Reinforcement learning driven heuristic for two-dimensional bandwidth minimization

Qing Zhou[a], Ming Gao[a], Jin-Kao Hao[b,*]

[a] *School of Business Administration, Northeastern University, 195 Chuangxin Road, Shenyang 110169, China, email: zhouqing@mail.neu.edu.cn; 2271262@stu.neu.edu.cn*
[b] *LERIA, Université d'Angers, 2 bd Lavoisier, 49045 Angers Cedex 01, France, email: jin-kao.hao@univ-angers.fr*

Applied Soft Computing; https://doi.org/10.1016/j.asoc.2025.114327

Abstract

The two-dimensional bandwidth minimization problem (2DBMP) is a representative graph layout problem, which originates from the circuit layout model where wires are traced horizontally or vertically. The problem involves embedding a guest graph into a grid-based host graph while minimizing the maximum distance between any two adjacent vertices. 2DBMP has many applications in telecommunications, implementing matrix decomposition, very large-scale integration (VLSI) design, and so on. However, the problem is computationally challenging because it is known to be NP-hard. Due to the inherent difficulty of solving the problem exactly, it is critical to design heuristic methods that are capable of producing high quality solutions in a reasonable amount of time. This study introduces an effective heuristic approach designed to address 2DBMP. The algorithm combines a reinforcement learning process for identifying promising search regions and a tabu search procedure for in-depth exploitation of those areas. Extensive experimentation on 90 commonly used benchmark instances demonstrates the competitiveness of the proposed algorithm in comparison to existing approaches from the literature. In particular, it achieves significant improvements in 22 cases and matches the best-known solutions for the remaining cases with only 4 exceptions. In addition, we report numerical results on 60 large and dense graphs used in many other bandwidth problems. We examine key algorithmic ingredients to understand their roles.

Keywords: Learning-based optimization; tabu search; reinforcement learning; bandwidth minimization.

1. Introduction

Let $G = (V_G, E_G)$ be an undirected guest graph, where $V_G = \{v_1, v_2, ..., v_n\}$ is the set of n vertices and $E_G \subset V_G \times V_G$ is the set of m edges. The host graph $H = (V_H, E_H)$ corresponding to the guest graph is a two-dimensional *grid graph* with the vertex set V_H ($|V_H| = \lceil\sqrt{n}\rceil \times \lceil\sqrt{n}\rceil$) and the edge set E_H. Each vertex of the grid host graph is represented as a pair (x, y) in the plane, where x, y are integer numbers and $1 \leq x, y \leq \lceil\sqrt{n}\rceil$.

Given a guest graph $G = (V_G, E_G)$ and its host graph $H = (V_H, E_H)$, an injective function establishes a particular relationship between the guest graph and the host graph, where each vertex in the guest graph has one and only one determined injective vertex in the host graph

[*] Corresponding author

and each vertex in the host graph holds at most one vertex from the guest graph. The injective function φ (also called embedding or labeling) can be defined as

$$\varphi : V_G \to V_H, \forall v \in V_G, \exists!\,(x,y) \in V_H \mid \varphi\,(v) = (x,y) \ \wedge \ \forall u \in V_G, u! = v \mid \varphi(u) \neq \varphi(v) \qquad (1)$$

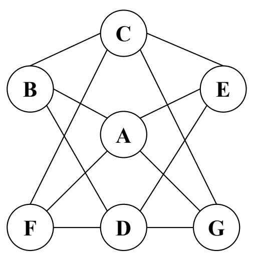
where $\varphi\,(v) = (x,y)$ is the label in the host graph of vertex v of the guest graph.

For a given embedding φ of the graph G, its bandwidth $BW(G,\varphi)$ represents the maximum distance between any two neighboring vertices of the host graph according to a distance metric d:

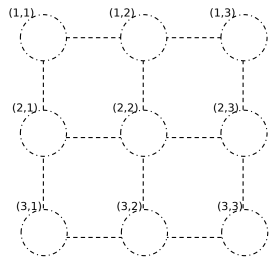
$$BW(G,\varphi) = \max_{(v,u)\in E_G} \{d[\varphi(v),\varphi(u)]\} \qquad (2)$$

The two-dimensional bandwidth minimization problem (2DBMP) originates from the circuit layout model in which wires are traced horizontally or vertically, and then the metric is usually expressed as a straight-line distance calculated using the L_1-*norm* [25] (also known as the Taxicab norm distance or the Manhattan distance [16]). Therefore, the distance between any two vertices (x_1,y_1) and (x_2,y_2) of the host graph is given by:

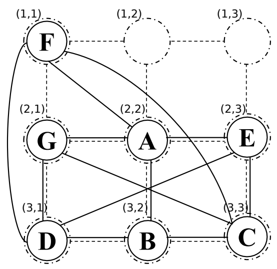
$$d\,[(x_1,y_1),(x_2,y_2)] = |x_1 - x_2| + |y_1 - y_2| \qquad (3)$$



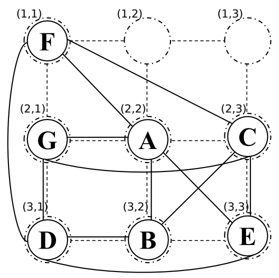
(a) A guest graph G



(b) A host graph H



(c) An example of an embedding φ_1



(d) An example of the optimal embedding φ_o

Figure 1: An example of two embeddings from G to H.

The goal of 2DBMP is then to ascertain an injective function φ^* that minimizes the bandwidth for the given guest graph G, i.e.,

$$\varphi^* = \arg\min_{\varphi\in\Phi}\{BW(G,\varphi)\} \qquad (4)$$

where Φ denotes the set of all potential embeddings of the guest graph.

Fig. 1 shows a guest graph $G = (V_G, E_G)$ (Fig. 1(a)) with 7 vertices and 12 edges and the corresponding host graph $H = (V_H, E_H)$ (Fig. 1(b)) with 9 vertices ($|V_H| = \lceil\sqrt{7}\rceil \cdot \lceil\sqrt{7}\rceil$), where each vertex is denoted by its (x, y) coordinates, i.e., $V_H = \{(1,1), ..., (3,3)\}$ and an edge exists between two vertices if they are at distance 1. Fig. 1(c) shows a suboptimal embedding φ_1, while Fig. 1(d) shows the optimal embedding φ_o. For instance, with embedding φ_1, vertex C of G receives the label (3, 3) in the host graph, while the optimal embedding gives the label (2, 3) to vertex C.

To assess the bandwidth of an embedding, it is essential to compute the distances between each pair of neighboring vertices in G, following Eq. (2) and Eq. (3). For the embedding φ_1 (Fig .1(c)), considering the adjacent vertices C and F of G, with $\varphi_1(C) = (3,3)$ and $\varphi_1(F) = (1,1)$, the associated distance $d[\varphi_1(C), \varphi_1(F)]$ is $|3-1| + |3-1| = 4$. Therefore, the bandwidth of the embedding φ_1 is given by

$$\begin{aligned} BW(G,\varphi_1) = \max\{&d[\varphi_1(A),\varphi_1(B)], d[\varphi_1(A),\varphi_1(E)], d[\varphi_1(A),\varphi_1(F)], d[\varphi_1(A),\varphi_1(G)],\\ &d[\varphi_1(B),\varphi_1(C)], d[\varphi_1(B),\varphi_1(D)], d[\varphi_1(C),\varphi_1(E)], d[\varphi_1(C),\varphi_1(F)],\\ &d[\varphi_1(C),\varphi_1(G)], d[\varphi_1(D),\varphi_1(E)], d[\varphi_1(D),\varphi_1(F)], d[\varphi_1(D),\varphi_1(G)]\}\\ =& \max\{1,1,2,1,1,1,1,4,3,3,2,1\} = 4 \end{aligned} \tag{5}$$

The bandwidth of the embedding φ_o is determined as follows.

$$\begin{aligned} BW(G,\varphi_o) = \max\{&d[\varphi_o(A),\varphi_o(B)], d[\varphi_o(A),\varphi_o(E)], d[\varphi_o(A),\varphi_o(F)], d[\varphi_o(A),\varphi_o(G)],\\ &d[\varphi_o(B),\varphi_o(C)], d[\varphi_o(B),\varphi_o(D)], d[\varphi_o(C),\varphi_o(E)], d[\varphi_o(C),\varphi_o(F)],\\ &d[\varphi_o(C),\varphi_o(G)], d[\varphi_o(D),\varphi_o(E)], d[\varphi_o(D),\varphi_o(F)], d[\varphi_o(D),\varphi_o(G)]\}\\ =& \max\{1,2,2,1,2,1,1,3,2,2,2,1\} = 3 \end{aligned} \tag{6}$$

Obviously, φ_o has a better objective value and this optimal embedding can be obtained from φ_1 by re-labeling the vertex E with $(3,3)$ and vertex C with $(2,3)$.

2DBMP has a number of important applications, especially in very large scale integration (VLSI) circuit modeling [2] and telecommunications [1, 8]. Meanwhile, the problem poses a significant computational challenge due to its NP-hard complexity [28]. Due to the relevance of 2DBMP, several solution methods have been proposed in the literature as reviewed in Section 2. We find that compared to other graph layout problems, research on 2DBMP is quite limited. These existing 2DBMP approaches have contributed to finding satisfactory solutions for several benchmark instances of 2DBMP. However, their performance varies depending on the test graphs, and they face the challenge of consistently generating high-quality solutions for large graphs. In this study, our goal is to advance the state of the art for effectively addressing 2DBMP by proposing a very competitive heuristic method that is able to obtain high quality solutions in a short computing time. The main contributions of this work are summarized as follows.

We propose the first reinforcement learning based tabu search algorithm (RLTS) for 2DBMP, which uses two complementary processes. The construction process relies on a probability matrix obtained through reinforcement learning to generate diverse and qualified starting solutions. The improvement process uses tabu search to explore a constrained neighborhood to improve the input solutions. Until now, the solution methods designed for 2DBMP rely on classical metaheuristic approaches and their combination. This work is the first study that shows that combining learning and metaheuristic methods can effectively solve the challenging 2DBMP problem.

Extensive experimental evaluations confirm that the proposed method possesses a remarkable competitive edge over the state-of-the-art methods in existing literature. Given that several practical problems can be recast as 2DBMP, the proposed algorithm can contribute to finding better solutions for these related applications and the program codes that we will make available to the community facilitate such applications.

Finally, the design principle of using reinforcement learning to generate initial solutions for further improvement is general. Therefore, it can be advantageously applied to other layout and bandwidth optimization problems.

The rest of the paper is organized as follows. Section 2 reviews existing related works. Section 3 outlines the methodology employed by the proposed method. Section 4 showcases the computational results and compares them with state-of-the-art algorithms. Following this, Section 5 provides an analysis of the essential components of the algorithm. Lastly, Section 6 summarizes the findings and offers perspectives for the future.

2. Literature review

Graph layout problems (GLPs) cover a wide range of classical combinatorial optimization models of great practical and theoretical importance. In GLPs, an input graph (or guest graph) is embedded in the output graph (or host graph) by an injective function that maps the vertices of the input graph to the output graph. Existing research on GLPs mainly focuses on regular host graphs, such as the path graphs [22], the cycle graphs [29], the tree graphs [10], the grid graphs [4], and others. Meanwhile, different GLPs have different optimization objective functions, such as the minimum cutwidth [18], the minimum sum of bandwidth [5], the minimum profile of graph [14], and so on.

2DBMP is a special graph layout problem where the host graph is a two-dimensional grid, with the goal of minimizing the bandwidth of the layout of the host graph. 2DBMP has practical applications in domains such as very large-scale integration (VLSI) circuit modeling [2], task scheduling for parallel processing systems [23], solving equations systems [23], and matrix decomposition [7]. In recent times, there has been a notable emphasis among researchers on the development of advanced metaheuristic algorithms, capable of delivering excellent solutions within acceptable computational times.

Rodríguez-García et al. [30] proposed a hybrid algorithm combining the GRASP method with path relinking to explore the solution space of 2DBMP. Later, Rodríguez-García et al. [31] developed a basic variable neighborhood search (BVNS) that integrates two constructive procedures and three local search procedures to solve 2DBMP. To assess the performance of BVNS, the authors designed three constraint satisfaction problem (CSP) models as the comparative algorithms and the experimental results demonstrated the effectiveness of BVNS.

Recently, Cavero et al. [4] proposed the Iterated Greedy (IG) algorithm, which exhibits very competitive performance. IG includes a greedy construction process to generate high-quality solutions, a destruction process and a reconstruction process balancing between intensification and diversification to perturb the incumbent solution. In addition, a novel local search procedure is integrated into IG, which includes three advanced enhancement techniques, i.e., the neighborhood reduction strategy, the tie-break criterion to deal with 'flat landscapes', and the fast evaluation of candidate solutions. Their computational results demonstrate that IG performs well compared to the most advanced methods available.

Very recently, Khandelwal et al. [20] studied the grid bandwidth minimization problem (denoted as GBMP), and the only difference between GBMP and 2DBMP is that GBMP considers the host graph to be a $l_1 \times l_2$ grid (l_1 and l_2 are input parameters) while 2DBMP considers the host graph to be a $\lceil\sqrt{n}\rceil \times \lceil\sqrt{n}\rceil$ square grid. For $l_1 = l_2 = \lceil\sqrt{n}\rceil$, GBMP is equivalent to 2DBMP, therefore 2DBMP is a special case of GBMP. In their work, the authors presented a simulated annealing algorithm for GBMP that uses a two problem-specific construction procedure to generate initial solutions and four neighborhoods to explore the search space. They included computational results on 2DBMP.

In [21], the same authors proposed a reduced variable neighborhood search algorithm (RVNS) that combines four construction heuristics, two new neighborhood search operators and a shaking procedure to explore the search space. They also reported computational results on 2DBMP and showed the superiority of RVNS over existing 2DBMP methods including BVNS [31].

According to the reported computational results, BVNS [31], RVNS [21], and IG [4] are the state-of-the-art algorithms for 2DBMP from the literature. However, there is still significant room for improvement in solution quality and computational efficiency. In fact, existing 2DBMP algorithms do not perform consistently well on the benchmark instances, and no single method is able to achieve all best-known results on the benchmark instances. In addition, some methods, such as BVNS, take a long time to achieve their results. Moreover, reinforcement learning techniques have not been incorporated in existing 2DBMP algorithms, although reinforcement learning-based methods have shown very competitive performance for effectively tackling several complex combinatorial optimization problems, as exemplified in [17, 19, 24, 34, 35, 36]. Finally, tabu search is one of the most powerful metaheuristics for solving complex optimization problems. Until now, the reinforcement learning strategies and tabu search are still unexplored for 2DBMP. This work therefore aims to fill these gaps by investigating, for the first time, the potential of reinforcement learning-based tabu search to tackle 2DBMP. As shown in the computational results of Section 4, this algorithm is indeed highly competitive compared to the existing 2DBMP methods.

Meanwhile, several advanced heuristics are also proposed to address highly related bandwidth problems, such as the multi-start search for the cyclic cut-width minimization problem [6], the adaptive large neighborhood search for the cut-width minimization problem [32], and the dual representation simulated annealing algorithm for the bandwidth minimization problem [33].

3. Reinforcement learning based tabu search

In this section, we present our approach, denoted as RLTS (reinforcement learning based tabu search), designed for 2DBMP. RLTS integrates a reinforcement learning strategy with a dedicated tabu search procedure.

3.1. General scheme

Reinforcement learning has the capacity to extract valuable insights from explored local optima, which can subsequently be leveraged to steer the algorithm towards promising uncharted territories during the ongoing search process. Reinforcement learning has been successfully integrated into heuristics to tackle a variety of challenging optimization problems, e.g., [17, 19, 24, 34, 35, 36]. In this work, a reinforcement learning based tabu search algorithm (RLTS) is proposed to solve 2DBMP, which integrates a probability learning strategy through reinforcement learning to produce diversified and high-quality initial solutions and an advanced tabu search algorithm to perform effective local optimization.

The pseudocode of the proposed RLTS is presented in Algorithm 1. RLTS starts with an initial probability matrix P_0 (more details of P_0 is given in Section 3.4) (line 1). Then, RLTS initializes the current best solution φ_{best} with the constructive procedure introduced in Section 3.2 (line 2). Subsequently, the algorithm proceeds through a sequence of iterations (lines 3-9). Within each iteration, an initial solution, denoted as φ_{start}, is created based on the probability matrix P (line 4). This initial solution is then subjected to further enhancements through the tabu search process outlined in Section 3.3, ultimately leading to a local optimum represented as φ_{lo} (line 5). After that, the probability matrix P is updated based on the newly reached local

Algorithm 1: The probability learning based tabu search for 2DBMP

```
   Input: Input graph $G$, cutoff time $t_{max}$
   Output: Best solution $\varphi_{best}$ found
1  $P \leftarrow P_0$ /* Initialize the probability matrix $P$ with $P_0$, see Section 3.4 */
2  $\varphi_{start} \leftarrow Construction\_using\_probability(P)$ /* Construct a solution based on $P$, Section 3.2 */
3  $\varphi_{best} \leftarrow \varphi_{start}$ /* Update the best found solution */
4  while $t_{max}$ is not reached do
5      $\varphi_{start} \leftarrow Construction\_using\_probability(P)$ /* Generate a starting solution, Section 3.2 */
6      $\varphi_{lo} \leftarrow Tabu\_search(\varphi_{start})$ /* Obtain a local optimal solution, Section 3.3 */
7      $P \leftarrow Probability\_updating(\varphi_{start}, \varphi_{lo}, P)$ /* Update the probability matrix, Section 3.4 */
8      $P \leftarrow Probability\_smoothing(P)$ /* Make probability smoothing, Section 3.4 */
9      if $BW(G, \varphi_{lo}) < BW(G, \varphi_{best})$ then
10         $\varphi_{best} \leftarrow \varphi_{lo}$ /* Update the best solution $\varphi_{best}$ */
11 return $\varphi_{best}$
```

optimal solution as described in Section 3.4 (lines 6-7). Furthermore, the algorithm updates the current best solution φ_{best} (lines 8-9) based on the resulting solution φ_{lo}, provided that φ_{lo} surpasses φ_{best} in terms of the bandwidth function (as defined in Eq. (2)). This iterative process continues until the predefined time limit is met, at which point the algorithm outputs the best solution discovered so far, φ_{best}.

3.2. Construction procedure

The construction procedure (denoted as CP) uses the probability matrix P to produce an initial (input) solution for the tabu search algorithm.

For ease of description, for the host graph V_H, we also call each vertex $(x, y) \in V_H$ $(1 \leq x, y \leq \lceil\sqrt{n}\rceil)$ a vertex j $(j \in \{1, ..., N\})$ by mapping the vertex $(1, 1)$ to vertex 1, $(1, 2)$ to vertex 2, ..., $(\lceil\sqrt{n}\rceil, \lceil\sqrt{n}\rceil)$ to vertex N. The probability matrix P is a two-dimensional matrix of size $n \times N$ $(N = \lceil\sqrt{n}\rceil^2)$, where each element p_{vj} $(1 \leq v \leq n, 1 \leq j \leq N)$ represents the probability of labeling a vertex v of the guest graph with the vertex j of the host graph, satisfying that $0 \leq p_{vj} \leq 1$ and $\sum_{j=1}^{N} p_{vj} = 1$ for each vertex $v = 1, ..., n$ and $j = 1, ..., N$.

As shown in Algorithm 2, CP first calculates the difference value *delta* between the largest and the second largest probabilities for each row of the probability matrix P (lines 1-6). Then CP determines the set of vertices *vertex_list* sorted by the *delta* values in a descending order (line 7). After that, CP traverses the set *vertex_list* to allocate each vertex of *vertex_list* to a vertex in the host graph in the roulette selection manner (lines 10-22). Specifically, for each vertex v, a roulette having un sections is constructed where un is equal to the number of unoccupied vertices of the host graph (lines 17-22). Moreover, the area of each section j of the roulette is equal to the probability p_{vj}. By using the roulette selection, a larger p_{vj} indicates that the vertex v has a larger probability to be allocated to the vertex j of the host graph. Once all the n vertices of *vertex_list* are allocated, an initial solution is created and the construction procedure is terminated.

Let us examine the time complexity of CP. The *delta* values of n vertices (lines 1-6) can be identified in $O(n \times N)$ time. The *vertex_list* $(|vertex\_list| = n)$ is sorted as a descending sequence having the time complexity of $O(nlogn)$. For each vertex v in *vertex_list*, the selection operation based on roulette wheel (lines 11-22) embedding v into the host graph is performed in

Algorithm 2: The construction procedure based on probability matrix

```
    Input: Probability matrix $P$
    Output: Constructed solution $\varphi_{start}$
 1  for $v \leftarrow 1$ to $n$ do
 2      $maxPro \leftarrow 0$ /* $maxPro$ records the maximum value of the probabilities in row $v$ of $P$ */
 3      for $j \leftarrow 1$ to $N$ do
 4          if $maxPro \leq p_{vj}$ then
 5              $delta[v] \leftarrow p_{vj} - maxPro$ /* $delta[v]$ indicates the difference value between the
                largest and the second largest probabilities in row $v$ of $P$ */
 6              $maxPro \leftarrow p_{vj}$
 7  $vertex\_list \leftarrow sort\_descending(delta)$ /* $vertex\_list$ keeps the number of $n$ vertices sorted
    by the $delta$ values in a descending order */
 8  for $j \leftarrow 1$ to $N$ do
 9      $host[j] \leftarrow false$ /* Indicate that no vertex in the host graph is assigned */
10  foreach $v$ in $vertex\_list$ do
11      $sumPro \leftarrow 0$
12      for $j \leftarrow 1$ to $N$ and the vertex $j$ in the host graph is unoccupied do
13          $sumPro \leftarrow sumPro + p_{vj}$
14      $r \leftarrow random() \times sumPro$ /* Function $random()$ generates a real number between 0 and 1 */
15      /* Roulette selection of a vertex $j$ from the host graph for vertex $v$ */
16      $roulette \leftarrow 0$
17      for $j \leftarrow 1$ to $N$ and the vertex $j$ in the host graph is unassigned do
18          $roulette \leftarrow roulette + p_{vj}$
19          if $r < roulette$ then
20              $\varphi_{start}(v) = j$
21              $host[j] = true$ /* Indicate that the vertex $j$ is assigned */
22              break
23  return $\varphi_{start}$
```

$O(N)$ time. The time complexity of the construction procedure is thus bounded by $O(n \times N \times 2 + nlogn)$.

3.3. Tabu search procedure

The proposed algorithm RLTS uses the tabu search (denoted by TS) procedure for local improvement, which relies on the general tabu search metaheuristic [12]. Our TS procedure is characterized by its constrained neighborhood and an advanced exploration strategy described in the following subsections.

3.3.1. Constrained neighborhood

To improve the given solution, TS iteratively transits from the current solution φ to a neighboring solution. For this, the move operator (denoted by $Move(\varphi, v, (x', y')))$ proposed in [4] is used, which is defined as follows. Given the current solution φ, suppose that the vertex v of the guest graph V_G is labeled with the vertex (x, y) of the host graph V_H (i.e., $\varphi(v) = (x, y)$). Then the move $Move(\varphi, v, (x', y'))$ replaces the current label (x, y) of v with a different label (x', y'). It should be noted that if (x', y') is the label of another vertex u before the move, a second move is performed to label the vertex u with (x, y). In this case, this double move is equivalent to exchange the label of v and the label of u.

The neighborhood induced by this $Move$ operator is given by:

$$N(\varphi) = \{\varphi' \mid \varphi \oplus Move(\varphi, v, (x', y')), v \in V_G, (x', y') \in V_H, \varphi(v) \neq (x', y')\} \tag{7}$$

We show two examples to illustrate this move in Fig. 2. Fig. 2(a) presents an embedding φ, where vertex E of the guest graph is allocated to vertex $(2, 3)$ before performing the $Move(\varphi, E, (1, 2))$ move. Then, Fig. 2(b) shows the resulting embedding φ' after the move, where E is allocated to $(1, 2)$, leaving $(2, 3)$ unallocated. In Fig .2(d), we show the new embedding φ' after performing $Move(\varphi, E, (3, 3))$, where E is assigned to $(3, 3)$ and simultaneously C receives the label $(2, 3)$.

To ensure a high search efficiency, TS uses a constrained move operator which is based on the bandwidth of the vertices of the guest graph and the region of interest (RoI) of vertices. For each vertex v of the guest graph G, the bandwidth of v is expressed as [27]:

$$BW_G(v, \varphi) = \max_{u \in V_G(v)} \{d(\varphi(v), \varphi(u))\} \tag{8}$$

where $V_G(v)$ is the set of adjacent vertices of v, i.e., $V_G(v) = \{u \in V_G \mid (u, v) \in E_G\}$. Therefore, the bandwidth of the embedding can be also presented as:

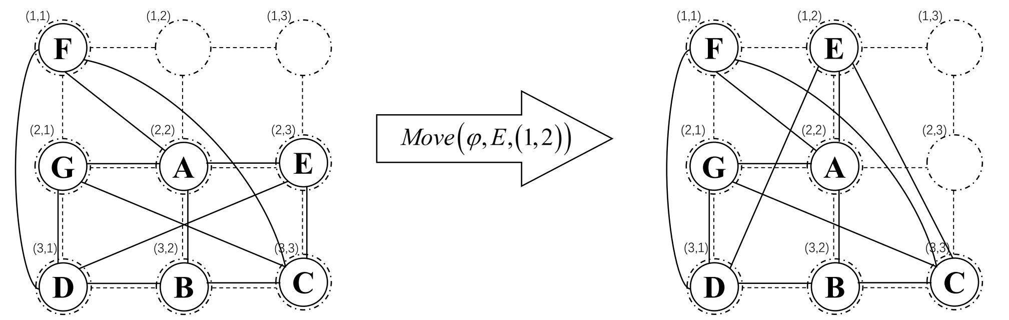
$$BW(G, \varphi) = \max_{v \in V_G} \{BW_G(v, \varphi)\} \tag{9}$$

RoI of a vertex $v \in V_G$ thus is a set of vertices $RoI(v)$ of the host graph such that the distance between the vertex $(x, y) \in RoI(v)$ and any adjacent vertex of v is not greater than the bandwidth of the embedding φ [4]. RoI of v can be defined as:

$$RoI\,(v) = \{(x, y) \in V_H \mid \forall u \in V_G\,(v)\,, d\,[\varphi\,(u)\,, (x, y)] \leq BW(G, \varphi)\} \tag{10}$$

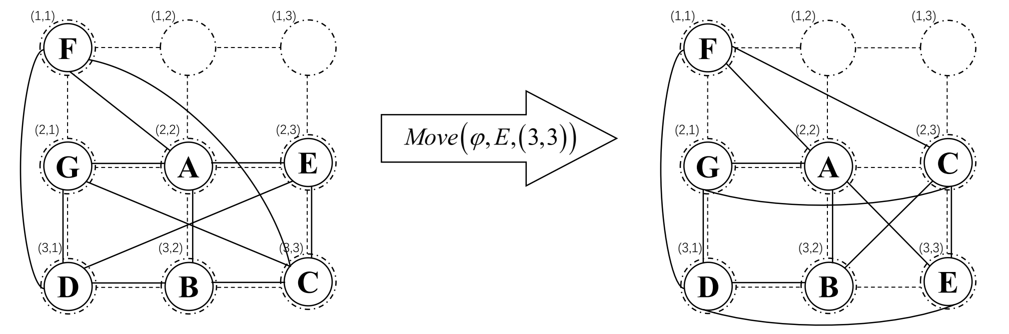
Let PV designate the set of potential vertices that collects the vertices of the guest graph with the largest bandwidths, i.e.,

$$PV(G, \varphi) = \{v \in V_G \mid BW_G\,(v, \varphi) = BW\,(G, \varphi)\} \tag{11}$$



(a) An embedding φ

(b) The resulting embedding φ' obtained by performing $Move(\varphi, E, (1,2))$



(c) An embedding φ

(d) The resulting embedding φ' obtained by performing $Move(\varphi, E, (3,3))$

Figure 2: Examples of performing $Move(\varphi, v, (x', y'))$ operation.

Considering that the key idea to improve the solution quality is to consider the vertices of PV, for each $Move(\varphi, v, (x', y'))$ operation, we restrict the vertex v to involve vertices of PV instead of all the vertices of the guest graph, and the vertex (x', y') is selected from the $RoI(v)$. For the current solution φ, the constrained neighborhood $CN(\varphi)$ induced by the constrained $Move(\varphi, v, (x', y'))$ operator is then defined by:

$$CN(\varphi) = \{Move(\varphi, v, (x', y')) \mid v \in PV, (x', y') \in RoI(v), \varphi(v) \neq (x', y')\} \tag{12}$$

Note that IG [4] uses a neighborhood reduction technique to reduce the size of $Move$ neighborhood $N(\varphi)$ by exploring the most promising neighbor solutions based on the RoI of each vertex of the guest graph. However, the proposed TS further reduces the neighborhood by restricting the candidate vertices of V_H to the RoI of each potential vertex only from PV rather than from all the vertices of the guest graph.

The size of the neighborhood $N(\varphi)$ without any reduction equals $n \times (N - 1)$. With the reduction strategy of RoI [4], the size of the $Move$ neighborhood is reduced to $n \times |RoI|$, where $|RoI|$ is mainly determined by the bandwidth of the current solution and the edge density of the guest graph according to Eq. (10) and satisfies $|RoI| < N$ in most cases. On the basis of RoI reduction strategy, the PV strategy used in this work further reduces the neighborhood size to $|PV| \times |RoI|$ and the size of PV always satisfies $|PV| < n$ according to Eq. (11). In Section 5.3, we demonstrate the benefit of the constrained neighborhood to the overall algorithm's performance.

3.3.2. Exploration with tabu search

Our TS procedure uses the $Move(\varphi, v, (x', y'))$ operator to explore the constrained neighborhood explained in the last section. To avoid short-term search cycling, TS employs a tabu list to avoid revisiting candidate solutions that were recently encountered. Specifically, after a $Move(\varphi, v, (x', y'))$ operation ($v \in V_G, (x', y') \in V_H$), the involved vertex v (and possibly vertex u allocated to (x', y') for the current solution φ) of the guest graph are recorded in the tabu list and will not be able to participate in any move operations for the subsequent L iterations (L is referred to as the tabu tenure).

In the proposed TS procedure, the tabu list is denoted as a $n \times N$ matrix tl where n and N represent respectively the number of vertices in the guest graph and the host graph with $N = \lceil\sqrt{n}\rceil \times \lceil\sqrt{n}\rceil$. Each element $tl(v, j)$ of tl ($v = 1, ..., n$, $j = 1, ..., N$) records the number of iterations that a vertex v from the guest graph cannot be allocated to a vertex j in the host graph. When a move operation $Move(\varphi, v, (x', y'))$ is performed, the tabu list is updated as [13]:

$$tl\,(v, \varphi(v)) = iter + L \quad (and\, tl(u, \varphi(u)) = iter + L,\, if\, \varphi(u) = (x', y')) \tag{13}$$

where L is the tabu tenure and follows a uniform distribution over the interval $[\mu(1-\varepsilon) \times N, \mu(1+\varepsilon) \times N]$, from which a value is randomly drawn at each time. The parameter $\mu \in [0, 1]$ controls the mean value of the tabu tenure and the parameter $\varepsilon \in [0, 1]$ determines the variation range of the tabu tenure.

Moreover, to guarantee the high computational efficiency of the search, TS uses the fast move calculation technique [4] to evaluate the neighboring solutions and the tie-break criterion [4] to distinguish between the neighboring solutions of the same quality.

The pseudocode of TS is depicted in Algorithm 3. TS first initializes the solutions φ_{lo} and φ_{temp} with the input solution φ_{start} (lines 1-2), and sets each element of the tabu list tl to 0 (lines 4-6). Then TS conducts a sequence of iterations (lines 9-39). During each iteration, a set of potential vertices PV is built by calculating the bandwidth of vertices (line 15, the definition of PV is given in Eq. (11)). Then each potential vertex from PV is considered by the move

Algorithm 3: The tabu search procedure

```
Input: Start solution $\varphi_{start}$, probability matrix $P$, search depth of tabu search $\omega$, tabu
       tenure $L$, cutoff time $t_{max}$
Output: Final solution $\varphi_{lo}$
1  $\varphi_{lo} \leftarrow \varphi_{start}$
2  $\varphi_{temp} \leftarrow \varphi_{start}$
3  /* Initialize the tabu list $tl$ */
4  for $v \leftarrow 1$ to $n$ do
5      for $j \leftarrow 1$ to $N$ do
6          $tl[v][j] \leftarrow 0$
7  $stagnate \leftarrow 0$
8  $iter \leftarrow 0$ /* Iteration counter */
9  while $Time() < t_{max}$ and $stagnate < \omega$ do
10     $min\_obj \leftarrow +\infty$
11     $(node, pos) \leftarrow (-1, -1)$
12     $stagnate \leftarrow stagnate + 1$
13     $iter \leftarrow iter + 1$
14     $exit\_signal \leftarrow false$
15     $PV(\varphi_{temp}) \leftarrow Build\_Vertices(\varphi_{temp})$ /* Section 3.3.1 */
16     foreach $v$ in $PV(\varphi_{temp})$ do
17         if $exit\_signal$ is $true$ then
18             break
19         $RoI(v) \leftarrow Build\_Region(\varphi_{temp}, v)$ /* Section 3.3.2*/
20         foreach vertex $j$ in the $RoI(v)$ do
21             /* The pair$< v, j >$ is not flagged as tabu */
22             if $tl[v][j] <= iter$ then
23                 $obj \leftarrow Fast\_Evaluate(\varphi_{temp}, v, j)$ /* Section 3.3.2 */
24                 if $obj < min\_obj$ then
25                     $min\_obj \leftarrow obj$
26                     $(node, pos) \leftarrow (v, j)$
27                 if $min\_obj < BW(G, \varphi_{temp})$ then
28                     $exit\_signal \leftarrow true$
29                     break /* An improved solution has been found, then exit the inner loop,
                         and pass the exit signal to the outer loop */
30     /* Execute the $Move$ operation, and update the tabu list */
31     if $node! = -1$ and $pos! = -1$ and the vertex $pos$ is unoccupied then
32         $j = \varphi_{temp}(node)$, $\varphi_{temp}(node) = pos$, $tl[node][j] = iter + L$
33     if $node! = -1$ and $pos! = -1$ and the vertex $pos$ is occupied then
34         $v_h \leftarrow \varphi_{temp}(pos)$ /* $v_h$ is the vertex hold by the vertex $pos$ of current solution */
35         $j \leftarrow \varphi_{temp}(node)$, $\varphi_{temp}(node) = pos$, $\varphi_{temp}(v_h) = j$
36         $tl[node][j] = iter + L$, $tl[v_h][pos] = iter + L$
37     if $BW(G, \varphi_{temp}) < BW(G, \varphi_{lo})$ then
38         $\varphi_{lo} \leftarrow \varphi_{temp}$
39         $stagnate \leftarrow 0$
40 return $\varphi_{lo}$
```

operation (line 16). For each potential vertex v ($v \in PV$), its region of interest $RoI(v)$ is built and each vertex j in $RoI(v)$ is visited for considering moving (lines 19-20). For each non-tabu candidate move operation $Move(\varphi, v, j)$, the fast evaluation technique is used to compute quickly the bandwidth of each candidate solution (line 23). Note that TS uses the first improvement strategy to visit $Move$ neighborhood, in other words, whenever an improved non-tabu candidate solution is encountered, TS performs the move (lines 27-29). Each time a $Move(\varphi, v, j)$ operation is executed, the involved vertex v (and possibly vertex u of the guest graph satisfying $\varphi(u) = j$) are appended to the tabu list (lines 31-36). The best solution discovered is refreshed whenever a superior solution is attained based on the bandwidth function (lines 37-39). TS continues until the best recorded solution φ_{lo} remains unchanged for a continuous span of ω iterations. Here, ω represents a parameter referred to as the search depth of TS. In each iteration (lines 10-29), we first determine the set of potential vertices PV (line 15) in $O(n)$ time by comparing the bandwidth of a number of n vertices according to Eq. (11). We then calculate $RoI(v)$ of each vertex v in PV (line 19) in $O(n \times N)$ time. The evaluation of move value of each $Move$ operation (line 23) can be accomplished in $O(n)$ benefiting from the fast evaluation technique [4]. Thus, the time complexity of each iteration of tabu search is bounded by $O(n + |PV| \times n \times N + |PV| \times |RoI| \times n)$.

3.4. Reinforcement learning

Reinforcement learning is a versatile learning approach that strives to acquire optimal actions from a finite selection of available actions by engaging in ongoing interactions with an unfamiliar environment. Unlike supervised learning methods, reinforcement learning does not require a knowledgeable agent to provide correct actions. Instead, it adapts its future actions based on the feedback signals it receives from the environment [15].

Our reinforcement learning process is implemented based on the probability matrix P (see Section 3.2). Recall that the probability matrix P is a $n \times N$ matrix, where each element p_{vj} ($1 \leq v \leq n$, $1 \leq j \leq N$) represents the probability of allocating a vertex v of the guest graph to the vertex j of the host graph, satisfying that $0 \leq p_{vj} \leq 1$ and $\sum_{j=1}^{N} p_{vj} = 1$ for each vertex $v = 1, ..., n$ and $j = 1, ..., N$.

At the beginning of the proposed algorithm, the probability matrix P is initiated as P_0 with its element $p_{vj} = 1/N$ ($v = 1, ..., n$, $j = 1, ..., N$). The reinforcement learning procedure is activated to update the probability matrix P when the tabu search procedure is terminated by comparing two embeddings φ and φ', which represent the input and output solutions of the tabu search, respectively. Specifically, for each vertex $v \in V_G$ of the guest graph, if φ and φ' have the same value, then we reward its original embedded vertex in the host graph as follows (t is the current iteration number of the 'while' loop in Algorithm 1):

$$p_{vj}(t+1) = \begin{cases} \alpha + (1-\alpha)p_{vj}(t) & j = \varphi(v) \\ (1-\alpha)p_{vj}(t) & j \in V_H \backslash \{\varphi(v)\} \end{cases} \tag{14}$$

Otherwise, we penalize the vertex $\varphi(v)$ and compensate the vertex $\varphi'(v)$ of the host graph. The probability matrix P is then updated as follows:

$$p_{vj}(t+1) = \begin{cases} (1-\gamma)(1-\beta)p_{vj}(t) & j = \varphi(v) \\ \gamma + (1-\gamma)\frac{\beta}{N-1} + (1-\gamma)(1-\beta)p_{vj}(t) & j = \varphi'(v) \\ (1-\gamma)\frac{\beta}{N-1} + (1-\gamma)(1-\beta)p_{vj}(t) & j \in V_H \backslash \{\varphi(v), \varphi'(v)\} \end{cases} \tag{15}$$

where α ($0 < \alpha < 1$) is a reward factor, β ($0 < \beta < 1$) is a penalization factor and γ ($0 < \gamma < 1$) is a compensation factor.

Obviously, each update of P requires traversing and calculating all values p_{vj} ($1 \leq v \leq n$, $1 \leq j \leq N$), and then the time complexity of updating P is bounded by $O(n \times N)$. Following the work of [35], we additionally uses a probability smoothing technique, which operates as follows: for each element p_{vj} ($1 \leq v \leq n$, $1 \leq j \leq N$), if its value reaches a predetermined threshold p_0, then p_{vj} is decreased by multiplying a smoothing coefficient ρ ($0 < \rho < 1$) [35]. To ensure that the sum of the probabilities for each row of P equals 1 after probability smoothing, we adjust all probabilities p_{vj} ($1 \leq j \leq N$) by dividing them by a coefficient $1 - (1 - \rho) \times p_{vj}$.

4. Computational experiments

In this section, we present comprehensive computational results obtained using the proposed RLTS algorithm on widely recognized benchmark instances. We conduct comparisons with several reference algorithms for a thorough evaluation.

4.1. Benchmark instances and experimental settings

The computational evaluations are based primarily on the two sets of 90 benchmark instances[1], which have been used in the 2DBMP literature [4, 31]. The first set of instances (named *Regular*) consists of 45 topologically diverse graphs of small size, with the number of vertices $n \in [5, 21]$, the number of edges $m \in [6, 190]$ and the density of the graph $\theta \in [9.52\%, 100.00\%]$. The second set of instances (called *Harwell-Boeing*) contains 45 representative and diverse graphs from the Harwell-Boeing Sparse Matrix Collection, where the number of vertices ranges from 48 to 960 and the number of edges ranges from 78 to 7442 and most of the graphs have a low density with $\theta < 5\%$.

In addition, we assess the performance of RLTS using three sets of widely used bandwidth benchmarks [11][2] which have not been used for testing 2DBMP methods to further understand the behavior of RLTS on large graphs and dense graphs. The first set (named *Grid*) contains 12 two-dimensional grid graphs where the number of vertices $n \in [960, 1170]$. The second set (denoted as *Hamming*) consists of 12 large regular graphs ($840 \leq n \leq 1152$). The third set (denoted as *Harwell-Boeing* V_2) consists of 36 more dense graphs from the Harwell-Boeing Sparse Matrix Collection with $\theta \in [5.02\%, 89.09\%]$. The information about the five sets of instances is given in Table 1.

Table 1: Information of the sets of benchmark instances.

Set	Size	n	m	θ
Regular	45	$[5, 21]$	$[6, 190]$	$[9.52\%, 100.00\%]$
Harwell-Boeing	45	$[48, 960]$	$[78, 7442]$	$[0.35\%, 20.33\%]$
Grid	12	$[960, 1170]$	$[1792, 2112]$	$[0.35\%, 0.39\%]$
Hamming	12	$[840, 1152]$	$[6480, 9216]$	$[1.39\%, 2.03\%]$
Harwell-Boeing V_2	36	$[9, 765]$	$[32, 44899]$	$[5.02\%, 89.09\%]$

The experiments were carried out on a computing platform equipped with an Intel Xeon E5-2695 v4 processor (2.1 GHz) and 1 GB of RAM, executing under the Linux operating system. The proposed RLTS algorithm was implemented in the C++ language [3] without relying on any special libraries and compiled using the g++ 7.3.0 compiler with the -O3 optimization option.

[1]The instances are publicly available at https://www.heuristicas.es/

[2]Extended instances have been made publicly available at https://grafo.etsii.urjc.es/optsicom/abp.html

[3]The source code of the proposed RLTS algorithm will be accessible upon the publication of the paper at `https://github.com/neteasefans/two-dimensional-bandwidth-minimization-problem`

4.2. Parameter tuning

Table 2: Parameter values tuned by 'irace' software.

Parameter	Section	Description	Candidate values	Final value
μ	3.3	control the mean value of tabu tenure	{0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9}	0.8
ε	3.3	determine the variation range of the tabu tenure	{0.2,0.3,0.4,0.5,0.6,0.7,0.8}	0.5
ω	3.3	search depth of tabu search	{1000,2000,3000,4000,5000,6000}	3000
α	3.4	reward factor	{0.1,0.2,0.3,0.4,0.5}	0.4
β	3.4	penalization factor	{0.1,0.2,0.3,0.4,0.5}	0.2
γ	3.4	compensation factor	{0.1,0.2,0.3,0.4,0.5}	0.3
p_0	3.4	smoothing threshold	{0.5,0.6,0.7,0.8,0.9}	0.7
ρ	3.4	smoothing coefficient	{0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9}	0.9

The proposed RLTS algorithm requires eight parameters: μ (to control the mean value of the tabu tenure), ε (to determine the variation range of the tabu tenure), ω (search depth of tabu search), α (reward factor), β (penalization factor), γ (compensation factor), p_0 (smoothing threshold), ρ (smoothing coefficient). The first 3 parameters are used to control the tabu search procedure, while the remaining 5 parameters are used to regulate the reinforcement learning process.

The tuning of these parameters is performed using an automated tuning package known as 'irace' [26], which is specifically designed for off-line parameter configuration of parameterized approaches. For the tuning experiment, a collection of 20 randomly chosen instances from the first four sets of 150 benchmark instances were used. The tuning budget (i.e., the maximum number of RLTS executions) was configured to 1000 runs, with each run allotted a time limit of 50 seconds. The candidate values of each parameter serving as the input of the 'irace' software, are chosen based on our preliminary experiments and are also inspired by the work of [24]. Table 2 illustrates the candidate values and the final value determined by 'irace' for each parameter. The reported experiments are based on these final parameter values.

4.3. Computational results and comparisons

Table 3: Comparative results between RLTS and the three reference methods including BVNS [31], RVNS [21] and IG [4] on the *Regular* and *Harwell-Boeing* benchmark sets.

Instance set	BVNS	RVNS	IG	RLTS
Regular (45 instances)				
#Best	39/–	–/–	45/44	45/45
#Improve/#Match	0/39	–/–	0/45	0/45
Average Time(s)	4.81	–	0.04	0.01
Average g_{best}/g_{avg}(%)	11.11/–	–/–	0.00/0.37	0.00/0.00
p-value$_{best}$	0.01	–	1.00	
p-value$_{avg}$	–	–	0.32	
Harwell-Boeing (45 instances)				
#Best	1/–	7/–	31/11	41/37
#Improve/#Match	0/4	0/13	15/26	22/19
Average Time(s)	439.63	–	116.05	113.94
Average g_{best}/g_{avg}(%)	50.67/–	34.62/–	-7.62/4.05	-8.87/-0.90
p-value$_{best}$	3.28E-011	4.60E-08	3.89E-03	
p-value$_{avg}$	–	–	3.12E-05	

To evaluate the performance of RLTS, this section presents computational results and compares RLTS with several state-of-the-art 2DBMP methods across the five sets of benchmark

Table 4: Comparative results between RLTS and the reference algorithm IG [4] on the *Grid*, *Hamming* and *Harwell-Boeing* V_2 benchmark sets.

Instance set		IG	RLTS
Grid (12 instances)			
	#Best	6/4	12/9
	Average Time(s)	19.05	34.91
	p-value$_{best}$	1.43E-02	
	p-value$_{avg}$	1.32E-01	
Hamming (12 instances)			
	#Best	1/3	12/9
	Average Time(s)	79.59	95.15
	p-value$_{best}$	9.11E-04	
	p-value$_{avg}$	8.33E-02	
Harwell-Boeing V_2 *(36 instances)*			
	#Best	32/16	36/33
	Average Time(s)	92.28	66.65
	p-value$_{best}$	4.55E-02	
	p-value$_{avg}$	1.17E-04	

Table 5: Results of the Wilcoxon signed-rank test for RLTS and the reference methods including BVNS [31], RVNS [21] and IG [4] on the five sets of instances, with a significance level of 0.05.

Instance set	Comparison	R^+_{best}	R^-_{best}	p-value	R^+_{avg}	R^-_{avg}	p-value
Regular (45 instances)							
	RLTS vs. BVNS	6	0	0.01	–	–	–
	RLTS vs. IG	0	0	1.00	1	0	0.32
Harwell-Boeing (45 instances)							
	RLTS vs. BVNS	44	0	6.17E-09	–	–	–
	RLTS vs. RVNS	38	3	9.95E-08	–	–	–
	RLTS vs. IG	11	1	2.01E-02	34	8	6.10E-04
Grid (12 instances)							
	RLTS vs. IG	6	0	0.01	8	3	0.13
Hamming (12 instances)							
	RLTS vs. IG	11	0	1.77E-03	9	3	6.51E-02
Harwell-Boeing V_2 (36 instances)							
	RLTS vs. IG	4	0	4.55E-02	20	3	1.32E-04

instances. According to numerical results presented in two latest studies on 2DBMP [4, 21], the iterated greedy (IG) algorithm [4], the basic variable neighborhood search (BVNS) [31], and the reduced variable neighborhood search (RVNS) algorithm [21] are the best performing methods for 2DBMP. Hence, we consider these algorithms as our primary reference points for the computational comparisons. Given that the source code for BVNS and RVNS is not accessible, the numerical results attributed to BVNS and RVNS are extracted directly from the relevant literature. The source code of IG is generously provided by the authors, and we ran independently IG and RLTS 20 times for each instance by using different random seeds, where for the z-th ($z = 1, 2, ..., 20$) execution, the random seed was set to be z, with a time limit of 500 seconds per run on the same computing platform detailed in Section 4.1.

Tables 3 and 4 summarize the comparison results on the five benchmark sets *Regular*, *Harwell-Boeing*, *Grid*, *Hamming* and *Harwell-Boeing* V_2. Row '#Best' displays the count of instances for which each respective approach produces the best results among all the compared methods in both best and average objective values. Row '#Improve/#Match' indicates the number of instances in which each algorithm either improves upon or matches the best-known solutions from the literature. Row 'Average Time(s)' provides the average runtime in seconds required by each algorithm to attain the final objective value across all test graphs[4]. Furthermore, row 'Average $g_{best}/g_{avg}(\%)$' presents the average percentage gap from the best and average objective values to the best-known solutions from the literature. In cases where a particular result is not provided by a compared algorithm, it is denoted by the symbol '–'. To assess whether there are statistically significant differences between RLTS and the compared algorithms regarding their best and average objective values, the p-values from the non-parametric Friedman test [9] are presented in the 'p-value$_{best}$' and 'p-value$_{avg}$' rows. A statistically significant difference (p-value ≤ 0.05) between the best and average performances of the compared algorithms suggests that the observed differences are meaningful and unlikely to result from random variation. Tables A.1-A.5 of the Appendix provide the detailed computational results of the compared algorithms on the five sets of benchmark instances.

One can observe from Table 3 that RLTS delivers highly competitive results against the state-of-the-art methods on the set of *Regular* and *Harwell-Boeing* instances, which are widely tested in 2DBMP literature. Specifically, in terms of the best objective values, RLTS reports the best results on 45 (41) cases out of 45 *Regular* (*Harwell-Boeing* instances), while BVNS, RVNS and IG yield the best results on 39 (1), – (7) and 45 (31) instances, respectively. Concerning the average objective values, RLTS and IG yield the best results on 45 (37) and 44 (11) instances. In addition, RLTS improves the best-known solutions in the literature for 22 instances and matches the best-known solutions for the remaining instances except 4 cases out of the two sets of 90 instances. The average running time in seconds needed to achieve the final objective value suggests that RLTS and IG demonstrate comparable performance, and has an obviously better performance than BVNS on both *Regular* and *Harwell-Boeing* sets. For both sets, RLTS obtains the smallest average percentage gap between the best or average objective value and the best-known solution from the literature (0.00%/0.00% and -8.87%/-0.90%). The p-values (p-values ≤ 0.05) from the non-parametric Friedman test reveal a statistically significant difference between RLTS and BVNS concerning the best or average objective values for *Regular* and *Harwell-Boeing* benchmarks. Additionally, a statistically significant difference is observed between IG and RLTS in the *Harwell-Boeing* instances with respect to the best/average performance.

To further evaluate the behavior of RLTS on large scale graphs and dense graphs, Table 4

[4]The result of BVNS was compiled from the original literature [31] and BVNS was executed on an Intel Core^{TM}i5-3470 CPU with 3.20 Ghz which is faster than that used by IG and RLTS according to the CPU frequency. RVNS [21] did not provide the running time information.

provides a summary of the comparative results between RLTS and IG [4] on the *Grid* , *Hamming* and Harwell-Boeing V_2. Table 4 shows that RLTS outperforms IG by providing 12/9, 12/9 and 36/33 best results with respect to the best/average objective value, while IG reports the best results on 6/4, 1/3 and 32/16 cases for the instances of *Grid*, *Hamming* and *Harwell-Boeing* V_2 sets. In terms of the computational efficiency, RLTS requires a slightly more running time in seconds to find the final solution on the *Grid* and *Hamming* benchmarks, while showing marginally better performance on the *Harwell-Boeing* V_2 set. The small p-values suggest that there exist statistically significant differences between RLTS and IG concerning the best and average performance.

Table 5 presents the Wilcoxon signed-rank test [3, 9] results between RLTS and the compared algorithms. Column '$\mathrm{R}^{+}_{\mathrm{best}}$' ('$\mathrm{R}^{+}_{\mathrm{avg}}$') represents the sum of ranks where RLTS surpasses the compared methods with respect to the best (average) objective value. Conversely, column '$\mathrm{R}^{-}_{\mathrm{best}}$' ('$\mathrm{R}^{-}_{\mathrm{avg}}$') reports the sum of ranks for situations where RLTS performs worse in comparison. Table 5 discloses that RLTS is statistically superior to the compared algorithms across the five sets of instances with p-values < 0.05. These observations show the advantages of RLTS in both solution quality and computational efficiency, comparing with the best performing 2DBMP methods in the literature.

In summary, when comparing with the state-of-the-art 2DBMP approaches, the proposed RLTS algorithm exhibits very competitive performance in both solution quality and computational efficiency for various test graphs. Moreover, as shown in Tables A.1-A.5 of the Appendix, RLTS shows a certain level of stability by yielding small standard deviation of objective values across 20 independent runs. However, the performance of the algorithm is, to some extent, dependent on parameter settings, and the presence of eight parameters makes the tuning process relatively challenging. As a heuristic algorithm, RLTS does not provide guarantees on solution optimality, making it impossible to precisely quantify the gap between the obtained solution and the global optimum.

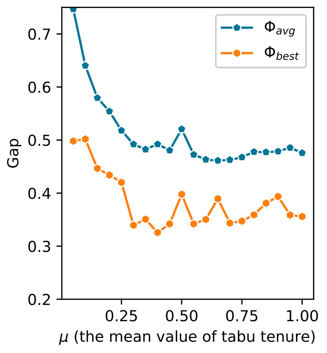
5. Analysis

This section analyzes some essential aspects of the proposed RLTS algorithm, including a sensitivity analysis of key parameters, the influence of the reinforcement learning strategy and the effect of the constrained neighborhood.

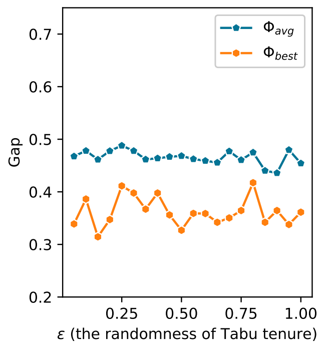
5.1. Sensitivity analysis of the parameters

The proposed RLTS algorithm necessitates eight parameters as depicted in Table 2. To understand how the parameters influence the performance of RLTS, we perform a sensitivity analysis by adjusting the values of each parameter within a reasonable range. For each candidate value of the parameters, RLTS was run 20 times independently, with a time limit of 500 seconds per run. Fig. 3 reports the obtained best objective value (referred to as Φ_{best}) and the average objective value (referred to as Φ_{avg}) across 20 randomly selected instances used in the parameter tuning experiment of Section 4.2. The X-axis represents the values of each parameter, and the Y-axis displays the average gap of the best/avearage objective value to the best result achieved in the experiments over the 20 instances.

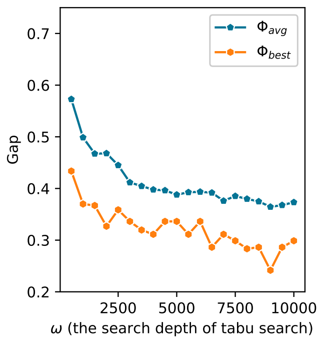
In order to ascertain whether there is a statistically significant difference in the solution quality obtained with different parameter values, the non-parametric Friedman tests were conducted and the results (p-value$_{best}$ / p-value$_{avg}$) are provided in Fig. 3. The tests indicate that RLTS is sensitive to the settings of parameters μ (p-value$_{best} = 1.10E-08$), ω (p-value$_{best} = 8.79E-10$), α (p-value$_{best} = 4.00E-06$), while this is not observed for the other parameters. For instance, RLTS exhibits sensitivity to μ, possibly attributed to the fact that a too small tabu tenure will



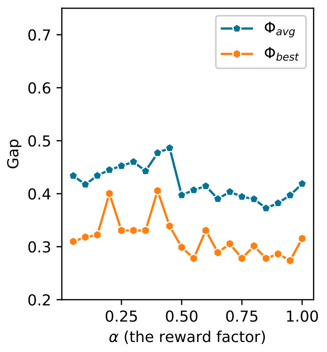
(a) p-value = 1.10E-08 / 1.11E-21



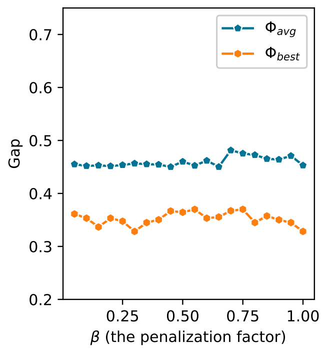
(b) p-value = 4.04E-01 / 8.24E-01



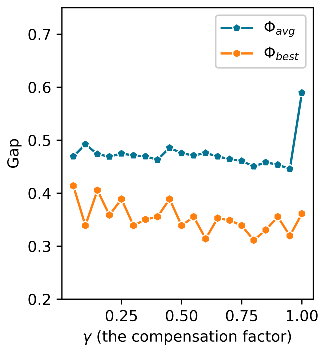
(c) p-value = 8.79E-10 / 7.20E-19



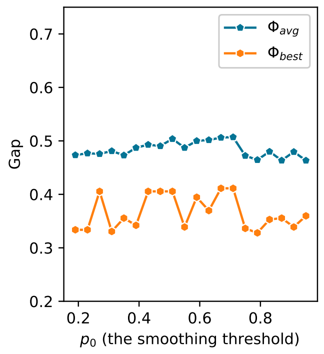
(d) p-value = 4.00E-06 / 4.60E-12



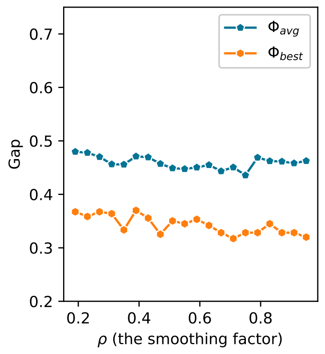
(e) p-value = 8.63E-01 / 1.08E-01



(f) p-value = 2.23E-01 / 2.20E-05



(g) p-value = 7.02E-01 / 5.27E-02



(h) p-value = 7.85E-01 / 2.02E-02

Figure 3: Sensitivity analysis of the parameters.

make it difficult for the algorithm to escape the local optimum traps, while a too long tabu tenure can prevent the algorithm from reaching some promising solutions. Considering that the tabu tenure L plays an important role in the tabu search process, and recalling that L is assumed to follow a uniform distribution over the interval $[\mu(1-\varepsilon) \times N, \mu(1+\varepsilon) \times N]$, from which a value is randomly selected at each time. As shown in Fig. 3(a) and 3(b), a higher value of u, approaching 1, contributes positively to the performance of the proposed algorithm, while the value of ε appears to have a relatively minor impact. This observation suggests that a relatively large mean value of L tends to yield better performance.

5.2. Effect of the reinforcement learning strategy and the tabu search procedure

To verify the effect of the reinforcement learning strategy, we made a comparison between RLTS and its two variants (denoted as ITS and ITS$_g$) such that the reinforcement learning strategy is disabled on the five sets of 150 benchmark instances. For ITS, each input solution for the tabu search procedure is generated by a random construction procedure (denoted by RCP), which operates as follows. For every vertex v in the input graph, RCP selects randomly an unoccupied vertex of the host graph to embed v. RCP repeats this process until all the vertices of the input graph are assigned. For ITS$_g$, the greedy construction method employed by IG [4] is used to generate an input solution for the tabu search procedure. For a fair comparison, RLTS, ITS and ITS$_g$ were run 20 times independently for each test graph.

Table 6: Comparative results between RLTS and its two variants ITS, ITS$_g$ across the five instance sets.

Instance set		ITS	ITS$_g$	RLTS
Regular (45 instances)				
	#Best	45/45	45/45	45/45
	Average Sd	0.00	0.00	0.00
	Average Time(s)	0.02	0.02	0.01
	p-value$_{best}$	1.00	1.00	
	p-value$_{avg}$	1.00	1.00	
Harwell-Boeing (45 instances)				
	#Best	17/8	29/22	45/40
	Average Sd	0.26	0.20	0.28
	Average Time(s)	79.66	39.62	111.37
	p-value$_{best}$	1.54E-08	2.71E-05	
	p-value$_{avg}$	2.50E-07	2.28E-05	
Grid (12 instances)				
	#Best	0/0	11/9	12/3
	Average Sd	0.38	0.37	0.51
	Average Time(s)	141.84	72.81	36.71
	p-value$_{best}$	2.22E-03	6.95E-01	
	p-value$_{avg}$	1.55E-03	8.33E-02	
Hamming (12 instances)				
	#Best	0/0	0/0	12/12
	Average Sd	0.63	0.54	0.74
	Average Time(s)	191.75	176.60	95.15
	p-value$_{best}$	2.21E-03	2.32E-03	
	p-value$_{avg}$	1.69E-03	1.84E-03	
Harwell-Boeing V_2 (36 instances)				
	#Best	32/21	32/23	36/35
	Average Sd	0.19	0.10	0.10
	Average Time(s)	52.04	41.46	66.65
	p-value$_{best}$	1.28E-04	1.44E-03	
	p-value$_{avg}$	4.55E-02	1.57E-01	

The comparative results are outlined in Table 6. Row 'Average Sd' indicates the average standard deviation of the objective value across all the test graphs of the benchmark set and the

Table 7: Comparative results between the tabu search procedure including ITS and ITS_g and the state-of-the-art methods on the *Regular* and *Harwell-Boeing* benchmark sets.

Instance set		BVNS	RVNS	IG	ITS	ITS_g
Regular (45 instances)						
	#Best	39/–	–/–	45/44	45/45	45/45
	#Improve/#Match	0/39	–/–	0/45	0/45	0/45
	Average Time(s)	4.81	–	0.04	0.02	0.02
	Average g_{best}/g_{avg}(%)	11.11/–	–/–	0.00/0.37	0.00/0.00	0.00/0.00
	p-value$_{best}$	1.43E-02	–	1.00		1.00
	p-value$_{avg}$	–	–	3.17E-01		1.00
Harwell-Boeing (45 instances)						
	#Best	1/–	8/–	40/15	19/10	35/35
	#Improve/#Match	0/4	0/13	15/26	20/14	4/26
	Average Time(s)	439.63	–	116.05	79.66	39.62
	Average g_{best}/g_{avg}(%)	50.67/–	34.62/–	-7.62/4.05	10.66/19.38	-2.34/2.61
	p-value$_{best}$	3.60E-09	4.59E-06	7.56E-06		9.19E-06
	p-value$_{avg}$	–	–	3.84E-04		2.59E-08

Table 8: Comparative results between the tabu search procedure including ITS and ITS_g and the state-of-the-art methods on the *Grid*, *Hamming* and *Harwell-Boeing* V_2 benchmark sets.

Instance set		IG	ITS	ITS_g
Grid (12 instances)				
	#Best	6/4	0/0	11/9
	Average Time(s)	19.05	141.84	72.81
	p-value$_{best}$	1.96E-03		1.53E-03
	p-value$_{avg}$	2.20E-03		2.20E-03
Hamming (12 instances)				
	#Best	11/10	4/0	5/2
	Average Time(s)	79.59	191.75	176.60
	p-value$_{best}$	1.93E-02		5.64E-01
	p-value$_{avg}$	2.22E-03		2.20E-03
Harwell-Boeing V_2 (36 instances)				
	#Best	33/19	33/23	35/29
	Average Time(s)	92.28	52.04	41.46
	p-value$_{best}$	1.00		1.57E-01
	p-value$_{avg}$	8.44E-02		1.83E-02

other symbols have the same meanings as those in Tables 3 and 4. Table 6 indicates that RLTS performs better than ITS and ITS_g. In terms of the best/average objective value, RLTS obtains the best results on 150/135 cases across all the five sets of 150 instances, while ITS and ITS_g yield the best results on 94/74 and 117/99 cases. Table 6 also indicates that the greedy construction method appears to be more advantageous compared to the random construction method for the tabu search procedure by comparing ITS and ITS_g in both best and average performances across the five benchmark sets. Moreover, ITS_g requires the same or less average running time to find the final solutions for all the five benchmarks. When comparing the average running times to reach the final solutions across the five sets of instances, RLTS requires a little more running time than ITS and ITS_g on the *Harwell-Boeing* and *Harwell-Boeing* V_2 sets and a slightly less time on the others sets. The small average standard deviation Sd (< 1.00) for the compared algorithms on the five benchmark sets across 20 independent runs to some extent reveals the stability of the algorithms. The small p-values ($p < 0.05$) verify that there are statistically significant differences between RLTS and its two variants ITS and ITS_g for the best and average performances. This experiment shows the effectiveness of the reinforcement learning strategy.

To verify the advantage of the tabu search, we compared the tabu search procedure, including ITS and ITS_g with the state-of-the-art 2DBMP algorithms. Tables 7 and 8 show the comparative results, from which it can be observed that ITS and ITS_g, particularly ITS_g achieve very competitive performance compared to the state-of-the-art 2DBMP methods in both the best and average objective values across the five benchmark sets. For the average running time to find the final solution, ITS and ITS_g use a little less time than the compared methods on the *Regular*, *Harwell-Boeing* and *Harwell-Boeing* V_2 benchmarks, while spending a slightly more time on the *Grid* and *Hamming* sets. The small p-values ($p \leq 0.05$) reveal the statistically significant differences between the tabu search procedure (including ITS and ITS_g) and the state-of-the-art 2DBMP approaches with respect to the best and average objective values. This comparison shows the usefulness of the tabu search procedure.

5.3. Effect of the constrained neighborhood structure

As introduced in Section 3.3.1, RLTS uses a constrained neighborhood during its local optimization procedure. To examine the effect of the constrained neighborhood, we made a comparison between RLTS and a variant RLTS-NRS by replacing the constrained neighborhood with the neighborhood reduction strategy proposed by [4].

Table 9 shows that RLTS is clearly superior to RLTS-NRS by yielding the best results on 150/150 instances for both the best and average objective values across the five set of instances, while RLTS-NRS produces the best results on 73/43 cases only. Moreover, RLTS requires less average running times to find its final solutions on all the five benchmark sets. We observe that the performance of the proposed algorithm deteriorates significantly, when the constrained neighborhood structure is disabled. The small p-values ($p < 0.05$) affirm the statistically significant differences between RLTS-NRS and RLTS for both the best and average performances. This experiment shows the effectiveness of the constrained neighborhood for the performance of RLTS algorithm. Furthermore, it discloses that the constrained neighborhood plays a key role in the success of RLTS algorithm.

5.4. Convergence analysis and stability of RLTS

To analyze the convergence of the proposed algorithm, Fig. 4 shows, for six graphs of different types, the running profile of the cost function defined by $i \mapsto f$ where i is the number of iterations and f represents the best objective value. As can be seen in the figure, the convergence of the algorithm depends on the problem instance. In general, RLTS can reach high-quality

Table 9: Comparative results between RLTS and RLTS-NRS across the five instance sets.

Instance set		RLTS-NRS	RLTS
Regular (45 instances)			
	#Best	45/39	45/45
	Average Sd	0.05	0.00
	Average Time(s)	5.51	0.01
	p-value$_{best}$	1.44E-02	
	p-value$_{avg}$	1.00	
Harwell-Boeing (45 instances)			
	#Best	6/0	45/45
	Average Sd	0.73	0.28
	Average Time(s)	193.84	111.37
	p-value$_{best}$	5.18E-09	
	p-value$_{avg}$	8.16E-09	
Grid (12 instances)			
	#Best	0/0	12/12
	Average Sd	3.11	0.51
	Average Time(s)	295.37	36.71
	p-value$_{best}$	2.22E-03	
	p-value$_{avg}$	1.89E-03	
Hamming (12 instances)			
	#Best	0/0	12/12
	Average Sd	0.92	0.74
	Average Time(s)	429.91	95.15
	p-value$_{best}$	2.22E-03	
	p-value$_{avg}$	2.16E-03	
Harwell-Boeing V_2 (36 instances)			
	#Best	22/4	36/36
	Average Sd	0.40	0.10
	Average Time(s)	82.04	66.65
	p-value$_{best}$	2.49E-07	
	p-value$_{avg}$	1.99E-04	

solutions after several ten to several hundred iterations, and the number of iterations required for convergence tends to increase with the instance size and density of the graph.

In addition, as shown in Section 5.2, RLTS exhibits a certain stability by producing a small standard deviation in the objective value across 20 independent runs for the five benchmark sets. Furthermore, the results of Tables A.1-A.5 of the Appendix show that RLTS consistently hits its best objective value in multiple runs across various instances, with an average hit of more than 12 out of 20 runs for the *Regular*, *Harwell-Boeing*, *Grid* and *Harwell-Boeing* V_2 benchmark sets. However, its performance is less stable on the *Hamming* set.

6. Conclusions

This work proposes an effective probability leaning based tabu search algorithm for the two-dimensional bandwidth minimization problem (2DBMP), which has a variety of applications, but is computationally challenging. The proposed algorithm depends on a reinforcement learning process to construct an input solution, which is subsequently enhanced by an effective tabu search procedure. The probability learning process is rooted in the concept of gathering valuable insights from previously already explored solutions to generate new promising solutions. The tabu search procedure utilizes an refined neighborhood reduction strategy to improve its search efficiency.

We evaluated the performance of the proposed algorithm using two sets of 90 instances commonly employed in 2DBMP literature. Our experimental results reveal that the proposed algorithm exhibits strong competitiveness when compared to the best-performing reference algorithms. Specifically, it managed to discover 22 improved best-known solutions, establishing new upper bounds, and achieved results on par with the best-known solutions for all but four out of the 90 benchmark instances. The proposed method is also tested on three other sets of bandwidth instances to further observe its behavior on large and dense graphs. We have delved into the key ingredients of the proposed method to gain insights into their roles. The source code of the proposed algorithm will be made accessible to the public. This will provide a valuable resource for practitioners and researchers engaged in addressing 2DBMP and related problems, enabling them to utilize the code for addressing their specific challenges.

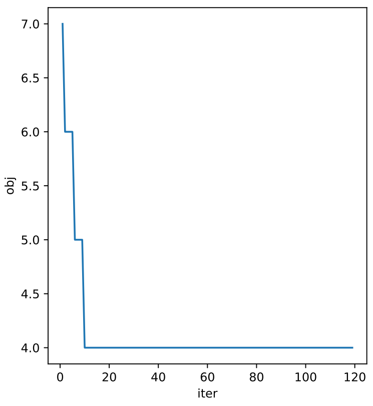
Since the proposed approach is a heuristic method, one cannot determine whether the best solutions found are optimal solutions. Consequently, there is a growing need for further exploration of exact methods, that can hopefully prove the solution optimality for instances of reasonable size. In addition, it would be intriguing to investigate the interest of basic ideas within the proposed algorithm, such as the reinforcement learning strategy, and the neighborhood reduction strategy for tackling related graph layout problems with complex constraints.

Acknowledgment

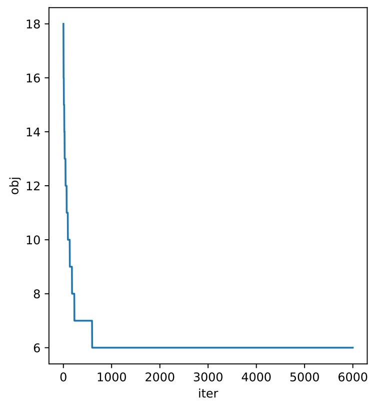
We are grateful to the reviewers for their comments that helped us to improve the paper. We would like to thank Dr. Sergio Cavero, Prof. Eduardo G. Pardo, and Prof. Abraham Duarte of [4] for sharing the source code of their IG algorithm.

References

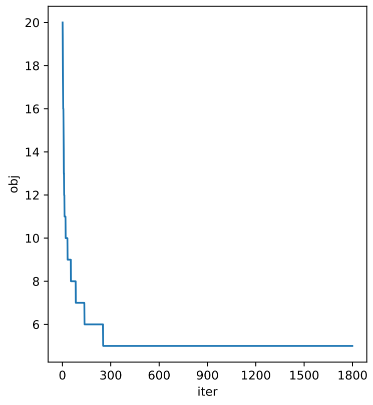
[1] Bezrukov, S. L., Chavez, J. D., Harper, L. H., Röttger, M., and Schroeder, U. P. (1998). Embedding of hypercubes into grids. In *Mathematical Foundations of Computer Science 1998: 23rd International Symposium, MFCS'98 Brno, Czech Republic, August 24–28, 1998 Proceedings 23*, pages 693–701. Springer.



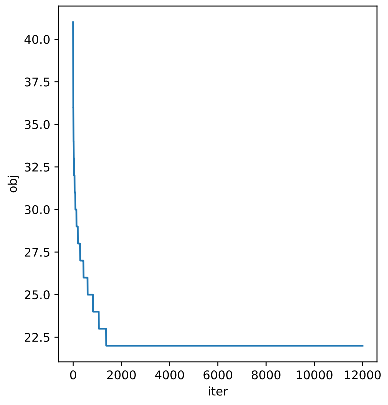
(a) k4xk5 (n=20)



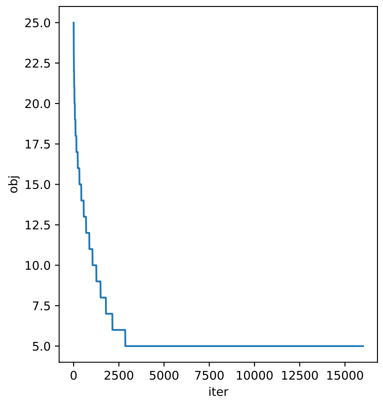
(b) lund_b (n=147)



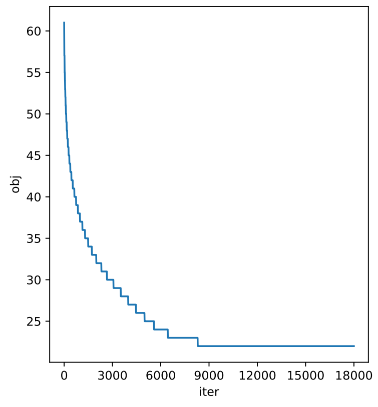
(c) ash292 (n=292)



(d) mbeause (n=492)



(e) mesh30_34 (n=1020)



(f) hamming3x4x4x4x6 (n=1152)

Figure 4: Convergence of RLTS.

[2] Bhatt, S. N. and Leighton, F. T. (1984). A framework for solving VLSI graph layout problems. *Journal of Computer and System Sciences*, 28(2):300–343.

[3] Carrasco, J., García, S., Rueda, M., Das, S., and Herrera, F. (2020). Recent trends in the use of statistical tests for comparing swarm and evolutionary computing algorithms: Practical guidelines and a critical review. *Swarm and Evolutionary Computation*, 54:100665.

[4] Cavero, S., Pardo, E. G., and Duarte, A. (2023). Efficient iterated greedy for the two-dimensional bandwidth minimization problem. *European Journal of Operational Research*, 306(3):1126–1139.

[5] Cavero, S., Pardo, E. G., Duarte, A., and Rodriguez-Tello, E. (2022). A variable neighborhood search approach for cyclic bandwidth sum problem. *Knowledge-Based Systems*, 246:108680.

[6] Cavero, S., Pardo, E. G., Laguna, M., and Duarte, A. (2021). Multistart search for the cyclic cutwidth minimization problem. *Computers & Operations Research*, 126:105116.

[7] Chinn, P. Z., Chvátalová, J., Dewdney, A. K., and Gibbs, N. E. (1982). The bandwidth problem for graphs and matrices–a survey. *Journal of Graph Theory*, 6(3):223–254.

[8] Chung, F. R. (1988). Labelings of graphs. In Beineke, L. W. and Wilson, R. J., editors, *Selected topics in graph theory*, pages 151–168. Academic Press.

[9] Derrac, J., García, S., Molina, D., and Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18.

[10] Ding, G. and Oporowski, B. (1995). Some results on tree decomposition of graphs. *Journal of Graph Theory*, 20(4):481–499.

[11] Duarte, A., Martí, R., Resende, M. G., and Silva, R. M. (2011). GRASP with path relinking heuristics for the antibandwidth problem. *Networks*, 58(3):171–189.

[12] Glover, F. (1989). Tabu search–part I. *ORSA Journal on Computing*, 1(3):190–206.

[13] Glover, F., Lü, Z., and Hao, J.-K. (2010). Diversification-driven tabu search for unconstrained binary quadratic problems. *4OR*, 8:239–253.

[14] Gonzaga de Oliveira, S. L., Bernardes, J. A., and Chagas, G. O. (2018). An evaluation of low-cost heuristics for matrix bandwidth and profile reductions. *Computational and Applied Mathematics*, 37:1412–1471.

[15] Gosavi, A. (2009). Reinforcement learning: A tutorial survey and recent advances. *INFORMS Journal on Computing*, 21(2):178–192.

[16] Han, J., Pei, J., and Tong, H. (2022). *Data mining: concepts and techniques.* Morgan Kaufmann.

[17] Han, Y., Peng, H., Mei, C., Cao, L., Deng, C., Wang, H., and Wu, Z. (2023). Multi-strategy multi-objective differential evolutionary algorithm with reinforcement learning. *Knowledge-Based Systems*, 277:110817.

[18] He, M., Wu, Q., and Lu, Y. (2022). Breakout local search for the cyclic cutwidth minimization problem. *Journal of Heuristics*, 28(5):583–618.

[19] Hu, Z., Gong, W., Pedrycz, W., and Li, Y. (2023). Deep reinforcement learning assisted co-evolutionary differential evolution for constrained optimization. *Swarm and Evolutionary Computation*, 83:101387.

[20] Khandelwal, A., Srivastava, K., and Saran, G. (2023a). Grid bandwidth minimization problem: simulated annealing approach. *Mapana Journal of Sciences*, 22:101–128.

[21] Khandelwal, A., Srivastava, K., and Saran, G. (2023b). A reduced variable neighbourhood search algorithm for grid bandwidth minimization problem. In *2023 7th International Conference on Intelligent Computing and Control Systems (ICICCS)*, pages 795–800. IEEE.

[22] Koohestani, B. (2023). On the solution of the graph bandwidth problem by means of search methods. *Applied Intelligence*, 53(7):7988–8004.

[23] Lai, Y.-L. and Williams, K. (1999). A survey of solved problems and applications on bandwidth, edgesum, and profile of graphs. *Journal of Graph Theory*, 31(2):75–94.

[24] Li, M., Hao, J.-K., and Wu, Q. (2022). Learning-driven feasible and infeasible tabu search for airport gate assignment. *European Journal of Operational Research*, 302(1):172–186.

[25] Lin, L. and Lin, Y. (2010). Two models of two-dimensional bandwidth problems. *Information Processing Letters*, 110(11):469–473.

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

[27] Mladenovic, N., Urosevic, D., Pérez-Brito, D., and García-González, C. G. (2010). Variable neighbourhood search for bandwidth reduction. *European Journal of Operational Research*, 200(1):14–27.

[28] Papadimitriou, C. H. (1976). The NP-Completeness of the bandwidth minimization problem. *Computing*, 16(3):263–270.

[29] Ren, J., Hao, J.-K., Rodriguez-Tello, E., Li, L., and He, K. (2020). A new iterated local search algorithm for the cyclic bandwidth problem. *Knowledge-Based Systems*, 203:106136.

[30] Rodríguez-García, M., Duarte, A., and Sánchez-Oro, J. (2018). GRASP with path relinking for 2D-bandwidth minimization problem. In *Proceedings of the International Conference on Learning and Optimization Algorithms: Theory and Applications*, pages 1–5.

[31] Rodríguez-García, M. Á., Sanchez-Oro, J., Rodriguez-Tello, E., Monfroy, E., and Duarte, A. (2021). Two-dimensional bandwidth minimization problem: Exact and heuristic approaches. *Knowledge-Based Systems*, 214:106651.

[32] Santos, V. G. M. and de Carvalho, M. A. M. (2021). Tailored heuristics in adaptive large neighborhood search applied to the cutwidth minimization problem. *European Journal of Operational Research*, 289(3):1056–1066.

[33] Torres-Jimenez, J., Izquierdo-Marquez, I., Garcia-Robledo, A., Gonzalez-Gomez, A., Bernal, J., and Kacker, R. N. (2015). A dual representation simulated annealing algorithm for the bandwidth minimization problem on graphs. *Information Sciences*, 303:33–49.

[34] Zhao, F., Zhuang, C., Wang, L., and Dong, C. (2024). An iterative greedy algorithm with q-learning mechanism for the multiobjective distributed no-idle permutation flowshop scheduling. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 54(5):3207–3219.

[35] Zhou, Y., Duval, B., and Hao, J.-K. (2018). Improving probability learning based local search for graph coloring. *Applied Soft Computing*, 65:542–553.

[36] Zhou, Y., Hao, J.-K., and Duval, B. (2016). Reinforcement learning based local search for grouping problems: A case study on graph coloring. *Expert Systems with Applications*, 64:412–422.

Appendix A. Detailed comparative results

Tables A.1-A.2 of the Appendix provide the comprehensive comparative results between the proposed RLTS algorithm and the best performing approaches in the literature across two sets of 90 widely used benchmark instances, including *Regular* and *Harwell-Boeing* under the condition indicated in Section 4.3. Note that RVNS [21] did not report results on the *Regular* benchmark. For a comprehensive evaluation of the behavior of RLTS on large scale graphs, Tables A.3-A.5 of the Appendix show the comparative results obtained by running RLTS and IG [4] on the same computing platform shown in Section 4.1 on the *Grid*, *Hamming* and *Harwell-Boeing* V_2 benchmarks. For each table, columns '*name*', 'n', 'm' and 'θ' give the name, the number of vertices, the number of edges and the density for each instance, respectively. Column 'f_{bk}' displays the best-known solution from the literature. Columns 'f_{best}', 'f_{avg}', 't_{avg}' and 'Sd' present the best objective value, the average objective value, the average running time in seconds to find the final objective value, and the standard deviation of the objective value across 20 independent runs, respectively. Column '$\#hit$' represents the number of runs reaching f_{best}. Row '#Best' denotes the number of cases that each algorithm produces the best results among all the compared methods concerning the best and average objective values. Row '#Improve/#Match' records the number of instances that each method either improves upon or matches the best-known solutions from the literature. Row 'Average' shows the average results across all the test graphs for the corresponding indicators. The best values among the compared results are highlighted in bold.

Table A.1: Comparison results between RLTS and the reference methods including BVNS [31] and IG [4] on the *Regular* set of instances.

Instance					BVNS		IG				RLTS				
name	n	m	θ	f_{bk}	f_{best}	t_{avg}	f_{best}	f_{avg}	t_{avg}	Sd	f_{best}	$\#hit$	f_{avg}	t_{avg}	Sd
wheel5	5	8	80.00%	**2**	**2**	0.12	**2**	2.00	0.01	0.00	**2**	20	2.00	0.00	0.00
k5	5	10	100.00%	**2**	**2**	0.12	**2**	2.00	0.01	0.00	**2**	20	2.00	0.00	0.00
p2xp3	6	7	46.67%	**1**	2	0.11	**1**	1.00	0.01	0.00	**1**	20	1.00	0.00	0.00
bipartite3x3	6	9	60.00%	**2**	**2**	0.18	**2**	2.00	0.01	0.00	**2**	20	2.00	0.00	0.00
p2xc3	6	9	60.00%	**2**	**2**	0.16	**2**	2.00	0.01	0.00	**2**	20	2.00	0.00	0.00
tree2x2	7	6	28.57%	**1**	**1**	0.11	**1**	1.00	0.01	0.00	**1**	20	1.00	0.00	0.00
bipartite3x4	7	12	57.14%	**3**	**3**	0.28	**3**	3.00	0.01	0.00	**3**	20	3.00	0.00	0.00
wheel7	7	12	57.14%	**2**	**2**	0.35	**2**	2.00	0.01	0.00	**2**	20	2.00	0.00	0.00
bipartite4x4	8	16	57.14%	**3**	**3**	0.56	**3**	3.00	0.02	0.00	**3**	20	3.00	0.00	0.00
p3xp3	9	12	33.33%	**1**	2	0.38	**1**	1.00	0.01	0.00	**1**	20	1.00	0.00	0.00
p3xc3	9	15	41.67%	**2**	**2**	0.54	**2**	2.00	0.02	0.00	**2**	20	2.00	0.00	0.00
c3xc3	9	18	50.00%	**2**	**2**	0.83	**2**	2.00	0.02	0.00	**2**	20	2.00	0.00	0.00
path10	10	9	20.00%	**1**	**1**	0.44	**1**	1.00	0.02	0.00	**1**	20	1.00	0.00	0.00
cycle10	10	10	22.22%	**1**	**1**	0.40	**1**	1.00	0.02	0.00	**1**	20	1.00	0.00	0.00
petersen	10	15	33.33%	**2**	**2**	0.77	**2**	2.00	0.02	0.00	**2**	20	2.00	0.00	0.00
wheel10	10	18	40.00%	**2**	**2**	1.17	**2**	2.00	0.02	0.00	**2**	20	2.00	0.00	0.00
cyclePow10-2	10	20	44.44%	**2**	**2**	1.12	**2**	2.00	0.02	0.00	**2**	20	2.00	0.00	0.00
bipartite5x5	10	25	55.56%	**3**	**3**	1.46	**3**	3.00	0.03	0.00	**3**	20	3.00	0.01	0.00
cyclePow10-10	10	45	100.00%	**4**	**4**	3.74	**4**	4.00	0.05	0.00	**4**	20	4.00	0.00	0.00
k10	10	45	100.00%	**4**	**4**	4.15	**4**	4.00	0.03	0.00	**4**	20	4.00	0.00	0.00
c3xc4	12	24	36.36%	**2**	**2**	2.13	**2**	2.00	0.03	0.00	**2**	20	2.00	0.00	0.00
p3xk4	12	26	39.39%	**2**	**2**	2.58	**2**	2.00	0.03	0.00	**2**	20	2.00	0.07	0.00
c3xk4	12	30	45.45%	**3**	**3**	2.59	**3**	3.00	0.04	0.00	**3**	20	3.00	0.00	0.00
k3xk4	12	30	45.45%	**3**	**3**	2.83	**3**	3.00	0.03	0.00	**3**	20	3.00	0.00	0.00
tree2x3	13	12	15.38%	**2**	**2**	0.99	**2**	2.00	0.02	0.00	**2**	20	2.00	0.00	0.00
path15	15	14	13.33%	**1**	2	1.32	**1**	1.00	0.02	0.00	**1**	20	1.00	0.00	0.00
tree3x2	15	14	13.33%	**2**	**2**	1.25	**2**	2.00	0.03	0.00	**2**	20	2.00	0.00	0.00
cycle15	15	15	14.29%	**2**	**2**	1.19	**2**	2.00	0.03	0.00	**2**	20	2.00	0.00	0.00
wheel15	15	28	26.67%	**3**	**3**	4.13	**3**	3.00	0.03	0.00	**3**	20	3.00	0.00	0.00
cyclePow15-2	15	30	28.57%	**2**	**2**	3.58	**2**	2.00	0.04	0.00	**2**	20	2.00	0.00	0.00
bipartite7x8	15	56	53.33%	**4**	**4**	9.03	**4**	4.00	0.06	0.00	**4**	20	4.00	0.00	0.00
cyclePow15-10	15	105	100.00%	**6**	**6**	15.01	**6**	6.00	0.07	0.00	**6**	20	6.00	0.00	0.00
path20	20	19	10.00%	**1**	**1**	4.04	**1**	1.00	0.04	0.00	**1**	20	1.00	0.01	0.00
cycle20	20	20	10.53%	**1**	**1**	4.30	**1**	1.00	0.06	0.00	**1**	20	1.00	0.03	0.00
p4xp5	20	31	16.32%	**1**	2	7.84	**1**	1.00	0.05	0.00	**1**	20	1.00	0.16	0.00
p4xc5	20	35	18.42%	**2**	3	7.55	**2**	2.00	0.06	0.00	**2**	20	2.00	0.01	0.00
wheel20	20	38	20.00%	**3**	**3**	11.93	**3**	3.50	0.05	0.50	**3**	20	3.00	0.00	0.00
c4xc5	20	40	21.05%	**2**	**2**	9.15	**2**	2.00	0.06	0.00	**2**	20	2.00	0.08	0.00
cyclePow20-2	20	40	21.05%	**2**	3	9.26	**2**	2.00	0.05	0.00	**2**	20	2.00	0.01	0.00
p4xk5	20	55	28.95%	**3**	**3**	14.81	**3**	3.00	0.06	0.00	**3**	20	3.00	0.00	0.00
c4xk5	20	60	31.58%	**3**	**3**	17.86	**3**	3.00	0.08	0.00	**3**	20	3.00	0.01	0.00
k4xk5	20	70	36.84%	**4**	**4**	20.00	**4**	4.00	0.08	0.00	**4**	20	4.00	0.00	0.00
bipartite10x10	20	100	52.63%	**5**	**5**	20.00	**5**	5.00	0.19	0.00	**5**	20	5.00	0.00	0.00
cyclePow20-10	20	190	100.00%	**6**	**6**	20.00	**6**	6.00	0.12	0.00	**6**	20	6.00	0.00	0.00
tree2x4	21	20	9.52%	**2**	**2**	5.93	**2**	2.00	0.05	0.00	**2**	20	2.00	0.00	0.00
#Best				45	39		45	44			45		45		
#Improve/#Match					0/39		0/45				0/45				
Average			42.13%	2.42	2.56	4.81	2.42	2.43	0.04	0.01	2.42	20	2.42	0.01	0.00

Table A.2: Results obtained by RLTS and the compared methods including BVNS [31], RVNS [21] and IG [4] on the *Harwell-Boeing* set of instances.

Instance					BVNS		RVNS	IG				RLTS				
name	n	m	θ	f_{bk}	f_{best}	t_{avg}	f_{best}	f_{best}	f_{avg}	t_{avg}	Sd	f_{best}	$\#hit$	f_{avg}	t_{avg}	Sd
bcsstk01	48	176	15.60%	5	5	48.00	5	**4**	4.50	0.48	0.50	**4**	20	4.00	0.36	0.00
can___62	62	78	4.12%	**2**	**2**	62.00	3	**2**	2.30	0.39	0.46	**2**	20	2.00	0.33	0.00
nos4	100	247	4.99%	4	4	100.01	4	4	4.00	16.96	0.00	**3**	20	3.00	24.26	0.00
bcspwr03	118	179	2.59%	**3**	4	118.01	4	**3**	3.00	1.16	0.00	**3**	20	3.00	0.48	0.00
bcsstk04	132	1758	20.33%	**8**	9	132.01	**8**	**8**	8.50	52.80	0.50	**8**	20	8.00	48.40	0.00
bcsstk22	138	279	2.95%	**3**	4	138.01	4	**3**	3.10	3.00	0.30	**3**	20	3.00	2.46	0.00
can__144	144	576	5.59%	**4**	5	144.01	**4**	**4**	4.10	5.06	0.30	**4**	20	4.00	4.09	0.00
bcsstk05	153	1135	9.76%	**5**	7	153.01	**5**	6	6.80	62.69	0.40	6	0	6.38	80.99	0.48
can__161	161	608	4.72%	**4**	6	161.01	5	**4**	4.80	15.68	0.40	**4**	20	4.00	18.43	0.00
dwt__198	198	597	3.06%	**3**	5	198.01	**3**	**3**	3.80	24.44	0.40	**3**	20	3.00	31.11	0.00
dwt__209	209	767	3.53%	5	6	209.01	5	5	5.10	11.74	0.30	**4**	10	4.97	6.62	0.16
dwt__221	221	704	2.90%	**4**	5	221.01	6	**4**	4.10	11.38	0.30	**4**	20	4.00	11.44	0.00
can__229	229	774	2.96%	5	7	229.01	5	**4**	4.90	25.33	0.30	**4**	20	4.00	27.47	0.00
dwt__234	234	300	1.10%	**2**	4	234.01	**2**	3	3.50	9.28	0.50	3	0	3.00	5.49	0.00
nos1	237	390	1.39%	**3**	4	237.01	4	**3**	3.00	5.57	0.00	**3**	20	3.00	4.39	0.00
dwt__245	245	608	2.03%	**4**	6	245.02	5	**4**	4.70	14.27	0.46	**4**	20	4.00	12.49	0.00
lshp_265	265	744	2.13%	**3**	6	265.00	6	**3**	3.80	113.35	0.40	**3**	18	3.10	160.97	0.30
bcspwr04	274	669	1.79%	**4**	7	274.02	6	**4**	4.60	17.12	0.49	**4**	20	4.00	13.43	0.00
ash292	292	958	2.25%	**4**	6	292.00	5	**4**	4.20	22.68	0.40	**4**	20	4.00	25.45	0.00
can__292	292	1124	2.65%	6	7	292.00	8	**5**	6.00	119.47	0.45	**5**	20	5.00	155.42	0.00
dwt__307	307	1108	2.36%	5	7	307.00	6	**4**	4.70	58.17	0.46	**4**	20	4.00	70.82	0.00
dwt__310	310	1069	2.23%	**4**	5	310.00	6	**4**	4.10	37.57	0.30	**4**	20	4.00	47.07	0.00
dwt__361	361	1296	1.99%	5	8	361.01	6	**4**	4.80	97.75	0.40	**4**	19	4.05	127.28	0.22
plat362	362	2712	4.15%	6	8	362.01	6	6	6.10	162.65	0.30	**5**	15	5.25	197.71	0.43
bcsstk07	420	3720	4.23%	**6**	9	420.01	8	**6**	6.20	232.38	0.40	**6**	11	6.92	192.81	0.72
bcspwr05	443	590	0.60%	5	5	443.01	6	4	4.80	60.31	0.40	**3**	12	3.83	68.05	0.38
can__445	445	1682	1.70%	7	9	445.01	7	6	6.40	100.90	0.49	**5**	4	5.80	111.34	0.40
bcsstk20	485	1325	1.13%	**4**	6	485.01	6	**4**	4.00	55.25	0.00	**4**	20	4.00	57.66	0.00
494_bus	494	586	0.48%	5	6	494.01	6	4	4.80	45.46	0.40	**3**	11	3.92	42.68	0.26
dwt__503	503	2762	2.19%	6	8	503.01	8	6	6.50	158.73	0.50	**5**	14	5.62	176.00	0.48
lshp_577	577	1656	1.00%	5	8	577.00	6	**4**	4.20	184.17	0.40	**4**	18	4.17	241.17	0.38
dwt__607	607	2262	1.23%	**4**	9	607.00	**4**	5	5.90	171.23	0.30	5	0	5.22	176.02	0.42
662_bus	662	906	0.41%	5	7	662.01	6	5	5.20	175.06	0.40	**4**	19	4.08	220.74	0.26
nos6	675	1290	0.57%	5	14	960.01	10	**4**	4.70	161.26	0.46	**4**	19	4.05	214.92	0.22
685_bus	685	1282	0.55%	5	8	685.01	6	**4**	4.90	187.58	0.30	**4**	17	4.15	245.07	0.36
can__715	715	2975	1.17%	**8**	11	715.01	**8**	**8**	8.60	393.70	0.49	**8**	9	9.10	158.07	0.62
nos7	729	1944	0.73%	6	10	729.01	8	**5**	5.80	196.12	0.40	**5**	13	6.05	183.88	0.50
dwt__758	758	2618	0.91%	5	7	758.01	7	**4**	4.90	189.10	0.30	**4**	13	4.67	218.49	0.61
lshp_778	778	2247	0.74%	**4**	9	778.01	8	5	5.10	227.53	0.30	**4**	13	5.08	263.00	0.41
bcsstk19	817	3018	0.91%	6	9	817.00	8	6	6.50	312.34	0.50	**5**	11	5.88	233.02	0.40
dwt__878	878	3285	0.85%	**5**	9	878.00	8	**5**	5.60	258.44	0.49	**5**	8	6.20	218.24	0.81
gr_30_30	900	3422	0.85%	**2**	9	900.01	8	**2**	3.80	227.22	2.32	5	0	6.20	250.23	0.51
dwt__918	918	3233	0.77%	6	9	918.01	8	**5**	5.20	273.97	0.40	**5**	7	6.30	284.34	0.81
nos2	957	1590	0.35%	4	6	957.01	6	**3**	3.70	169.41	0.46	**3**	5	3.75	190.91	0.49
nos3	960	7442	1.62%	7	14	960.01	10	7	7.50	553.22	0.92	**6**	10	8.05	303.51	1.14
#Best				23	1		7	31	11			41		37		
#Improve/#Match					0/4		0/13	15/26				22/19				
Average			2.89%	4.69	7.09	439.63	6.07	4.44	4.95	116.05	0.42	4.27	14.58	4.66	113.94	0.26

Table A.3: Results obtained by RLTS and the compared method IG [4] on the *Grid* set of instances.

Instance				IG				RLTS				
name	n	m	θ	f_{best}	f_{avg}	t_{avg}	Sd	f_{best}	$\#hit$	f_{avg}	t_{avg}	Sd
mesh120_8	960	1792	0.39%	**4**	4.80	14.87	0.40	**4**	14	4.92	22.00	0.26
mesh110_9	990	1861	0.38%	**4**	4.30	13.24	0.46	**4**	19	4.22	33.42	0.55
mesh100_10	1000	1890	0.38%	**4**	4.90	17.12	0.54	**4**	14	4.88	30.37	0.64
mesh50_20	1000	1930	0.39%	5	5.10	16.51	0.30	**4**	18	4.33	34.02	0.57
mesh40_25	1000	1935	0.39%	5	5.80	16.91	0.40	**4**	15	5.28	33.01	0.50
mesh60_17	1020	1963	0.38%	5	5.30	16.27	0.64	**4**	3	4.85	33.85	0.54
mesh30_34	1020	1976	0.38%	5	6.40	25.99	0.92	**4**	15	5.15	36.62	0.76
mesh34_30	1020	1976	0.38%	5	6.40	26.11	0.92	**4**	16	5.12	36.86	0.75
mesh80_13	1040	1987	0.37%	**4**	4.90	14.28	0.70	**4**	11	5.33	42.12	0.65
mesh70_15	1050	2015	0.37%	**4**	4.90	19.01	0.30	**4**	10	5.53	34.90	0.59
mesh90_12	1080	2058	0.35%	5	5.00	14.83	0.00	**4**	5	4.75	39.08	0.73
mesh33_33	1089	2112	0.36%	**1**	5.20	33.45	2.79	**1**	6	5.20	42.66	1.63
#Best				6	4			12		9		
Average			0.38%	4.25	5.25	19.05	0.70	3.75	12.17	4.96	34.91	0.68

Table A.4: Results obtained by RLTS and the compared method IG [4] on the *Hamming* set of instances.

Instance				IG				RLTS				
name	n	m	θ	f_{best}	f_{avg}	t_{avg}	Sd	f_{best}	$\#hit$	f_{avg}	t_{avg}	Sd
hamming2x3x4x5x7	840	6720	1.91%	18	18.40	43.65	0.80	**16**	7	17.30	58.26	0.68
hamming2x2x5x6x7	840	7140	2.03%	17	17.40	43.65	0.49	**16**	13	17.07	65.58	0.82
hamming3x3x4x4x6	864	6480	1.74%	18	18.60	86.63	0.49	**17**	3	17.85	64.90	0.53
hamming2x3x4x6x6	864	6912	1.85%	17	17.40	73.31	0.49	**16**	7	17.30	62.41	0.95
hamming3x3x4x5x5	900	6750	1.67%	18	18.80	70.24	0.75	**17**	13	18.07	69.03	0.52
hamming3x4x4x4x5	960	7200	1.56%	18	19.20	79.34	0.75	**17**	3	18.70	72.67	0.78
hamming2x3x4x5x8	960	8160	1.77%	18	18.60	116.77	0.49	**17**	15	18.27	83.04	1.00
hamming3x3x4x4x7	1008	8064	1.59%	19	20.20	80.19	0.75	**18**	15	19.27	87.99	0.74
hamming2x3x3x7x8	1008	9072	1.79%	19	19.60	107.28	0.49	**18**	3	19.70	98.96	0.78
hamming3x3x4x5x6	1080	8640	1.48%	20	21.20	69.67	0.75	**18**	1	19.90	115.60	0.44
hamming2x3x5x6x6	1080	9180	1.58%	**19**	19.60	112.33	0.49	**19**	2	20.80	132.98	0.81
hamming3x4x4x4x6	1152	9216	1.39%	21	21.80	72.03	0.75	**20**	7	21.98	230.39	0.82
#Best				1	3			12		9		
Average			1.70%	18.50	19.23	79.59	0.62	17.42	7.42	18.85	95.15	0.74

Table A.5: Results obtained by RLTS and the compared method IG [4] on the *Harwell-Boeing* V_2 set of instances.

Instance				IG				RLTS				
name	n	m	θ	f_{best}	f_{avg}	t_{avg}	Sd	f_{best}	$\#hit$	f_{avg}	t_{avg}	Sd
jgl009	9	32	88.89%	**3**	3.00	0.01	0.00	**3**	20	3.00	0.00	0.00
jgl011	11	49	89.09%	**4**	4.00	0.02	0.00	**4**	20	4.00	0.00	0.00
can___24	24	68	24.64%	**3**	3.00	0.03	0.00	**3**	20	3.00	0.00	0.00
pores_1	30	103	23.68%	**3**	3.00	0.05	0.00	**3**	20	3.00	0.00	0.00
ibm32	32	90	18.15%	**4**	4.00	0.07	0.00	**4**	20	4.00	0.01	0.00
bcspwr01	39	46	6.21%	**2**	2.00	0.05	0.00	**2**	20	2.00	0.01	0.00
bcspwr02	49	59	5.02%	**2**	2.70	0.09	0.47	**2**	20	2.00	0.06	0.00
curtis54	54	124	8.67%	**3**	3.00	0.14	0.00	**3**	20	3.00	0.01	0.00
will57	57	127	7.96%	**3**	3.00	0.14	0.00	**3**	20	3.00	0.01	0.00
dwt___59	59	104	6.08%	3	3.00	0.12	0.00	**2**	20	2.00	2.70	0.00
impcol_b	59	281	16.42%	**5**	5.00	0.54	0.00	**5**	20	5.00	0.07	0.00
can___61	61	248	13.55%	**4**	4.10	0.27	0.31	**4**	20	4.00	0.11	0.00
dwt___66	66	127	5.92%	**2**	2.00	0.14	0.00	**2**	20	2.00	0.24	0.00
west0067	67	287	12.98%	**5**	5.85	0.60	0.37	**5**	20	5.00	0.38	0.00
can___73	73	152	5.78%	4	4.00	0.31	0.00	**3**	20	3.00	2.23	0.00
steam3	80	424	13.42%	**4**	4.35	0.33	0.49	**4**	12	4.40	63.44	0.50
ash85	85	219	6.13%	**3**	3.00	0.26	0.00	**3**	20	3.00	0.07	0.00
dwt___87	87	227	6.07%	**3**	3.95	0.46	0.22	**3**	20	3.00	1.52	0.00
can___96	96	336	7.37%	4	4.00	0.38	0.00	**3**	20	3.00	22.29	0.00
gent113	113	549	8.68%	**5**	5.75	288.04	0.79	**5**	1	5.95	42.96	0.22
arc130	130	715	8.53%	10	10.00	3.40	0.00	**9**	3	9.85	132.41	0.37
lund_b	147	1147	10.69%	**5**	5.25	2.33	0.44	**5**	19	5.05	95.01	0.22
lund_a	147	1151	10.73%	**5**	5.10	2.39	0.31	**5**	20	5.00	56.26	0.00
mcca	180	1680	10.43%	**8**	8.00	15.93	0.00	**8**	20	8.00	0.64	0.00
dwt__193	193	1650	8.91%	**6**	6.90	5.04	0.31	**6**	20	6.00	33.01	0.00
steam1	240	1761	6.14%	**6**	6.20	4.62	0.41	**6**	16	6.20	139.62	0.41
wm2	260	2375	7.05%	**10**	11.00	257.20	0.86	**10**	20	10.00	5.10	0.00
wm3	260	2379	7.07%	**10**	10.95	246.29	0.76	**10**	20	10.00	18.79	0.00
wm1	277	2389	6.25%	**10**	10.85	240.51	0.81	**10**	20	10.00	42.28	0.00
mbeause	492	36209	29.98%	**21**	22.15	269.87	0.81	**21**	4	21.80	200.82	0.41
mbeacxc	492	41686	34.51%	**22**	23.30	208.99	0.80	**22**	20	22.00	167.71	0.00
mbeaflw	492	41686	34.51%	**22**	23.20	274.51	0.77	**22**	20	22.00	168.21	0.00
beacxc	497	42175	34.22%	**22**	23.10	266.54	0.85	**22**	14	22.30	195.21	0.47
beause	507	39427	30.74%	**22**	23.30	418.65	0.80	**22**	20	22.00	191.91	0.00
beaflw	507	44899	35.00%	**22**	22.95	402.34	0.83	**22**	5	23.00	358.29	0.73
mcfe	765	15358	5.26%	**14**	14.90	411.32	0.85	**14**	3	14.85	458.08	0.37
#Best				32	16			36		33		
Average			18.19%	7.89	8.33	92.28	0.34	7.78	17.14	7.93	66.65	0.10