called Adaptive Memory Programming. Approaches such as Tabu Search (TS) or Path Relinking (PR) (Glover et al., 2000) are memory oriented methods in which records about past choices and decisions determine future strategies. Because of that, they are also called intelligent methods. Other methodologies, such as Simulated Annealing (SA) or Genetic Algorithms (GA), do not incorporate the explicit use of memory structures and are based on other efficient strategies and search mechanisms. In this paper, we undertake to explore a key component of memory-based methods related with the way in which memory is implemented.

As one of the most successful metaheuristic methodologies for solving a wide variety of combinatorial optimization problems (Zhou et al. (2016), Silvestrin and Ritt (2017), Zhou et al. (2018), and Li et al. (2018)), Tabu Search (Glover and Laguna, 1998) incorporates adaptive memory which allows the implementation of procedures that are capable of searching the solution space in an efficient way. The memory used in tabu search can be explicit or attributive. Explicit memory records complete solutions while attributive memory records attributes or properties usually related with moves. Most tabu search implementations are based on attributive memory structures (Glover and Laguna (1998), Peng et al. (2015) and Zhou et al. (2016)). We explore here an alternative based on the explicit use of memory, in which we include hash functions to speed up the evaluation process associated with recording complete solutions. In particular, we apply the solution-based tabu search, which usually implements a memory structure based on a hash function to prevent the cycling in the search, which creates an efficient intensification pattern (Woodruff and Zemel, 1993). Interestingly, solution-based tabu search methods began to attract attention only very recently, and have already presented competitive performance in several binary optimization problems (such as minimum differential dispersion problem (Wang et al., 2017), multidemand multidimensional knapsack problem (Lai et al., 2018a), and maximum min-sum dispersion problem (Lai et al., 2018c)).

Although permutation problems can be formulated as binary problems, they constitute a special class that preferably should be treated somewhat differently (Glover and Hao (2017)). The research on solution-based tabu search using hashing mechanism for permutation problems is relatively scare. Fink and Voß (2003) defined the so-called hash code to prevent the search to revisit the previously visited solutions for continuous flow-shop scheduling problem. Liao and Huang (2011) employed the tabu search based on hash mechanism for two-machine flowshop scheduling with batch processing machines problems. Kulturel-Konak (2012) utilized the linear programming embedded probabilistic tabu search based on hashing strategy for unequal-area facility layout problem.

In this paper we undertake to explore the solution-based tabu search for the DBDP. To the best of our knowledge, this method has never been used to address other graph drawing problems. This work can be a guide to the research based on solution-based tabu search for permutation problems. In particular, a dedicated hash function is proposed for DBDP. The hash function values of the neighborhood solutions can be quickly calculated at constant time complexity. In addition, we compare our explicit, or solution-based, tabu search with the best previous method (i.e., an attributive tabu search procedure) for the DBDP, providing the user with an empirical comparison of these alternative ways to implement memory structures. Therefore, the aim of this study is to employ solution-based tabu search approach to tackle an important permutation problem, the DBDP.

The main contributions of this paper are summarized as follows:

- We propose a constrained neighborhood structure with a fast evaluation mechanism, and a new hash function embedded in the proposed solution-based tabu method for search intensification.

- Our method includes an adaptive perturbation phase for effectively escaping from local optima (search diversification).

- We perform extensive experimentation on previously reported instances to compare our method with the best previous method. The comparison favors our proposal.

- Our study reveals that solution-based memory structures implemented with hash functions are very fast and can be more efficient than the traditional attributive structures in terms of search intensification in comparison with the previous work (Martí et al., 2018) for DBDP.

Given that the ideas of the iterative solution-based tabu search framework and the proposed hash function are quite general for permutation problems, they could be applied to solve other related combinatorial optimization problems.

The rest of the paper is organized as follows. Section 2 presents the problem description and mathematical model of the DBDP problem. Section 3 describes the proposed iterative solution-based tabu search algorithm. Section 4 reports experimental results and comparisons with state-of-the-art algorithms from the literature. Section 5 evaluates the effectiveness of several key ingredients of the proposed algorithm. Concluding remarks are given in Section 6.

2. Problem Description and Previous Methods

The classic bipartite drawing problem (BDP) has received a lot of attention since the seminal work by Eades and Kelly (1986) with heuristics based on simple ordering rules, to the complex branch and cut proposed by Jünger and Mutzel (1997). A bipartite (or two-layered) graph is defined as $G = (V_1,V_2,E)$ where V_1, V_2 and E, respectively denotes the vertices of the left layer, the vertices of the right layer, and the edges in graph G. The number of the vertices in each layer is denoted by $|V_1| = m_1$ and $|V_2| = m_2$. A drawing D (BDP solution) is determined by the ordering π_1 of V_1, and the ordering π_2 of V_2, which can be denoted by $D = (\pi_1, \pi_2)$. The position of vertex u in left layer and right layer are denoted by $\pi_1(u)$ and $\pi_2(u)$, respectively. If vertex u precedes vertex v then $\pi_1(u) < \pi_1(v)$. Likewise, $\pi_1(u) > \pi_1(v)$ when v precedes u.

Table 1: Symbols and definitions.

Symbol	Definition
G	Original graph $G = (V_1,V_2,E)$
V	Set of original vertices in G
E	Set of original edges in G
D	Original drawing $D = (\pi_1,\ \pi_2)$ for G
$\pi_k(u)$	Position of vertex u in its layer k (k = 1,2) of G
IG	Incremental graph $IG = (IV_1,IV_2$, IE)
IV_k	Set of all the vertices in layer k of IG, and $IV = IV_1 \cup IV_2$
AV_k	Set of incremental vertices in layer k of IG, and $AV = IV_1 \cup IV_2$
IE	Set of incremental edges in IG
$lf(u)$	Label of the layer (first or second layer) containing vertex u
n_k	Number of vertices in layer k (k = 1,2) for IG, and n = n_1 + n_2
S	Incremental drawing (i.e., solution) $S = (\Pi_1,\ \Pi_2)$ for IG
$\Pi_k(u)$	Position of vertex u in its layer k of IG
$\omega_k(i)$	Vertex locating i^{th} position in layer k (k = 1,2) 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 $\Pi(u) > \Pi(u^{'}) > \Pi(v)$; $lf(u) = lf(v) = lf(u^{'})$}

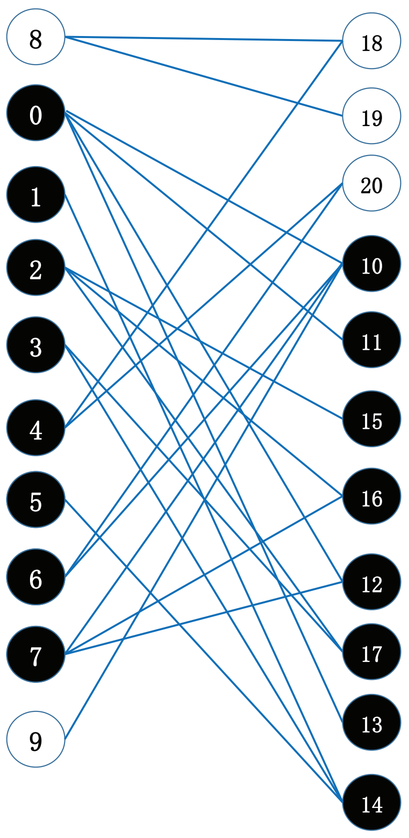
In this study, we tackle an extended version of the DBP called the dynamic bipartite drawing problem (DBDP) proposed by Martí et al. (2018), which results from adding two sets of vertices AV_k, with their corresponding edges AE_k ($k = 1, 2$) into two layers for obtaining an incremental graph.

Formally, the incremental graph is denoted by $IG = (IV_1, IV_2, IE)$ where $IV_k = V_k \cup AV_k$, $IE = E_k \cup AE_k$ and $|IV_k| = n_k$ $(k = 1, 2)$. In this study, we call each vertex in original graph G, as original vertex, while the vertex added to the incremental graph IG, as incremental vertex. The goal of the DBDP is to find a drawing with the minimum number of edge crossings while preserving the relative position of the original vertices. Obviously, the BDP problem can be considered as a special case of the DBDP problem if there exist no original vertices (i.e., the set of original vertices is null). We denote by $S = (\Pi_1, \Pi_2)$ to a solution of the DBDP over graph IG, where Π_1 and Π_2 respectively denote the permutation of the vertices in IV_1 and IV_2. In addition, the position of vertices u and v in the ordering of Π_1 and Π_2 can be denoted by $\Pi_1(u)$ and $\Pi_2(v)$, respectively. Table 1 summarizes all the symbols and definitions introduced in this study.

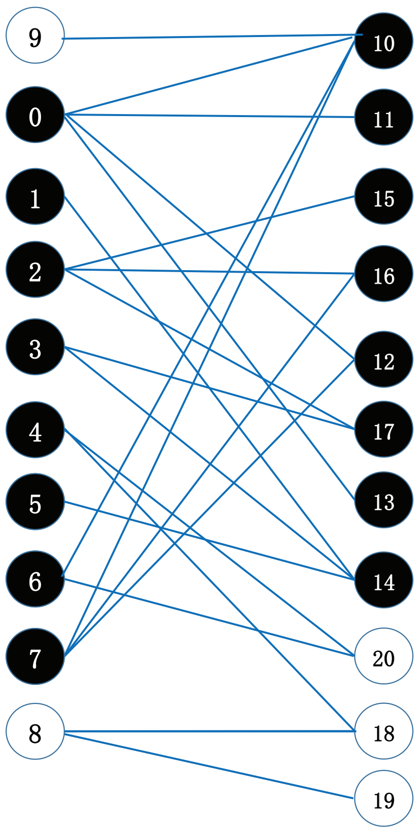
2.1. Problem motivation

It is well documented that incremental drawing is a very important area in graph representations. We can find lots of references highlighting this problem, or family of problems (Pinaud et al., 2004). In affiliation networks, individuals and groups are depicted with vertices, and edges represent the membership of individuals to those groups. These networks usually change in time, since new groups and members are systematically added. When these additions occur, it is desirable that the new layout is both aesthetically pleasing and preserves dynamic stability (i.e., it stands well into the sequence of drawings). Hence we can handle these vertices as the incremental vertices. The situation "dynamic stability" we deal with in this study is therefore very representative.

Figure 1 shows an example of two drawings (i.e., solutions) of a bipartite graph with 21 vertices for the incremental bipartite drawing problem. The black vertices denote the original elements that should keep the relative posi-



(a) The first drawing (79 edge crossings)



(b) The second drawing (44 edge crossings)

Figure 1: An example (GB_1_rnd1_01_0001_20 instance) of two drawings of a two-layered graph for the incremental bipartite drawing problem.

tion in the graph while the white vertices denote the new (incremental) ones added which can be arbitrarily moved. The number of the edge crossings in the original graph (without the incremental elements) is equal to 33. The first drawing in Figure 1 has 79 edge crossings, and can be optimized according to the dynamic graph drawing problem (i.e., by relocating the new white nodes in the position that minimizes the number of crossings while preserving the relative position of the original nodes), resulting in the second drawing with 44 edges crossings by moving the incremental (or white) vertices.

In some real dynamic scenarios, there exist not only the cases of adding vertices, but also the cases of deleting vertices from the original graph. The situation with only deleting some vertices from the original graph can be considered as the classic bipartite drawing problem, since the elements in the final graph do not have any additional constraint, which has been extensively studied in the literature (Eades and Wormald (1994), Valls et al. (1996), Martí (1998)). In this study, we consider both adding and deleting vertices in the original graph, which can be called the decremental bipartite drawing problem. Note that this is the first time that the decremental problem is studied.

Since the first procedure developed by Carpano (1980), most heuristics for the BDP are structured in a similar way. Specifically, the procedures first order one layer employing a simple rule, while keeping the position of the vertices in the other layer fixed. Then, the other layer is ordered and the process repeats until two successive iterations occur in which the relative positions of the vertices remain the same. The best-known approach to solve the BDP is the so-called Barycenter Method (Battista et al., 1998), which is similar to Carpano's algorithm. In this method, the position of a given vertex is calculated as the arithmetic mean of the positions of its adjacent vertices. The basic principle of this rule is that crossings are likely to be minimized by increasing the number of horizontal arcs. Mäkinen (1989) then developed the Median heuristic. This ordering rule is similar to the barycenter method, with the difference that it employs the median instead of the average to compute the position of each vertex. After that, Martí and Laguna (2003) proposed two metaheuristics, tabu search and GRASP to tackle the dense and low-density graph, respectively. Although these methods are proposed very early, part of them are still being used nowadays.

Recently, in the scientific literature, some efforts were carried out in order to solve dynamic (incremental) graph drawing problems. Martí and Estruch (2001) proposed a GRASP method based on the statistical sampling of the solution space for the DBDP. A randomization component in the construction phase has the objective of obtaining relatively diverse solutions, thus having candidate solutions in different regions of the search space. These solutions are then submitted to a local search to locate the corresponding local optima. Very recently, Martí et al. (2018) proposed a method combining tabu search and path relinking for the same problem obtaining better solutions than the previous GRASP. As it is customary in tabu search procedures, when the method selects a new vertex v and moves it, it is recorded in a dedicated memory structure (the so called tabu list), in order to prohibit to move it in the next iterations. In addition, the tabu search method is coupled with path relinking for creating paths between two high quality drawings in order to generate new and better solutions.

3. The Tabu Search Method

Solution-based tabu search approach is able to obtain a stronger intensification ability, which is crucial for locating good local optima, since it usually relies on a hash function and a hash vector to record the whole information of a solution instead of one move or its attribute typical in the standard attributive tabu search, thus reducing the probability of revisiting the same solution and inducing a strong intensification pattern. We propose an iterative solution-based tabu search (ISB-TS) combined with an adaptive perturbation mechanism for solving the DBDP. Its general scheme and key components are presented in the following subsections.

3.1. General scheme

Our ISB-TS algorithm follows the basic scheme of the iterated local search (ILS) (Lourenco et al., 2010). The basic idea is to apply a solution-based tabu search procedure to intensify the search in a given search space region, and to employ an adaptive perturbation mechanism to move to a new search region once a local optimum is attained. Inspired by the breakout local search (Benlic and Hao (2013); Fu and Hao (2014)), ISB-TS has a particular focus on the importance of the perturbation mechanism to introduce a suitable degree of diversification at a certain stage of search.

More precisely, the general scheme of ISB-TS presented in Algorithm 1 can be described as follows: Starting from an initial solution constructed by a dedicated constructive procedure (line 1), ISB-TS initializes parameters, i.e., the best-found solution S^{best}, the perturbation strength L and the counter ξ for consecutive non-improving local optima (lines 2-4). After the initialization, it applies a solution-based tabu search method to reach a local optimum S (line 6). Then the best solution S^{best} found so far, and the counter ξ of consecutive non-improving local optima are updated (lines 7-12). After each local optimization, ISB-TS tries to move from local optimum to another one by employing varying perturbations, depending on the state of the search. If the search finds a better local optima S than S_p, ISB-TS switches to weaker perturbations subsequently due to the search escaped from the previous local optimum (lines 13-14). Otherwise, ISB-TS perturbs S more strongly by increasing the perturbation strength L since the search returns to the previous local optima S_p (lines 15-16). After visiting a certain number ξ_{max} of local optima without improving the best solution found so far, ISB-TS applies a significantly stronger perturbation in order to drive definitively the search towards a new and more distant region in the search space (lines 17-20). When the perturbation strength is determined, the perturbation operator is employed to generate a new solution in a different area of the search s-pace, and the symbol S_p preserves the previous solution (lines 21-22). The solution-based tabu search and adaptive perturbation mechanism iteratively alternate until the stopping criterion (i.e., the maximum computing time T_{max}) is met, and finally returns the best found solution S^{best} as the final result (line 24).

The performance of the proposed ISB-TS algorithm relies on three key factors. First, the initialization procedure should be able to generate different solutions of reasonable quality, which serve as the restarting points of independent runs of ISB-TS. Second, the local optimization procedure is also a key component because different local optimization search strategies lead to different search trajectories, thus solutions of different qualities. Third, we should control the perturbation strength, denoted by L, which determines the perturbation intensity applied to the current solution. In our case, this corresponds to decide how many vertices to delete when perturbing the incumbent solution. Indeed, if L is too small, the search usually returns to the original local optimum, leading to search stagnation. Otherwise, if L is too large, the perturbation is reduced to random restarting. The components of the proposed ISB-TS algorithm for the DBDP are described in the following subsections.

3.2. Initial solution procedure

The ISB-TS algorithm uses a greedy randomized strategy to construct the initial solution. We basically implement a GRASP constructive method (Festa and Resende, 2010). Starting from a partial solution only containing all the original vertices, the constructive procedure iteratively inserts each

Algorithm 1: Framework of the iterative solution-based tabu search

```
Input: Benchmark instance (B); The maximum computing time ($T_{max}$)
Output: Best-found solution ($S^{best}$)
1  $S \leftarrow GenerateInitialSolution(B)$ ;
2  $S^{best} \leftarrow S$ ;
3  $L \leftarrow L_{min}$    /* L records the perturbation strength */ ;
4  $\xi \leftarrow 0$    /* $\xi$ denotes the counter for consecutive non-improving local optima */ ;
5  while The maximum computing time $T_{max}$ is not reached do
6      $S \leftarrow SolutionbasedTabuSearch(S)$ ;
       // Update the best solution $S^*$ found so far, and increase the counter $\xi$ 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
       // Determine the perturbation strength $L$ adaptively.
13     if $S$ is better than $S_p$ then
           // Search escaped from the previous local optimum, re-initialize
           perturbation strength.
14         $L \leftarrow L_{min}$ ;
15     else if $S$ is not better than $S_p$ and $\xi < \xi_{max}$ then
           // Search returned to the previous local optimum, increment perturbation
           strength.
16         $L \leftarrow Min(L + 1, L_{max})$ ;
17     else if $\xi \geq \xi_{max}$ then
           // Search seems to be stagnating, strong perturbation required.
18         $L \leftarrow L_{max}$;
19         $\xi \leftarrow 0$ ;
20     end
       // Perturb the current local optimum $S$ with perturbation strength $L$.
21     $S_p \leftarrow S$;
22     $S \leftarrow Perturb(L, S)$ ;
23 end
24 return $S^{best}$
```

incremental vertex into the initial partial solution to make it complete. The initial solution procedure is presented in Algorithm 2. More precisely, we first generate the initial partial solution S_0 consisting of the permutations π_1 and π_2 of original vertices V_1 and V_2 in the bipartite graph, and assign all the incremental vertices from AV_1 and AV_2 into AV (lines 1-3). Then, 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 (i.e., candidate list) records all the couples of remaining vertices in AV and its corresponding feasible positions (line 5). The restricted candidate list RCL is created, which contains the $\max\{1, \alpha * |CL|\}$ vertices with the minimum incremental objective value δ in CL (line 6). We 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 (lines 8-9). Note that, the notation $\oplus$ used in this study indicates the operation of applying the underlying move operator on solution S_0 to produce the corresponding neighboring solution. Hence, $S_0 \oplus (v_c, p_c)$ in line 8 denotes that we randomly choose a vertex v_c and insert it into the position p_c of the initial solution S_0. When vertex set AV becomes empty, the initial solution phase terminates and the complete solution S_0 found during the search process is returned as the output of the initial solution procedure (line 11).

3.3. Tabu search phase

Our ISB-TS algorithm adopts a solution-based tabu search procedure to locate local optima by employing a constrained neighborhood structure with its fast evaluation mechanism and the dedicated solution-based tabu strategy. The details of the solution-based tabu search phase are described as follows.

3.3.1. Neighborhood structure

To move from one solution to another one in search space, the previous work by Martí et al. (2018) only considers to insert one vertex from its

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

```
   Input: Benchmark instance (B);
   Output: Initial solution ($S_0$)
1  Generate $\pi_1$, $\pi_2$, $AV_1$ and $AV_2$ from benchmark instance $B$;
2  $S_0 = (\pi_1, \pi_2)$ ;
3  $AV \leftarrow AV_1 \cup AV_2$ ;
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      $RCL \leftarrow \{(v, p) \in CL : |\{(v', p') : \delta(v', p') \geq \delta(v, p)\}| \leq max\{1, \alpha * |CL|\}\}$;
7      /*Randomly choose a vertex $v_c$ and insert it into the position $p_c$ of the initial solution $S_0$
       (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 $S_0$
```

current position to the previous position or the posterior position, which is too limited to search extensively. On the other hand, moving a vertex to a new position that is far from its current position usually cannot significantly improve solution quality, we thus utilize the distance restriction technique to guide the constrained neighborhood search also based on the widely-used insertion move, i.e., removing a vertex from its position and inserting it into a new position.

To be specific, we first define the distance for an insert move as the number of vertices between the previous position and the new position to be inserted. Then, we introduce a distance threshold Φ for the insert move, such that moves with a distance larger than this threshold are not considered. These move restrictions help us to save a large amount of computing time without sacrificing much of solution quality.

Given an incremental graph $IG = (IV_1, IV_2, IE)$, a candidate solution S and two vertices u and v, the proposed constrained neighborhood structure (i.e., set of partial neighbor solutions) consists of the following two neighborhood operators (i.e., $CN(S) = N_1(S) \cup N_2(S)$) which can be defined as follows:

- Neighborhood operator N_1: Insert an incremental vertex upwards (or downwards) another vertex under the distance restriction. Formally, the corresponding neighborhood can be written as follows:

$$N_1(S) = \{S \oplus Insert(v, u) : \ v \in AV, \ u \in IV, u \neq v; \ |\Pi(u) - \Pi(v)| \leq \Phi\} \tag{1}$$

- Neighborhood operator N_2: Insert one original vertex upwards (or downwards) any other incremental vertex, while keeping the ordering of the original vertices under the distance restriction. Formally, the corresponding neighborhood can be given by:

$$\begin{aligned} N_2(S) = \{S \oplus Insert(v, u) : \ v \in V, \ u \in AV; \ IM(v, u) \cap AV = \emptyset; \\ |\Pi(u) - \Pi(v)| \leq \Phi\} \end{aligned} \tag{2}$$

where $IM(v, u)$ denotes all intermediate vertices between vertices u and v in the same layer.

The neighborhood move can be defined by vertex v removing from its current position in the solution S and inserting immediately upwards (or downwards) another vertex u if vertex v is downwards (or upwards) u ($u \neq v$), denoted by $Insert(v, u)$. If vertex v is an incremental vertex, the neighborhood move produces a total of n_1 - 1 (or $n_2 - 1$) possible candidate positions (i.e., solutions) for each vertex. In this case, all the neighboring solutions produced should be feasible. Nevertheless, if vertex v is an original vertex, the new generated solution has to keep the relative order of the original vertices. To exploit the above defined neighborhoods, our ISB-TS employs the best-improvement mechanism to accept the best one among the neighboring solutions each time.

If all the vertices in the incremental graph are incremental vertices, there would be n_1 (or n_2) candidate vertices to be moved in the first (or second) layer, and $n_1 - 1$ (or $n_2 - 1$) possible positions for each candidate vertex. The corresponding size of the neighborhood can be denoted by $SN = n_1 \times (n_1 - 1) + n_2 \times (n_2 - 1)$. Note that there usually exist some original vertices in the graph, and in this study we only explore a fraction of these solutions due to the distance constraint imposed in the exploration, Therefore, the size of the neighborhood employed in our method is less than SN. The complexity of both two neighborhood operators employed in our method is bounded by $O(n^2)$.

3.3.2. Fast neighborhood evaluation strategy

As stated above, there exist a number of candidate neighborhood solutions to be evaluated due to the corresponding neighborhood moves. In fact, after the neighborhood move, the crossing numbers of most vertices in graph remain unchanged thus it is unnecessary to recalculate the sum of the crossing numbers for all the vertices. In order to efficiently evaluate the changes of the objective values of the candidate neighborhood moves, we utilize two matrices to record the information with the number of edge crossings for two vertices.

Given two vertices u and v in the same layer of the graph, let NC_{uv} denote the number of crossings for the edges incident to u and the edges incident to v, when u precedes v in the same layer (i.e., $\Pi_k(u) < \Pi_k(v)$, k = $lf(v)$ = $lf(u)$) (and NC_{vu} when v precedes u). The matrices M_k in the k^{th} layer (i.e., first layer or second layer) thus can be given as follows:

$$M_k(u, v) = [NC_{uv}]; \quad \forall u \in IV_k, v \in IV_k, u \neq v, k \in \{1, 2\} \tag{3}$$

Figure 3 depicts the matrices of the example in Figure 2 to illustrate how they save the information with the number of edge crossing for each pair of vertices. The first matrix M_1 records the number of crossings for the edges

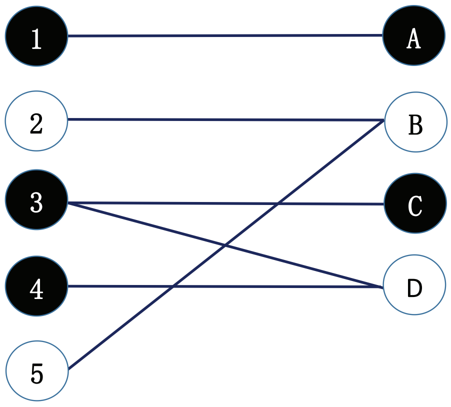


Figure 2: An example for one drawing for a two-layered graph with nine vertices.

$$M_1 = \begin{bmatrix} - & 0 & - & - & 0 \\ 1 & - & 0 & 0 & 0 \\ - & 2 & - & - & 2 \\ - & 1 & - & - & 1 \\ 1 & 0 & 0 & 0 & - \end{bmatrix}$$

(a) The first matrix

$$M_2 = \begin{bmatrix} - & 0 & - & 0 \\ 2 & - & 1 & 2 \\ - & 1 & - & 1 \\ 2 & 2 & 1 & - \end{bmatrix}$$

(b) The second matrix

Figure 3: The number of edge crossing for pairs of vertices in matrices

incident to the vertices in the first layer. Clearly, we can observe that the number of crossings for two edges incident to vertex 3 (i.e., (3, C) and (3, D)) and the edge incident to vertex 5 (i.e., (5, B)) is 2, since both edges (3, C) and (3, D) cross edge (5, B). Therefore, $NC_{35} = 2$ and $M_1(3, 5) = 2$, as shown in Figure 3(a) (row 3, column 5 in matrix M_1). On the other hand, edges (3, C) and (3, D) would not cross edge (5, B) if we insert vertex 5 upwards vertex 3 (i.e., after vertex 5 precedes vertex 3). That is why, in Figure 3(a), the value of row 5 and column 3 in matrix M_1 is 0 (i.e., $M_1(5, 3) = 0$), which means the number of crossings for the edges incident to 5 and the edges incident to 3 is 0, when vertex 5 precedes vertex 3 in the first layer.

Similarly, the second matrix M_2 records the number of crossings for the edges incident to the vertices in the second layer. $NC_{AB} = 0$ since the edge incident to vertex A (i.e., (1, A)) does not cross two edges incident to vertex B (i.e., (2, B) and (5, B)). As shown in Figure 3(b), the value of row 1 and column 2 in matrix M_2 is 0 (i.e., $M_2(1, 2) = 0$), which means the number of crossings for the edges incident to A and the edges incident to B is 0, when A precedes B in the second layer. Note that, A-D corresponds to $1 - 4$ in the matrix. When vertex B precedes vertex A (i.e., after vertex B is inserted upwards vertex A), the edge incident to vertex A (i.e., (1, A)) would cross two edges incident to vertex B (i.e., (2, B) and (5, B)). This is the reason why $NC_{BA} = 2$ and $M_2(2, 1) = 2$, as shown in Figure 3(b) (row 2, column 1 in matrix M_2). Note that the precedence between original vertices must be kept unchanged. Therefore $M_1(u, v)$ or $M_2(u, v)$ ($\forall u, v \in V$) is denoted by the symbol '-', which means that its value does not matter.

Making use of these two matrices, we propose a fast neighborhood evaluation mechanism to efficiently obtain the objective value of each move. For example, we insert vertex v immediately before (upwards) u_{j-1} or u_j in Figure 4. 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

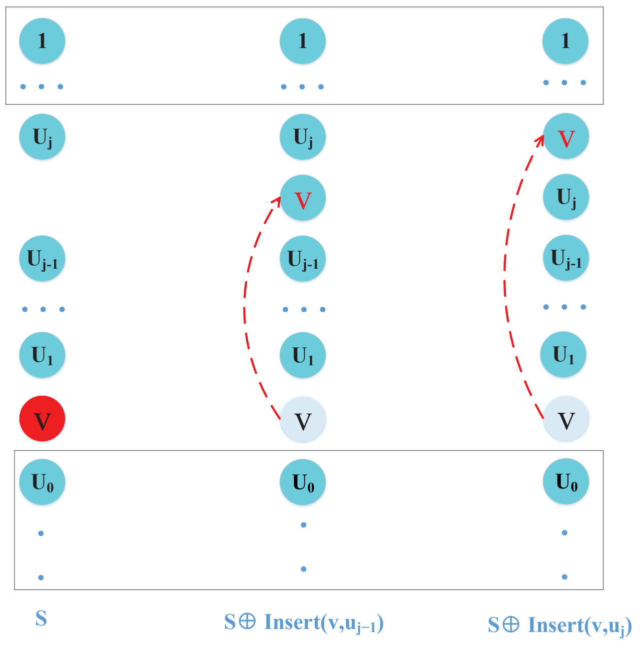


Figure 4: Insert vertex v immediately before (upwards) vertex u_{j-1} or u_j.

difference between u_{j-1} and v as the changed objective value with the two neighborhood moves. Given that 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}, while becoming $f(S \oplus Insert(v, u_j))$ after being inserted immediately before u_j, then we can calculate the objective value $f(S \oplus Insert(v, u_j))$ of the generated solution based on the neighborhood move $Insert(v, u_j)$ by the following formula:

$$f(S \oplus Insert(v, u_j)) = f(S \oplus Insert(v, u_{j-1})) - M(u_j, v) + M(v, u_j) \quad (4)$$

Where M denotes the corresponding evaluation matrix of vertex v.

To efficiently evaluate all the neighboring solutions, we can 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 one, according to the following equation:

$$f(S \oplus Insert(v, u_1)) = f(S) - M(u_1, v) + M(v, u_1) \tag{5}$$

Afterwards, making use of Equation 4 iteratively, we can quickly calculate the other objective values when vertex v is inserted immediately before other vertices. Likewise, the evaluation results of objective value downwards moves of vertex v immediately after other vertices can be iteratively calculated in a similar manner.

Algorithm 3: The procedure of updating the evaluation matrix for neighborhood move $Insert(v, u)$

```
Input: The previous evaluation matrix $M$
Output: The new evaluation matrix $M$ after updating strategy, where the matrix $M$
        corresponds to the opposite layer of vertex $v$
1  for $i \in Q$  where  $Q = \{IM(v, u) \cup \{u\}\}$ do
2      for $j \in A(i)$ do
3          $count(j) \leftarrow count(j) + 1$ ;
4      end
5  end
6  for $i \in A(v)$ do
7      for $j \in A(Q)$ do
8          $M(i, j) \leftarrow M(i, j)$ - $count(j)$;
9          $M(j, i) \leftarrow M(j, i) + count(j)$;
10         /*The $M$ matrix can be denoted by the number of crossings for the edges incident to
           two vertices according to the Equation 3.*/
11     end
12 end
13 return $M$
```

It is intuitive to obtain the objective value of each neighboring solution by the evaluation mechanism presented above. Nevertheless, how to update the values maintained in two evaluation matrices is another issue in this procedure. In fact, the matrix of the layer in which the moving vertex is located will remain unchange, while the move may affect that of its adjacent vertices. For example, when we insert vertex v immediately before vertex u, we need to consider the adjacent vertices of the specific vertices, where the specific vertices consist of u, v and all the intermediate vertices $IM(u, v)$. Precisely, Algorithm 3 presents the updating procedure of two evaluation matrices for the neighborhood move $Insert(v, u)$. The set Q records all the intermediate vertices $IM(u, v)$ and vertex u, and $A(Q)$ reserves all the vertices adjacent to vertices in Q. The number of vertices in Q adjacent to vertex j in $A(Q)$ are thus maintained in counter $count(j)$ (lines 1-5). After that, we update the matrix of the opposite layer to the vertex v according to the counter $count(j)$ (lines 6-12).

3.3.3. Solution-based tabu strategy

In our tabu search method, we propose a solution-based strategy to determine the tabu status of neighboring solutions. Specifically, the tabu list is based on one hash vector HV of length λ, where each position represents a binary variable, and the hash vector is associated with a hash function hf. In particular, the hash function maps a candidate solution S of the search space Ω to an index of hash vector as follows:

$$hf : S \in \Omega \rightarrow \{0, 1, 2, \dots, \lambda - 1\} \tag{6}$$

Based on the hash vector and the corresponding hash function, we determine the tabu status of candidate solutions by the following rule. Given a candidate solution S, the hash vector HV and the associated hash function hf, S is identified as a tabu solution if $HV(hf(S)) = 1$. Otherwise, S is determined as a non-tabu solution.

The previous hash functions proposed based on the binary decision variables in the literature (Wang et al. (2017), Lai et al. (2018a), Lai et al. (2018b), and Lai et al. (2018c)) are suitable for the associated binary problems. Unlike the binary optimization problems, the permutation problems constitute a special class that preferably should be treated somewhat differently. Indeed, a good hash function is able to distinguish different solutions and avoid hash conflict with maximum probability and based on it the hash value of the neighboring solution should be easily evaluated. Considering these factors, we propose the following hash function for the DBDP, which can be also applied to other permutation problems. Let S be a candidate solution, the proposed hash function hf can be formally defined as follows:

$$hf(S) = (\sum_{i=1}^{n1}(\omega_1(i) - \omega_1(i-1))^{\kappa} + \sum_{j=1}^{n2}(\omega_2(j) - \omega_2(j-1))^{\kappa}) \ mod \ \lambda \tag{7}$$

where parameter κ is used to define the hash function and λ is the length of hash vector that is set to 10^7 in line with the previous studies.

For the example depicted in Figure 4 (i.e., moving vertex v upwards vertex u_{j-1}), the hash value can be quickly calculated according to the following equation:

$$\begin{aligned} hf(S \oplus Insert(v, u_{j-1})) =& (hf(S) - (v - u_1)^{\kappa} - (u_0 - v)^{\kappa} + (u_0 - u_1)^{\kappa} \\ & + (v - u_{j-1})^{\kappa} + (u_j - v)^{\kappa} - (u_j - u_{j-1})^{\kappa}) \ mod \ \lambda \end{aligned} \tag{8}$$

Thus, the time complexity of determining the tabu status of a neighboring solution is O(1).

The procedure of the solution-based tabu search phase is given in Algorithm 4. The hash vector HV is initialized for only once in the whole ISB-TS algorithm (lines 1-6). After that, the procedure performs a number of iterations to improve the current solution until the consecutive non-improving local optima θ reaches the maximum threshold Θ (lines 9-27). At each iteration, the algorithm replaces the current solution S by a best non-tabu neighboring solution $S^{'}$ chosen from the constrained neighborhood structure $CN(S)$ according to the proposed tabu rule mentioned above (lines 10-12). During the search, the local optimum encountered S^{*} is updated each time a better solution is found (lines 13-18). To reduce the negative effects of hash conflict, ISB-TS allows to accept the tabu solution if it is better than the best found solution S^{best} so far (i.e., so-called tabu aspiration criterion) (lines 19-25). The hash vector is accordingly updated by the new solution (line 26). Finally, the algorithm terminates if the threshold Θ is reached, and then returns the local optimum S^* found during this search process (line 28).

3.4. Adaptive perturbation mechanism

The purpose of the perturbation mechanism is to allow ISB-TS to escape from the current local optima in order to discover other local optima with better solution quality. For this purpose, we apply an adaptive perturbation mechanism which varies the perturbation intensity, depending on the search status.

In addition to the dynamic perturbation strength (described in Section 3.1) to be applied for each perturbation, we also focus on the type of perturbation move (i.e., how to perturb based on the determined perturbation strength). The perturbation move consists of two procedures, the destruction procedure that removes some incremental vertices from the solution S, and the reconstruction procedure that iteratively reinserts them into the incumbent solution. The number of removed vertices is equal to the perturbation strength L and the rule for reinserting them is similar to the initial solution procedure presented in Section 2.

The perturbation operator is presented in Algorithm 5. First, we generate a partial solution S_c by randomly removing a number L of incremental vertices RV from local optimum S (line 1). Then, we iteratively insert each incremental vertex v from RV into the position p of the partial solution S_c until solution S_c becomes complete (lines 2-8). At each iteration, the set CL records all the couples of remaining incremental vertices in RV and its corresponding feasible positions (line 3). The restricted candidate list RCL is created in line 4, which contains the $\max\{1, \beta * |CL|\}$ vertices with the

Algorithm 4: Solution-based tabu search phase $SolutionbasedTabuSearch(S)$

```
Input: Initial solution $S$, hash vector $HV$ of length $\lambda$, hash function $hf$, depth $\Theta$ of tabu search,
        the best-found solution $S^{best}$ so far
Output: The local optima $S^*$ found so far
// Initialize the hash vector for only once in the whole ISB-TS algorithm.
1  if $Initial\_hash\_bool$ is False then
2      for $i \leftarrow 0$ to $L-1$ do
3          $HV[i] \leftarrow 0$;
4      end
5      $Initial\_hash\_bool \leftarrow$ True;
6  end
7  $\theta \leftarrow 0$ ;
8  $S^* \leftarrow S$ ;
   // Main search procedure
9  while The maximum threshold $\Theta$ is not reached, i.e., $\theta \le \Theta$ do
       // Find a best neighborhood solution $S^{'}$ satisfying that this solution is not in
       the tabu list.
10     $S^{'} \leftarrow \arg \min_{s \in CN(S)} \{s : HV(hf(s)) = 0\}$;
11     /* where $CN(S)$ is the combination of two neighborhood operator $N_1(S) \cup N_2(S)$ according
       to the Equations 1 and 2, and $hf$ function can be calculated according to the Equations 7
       and 8.*/;
12     $S \leftarrow S^{'}$ ;
13     if $f(S) < f(S^*)$ then
14         $S^* \leftarrow S$ ;
15         $\theta \leftarrow 0$ ;
16     else
17         $\theta \leftarrow \theta + 1$;
18     end
       // Tabu aspiration criterion: accept the neighboring solution with the better
       objective value than all previously visited solutions even if it is in tabu
       status.
19     $S^a \leftarrow \arg \min_{s \in CN(S)} \{s\}$;
20     /* where $CN(S)$ is the combination of two neighborhood operator $N_1(S) \cup N_2(S)$ according
       to the Equations 1 and 2.*/;
21     if $f(S^a) < f(S^{best})$ then
22         $S^* \leftarrow S^a$ ;
23         $S \leftarrow S^a$ ;
24         $\theta \leftarrow 0$ ;
25     end
26     $HV[hf(S)] \leftarrow 1$;
27 end
28 return $S^*$
```

Algorithm 5: Perturbation operator $Perturb(S, L)$

```
Input: Local optimum $S$, perturbation strength $L$
Output: A perturbed solution $S_c$
// Destruction procedure
1 Generate a partial solution $S_c$ by randomly removing a set of incremental vertices $RV$ from
  solution $S$;
// Reconstruction procedure
2 ‘ while the solution $S_c$ is not a complete solution, i.e., $|RV| = L > 0$ do
3     $CL \leftarrow \{(v, p) : v \in RV, p\ denotes\ each\ feasible\ position\ of\ vertex\ v\}$;
4     $RCL \leftarrow \{(v, p) \in RV : |\{(v', p') : \delta(v', p') \geq \delta(v, p)\}| \leq max\{1, \beta * |CL|\}\}$;
5     Randomly choose a vertex $v_c$ and insert it into the position $p_c$ (where $(v_c, p_c) \in RCL$);
6     $S_c \leftarrow S_c \oplus (v_c, p_c)$;
7     $RV \leftarrow RV \setminus \{v_c\}$;
8 end
9 return $S_c$
```

minimum incremental objective values δ in CL. The value of β can be set as L/L_{max} to balance the weight between greedy and random strategies according to the perturbation strength. When L becomes larger, the procedure tends to be more random, and on the contrary, it tends to be more greedy. We randomly choose a vertex v_c and insert it into the determined position p_c (line 5). The partial solution S_0 and the remaining vertex set RV are updated in lines 6 and 7. When the vertex set RV becomes empty, the perturbation procedure terminates and the complete solution S_c is returned as the result of the perturbation operator (line 9).

In general, the proposed perturbation phase is controlled by the parameter L of the jump magnitude by determining the number of removed vertices in the destruction procedure and the balance ratio between random and greedy strategies in the reconstruction procedure, i.e., the larger the magnitude L, the stronger the perturbation.

4. Computational Results

In this section, we report extensive computational experiments conducted to assess the performance of the proposed iterative solution-based tabu search algorithm (ISB-TS). Moreover, we compare our proposed ISB-TS algorithm with the state-of-the-art reference algorithms on solving public benchmark instances of DBDP.

4.1. Benchmark instances and experimental protocols

For experimental evaluations, we employ two sets of instances in our experimentation in line with the previous studies (Martí and Estruch, 2001; Martí et al., 2018). The first one containing 120 instances was proposed by Martí and Estruch (2001), with the original number of vertices (n_1,n_2) of each layer in the interval [25,50], and the graph density d in the interval [0.065, 0.300]. The instances are incremented 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, 2, γ = 1.2 and 1.6. The second set contains 1000 instances obtained with the generator described in Stallmann et al. (2001). The size of the first layer is in the range [10, 377], while the size of second layer is in the range [10, 190]. The number of edges ranges from 20 to 950. The value of γ is set as 1.1, 1.2 and 1.3. In line with the previous studies on this problem (i.e., (Martí and Estruch, 2001; Martí et al., 2018)), we maintain the setting of parameter γ in the two different instance sets considered.

We coded the ISB-TS algorithm in C++ and ran it on a PC with a 2.60GHz Intel Core i7-700HQ processor and 16GB of RAM with Windows 10 operating system. To evaluate the performance of ISB-TS, we perform comparisons with the following heuristics which are all conducted on a computer with a 2.8 GHz Intel Core i7 processor with 16GB of RAM in the literature:

- The greedy random adaptive search procedure ($GRASP$) method proposed by Martí and Estruch (2001).

- The tabu search and path-relinking algorithm (TS+PR) proposed by Martí et al. (2018).

- The Gurobi based on mathematical programming formulation.

For the purpose of fair comparison, we utilize the method based on the assumption that the CPU speed is approximately linearly proportional to the CPU frequency. According to www.cpubenchmark.net, the CPU speed in Martí et al. (2018) is faster than ours due to the same CPU frequency. We utilize the reported results of the corresponding results presented in Martí et al. (2018). Moreover, we perform 10 independent runs of ISB-TS for each problem instance, with the maximum time-limit per run set to the scaled CPU times used by the current best performing algorithms mentioned above. 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. We share all the 1120 benchmark instances and the executable files of our proposed ISB-TS on the website[1].

4.2. Parameter tuning

Table 2 presents the settings of the SB-ITS parameters used in this study. In line with the previous studies, we adopt a subset of 22 representative instances from the first set as the training instances to configure the best values of key parameters of our method. The parameters of (α, L_{min}, L_{max}, θ, Θ, ξ_{max}, and κ) were tuned with Iterated F-race (IFR) (Birattari et al., 2010), and an automated configure method that is part of the IRACE package (López-Ibáñez et al., 2016). The tuning was performed on all the 22 training

[1]https://github.com/283224262/DBDP

instances. For each parameter, IFR 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 execution of ISB-TS, with the time limit 360 seconds for each instance. The setting of parameters suggested by IFR is reported as *Final value* in Table 2.

Table 2: Settings of the parameters used in ISB-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}	Minimal perturbation strength	(n/6, n/7, n/8)	n/7
L_{max}	Maximal perturbation strength	(n, n/1.5, n/2)	n/1.5
Φ	Distance threshold for the insert move, (F(x) = Max(5, n/x)))	F(8), F(25),F(40)	F(25)
Θ	The maximum threshold of iterations without improving the local optima in solution-based tabu search phase	(7500, 10000,12500)	12500
ξ_{max}	The maximum threshold of iterations without improving the best solution found	(2000,3000,4000)	3000
κ	The parameter in the hash function	(1.2,2,4)	4

4.3. Comparisons with the state-of-the-art algorithms

We compare the proposed ISB-TS with the two best performing algorithms (i.e., GRASP and TS+PR) and a general optimization solver (Gurobi). As presented in Table 3, the first four columns give, for each instance, the numbers of vertices in each layer (n_1 and n_2), the density ($dens.$) and the percentage γ of the quantities in the original graph, respectively. The next column $BestKnown$ shows the best objective value among the three reference algorithms. The following twelve columns show the minimum crossing number $Cross$, the computing time $Time$ in seconds, and average percent deviation $DEV(\%)$ from the best-known solutions produced by all the four compared algorithms (i.e., GRASP, TS+PR, Gurobi, and ISB-TS (1 run)) in one run. Furthermore, in order to test the robustness of our ISB-TS, the average results obtained by our ISB-TS algorithm in ten runs are reported in the next four columns (labeled as ISB-TS (10 runs)), including the best and average objective value f_{best} and f_{avg}, the average running time T_{avg} to reach the best solution in each run and average percent deviation $DEV(\%)$. Note that the best results obtained by our ISB-TS and all the compared algorithms are indicated in bold.

The row $\#Avg$ indicates the average value of each measure. The row $\#Beq$ shows the number of instances for which the associated algorithm obtains the better results or match the best-known results among the compared algorithms. Furthermore, in order to check there exists a significant difference between the results of our ISB-TS algorithm and those obtained by the reference algorithms in terms of f_{best} or f_{avg}, we reported the $p-value$s from the non-parametric Friedman test in the last row of tables where a $p-value$ smaller than 0.05 implies a significant difference between the compared results.

Table 3 shows that our ISB-TS algorithm outperforms significantly the other three reference algorithms. In particular, in only one run, ISB-TS can improve best-known results for 13 out of 22 instances, and it matches the best known results for the remaining 5 ones. Meanwhile, the average computing time of our ISB-TS to reach the best objective values is lower than any other (15.17s vs 308.08s, 41.9s and 1102.4s). Furthermore, the number of instances for which ISB-TS can improve or match the best-known results increases up to 20 instances. ISB-TS is worse than Gurobi in only two instances with the larger number of edge crossings (19846 vs 19831 and 2173 vs 2169). In addition, the non-parametric Friedman tests ($p-values < 0.05$) confirms the significance of these differences, and the least average value of $DEV(\%)$ also shows the robustness of ISB-TS. All these outcomes indicate that ISB-TS has a strong search ability and a high computational efficiency on this part of benchmark instances.

From Table 4, which reports 120 instances of the first instance set, our ISB-TS algorithm dominates the competing algorithms including a very effective variant of TS+PR (TS(500)+PR) in terms of all the indicators. To be specific, our ISB-TS is able to obtain better solutions than the best reference algorithm TS(500)+PR (83264.97 vs. 83888.83) for 104 out of 120 instances according to the experimental results, while TS(500)+PR can only obtain better results in 10 instances. In particular, the average computing time of our ISB-TS method is almost 10 times faster than TS(500)+PR overall the 120 instances of the first instance set. The non-parametric Friedman tests ($p - value <$ 2.2e-16) indicates that there exists a significant difference between the ISB-TS algorithm and the reference algorithm TS(500)+PR in terms of f_{best} values on this set of instances.

Table 5 indicates that for the 1000 instances of the second set, the ISB-TS algorithm achieves the best results among the compared algorithms in most instances. Compared with the state-of-the-art algorithm TS+PR, ISB-TS can obtain better solutions for 687 instances out of 1000 instances, while the TS+PR can only obtain better result for 146 instances. In addition,

Table 3: Results for the performance of ISB-TS in comparison with the reference algorithms (i.e., GRASP, TS+PR, and Gurobi) for the 22 training instances.

n_1	n_2	*dens.*	γ	*Best Known*	GRASP			TS+PR			Gurobi			ISB-TS (1 run)			ISB-TS (10 runs)			
					Cross	*Time*	*DEV*(%)	*Cross*	*Time*	*DEV*(%)	*Cross*	*Time*	*DEV*(%)	*Cross*	*Time*	DEV(%)	f_{best}	f_{avg}	T_{av}	*DEV*(%)
25	25	0.065	1.2	305	316	6.1	3.6	**305**	0.4	0	**305**	0.4	0	**305**	0.01	0	**305**	305	1.22	0
25	25	0.065	1.6	979	1131	34.5	15.5	1106	1.8	13	**979**	237.7	0	1029	0.67	5.11	**979**	998.4	19.14	0
25	25	0.175	1.2	3922	4037	36.9	2.9	3927	0.4	0.1	**3922**	1.4	0	**3922**	0.09	0	**3922**	3922	0.47	0
25	25	0.175	1.6	11982	12374	120	3.3	11982	3.8	0	12444	1800.6	3.9	**11794**	1.38	-1.57	**11794**	11794	17.39	-1.57
25	25	0.3	1.2	15036	15185	60.9	1	15067	0.6	0.2	**15036**	9.6	0	**15036**	0.06	0	**15036**	15036	6.17	0
25	25	0.3	1.6	39657	40538	252.2	2.2	39657	7.7	0	41705	1801.6	5.2	**39465**	0.02	-0.48	**39465**	39465	6.93	-0.48
25	50	0.065	1.2	2173	2218	27.8	2.1	2185	0.7	0.6	**2173**	1.5	0	2175	1.59	0.09	**2173**	2174.8	29.38	0
25	50	0.175	1.2	19794	20112	201.7	1.6	19861	2.1	0.3	**19794**	193.3	0	**19794**	1.98	0	**19794**	19794.1	4.61	0
25	50	0.3	1.2	62986	64113	495.3	1.8	63312	4.8	0.5	62986	1802.3	0	**61813**	2.04	-1.86	**61813**	61813	4.35	-1.86
25	50	0.3	1.6	175764	175764	519	0	176909	52.8	0	184788	1831.6	4.5	**172837**	7.45	-1.67	**172837**	172882.5	4.64	-1.67
50	25	0.065	1.2	2169	2230	42.2	2.8	2191	0.8	1	**2169**	1.6	0	2185	0.62	0.74	2173	2178.6	34.21	0.18
50	25	0.065	1.6	5828	6459	267.3	10.8	5828	12.4	0	6155	1801	5.6	5554	9.36	-4.7	**5552**	5552	69.22	-4.74
50	25	0.175	1.2	19831	20265	228.9	2.2	19890	2	0.3	**19831**	336.2	0	19850	4.33	0.1	19846	19865.2	30.11	0.08
50	25	0.175	1.6	54004	57004	517.7	5.6	54004	31.3	0	63287	1802.4	17.2	**53810**	61.53	-0.36	**53810**	53813.2	45.43	-0.36
50	25	0.3	1.2	65593	66253	502.8	1	65593	4.4	0	66319	1802	1.1	**65557**	1.09	-0.05	**65557**	65571.4	39.71	-0.05
50	25	0.3	1.6	179282	186429	538.6	4	179282	58.6	0	189119	1805.3	5.5	**176557**	60.22	-1.52	**176557**	176561	32.61	-1.52
50	50	0.065	1.2	7637	7859	297.3	2.9	7664	3.9	0.4	**7637**	3.5	0	**7637**	1.34	0	**7637**	7637.6	16.09	0
50	50	0.065	1.6	24933	25545	506.5	2.5	24933	67	0	27110	1802.8	8.7	24133	24.80	-3.21	**24103**	24126.2	15.92	-3.33
50	50	0.175	1.2	77253	78717	505.5	1.9	77253	14.3	0	77480	1802.7	0.3	**77205**	2.63	-0.06	**77205**	77205	0.2	-0.06
50	50	0.175	1.6	233326	238979	504.1	2.4	233326	209.3	0	256917	1808.6	10.1	230791	67.86	-1.09	**230789**	230789	66.81	-1.09
50	50	0.3	1.2	248454	251277	536.8	1.1	248454	26.1	0	258330	1806.3	4	**248172**	4.92	-0.11	**248172**	248172	6.69	-0.11
50	50	0.3	1.6	712459	728794	575.7	2.3	712459	416.7	0	757750	1800.3	6.4	709599	79.72	-0.4	**709598**	709598.1	201.21	-0.4
#Avg				89243.95	91163.59	308.08	3.34	89326.72	41.90	0.75	94374.36	1102.40	3.30	88600.91	15.17	-0.50	88596.23	88602.46	29.66	-0.77
#Beq					1			12			10			18			20	16		
$p-value$					2.73e-06			4.59e-06			2.91e-2						4.68e-3			

ISB-TS can obtain better results than TS+PR in terms of f_{best} (36448.09 vs. 36928.31) and T_{avg} (49.05s vs. 83.67s). The non-parametric Friedman tests ($p-value < 2.2\text{e-}16$) demonstrates that the ISB-TS algorithm and the reference algorithm TS+PR is significantly different in terms of f_{best} on the second instance set.

In summary, the experimental results reported above show clearly that the proposed ISB-TS algorithm is very competitive compared to the state-of-the-art algorithms in the literature both in terms of solution quality and computational efficiency.

Table 4: Results for the performances of ISB-TS in comparison with the reference algorithms (i.e., GRASP, TS(500)+PR, TS+PR, and Gurobi) in the first instance set with $\gamma = 1.2$ and 1.6.

Algorithm	$\gamma = 1.2$			$\gamma = 1.6$		
	Cross	*Time*	*DEV*(%)	*Cross*	*Time*	*DEV*(%)
Small size (n_1+n_2 = 50)						
Gurobi	6230.00	27.18	0.00	18553.53	1224.33	4.24
GRASP	6317.47	33.23	2.17	18056.27	141.83	7.44
TS(500) + PR	6232.20	48.76	0.10	17459.13	1039.53	1.01
TS + PR	6234.33	8.15	0.16	17489.07	90.12	1.71
ISB-TS (1 run)	**6230.00**	**1.21**	**0.00**	**17397.07**	**71.95**	**0.96**
Medium size (n_1+n_2 = 75)						
Gurobi	28522.20	781.71	0.24	84088.57	1808.00	7.50
GRASP	28776.57	228.98	2.52	80917.97	409.31	5.15
TS(500) + PR	28379.10	621.53	0.16	78959.23	1718.27	0.27
TS + PR	28390.17	61.11	0.21	79161.07	386.90	0.61
ISB-TS (1 run)	**28365.33**	**42.57**	**0.08**	**77817.4**	**160.66**	**0.23**
Large size (n_1+n_2 = 100)						
Gurobi	112572.27	1207.01	1.10	343478.67	1891.89	8.70
GRASP	111381.07	423.60	1.94	328878.53	532.43	3.86
TS(500) + PR	110214.33	1516.66	0.08	322528.33	1687.68	0.00
TS + PR	110233.07	292.30	0.12	322831.60	514.51	0.23
ISB-TS (1 run)	**110172.60**	**103.84**	**-0.04**	**319490**	**211.41**	**-0.94**
#Total			#Better			
TS(500) + PR	83888.83	1121.52	10 (120)			
ISB-TS (1 run)	**83264.97**	**137.24**	**104 (120)**			

Table 5: Results for the performances of ISB-TS in comparison with the reference algorithms (i.e., GRASP, TS+PR, and Gurobi) in the second set of instances.

Algorithm	*Cross*	*Time*	*DEV*(%)
Small size ($21 \leq n_1 + n_2 \leq 57$)			
GRASP	510.82	0.40	4.09
TS + PR	497.77	**0.09**	2.03
ISB-TS (1 run)	**489.93**	0.02	**1.94**
Medium size ($111 \leq n_1 + n_2 \leq 122$)			
GRASP	22742.99	18.83	4.69
TS + PR	22394.12	7.80	0.06
ISB-TS (1 run)	**22228.50**	**3.72**	**-0.74**
Large size ($231 \leq n_1 + n_2 \leq 471$)			
GRASP	87844.49	319.53	4.97
TS + PR	85816.74	232.27	0.01
ISB-TS (1 run)	**84597.47**	**113.68**	**-1.42**
#Total			#Better
TS+ PR	36928.31	83.67	146 (1000)
ISB-TS (1 run)	**36448.09**	**40.91**	**687 (1000)**

5. Analysis and Discussions

In this section we pay attention to the analysis of several key elements of the proposed algorithm (i.e., the constrained neighborhood structure, solution-based tabu strategy and the diversified perturbation strength technique) used in our proposed ISB-TS algorithm.

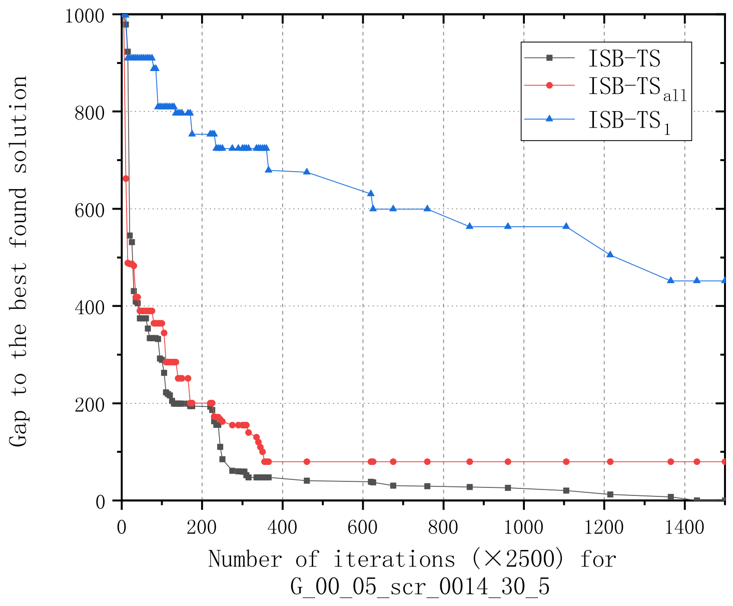
5.1. Impact of the constrained neighborhood structure

ISB-TS adopts a constrained neighborhood structure to achieve the quality and efficiency of the search by constricting the distance of moving vertex with Φ. To evaluate the impact of constrained neighborhood structure, we conduct an experiment to compare the performance of ISB-TS with its alternative versions, ISB-TS$_{all}$ which evaluates all the neighboring solutions without this distance constriction mechanism, and ISB-TS$_1$ which only evaluates two neighboring solutions each time by considering moving the current vertex to the previous position or the posterior position, similar to the method proposed in Martí et al. (2018). We obtain ISB-TS$_{all}$ and ISB-TS$_1$ by setting the parameter Φ with the value of $\max(n_1,n_2)$ and 1, respectively, and select two representative instances, i.e., $G\_00\_05\_scr\_0014\_30\_5$ and $G\_21\_06\_scr\_0002\_30$ with the number of vertices 471 and 242, to test their performance. For each test variant and each instance considered, the corresponding algorithm is independently performed 10 times with a maximum number of 1500 ($\times$2500) iterations, and the gap of the best found solution (f^*) in each variant to the current best known result (f^{best}), i.e., $f^* - f^{best}$, is recorded as a function of the number of iterations. The evolution of the gap value is respectively plotted in Figure 5 for each instance and each variant.

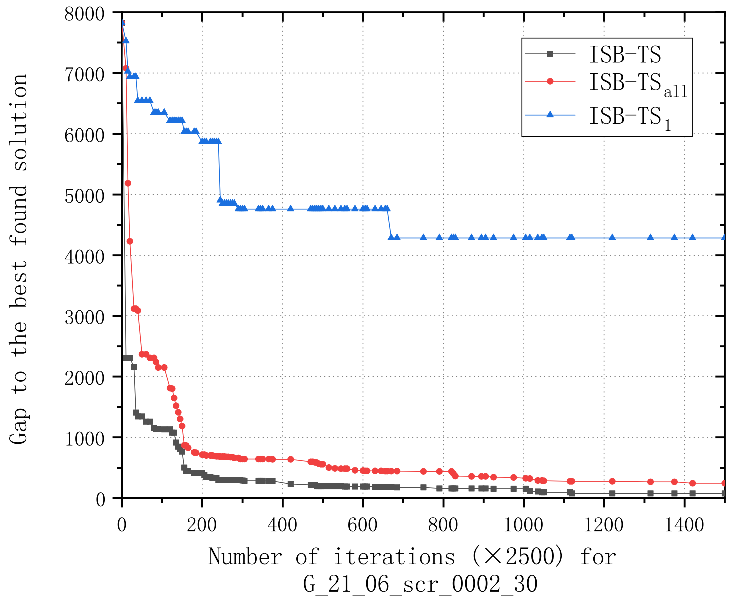
One observes from Figure 5 that the performance of the compared algorithms are significantly influenced by the different size of neighborhood structure. The normal version of ISB-ITS employed a constrained neighborhood structure performs the best, although ISB-ITS$_{all}$ exploits a close performance with it. ISB-TS$_1$ clearly performs the worst, thus it is important to avoid making the distance threshold Φ too small during the search since this will restrict the search region of the algorithm too much, causing the search to miss high-quality solutions. In general, the constrained neighborhood structure employed in this study will lead to a good performance in terms of both the computing speed and solution quality.

5.2. Impact of solution-based tabu strategy

The solution-based tabu strategy is a crucial ingredient of our ISB-TS algorithm. To show its importance with respect to the popular attribute-based tabu strategy, we produce a variant IAB-TS of the ISB-TS method by replacing the solution-based tabu strategy with the attribute-based tabu strategy typically employed in tabu search implementations, while keeping other ISB-TS components unchanged according to the experimental protocol



(a)



(b)

Figure 5: Comparative results of ISB-TS with ISB-TS$_{all}$ and ISB-TS$_1$. The vertical axis represents the gap to the best found solution and the horizontal axis represents the number of iterations.

Table 6: Comparison between the ISB-TS and its variant IAB-TS with attribution-based tabu strategy on the 30 representative large instances.

Instance	f_{best}		f_{avg}		T_{avg}	
	IAB-TS	ISB-TS	IAB-TS	ISB-TS	IAB-TS	ISB-TS
$G\_00\_04\_scr\_0001\_10$	10854	**10851**	10856.1	**10851**	33.20	27.73
$G\_00\_04\_scr\_0001\_20$	9534	**9515**	9548.9	**9536.5**	64.51	67.69
$G\_00\_04\_scr\_0001\_30$	7922	**7909**	7958.9	**7931.4**	114.48	126.66
$G\_00\_04\_scr\_0002\_10$	10405	**10400**	10415.5	**10413.7**	24.38	45.97
$G\_00\_04\_scr\_0002\_20$	8451	**8433**	8461	**8454.4**	152.67	110.98
$G\_00\_04\_scr\_0002\_30$	**7345**	7352	**7365.5**	7374.5	96.71	103.25
$G\_00\_04\_scr\_0003\_10$	11246	**11242**	11254.1	**11249.2**	118.97	125.22
$G\_00\_04\_scr\_0003\_20$	9307	**9303**	9336.1	**9326.2**	82.42	176.38
$G\_00\_04\_scr\_0003\_30$	8122	**8104**	8169.9	**8133.3**	119.31	156.90
$G\_00\_04\_scr\_0004\_10$	10270	**10264**	10282.9	**10275.3**	148.87	156.53
$G\_00\_05\_scr\_0012\_30$	**31923**	31992	**32047.1**	32052.4	149.01	150.76
$G\_00\_05\_scr\_0013\_10$	47779	**47748**	47813.1	**47799.5**	91.92	134.02
$G\_00\_05\_scr\_0013\_20$	**40760**	40777	**40851.7**	40867.5	25.05	152.06
$G\_00\_05\_scr\_0013\_30$	33786	**33666**	33885.3	**33764.9**	46.71	71.74
$G\_00\_05\_scr\_0014\_10$	**46677**	46689	46735.8	**46717.7**	4.73	3.94
$G\_00\_05\_scr\_0014\_20$	38178	**38167**	**38197.9**	38228.1	161.21	147.89
$G\_00\_05\_scr\_0014\_30$	32241	**32096**	32371.2	**32260.4**	214.17	227.81
$G\_00\_05\_scr\_0015\_10$	**44895**	44918	**44961.5**	44976.2	90.03	106.81
$G\_00\_05\_scr\_0015\_20$	37560	**37537**	37614.3	**37610.4**	89.52	154.42
$G\_00\_05\_scr\_0015\_30$	32368	**32290**	32449.5	**32365.8**	205.28	131.27
$G\_21\_06\_scr\_0001\_10$	**192094**	**192094**	**192094**	**192094**	46.03	6.85
$G\_21\_06\_scr\_0001\_20$	173515	**173377**	173574	**173480**	176.53	30.69
$G\_21\_06\_scr\_0001\_30$	133086	**132968**	133153	**133092.6**	223.60	232.39
$G\_21\_06\_scr\_0002\_10$	**195546**	195552	**195549.6**	195552	115.97	68.58
$G\_21\_06\_scr\_0002\_20$	178140	**178042**	178190.8	**178119.6**	139.98	163.28
$G\_21\_06\_scr\_0002\_30$	164395	**164379**	164608.2	**164498.2**	133.88	180.04
$G\_21\_06\_scr\_0003\_10$	192955	**192947**	192957.8	**192947.8**	104.91	125.25
$G\_21\_06\_scr\_0003\_20$	**170026**	**170026**	**170055**	170086.6	182.39	114.21
$G\_21\_06\_scr\_0003\_30$	159157	**159011**	159457.3	**159286.7**	204.60	51.98
$G\_21\_06\_scr\_0004\_10$	**194128**	194186	**194137.6**	194187.8	194.27	188.00
$\#Best$	9	**23**	9	**22**	**19**	11
$\#Avg$	74422.17	**74394.5**	74478.45	**74451.12**	118.51	**117.98**
$p-value$	8.15e-3		1.58e-2			

given in Table 2. Specifically, when we select a new vertex v and move it, vertex v is maintained in tabu list for the number of the current iteration, in order to prohibit to move it in the next tt iterations (tt is the tabu tenure). We set $tt = 5$ in line with the previous setting in Martí et al. (2018). Finally, we complement the aspiration criterion that the tabu status of a move is disabled if the generated solution leads to a solution better than all previously visited solutions.

To compare the performance between IAB-TS and ISB-TS, we carry out an experiment based on a set of 30 large instances with the number of vertices in the range of 231, 242, 471, where both methods were run 10 times, and terminated within 360 seconds for each time. The experimental results are summarized in Table 6 where we present for each algorithm the best results f_{best} obtained over 10 runs, the average results f_{avg}, and the average computing time T_{av}. The row $\#Avg$ indicates the average value of each measure. The row $\#Best$ shows the number of instances for which the associated algorithm obtains the best results in terms of f_{best} or f_{avg} among the compared algorithms. The $p - value$s from the non-parametric Friedman test in the last row of tables where a $p - value$ smaller than 0.05 implies a significant difference between the compared results.

We can observe from Table 6, the ISB-TS can obtain better results in comparison with IAB-TS in terms of f_{best} and f_{avg} for 23 and 22 instances out of 30 instances, respectively. In addition, the average values of $\#avg$ of ISB-TS are superior to the results of IAB-TS in terms of f_{best} (74394.5 vs. 74422.17), f_{avg} (74451.12 vs. 74478.45) and T_{avg} (117.98s vs. 118.51s). The $p - value$s from the non-parametric Friedman test in the last row of tables where both $p - value$s (8.15e-3 and 1.58e-2) smaller than 0.05 implies the ISB-TS is significantly different from IAB-TS in terms of f_{best} and f_{avg}. This experiment thus concludes that the proposed solution-based tabu strategy is more suitable than the attribute-based tabu strategy to combine its iterative perturbation mechanism for solving the DBDP.

5.3. Impact of the adaptive strength technique

Table 7: Comparison between the ISB-TS and its variant ISB-TS$_{sp}$ without adaptive strength technique on the 30 representative large instances.

Instance	f_{best}		f_{avg}		T_{avg}	
	ISB-TS$_{sp}$	ISB-TS	ISB-TS$_{sp}$	ISB-TS	ISB-TS$_{sp}$	ISB-TS
*G*_00_04_*scr*_0001_10	10854	**10851**	10877.5	**10851**	125.76	27.73
*G*_00_04_*scr*_0001_20	9555	**9515**	9594.3	**9536.5**	192.54	67.69
*G*_00_04_*scr*_0001_30	7956	**7909**	7989.2	**7931.4**	121.48	126.66
*G*_00_04_*scr*_0002_10	10423	**10400**	10441.5	**10413.7**	127.34	45.97
*G*_00_04_*scr*_0002_20	8463	**8433**	8496.5	**8454.4**	107.26	110.98
*G*_00_04_*scr*_0002_30	7373	**7352**	7437.4	**7374.5**	81.99	103.25
*G*_00_04_*scr*_0003_10	11257	**11242**	11285.5	**11249.2**	147.74	125.22
*G*_00_04_*scr*_0003_20	9310	**9303**	9346.7	**9326.2**	126.54	176.38
*G*_00_04_*scr*_0003_30	8168	**8104**	8200.2	**8133.3**	87.90	156.90
*G*_00_04_*scr*_0004_10	10291	**10264**	10301.6	**10275.3**	167.25	156.53
*G*_00_05_*scr*_0012_30	32007	**31992**	32088.2	**32052.4**	196.86	150.76
*G*_00_05_*scr*_0013_10	47769	**47748**	47812.3	**47799.5**	278.09	134.02
*G*_00_05_*scr*_0013_20	40790	**40777**	**40851.2**	40867.5	174.99	152.06
*G*_00_05_*scr*_0013_30	**33618**	33666	**33697.1**	33764.9	195.54	71.74
*G*_00_05_*scr*_0014_10	**46679**	46689	46718.3	**46717.7**	176.02	3.94
*G*_00_05_*scr*_0014_20	**38147**	38167	**38207**	38228.1	162.74	147.89
*G*_00_05_*scr*_0014_30	**32062**	32096	**32158.6**	32260.4	250.27	227.81
*G*_00_05_*scr*_0015_10	44947	**44918**	45014	**44976.2**	194.34	106.81
*G*_00_05_*scr*_0015_20	**37490**	37537	**37552.8**	37610.4	220.39	154.42
*G*_00_05_*scr*_0015_30	**32083**	32290	**32280.4**	32365.8	194.14	131.27
*G*_21_06_*scr*_0001_10	192095	**192094**	192116.3	**192094**	36.68	6.85
*G*_21_06_*scr*_0001_20	173423	**173377**	**173465.8**	173480.0	184.69	30.69
*G*_21_06_*scr*_0001_30	133000	**132968**	133115.1	**133092.6**	171.06	232.39
*G*_21_06_*scr*_0002_10	195575	**195552**	195591.7	**195552**	100.18	68.58
*G*_21_06_*scr*_0002_20	178098	**178042**	178237.2	**178119.6**	29.28	163.28
*G*_21_06_*scr*_0002_30	164520	**164379**	164611.6	**164498.2**	147.72	180.04
*G*_21_06_*scr*_0003_10	192952	**192947**	192980.1	**192947.8**	47.91	125.25
*G*_21_06_*scr*_0003_20	170069	**170026**	170115.6	**170086.6**	128.01	114.21
*G*_21_06_*scr*_0003_30	**158944**	159011	159412.7	**159286.7**	215.39	51.98
*G*_21_06_*scr*_0004_10	194195	**194186**	194244.5	**194187.8**	78.54	188.00
#Best	7	**23**	7	**23**	9	**21**
#Avg	74403.8	**74394.5**	74474.70	**74451.12**	148.95	**117.98**
$p - value$	3.48e-3		3.49e-3			

To highlight the importance of the adaptive strength strategy in adaptive perturbation phase, we compare our algorithm with a variant of ISB-TS (denoted as ISB-TS$_{sp}$) obtained by removing the adaptive strength mechanism and keeping the other ingredients unchanged. Compared with the adaptive strength strategy employed in ISB-TS, which relies on the historical search information (such as, the previous local optimum S_p, the counter ξ of consecutive non-improving best found solution), the variant ISB-TS$_{sp}$ adopts a fixed perturbation strength L_{sp}. The value of L_{sp} is empirically set as $(n_1 + n_2)/2$.

To compare ISB-TS and ISB-TS$_{sp}$, we carried out an experiment based on a set of 30 large instances with the number of vertices in the range of 241, 242, 471. One observes from Table 7 that the ISB-TS performs better than the compared variant ISB-TS$_{sp}$ in terms of f_{best}, f_{avg}, and T_{avg}. To be specific, ISB-TS is able to find better results in terms of f_{best} and f_{avg} than ISB-TS$_{sp}$ within two indicators $\#Best$ (23 vs. 7, 23 vs. 7), and $\#Avg$ (74394.5 vs. 74403.8, 74451.12 vs. 74474.70) within a faster average computing time (117.98s vs. 148.95s). Furthermore, the $p - value$s (3.48e-3 and 3.49e-3) smaller than 0.05 indicate that both two algorithms are significantly different from each other in terms of f_{best} and f_{avg}. This experiment thus confirms the effectiveness of the adaptive strength mechanism in perturbation phase.

5.4. Effectiveness of the iterative mechanism

To study the impact of the iterative mechanism, we created a variant of the ISB-TS algorithm (denoted as SB-TS), where we disable the perturbation procedure, while keeping other components unchanged. We compare SB-TS and ISB-TS based on the set of 30 large instances with the number of vertices including 241, 242, and 471. We ran both SB-TS and ISB-TS 10 times to solve each instance. To perform a fair comparison, we run SB-TS with the same computing time as that of ISB-TS presented in Table 7.

Table 8: Comparison between the ISB-TS and its variant SB-TS without iterative mechanism on the 30 representative large instances.

Instance	*CT(s)*	f_{best}		f_{avg}	
		SB-TS	ISB-TS	SB-TS	ISB-TS
G_00_04_scr_0001_10	27.73	10886	**10851**	11130.7	**10851**
G_00_04_scr_0001_20	196.23	9609	**9515**	10018.4	**9536.5**
G_00_04_scr_0001_30	126.66	8070	**7909**	8204.3	**7931.4**
G_00_04_scr_0002_10	222.85	10436	**10400**	10492.5	**10413.7**
G_00_04_scr_0002_20	110.98	8505	**8433**	8592	**8454.4**
G_00_04_scr_0002_30	103.25	7510	**7352**	7833.4	**7374.5**
G_00_04_scr_0003_10	125.22	11309	**11242**	11345.2	**11249.2**
G_00_04_scr_0003_20	188.91	9469	**9303**	9661	**9326.2**
G_00_04_scr_0003_30	156.90	8323	**8104**	8491.4	**8133.3**
G_00_04_scr_0004_10	156.53	10306	**10264**	10383.9	**10275.3**
G_00_05_scr_0012_30	150.76	32120	**31992**	32549.9	**32052.4**
G_00_05_scr_0013_10	134.02	47870	**47748**	47954.5	**47799.5**
G_00_05_scr_0013_20	297.29	40793	**40777**	41202.5	**40867.5**
G_00_05_scr_0013_30	71.74	33702	**33666**	34256.5	**33764.9**
G_00_05_scr_0014_10	3.94	46697	**46689**	47144.5	**46717.7**
G_00_05_scr_0014_20	147.89	38307	**38167**	38494.7	**38228.1**
G_00_05_scr_0014_30	227.81	32295	**32096**	32893.2	**32260.4**
G_00_05_scr_0015_10	106.81	45031	**44918**	45152.9	**44976.2**
G_00_05_scr_0015_20	154.42	37561	**37537**	38030.1	**37610.4**
G_00_05_scr_0015_30	131.27	32400	**32290**	32449.5	**32365.8**
G_21_06_scr_0001_10	6.85	192134	**192094**	192180.5	**192094**
G_21_06_scr_0001_20	30.69	173849	**173377**	173860.8	**173480**
G_21_06_scr_0001_30	232.39	133368	**132968**	133891.5	**133092.6**
G_21_06_scr_0002_10	68.58	195596	**195552**	196022.7	**195552**
G_21_06_scr_0002_20	163.28	178098	**178042**	178120.8	**178119.6**
G_21_06_scr_0002_30	280.04	164531	**164379**	164924	**164451.2**
G_21_06_scr_0003_10	125.25	193458	**192947**	193684.1	**192947.8**
G_21_06_scr_0003_20	114.21	171088	**170026**	171181.8	**170086.6**
G_21_06_scr_0003_30	51.98	159846	**159011**	159884.2	**159286.7**
G_21_06_scr_0004_10	188.00	194456	**194186**	194488.6	**194187.8**
$\#Best$		0	**30**	0	**30**
$\#Avg$		74587.43	**74394.5**	74817.33	**74449.55**
$p-value$		4.32e-08		4.32e-08	

The experimental results are summarized in Table 8. It clearly shows that the ISB-TS algorithm performs consistently much better than SB-TS over all performance indicators considered within the same running time (reported in the second column CT(s)), and on all the tested instances, as confirmed by the small p-values (4.32e-08). This outcome shows that the iterative mechanism plays a very positive role in the performance of the ISB-TS algorithm.

5.5. Experimental results for the decremental bipartite drawing problem

In this section, we employed the proposed ISB-TS to tackle the decremental bipartite drawing problem, which considers both adding the incremental vertices and deleting some of the original vertices from the graph. For this purpose, we conducted the following experiment on 30 new decremental graph drawing instances. These instances are constructed by removing 10 percent of the vertices from the representative 30 large DBDP instances, as well as the corresponding edges incident to these vertices. We then compare the proposed ISB-TS with the best-performing algorithm TS+PR for these instances. We ran both ISB-TS and TS+PR 10 times to solve each instance. The experimental results are summarized in Table 9. For a fair comparison, we limit the running time of the two reference algorithms, as shown in the column $CT(s)$ of Table 9.

As presented in Table 9, the ISB-TS performs better than the compared best-performing algorithm TS+PR in terms of the indicators f_{best}, f_{avg}. To be specific, ISB-TS is able to find better results in terms of two indicators $\#Best$ (28 vs. 2, 30 vs. 0), and $\#Avg$ (42751.56 vs. 40600.5, 43762.02 vs. 40921.55). Furthermore, the $p-value$s (2.06e-06 and 3.18e-07) of the two statistical tests considered are smaller than 0.05, which indicate that both two algorithms are significantly different from each other in terms of both indicators f_{best} and f_{avg}. This experiment thus confirms that our pro-

Table 9: Comparison between the ISB-TS and the best-performing algorithm TS+PR on the 30 new decremental graph drawing instances.

Instance	*CT(s)*	f_{best}		f_{avg}	
		TS+PR	ISB-TS	TS+PR	ISB-TS
Cut_G_00_04_scr_0001_10	27.73	7766	**7717**	7816.5	**7725.1**
Cut_G_00_04_scr_0001_20	196.23	6189	**6106**	6334.1	**6176.8**
Cut_G_00_04_scr_0001_30	126.66	5525	**5358**	5594.7	**5375.6**
Cut_G_00_04_scr_0002_10	222.85	6877	**6831**	6918.8	**6838**
Cut_G_00_04_scr_0002_20	110.98	5922	**5829**	6013.8	**5840**
Cut_G_00_04_scr_0002_30	103.25	4962	**4875**	5117	**4907.9**
Cut_G_00_04_scr_0003_10	125.22	7899	**7869**	7936.7	**7872.4**
Cut_G_00_04_scr_0003_20	188.91	6314	**6228**	6452.5	**6245.4**
Cut_G_00_04_scr_0003_30	156.90	5293	**5225**	5430.6	**5271.4**
Cut_G_00_04_scr_0004_10	156.53	7220	**7177**	7268.2	**7181.3**
Cut_G_00_05_scr_0012_30	150.76	**19322**	19400	19564.2	**19466.1**
Cut_G_00_05_scr_0013_10	134.02	33463	**33312**	33500.5	**33345.3**
Cut_G_00_05_scr_0013_20	297.29	27240	**27195**	27812.8	**27263.8**
Cut_G_00_05_scr_0013_30	71.74	22286	**22280**	22626.5	**22367.1**
Cut_G_00_05_scr_0014_10	3.94	31002	**30974**	31170.3	**31012.8**
Cut_G_00_05_scr_0014_20	147.89	24580	**24537**	24923.2	**24611.2**
Cut_G_00_05_scr_0014_30	227.81	**20446**	20705	20600.5	**20835.9**
Cut_G_00_05_scr_0015_10	106.81	30139	**30059**	30228.3	**30100.3**
Cut_G_00_05_scr_0015_20	154.42	24823	**24746**	25130.4	**24825.4**
Cut_G_00_05_scr_0015_30	131.27	18793	**18634**	19141.6	**18749.7**
Cut_G_21_06_scr_0001_10	6.85	138113	**137306**	139750.6	**138000.7**
Cut_G_21_06_scr_0001_20	30.69	80882	**76335**	82092.6	**77006.6**
Cut_G_21_06_scr_0001_30	232.39	33178	**21673**	35643.2	**23295.9**
Cut_G_21_06_scr_0002_10	68.58	130218	**124961**	132355	**126067.7**
Cut_G_21_06_scr_0002_20	163.28	102674	**101564**	104578.5	**102078.5**
Cut_G_21_06_scr_0002_30	280.04	59758	**51735**	61740.8	**52022.9**
Cut_G_21_06_scr_0003_10	125.25	120764	**118952**	124422.3	**119254.4**
Cut_G_21_06_scr_0003_20	114.21	109468	**101693**	113464.9	**102963.6**
Cut_G_21_06_scr_0003_30	51.98	70011	**53221**	75523.2	**54809.5**
Cut_G_21_06_scr_0004_10	188.00	121420	**115518**	123708.3	**116135.2**
#Best		2	**28**	0	**30**
#Avg		42751.56	**40600.5**	43762.02	**40921.55**
$p-value$		2.06e-06		3.18e-07	

posed ISB-TS is highly competitive in comparison with the state-of-the-art algorithm TS+PR for the decremental bipartite drawing problem.

6. Conclusion

We present an iterative solution-based tabu search algorithm (ISB-TS) for solving the dynamic bipartite drawing problem (DBDP). The novel and key features of our algorithm include an efficient solution-based tabu search that uses constrained neighborhood structure, a fast evaluation strategy for solution improvement, and an adaptive perturbation phase with the diversified strength strategy to encourage the search to explore new regions in the search space.

Experimental evaluations on extensive benchmarks show that our ISB-TS competes very favourably with the current state-of-the-art algorithms. In addition, we carry out experimental analysis to reveal the effectiveness of the new features incorporated in the ISB-TS algorithm.

The main advantages of the proposed solution-based tabu search can be summarized as follows: First, the solution-based tabu search relying on the explicit memory mechanism (hash function and hash vector) can record the whole information of a solution instead of one move (or its typical attribute) as proposed in the standard attribute tabu search, thus reducing the probability of revisiting the same solution and inducing a stronger intensification pattern. Second, the solution-based tabu search has a simpler implementation structure, since it does not contain the tabu tenure and tabu aspiration, which are important components in standard attribute tabu search. On the other hand, the limitation of the proposed solution-based tabu search is that it requires larger storage space than the attribute-based tabu search, to keep the whole information of the visited solution permanently without the tabu tenure strategy.

An important conclusion of this study is that explicit memory (solution-based) can be more efficient than attributive memory. This is very relevant since most tabu search methods are based on attributive memory. Researchers usually prefer them since they are easily implemented, maintained, and operated, thus resulting in faster methods. However, our study reveals that solution-based memory structures implemented with hash functions can also be very fast and more efficient than the traditional attributive structures in terms of search intensification.

Acknowledgment

This research was supported by the Fundamental Research Funds under grant numbers JBK1901011 and JBK190504 for the Central Universities of China and National Natural Science Foundation of China under grant numbers 71320107001, and Spanish Ministerio de Ciencia, Innovación y Universidades with code PGC2018-095322-B-C21 (MCIU/AEI/FEDER, UE).

References

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

Benlic, U., Hao, J. K., 2013. Breakout local search for the quadratic assignment problem. Applied Mathematics & Computation 219 (9), 4800–4815.

Birattari, M., Yuan, Z., Balaprakash, P., Stützle, T., 2010. F-race and iterated f-race: An overview. In: Experimental methods for the analysis of optimization algorithms. Springer, pp. 311–336.

Burch, M., Müller, C., Reina, G., Schmauder, H., Greis, M., Weiskopf, D., 2012. Visualizing dynamic call graphs. In: VMV. pp. 207–214.

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

Eades, P., Kelly, D., 01 1986. Heuristics for drawing 2-layered networks. Ars Combinatoria - ARSCOM 21.

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.

Eades, P., Wormald, N. C., 1994. Edge crossings in drawings of bipartite graphs. Algorithmica 11 (4), 379–403.

Erten, C., Harding, P. J., Kobourov, S. G., Wampler, K., Yee, G., 2003. Graphael: Graph animations with evolving layouts. In: Graph Drawing, International Symposium, Gd 2003, Perugia, Italy, September 21-24, 2003, Revised Papers. pp. 98–110.

Festa, P., Resende, M. G. C., 2010. An annotated bibliography of graspc-part ii: Applications. International Transactions in Operational Research 16 (2), 131–172.

Fink, A., Voß, S., 2003. Solving the continuous flow-shop scheduling problem by metaheuristics. European Journal of Operational Research 151 (2), 400–414.

Fu, Z. H., Hao, J. K., 2014. Breakout local search for the steiner tree problem with revenue, budget and hop constraints. European Journal of Operational Research 232 (1), 209–220.

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

Glover, F., Hao, J. K., 2017. Diversification-based learning in computing and optimization. Journal of Heuristics, 1–17.

Glover, F., Laguna, M., 1998. Tabu search. In: Handbook of combinatorial optimization. Springer, pp. 2093–2229.

Glover, F., Laguna, M., Martí, R., 2000. Fundamentals of scatter search and path relinking. Control and Cybernetics 29 (3), 653–684.

Herman, I., Melançon, G., Marshall, M. S., 2000. Graph visualization and navigation in information visualization: A survey. IEEE Transactions on visualization and computer graphics 6 (1), 24–43.

Johnson, D. S., 1982. The NP-completeness column: An ongoing guide. Journal of Algorithms 9 (3), 426–444.

Jünger, M., Mutzel, P., 1997. 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms. Journal of Graph Algorithms & Applications 1 (1), 1–25.

Kulturel-Konak, S., 2012. A linear programming embedded probabilistic tabu search for the unequal-area facility layout problem with flexible bays. European Journal of Operational Research 223 (3), 614–625.

Lai, X., Hao, J. K., Dong, Y., 2018a. Two-stage solution-based tabu search for the multidemand multidimensional knapsack problem. European Journal of Operational Research 274, 35–48.

Lai, X., Hao, J. K., Glover, F., Lü, Z., 2018b. 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., 2018c. Solution-based tabu search for the maximum min-sum dispersion problem. Information Sciences 441, 79–94.

Li, X., Yue, C., Aneja, Y. P., Chen, S., Cui, Y., 2018. An iterated tabu search metaheuristic for the regenerator location problem. Applied Soft Computing 70, 182–194.

Liao, L.-M., Huang, C.-J., 2011. Tabu search heuristic for two-machine flowshop with batch processing machines. Computers & Industrial Engineering 60 (3), 426–432.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., Stützle, T., 2016. The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives 3, 43–58.

Lourenco, H. R., Martin, O. C., Stützle, T., 2010. Iterated local search: Framework and applications. Handbook of Metaheuristics 146, 363–397.

Mäkinen, E., 1989. A note on the median heuristic for drawing bipartite graphs. Fundamenta Informaticae XII (4), 563–569.

Martí, R., 1998. A tabu search algorithm for the bipartite drawing problem. European Journal of operational research 106 (2-3), 558–569.

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

Martí, R., Laguna, M., 2003. Heuristics and meta-heuristics for 2-layer straight line crossing minimization. Discrete Applied Mathematics 127 (3), 665–678.

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.

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.

Peng, B., Lü, Z., Cheng, T. C. E., 2015. A tabu search/path relinking algorithm to solve the job shop scheduling problem. Computers & Operations Research 53 (53), 154–164.

Pinaud, B., Kuntz, P., Lehn, R., 2004. Dynamic graph drawing with a hybridized genetic algorithm. Adaptive Computing in Design & Manufacture VI, 365–375.

Silvestrin, P. V., Ritt, M., 2017. An iterated tabu search for the multi-compartment vehicle routing problem. Computers & Operations Research 81, 192–202.

Stallmann, M., Brglez, F., Ghosh, D., 2001. Heuristics, experimental subjects, and treatment evaluation in bigraph crossing minimization. Journal of Experimental Algorithmics 6, 8.

Sugiyama, K., Tagawa, S., Toda, M., 1981. Methods for visual understanding of hierarchical system structures. IEEE Transactions on Systems Man & Cybernetics 11 (2), 109–125.

Valls, V., Martí, R., Lino, P., 1996. A branch and bound algorithm for minimizing the number of crossing arcs in bipartite graphs. European journal of operational research 90 (2), 303–319.

Wang, Y., Wu, Q., Glover, F., 2017. Effective metaheuristic algorithms for the minimum differential dispersion problem. European Journal of Operational Research 258 (3), 829–843.

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

Wu, S. D., Tsai, C. C., Yang, M., 2006. A vlsi layout legalization technique based on a graph fixing algorithm. In: International Symposium on Vlsi Design, Automation and Test. pp. 1–4.

Zhou, T., Wang, Y., Ding, J., Peng, B., 2016. Multi-start iterated tabu search for the minimum weight vertex cover problem. Journal of Combinatorial Optimization 32 (2), 368–384.

Zhou, Y., Wang, J., Wu, Z., Wu, K., 2018. A multi-objective tabu search algorithm based on decomposition for multi-objective unconstrained binary quadratic programming problem. Knowledge-Based Systems 141, 18–30.