A fast path relinking algorithm for the min–max edge crossing problem

Bo Peng[a], Lunwen Wu[a], Rafael Martí[b,*], Jiangshui Ma[a]

[a]*School of Business Administration, Southwestern University of Finance and Economics, Chengdu 610074, P.R. China*
[b]*Departamento de Estadística e Investigación Operativa, Universidad de Valencia, Spain*

Abstract

The min–max edge crossing problem (MMECP) is a challenging and important problem arising in integrated-circuit design, information visualization, and software engineering. Drawing edges as straight lines in accordance with the hierarchical graph drawing standard, the goal is to reduce the maximum number of edge crossings in graphs. In this study, we propose a fast path relinking (FPR) method based on dynamic-programming local search to tackle the MMECP, where an efficient neighborhood reduction mechanism is employed to evaluate only the so-called critical vertices instead of all the vertices. Moreover, the proposed FPR can simultaneously manage a number of neighborhood moves at each search iteration, which is significantly different from all the previous approaches based on one neighborhood in the literature. Extensive computational experiments on MMECP instances show that our proposed FPR approach is relatively competitive compared to the best-performing heuristics and the optimization Gurobi solver. In particular, our algorithm improved the best-known solutions for 104 of the 301 publicly available benchmark instances. Additional experiments were conducted to

*Corresponding author.
Email addresses: pengbo@swufe.edu.cn (Bo Peng), wulunwen@swufe.edu.cn (Lunwen Wu), rafael.marti@uv.es (Rafael Martí)

elucidate the key elements and search parameters of the proposed FPR. Furthermore, we made the source code of the algorithm publicly available to facilitate its use in real applications and future research.

Keywords: Metaheuristics; min–max edge crossing; graph drawing; dynamic programming; path relinking.

1. Introduction

The rise of big data has created a demand for effective data visualization techniques to analyze and present complex networks and knowledge-based systems. Graphs have now become a widely adopted model for representing this information, and are considered as a standard approach for facilitating data analysis and presentation. The crossing minimization problem in graphs has attracted considerable attention owing to its applicability and complexity. Particularly, the problem in layered graphs received considerable attention after the seminal work of Sugiyama et al. (1981). The authors developed a framework suggesting that any directed acyclic graph can be transformed into a layered structure, and many of the methods developed for layered graphs are relevant to more general graphs. The crossing minimization problem basically entails ordering the vertices in each layer (where they are arranged) to minimize the total number of intersections or crossings between edges in a graph layout.

Research on the problem in bipartite graphs has been ongoing for over 40 years, beginning with the introduction of the relative degree algorithm in Carpano's work in 1980 (Carpano, 1980). Initially, simple ordering principles were used as the basis for heuristics to quickly find solutions of acceptable quality, as both academics and practitioners sought practical solutions. However, recent advances in optimization have led to the development of more complex methods in both exact and heuristic domains (Martí, 1998; Battista et al., 1998; Chimani et al., 2011; Peng et al., 2020a; Zehavi, 2022).

Sugiyama's method (1981) represents graphs with the layered standard, and permits the application of crossing minimization algorithms to any given graph to improve its readability. This method first assigns vertices to layers (arranging nodes on parallel lines). Then, it orders the vertices in each layer for edge-crossing minimization. Finally, the method assigns vertices to specific locations within their respective layers to reduce the length and curvature of long edges. This limits the overall length of edges, resulting in a more compact and visually appealing graph representation.

In this research, we focus on an important application problem in VLSI circuit design (Bhatt and Leighton, 1984). The presence of a large number of wire crossings is an unfavorable aspect of the VLSI layout. More specifically, when all crossing wires carry the same signal concurrently, the wires crossed by many other wires are easily affected by crosstalk, which compromises the circuit performance. In contrast, when a network has a low number of wire crossings, it also requires fewer contact cuts, resulting in better signal quality. To ensure optimal performance, it is essential to minimize the number of crossings for each edge, rather than solely focusing on reducing the overall number of crossings. This application inspired a previous study (Stallmann, 2012) that proposed a heuristic method for minimizing the maximum number of intersections between edges. In addition, the graph drawing problem has a wide variety of applications in a lot of fields, e.g., network management (Kriegel et al., 2008), decision diagrams (Mateescu et al., 2008), bioinformatics (Lemons et al., 2011), software engineering (Burch et al., 2012), and database modeling (Hu et al., 2016).

Population-based metaheuristic methods (such as evolutionary or memetic algorithms) typically handle a large number of high-quality individuals to diversify the search process. Although these methods can generate high-quality solutions to address various optimization problems, they are often time-consuming, requiring intricate mechanisms to manage a large number

of individuals. On the other hand, one-solution-based search algorithms, such as local search, have a simple structure but converge too fast, making it easy to fall into local optima. To address these limitations, two-solution meta-heuristic algorithms have gained attention in recent years, demonstrating competitive performance in several classic combinatorial optimization problems such as graph coloring (Moalic and Gondran, 2018), flexible job shop scheduling (Ding et al., 2019), and satellite broadcast scheduling (Peng et al., 2020b).

In this study, we explore a fast path-relinking (FPR) method based on two solutions for the min–max edge crossing problem (MMECP). Path relinking was originally proposed as a search strategy in the tabu search methodology (Glover et al., 2021), and was adapted to the GRASP methodology by Laguna and Martí (1999) to solve a graph drawing problem. In their implementation, solutions generated via the greedy randomized adaptive search procedure (GRASP) were relayed to an improvement system by employing path relinking from these solutions to the elite solutions identified so far. Notably, the novelty of our proposal lies in the number of individuals that achieve the crossover or relink, and thus it can be a guide to research based on the two-individual method for permutation problems on graphs. Furthermore, in the traditional search procedure (such as classic local search or tabu search), we usually select only one neighborhood move to operate. In this study, taking into account the independence of moves between layers, we propose a dynamic programming mechanism embedded into the local search phase to accelerate the search process by selecting several neighborhood moves simultaneously. We will show that the complete algorithm achieves a reasonable trade-off between diversification and intensification, as demonstrated in our extensive experiments.

The main contributions of this study include:

- A fast path-relinking method to tackle the graph drawing problem by

managing only two individuals to achieve a better balance between solution quality and search efficiency.

- A dynamic programming–based local search to simultaneously manage several neighborhood moves in different layers at each iteration, which is significantly different from all the previous approaches based on one neighborhood in the literature.

- A neighborhood reduction strategy that only evaluates the critical vertices instead of all the vertices in the neighborhood to enhance the search process. We present a mathematical proposition to justify the proposed neighborhood structure.

- Extensive experimentation on public benchmark instances to compare the proposed method with the best-performing heuristics and the optimization solver, Gurobi. In particular, the FPR algorithm improved the best-known solutions for 104 of the 301 benchmark problem instances.

It should be noted that this study transcends the application of a solving method to a specific problem. The search strategies and optimization methods in the proposed fast path relinking and dynamic-programming local search are quite general and can provide new references for other graph drawing and permutation problems.

The remainder of this paper is organized as follows. Section 2 presents the MMECP problem and previous methods. Section 3 describes the FPR method. Section 4 presents the computational results and comparisons with state-of-the-art algorithms in literature. The effectiveness of several important components of the proposed method is discussed in Section 5, and concluding observations are summarized in Section 6.

2. Problem Description and Previous Methods

Crossing minimization is a well-known problem in graph drawing, as it takes place widely in various fields, such as VLSI circuit design (Bhatt and Leighton, 1984), information visualization (Herman et al., 2000), and software engineering (Latoza and Myers, 2011). In this study, we focus on an important variant, the min–max edge crossing problem (MMECP) proposed by Stallmann (2012), which seeks to minimize the maximum number of edge crossings in a hierarchical graph.

To describe the MMECP more precisely, we first denote a hierarchical graph as $G = (V,E,L,K)$, where V, E, and K are, respectively, the set of vertices, edges, and the number of layers. For a hierarchical graph, the vertex set V is partitioned into mutually disjoint subsets, where each subset represents one layer. Edges only exist between two consecutive vertex subsets, which also means that there are no edges that connect vertices in nonadjacent layers. In addition, the function $L(u)$ denotes the index of the layer where each vertex $u \in V$ resides by a mapping $V \to \{1, \ldots, K\}$. A configuration (drawing) of G is defined as $S = (\omega_1, \omega_2, ..., \omega_K)$. Each ω_k denotes the permutation of vertex set in the kth layer. Let $\psi(v)$ be the function that describes the position of vertex v from its residing layer such that if $v = \omega_k[j]$, then $\psi(v) = j$. Subsequently, an edge crossing is generated between edges (u, v) and (p, q), where vertices u and p are located in layer k, and vertices v and q are located in layer $k + 1$, precisely when the symbol c_{uvpq} takes a value of 1, as shown below:

$$c_{uvpq} = \begin{cases} 1, & \text{if}((\psi(u) < \psi(p)) \wedge (\psi(v) > \psi(q)) \;\; or \;\; (\psi(u) > \psi(p)) \wedge (\psi(v) < \psi(q))); \\ 0, & \text{otherwise}; \end{cases} \tag{1}$$

where vertices u, $p \in V_k$ (i.e., the vertex set in kth layer) and v, $q \in V_{k+1}$ (i.e., the vertex set in $(k + 1)$th layer) for the layers $k = 1, \ldots, K - 1$.

In addition, we denote by $C(e)$ the number of edges in E that cross with edge e. Then, the objective of MMECP is to find a drawing S such that $f(S) = \max\{C(e) : e \in E\}$ is minimized. Figure 1 shows two drawings

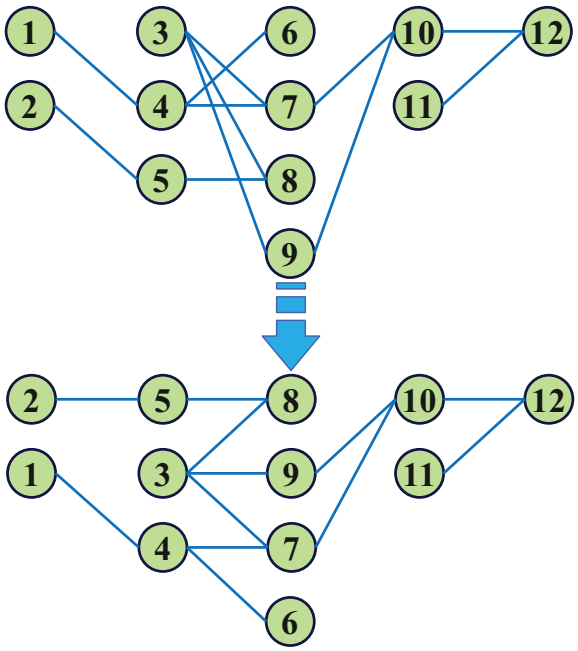


Fig. 1. Initial (top) and optimal (bottom) drawing of a Rome graph.

(i.e., solutions) of a five-layered graph for the MMECP (where the layers are arranged as parallel vertical lines or columns and the nodes are located in these columns). The maximum number of edge crossings in the first drawing (top) in Figure 1 is equal to 3, since edges (3,9) and (4,6) have this maximum number of crossings. The second drawing (bottom) in Figure 1 is optimized from the first drawing by relocating nodes to suitable positions, resulting in 0 edge crossings. To emphasize the significance of MMECP, we present an example in Figure 2 where the min-max objective is improved as the total sum objective increases. Specifically, the maximum number of edge crossings in the left drawing is 2, compared to 1 in the right drawing. In terms of the

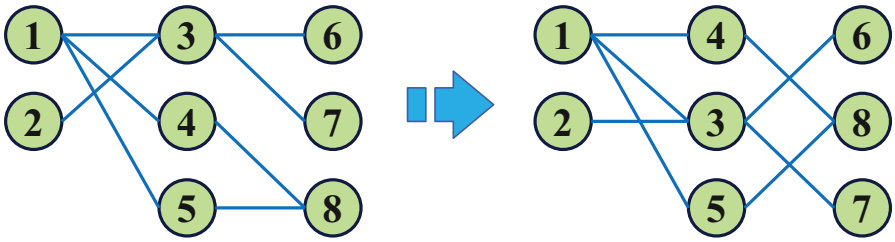


Fig. 2. Two drawings of a graph.

sum of edge crossings, the left drawing has 2, while the right drawing has 3.

As mentioned in the Introduction, the MMECP was first introduced by Stallmann (2012), who proposed a heuristic called the maximum-crossings edge (MCE) to solve this graph crossing problem. The MCE heuristic is based on the shifting heuristic presented by Matuszewski et al. (1999), with results generally superior to those of other heuristics based on the so-called barycenter method.

Martí et al. (2018) investigated the adaptation of the greedy randomized adaptive search procedure (GRASP) and strategic oscillation (SO) methods for solving this problem. In particular, the authors focused on the effect of the balance between randomization and greediness by employing a multi-start heuristic search method to solve this problem.

Pastore et al. (2020) proposed a tabu search method by implementing two memory structures, short-term and long-term, in the search process. The reported computational results demonstrated the excellent performance of their proposed method, and also revealed that CPLEX can only solve instances with small size and low density.

Recently, Wu et al. (2021) presented a variable depth neighborhood search algorithm to address the MMECP. Their method takes advantage of an efficient neighborhood search strategy based on a so-called ejection chain scheme to produce superior outcomes.

Despite these recent research developments on this problem mentioned above, the optimal solution has not yet been found for most instances of this difficult NP-hard optimization problem. To obtain high-quality solutions in the short computational times required by graph drawing systems, we have to apply complex metaheuristics. Currently, this problem is a challenge to test solving methodologies, which motivated this study to explore whether path relinking implemented as a two-solution search method can compete with these previous proposals within short time horizons.

3. Path relinking method

Path relinking (PR) is a technique proposed to essentially combine intensification and diversification in the search process (Glover et al., 2021). This method finds new solutions by following paths that connect high-quality solutions in the search space, starting from one solution and moving towards another one. To this end, elements or attributes in the final solution are added to the initial solution to obtain an intermediate solution in the path.

Laguna and Martí (1999) adapted PR to GRASP, opening its application to any other method. In this adaptation, relinking involves creating a path between two GRASP solutions.

3.1. General scheme

The proposed FPR algorithm is a hybrid method that integrates iterated local search and path relinking. It utilizes a dynamic-programming local search to intensify the search within a certain region of the solution space and then applies an adaptive perturbation mechanism and relinking operator to transition to a new search region upon reaching a local optimum.

Specifically, FPR consists of three major components: the $InitialSolution$ phase to produce a random initial solution, the local search based on dynamic programming ($DPLS$) to optimize the current solution, and the *Relinking*

operator to generate a combined solution (sometimes referred to as offspring in the literature on evolutionary methods).

Algorithm 1: The fast path-relinking method for MMECP

1: **Input**: Graph G; Large number of iterations li;
2: **Output**: The best solution S_{best} found so far
3: $S_1, S_2 \leftarrow InitialSolution(G)$, $S_{best} \leftarrow SaveBest(S_1, S_2)$
4: **while** the maximum computing time T_{max} is not reached **do**
5: $S_1^p \leftarrow S_1$, $S_2^p \leftarrow S_2$
6: $S_1^c \leftarrow Relinking(S_1, S_2)$, $S_2^c \leftarrow Relinking(S_2, S_{best})$
7: $S_1^c \leftarrow DPLS(S_1^c, li)$, $S_2^c \leftarrow DPLS(S_2^c, li)$
8: $S_{best} \leftarrow SaveBest(S_1^c, S_2^c, S_{best})$
9: **if** S_1^c is close to S_2^c or (S_1^c and S_2^c are close to S_1^p and S_2^p, respectively) **then**
10: $S_1 \leftarrow S_{best}$, $S_2 \leftarrow InitialSolution(G)$
11: **else**
12: $S_1 \leftarrow S_1^c$, $S_2 \leftarrow S_2^c$
13: **end if**
14: **end while**
15: return S_{best}

A pictorial representation of one cycle of the algorithmic framework is given in Fig. 3, followed by a formal description of the algorithm in Algorithm 1. Starting from two initial solutions S_1 and S_2 generated by the procedure *InitialSolution*, with the best solution of S_1 and S_2 saved as S_{best} (line 3 in Algorithm 1), the algorithm then alternates between two main procedures, $DPLS$ and *Relinking*, until a pre-specified value T_{max} on the computing time has elapsed. In each generation of the iterative procedure, S_1^p and S_2^p first save the values of two individuals to later evaluate whether the search process can continue to improve the incumbent solutions.

Subsequently, two offspring solutions S_1^c and S_2^c are generated by the dedicated *Relinking* procedure. More precisely, the first *Relinking* part connects the two incumbent solutions S_1 and S_2 to generate a new solution S_1^c, whereas the second *Relinking* part connects S_2 and the best found solution S_{best} (line 6). Next, FPR applies $DPLS$ with a large number of iterations (li) to attempt to improve the two generated solutions S_1^c and S_2^c (line 7). The

best solution in the current iteration S_{best} is updated if S_1^c or S_2^c is better than S_{best} (line 8). As soon as S_1^c is 'close' to S_2^c (i.e., the maximum crossing

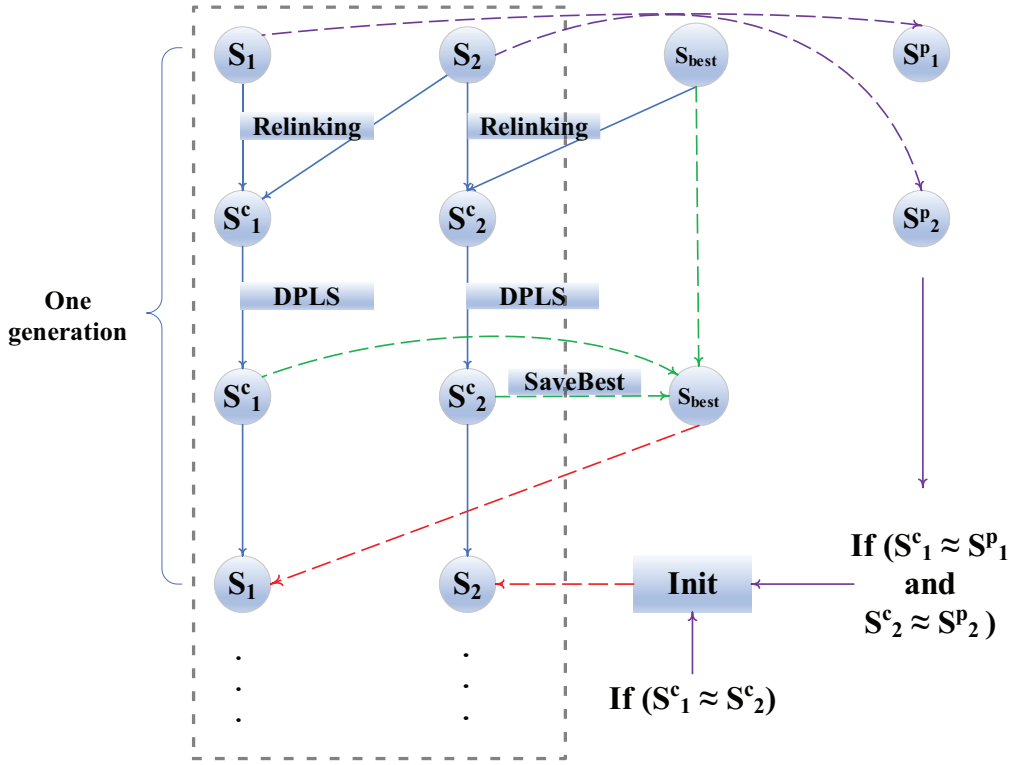


Fig. 3. Diagram of FPR.

value of S_1^c is equal to that of S_2^c) or S_1^c and S_2^c are close to the previously saved solutions S_1^p and S_2^p, respectively, both S_1 and S_2 are replaced by the best found solution S_{best} and a new random initial solution generated by $InitialSolution$, respectively, to ensure the intensification and diversification of the search (lines 9−10). Note that comparing the structures of two solutions to define their similarity is ideal but time-consuming. Therefore, here we adopt an approximate approach based on the objective function values to assess the similarity of solutions. In some special cases, specifically when there are two different solutions with the same objective function values, the

reconstruction process will also replace them (line 10). Finally, the best solution, S_{best}, is returned (line 15). The components of the proposed FPR method for the MMECP are described in the following subsections.

3.2. Solution space and evaluation function

Since the MMECP is a permutation problem, any m-layer permutation of n vertices is a feasible solution. Thus, the search space to be explored using the proposed method can be defined as follows:

$$\Omega = \{(\omega_1, \omega_2, \ldots, \omega_K) : |\omega_i| = n_i, 1 \leq i \leq K\}, \tag{2}$$

where ω_i and $|\omega_i|$ denote the permutation and the number of vertices in layer i, respectively. Thus, the size of the search space is equal to $\prod_{i=1}^{K} n_i!$, where n_i denotes the number of vertices in layer i. Given the min–max objective function, there are many solutions with the same objective value. As is well documented in heuristic-optimization literature, the objective value hardly represents the significant difference between different solutions with the same value (i.e., the maximum number of edge crossings); therefore, it is not a good indicator to guide the search. To distinguish solutions with the same objective value, Pastore et al. (2020) employed a δ-evaluation function based on a complex calculation expression. Because their strategy produced a very effective technique of exploring the search space, we now propose an alternative expression based on a relatively simple function to evaluate the solution S:

$$h(S) = f(S) * |V| + mcn(S), \tag{3}$$

where $f(S)$ denotes the objective value, i.e., the maximum crossing number of the edges for solution S in the graph, and $mcn(S)$ denotes the number of maximum-crossing edges (i.e., the edges with the maximum crossing number) for solution S in the graph. Therefore, we can distinguish numerous

similar valued solutions, i.e., those with the same objective function but different solution structures, by combining the number of maximum crossing edges with the original objective values (i.e., the maximum number of edge crossings) as the evaluation function in Formula 3.

3.3. Initial solution process

In this study, the proposed FPR algorithm produces an initial solution by randomly generating a permutation of the vertices in each layer. This random solution is then improved by the dynamic programming–based local search.

3.4. Dynamic programming–based local search phase

The local search procedure based on dynamic programming expedites the search process by selecting several neighborhood moves simultaneously at each iteration.

This procedure is presented in Algorithm 2, which can be described as follows: It first initializes variables, i.e., the best found solution S^*, the previous solution S_p, the perturbation strength ζ, and the number of iterations θ (lines 1–4). Next, the dynamic programming–based local search mechanism and the adaptive perturbation mechanism iteratively alternate until the stopping criterion is met (i.e., the maximum number of iterations θ reaches the maximum threshold Θ) (lines 5–23). Specifically, several iterations are first performed to improve the current solution S by selecting a set of neighborhood moves according to the dynamic-programming mechanism until there are no better neighboring solutions (lines 6–11). At each iteration, the algorithm first constructs the critical-edge set $E_c(S)$, the critical-vertex set $V_c(S)$, and the critical vertex–based neighborhood move set $M_c(S)$ for the incumbent solution S according to Equations (5–7). Then, the algorithm improves the current solution based on the chosen neighborhood move set NMS in different layers according to the dynamic-programming mechanism

described in Equations (8–11). Afterwards, the algorithm attempts to evaluate the search state to update the best solution S^*, the number of iterations θ, and the perturbation strength ζ. If the search escapes the previous local optimum, the algorithm decreases the perturbation strength (lines 16–17). Otherwise, the algorithm increases the perturbation strength (lines 18–20). After determining the perturbation strength, the algorithm uses the perturbation operator to generate a new solution in a distinct region of the search space while preserving the previous solution using the symbol S_p (lines 21–22). Finally, the algorithm returns the best solution S^* as the final output (line 24). The specifics of the dynamic programming–based local search are provided in the following subsections.

3.4.1. Neighborhood move set

The approach proposed by Martí et al. (2018) for moving between solutions in the search space of MMECP involves considering nearby positions of the barycenter of the selected vertex, whereas the tabu search method developed by Pastore et al. (2020) evaluates non-tabu vertices in the layer that have an edge with a crossing count exceeding or equal to a predefined threshold. In addition, the recent best-performing method proposed by Wu et al. (2021) employed a variable depth neighborhood search algorithm based on swap moves. Our FPR method employs only the insertion move based on critical vertices, which are the endpoints of the maximum crossing edges and the edges crossing them. Specifically, the $Insert(v, u)$ move can be defined by removing vertex v from its current position in the solution S and inserting it at the previous position of vertex u if it precedes v, or in the posterior position of u if v precedes u. The neighborhood move creates a total of n_k - 1 ($1 \leq k \leq K$) possible candidate positions (i.e., solutions) for each vertex. Figure 4 presents an example in which the $Insert(v_{1,1}, v_{1,2})$ move re-inserts vertex $v_{1,1}$ below $v_{1,2}$.

Algorithm 2: The dynamic programming–based local search procedure

Input: Current solution S; Maximum number of iterations (Θ)

Output: Best found solution (S^*) in current procedure

$S^* \leftarrow S$;

$S_p \leftarrow S$;

$\zeta \leftarrow \zeta_{min}$ /* ζ records the perturbation strength */ ;

$\theta \leftarrow 0$ /* θ denotes the number of iterations */ ;

while *The maximum number of iterations* Θ *is not reached, i.e.,* $\theta < \Theta$ **do**

    // Dynamical programming based local search procedure

    repeat

        $S^{'} \leftarrow S$;

        $[E_c(S), V_c(S), M_c(S)] \leftarrow ConstructNeighborhoodTriple(S)$ /* Equations (5–7)*/;

        $NMS(S) \leftarrow DynamicProgrammingSelectMoveSet(S, M_c(S))$ /* Equations (8–10)*/ ;

        $S \leftarrow S \oplus NMS(S)$ /* Equation 11*/ ;

    until *The generated solution* S *is not better than the saved solution* $S^{'}$ *(i.e.,* $h(S) >= h(S^{'})$*)*;

    // Update the best solution S^* found in current local search phase, re-initialize perturbation strength if the new best solution is found

    if S *is better than* S^* **then**

        $S^* \leftarrow S$;

        $\zeta \leftarrow \zeta_{min}$;

    end

    // Dynamically determine the perturbation strength ζ

    if S *is better than* S_p **then**

        // Search escaped from the previous local optimum, decrement perturbation strength

        $\zeta \leftarrow Max(\zeta/2, \zeta_{min})$;

    else if S *is not better than* S_p *and* $\zeta < \zeta_{max}$ **then**

        // Search returned to the previous local optimum, increment perturbation strength

        $\zeta \leftarrow Min(\zeta * 2, \zeta_{max})$;

    end

    $S_p \leftarrow S$, $\theta \leftarrow \theta + 1$;

    // Adaptive Perturbation Procedure

    $S \leftarrow AdaptivePerturbation(S^*, \zeta)$;

end

return S^*

To explore the solution space more efficiently, we propose a neighborhood reduction strategy that evaluates only the so-called critical vertices, defined as follows. Specifically, we first compute the set of edges with the maximum crossing number E_m, denoted by

$$E_m = \{(i_o, j_o) \in E : C(i_o, j_o) = \max_{(i,j)\in E} \{C(i,j)\}. \tag{4}$$

Candidate list strategies have been extensively used in local search methods, such as tabu search, to reduce the size of the neighborhood (Glover et al., 2021). It is usually more efficient to define a large neighborhood and then reduce it to explore only the most promising solutions, as opposed to directly defining a small neighborhood.

Definition 1 (Critical-edge set E_c). The critical edges include the maximum crossing edges and the edges crossing these in the graph. These edges are called critical edges because they determine the value of the objective function in the graph, which can be written as

$$E_c = E_m \cup \{(i_o, j_o) \in E : c_{i_o j_o i_1 j_1} = 1 \text{ or } c_{i_1 j_1 i_o j_o} = 1; (i_1, j_1) \in E_m\}. \tag{5}$$

Definition 2 (Critical-vertex set V_c). We call the vertex incident with the critical edges the critical vertex, which is given as follows:

$$V_c = \{i_o \in V : (i_o, j) \in E_c; j \in V\}. \tag{6}$$

To efficiently explore the solution space, we propose a neighborhood reduction strategy based on the critical vertices according to the following proposition:

Proposition 1. *If an insertion move based on any two vertices (e.g., $Insert(u, v)$) can improve the current solution (e.g., S) in terms of the evaluative function ($h(S)$) or objective value ($f(S)$), then at least one of these two vertices (u and v) is a critical vertex.*

Proof. 1. Without loss of generality, we assume that a basic insertion move based on two vertices can improve the current solution by reducing the crossing number of at least one maximum crossing edge (e.g., (p_1, q_1)). Here, we call these two vertices for the insertion move as insertion vertices. Then, there exists at least an edge (e.g., (p_2, q_2)) previously crossing with edge (p_1, q_1) that becomes non-crossing after the insertion move, thereby reducing the crossing value. In other words, $c_{p_1q_1p_2q_2} = 1$ becomes $c_{p_1q_1p_2q_2} = 0$ after the insertion move.

2. The relative positions of vertices p_1 and p_2 (or vertices q_1 and q_2) in the same layer have to change since the previous crossing edge (p_2, q_2) becomes non-crossing to (p_1, q_1) after the insertion move. In other words, the formula $(\psi(p_1) - \psi(p_2)) * (\psi^{'}(p_1) - \psi^{'}(p_2)) < 0$ or $(\psi(q_1) - \psi(q_2)) * (\psi^{'}(q_1) - \psi^{'}(q_2)) < 0$ holds, where the function $\psi^{'}(v)$ denotes the new position of vertex v after the insertion move. Then, at least one of vertices p_1 and p_2 (or vertices q_1 and q_2) is an insertion vertex.

3. Owing to the maximum crossing edge (p_1, q_1), all these four vertices (i.e., p_1, q_1, p_2, and q_2) are critical vertices according to Definition 2. Thus, at least one of the critical vertices is an insertion vertex according to the second point of this proof above, which also means that, at least one of insertion vertices is a critical vertex. Clearly, the original proposition is true.

□

Definition 3 (Critical vertex–based neighborhood move M_c). The proposed critical vertex–based neighborhood move considers only the first better solutions with respect to the current solution after moving the critical vertices upwards (or downwards) to the near position by satisfying the first

improvement rule, which can be expressed as follows:

$$M_c(u) = \{Insert(u,v) : h(S^{'}) < h(S); \quad S^{'} = S \oplus Insert(u,v); \\ u \in V_c(S), v = \arg \min_{L(v)=L(u)} (|\psi(u) - \psi(v)|)\}. \tag{7}$$

Here, $V_c(S)$ denotes all the critical vertices of solution S, and $\psi(u)$ and $\psi(u)$ represent the positions of vertices u and v, respectively. The operator $\oplus$ represents the application of the move operator to the current solution, resulting in a new solution. For instance, $S \oplus Insert(u,v)$ implies that a vertex u is selected and inserted at the position of vertex v in solution S.

Clearly, at most, one neighborhood move should be chosen for each critical vertex (e.g., $M_c(u)$), while there exist several critical vertex–based neighborhood moves for one solution (e.g., $M_c(S)$). It is worth noting that the neighborhood moves on vertices in the first layer will not affect the neighborhood moves on vertices in nonadjacent layers (e.g., the third layer). Thus, there exist some independent neighborhood moves in different layers, which means that we can select a set of these independent moves to execute in one iteration to shorten the improvement process, which can be presented as follows:

Definition 4 (Neighborhood move set NMS in different layers).

$$NMS = \{M_c(u_x) : x \in \{1, \ldots, K\}; \forall 1 \leq i < j \leq K, |L(u_i) - L(u_j)| > 1\}, \tag{8}$$

where vertices u_1, u_2, $\ldots$, u_x ($x \leq K$) are critical vertices located in different layers.

To identify a good set of moves in different layers in one iteration for MMECP, we apply a dynamic-programming mechanism inspired by the dynasearch method for the single scheduling problem (Congram et al., 2002). We first denote $\delta(i)$ as the minimum incremental (or changing) value of the evaluation function (i.e., $h(S)$) after one neighborhood move for all critical

vertices in layer i or 0 if all the moves in layer i cannot improve the current solution. The symbol $\delta(i)$ can be defined according to the following equation:

$$\delta(i) = \min_{\forall u \in V_c, L(u)=i} \{(h(S \oplus M_c(u)) - h(S)), 0\}. \quad (9)$$

Let us evaluate the example in Figure 4, where 15 vertices are located in five

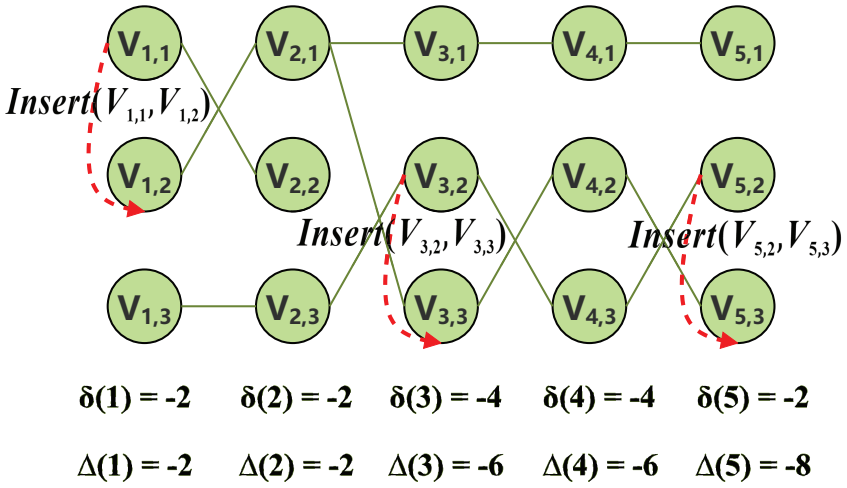


Fig. 4. An example with multiple moves (i.e., Insert$(v_{1,1}, v_{1,2})$, Insert$(v_{3,2}, v_{3,3})$, and Insert$(v_{5,2}, v_{5,3})$).

layers. The objective function of the current solution is equal to 1 because there are eight edges with one crossing. As shown in Figure 4, δ (1) takes the value of -2, which denotes the minimum incremental value of h based on the move Insert$(v_{1,1}, v_{1,2})$ in the first layer, since the number of maximum crossing edges can decrease the value in two units. Similarly, δ (3) takes the value of -4, which denotes the minimum incremental value of the evaluation function based on the move Insert$(v_{3,2}, v_{3,3})$ in layer 3, since the number of maximum crossing edges can decrease the value of 4.

Evidently, the δ value evaluates only the best neighborhood move within a layer. To evaluate the move gain based on neighborhood moves among

different layers, we define $\Delta(i)$ as the minimum incremental (or changing) value of the evaluation function (i.e., $h(S)$) based on the multiple moves between layers 1 and i.

As presented in Figure 4, $\Delta(3)$ denotes the minimum incremental evaluation-function value (i.e., -6) after applying multiple moves (i.e., Insert$(v_{1,1}, v_{1,2})$, Insert$(v_{3,2}, v_{3,3})$) from layer 1 to layer 3, since the evaluation function (i.e., $h(S)$) increases -6 or decreases 6, which means that six previous crossing edges (i.e., $(v_{1,1}, v_{2,2})$, $(v_{1,2}, v_{2,1})$,$(v_{2,1}, v_{3,3})$, $(v_{2,3}, v_{3,2})$,$(v_{3,2}, v_{4,3})$, and $(v_{3,3}, v_{4,2})$) become non-crossing after performing the multiple moves from layer 1 to layer 3.

For the border case, $\Delta(1)$ denotes the minimum incremental value of the evaluation function after conducting the neighborhood move in the first layer, which is clearly equal to $\delta(1)$. As shown in Figure 4, $\Delta(1)$ is equal to -2 since the value of the evaluation function would decrease 2 when the move Insert$(v_{1,1}, v_{1,2})$ is performed in the first layer. The symbol $\Delta(K)$ (e.g., $\Delta(5)$ in Figure 4) denotes the incremental value of the evaluation function of the best multiple moves for all the layers. Since the neighborhood moves in nonadjacent layers are independent, we can calculate $\Delta(i)$ using the following equation:

$$\Delta(i) = \min\{\Delta(i-1), \Delta(i-2) + \delta(i)\}, \tag{10}$$

which implies that the best incremental value of the evaluation function from layer 1 to layer i is equal to either that from layer 1 to layer $i-1$, or that from layer 1 to layer $i-2$ plus the minimum incremental evaluation-function value $\delta(i)$ in layer i. In the border case, $\Delta(0)$ is equal to zero. The time complexity for computing the move gain $\delta(i)$ of a layer is $O(n_i^2 * m^2)$, where n_i denotes the number of vertices in the ith layer, and m denotes the average degree of one vertex in graph, since the neighborhood moves should be evaluated based on two vertices in the same layer. Thus, the time complexity required

to compute the move gain $\Delta(K)$ of multiple layers is $O(\sum_{i=1}^{K} n_i^2 * m^2)$ because the Δ value must traverse all the layers.

Let us review the complete process presented in Figure 4. First, we can obtain the values of δ in five layers, where $\delta(1)$ to $\delta(5)$ are equal to -2, -2, -4, -4, and -2, respectively. According to Equation 10, we can then easily obtain the value of $\Delta(5)$ equal to -8, which means that we can choose vertices $v_{1,1}$, $v_{3,2}$, and $v_{5,2}$ to move downwards together after the neighborhood evaluation; the number of maximum crossing edges can decrease the value of 8. However, if we applied the traditional evaluation strategy reported in the literature, we would typically select the best neighborhood move to operate. There are two best neighborhood moves (i.e., Insert($v_{3,2}$, $v_{3,3}$) and Insert($v_{4,2}$, $v_{4,3}$)); in this case, it is difficult to justify which one is better. In particular, the search falls into the local-optimum trap if we select the latter neighborhood move to insert vertex $v_{4,2}$ to $v_{4,3}$ downwards. Hence, this example demonstrates the proposed dynamic-programming mechanism as well as the advantage of the proposed strategy in neighborhood search procedure.

Therefore, based on the proposed dynamic-programming mechanism, we can select the set of multiple neighborhood moves after each neighborhood evaluation. Finally, the evaluation value of the new solution after the neighborhood moves can be expressed as follows:

$$h(S \oplus NMS(S)) = h(S) + \Delta(K). \tag{11}$$

Note that the evaluation value of the new solution after the moves is recalculated if the number of maximum crossing edges (i.e., mcn) decreases to 0.

3.5. Adaptive perturbation mechanism

The approach used in this study involves an adaptive perturbation mechanism that adjusts the intensity of perturbations according to the state of

Algorithm 3: Adaptive perturbation procedure

Input: Current solution S, perturbation strength ζ depending on the state of search in Algorithm 2

Output: New solution S_c obtained from perturbation

1 $S_c \leftarrow S$;

// Destruction part

2 **for** $i \in \{1, \ldots, \zeta\}$ **do**

3 | Randomly choose a vertex u_i of solution S_c, delete it from the incumbent solution S_c;

4 **end**

// Reconstruction part

5 **for** $i \in \{1, \ldots, \zeta\}$ **do**

6 | Randomly re-insert the removed vertex u_i into the incumbent solution S_c ;

7 **end**

8 **return** S_c

the search. This allows the proposed algorithm to escape local optima and explore new regions of the solution space.

As presented in Algorithm 2, the adaptive method determines the perturbation strength ζ, which here is equal to the number of chosen vertices. If the current solution S is better than the previous solution S_p, the perturbation strength is decreased by decreasing the value of $\zeta/2$. Meanwhile, if the current solution S is not better than the previous solution S_p, the perturbation strength is increased by increasing the value of ζ*2. In particular, we re-initialize the perturbation strength to the minimum strength ζ_{min} if a new best solution is found. In general, the perturbation procedure presented in Algorithm 3 comprises two parts—i.e., the destruction part (lines 2–4) and reconstruction part (lines 5–7)—by randomly removing selected vertices from the current solution and reinserting them into the incumbent solution. Overall, the strength of the adaptive perturbation phase is determined by the parameter ζ, which means that a higher value of ζ results in a more powerful perturbation.

3.6. The relinking operator

The relinking operator is applied to obtain a high-quality solution by exploring the trajectories connecting two such solutions, namely, the initial solution and the guiding solution. The objective is to find a promising solution from a sequence of newly generated solutions.

Algorithm 4: Relinking operator

Input: Initial solution S^I, Guiding solution S^G
Output: The offspring solution S^O

$S^r \leftarrow S^I$, $S^O \leftarrow S^I$;
for $k \in 1, \ldots, K$ **do**
  Randomly choose a layer that has never been selected to relink, the number of which can be denoted by i;
  Select the permutation ω_i^G of the vertices of layer i in the guiding solution S^G;
  Replace the permutation ω_i^r of the vertices of layer i in the intermediate solution S^r with the previously chosen permutation ω_i^G, (i.e., $S^r \leftarrow \{\omega_1^r, \ldots, \omega_i^G, \ldots, \omega_K^r\}$);
  $S^{r'} \leftarrow S^r$;
  $S^{'} \leftarrow DPLS(S^{r'}, si)$;
  if $h(S^{'}) < h(S^O)$ **then**
    $S^O \leftarrow S^{'}$;
  end
end
return S^O

Our relinking operator is similar to the operator introduced in Napoletano et al. (2019), as each move generated by the relinking operator involves replacing an entire layer of the current solution with a layer of the guiding solution. The operator introduced in Napoletano et al. (2019) greedily selects the best layer, while the proposed operator in this study randomly selects it, introducing randomness and diversification to the search.

Algorithm 4 outlines the steps involved in the relinking procedure proposed. Specifically, we first let S^I be the initiating solution, and S^G be the guiding solution. The proposed operator then iteratively produces intermediate solutions between S^I and S^G using a sequence of layer-based operations

(lines 3–5). More precisely, we iteratively replace the permutation of one randomly chosen layer in an intermediate solution with the permutation of the corresponding layer in the guiding solution. To obtain a local optimum after obtaining an intermediate solution (as described in Algorithm 1), a weak local optimization method can be used (e.g., DPLS with a small number of iterations, si, as outlined in lines 6–7). It is worth noting that this approach differs from the strong local optimization method (e.g., DPLS with a large number of iterations, li) used in Algorithm 1. The reason is that the weak one tends to find a promising solution area with a moderate computational effort, whereas the strong one aims to obtain better local optima. Note that the intermediate solution S^r should not be modified inside DPLS in line 7; accordingly, we must use an intermediate symbol $S^{r'}$ to connote its value in line 6. Then, in the next iteration of the cycle, the statement in line 5 is executed on the previous S^r. Subsequently, we refresh the offspring solution if the obtained local optimum is better than the saved current offspring solution S^O (lines 8–10). After K iterations, the method terminates and returns the offspring solution S^O (line 12).

4. Computational Results

The performance of the proposed FPR was evaluated through a series of computational experiments, which are reported in this section. The experiments were designed to comprehensively assess the effectiveness of the proposed method.

4.1. Benchmark instances and experimental protocols

Consistent with earlier studies (Stallmann, 2012; Martí et al., 2018; Pastore et al., 2020; Wu et al., 2021), four instance sets were used in the experiments. The total number of vertices, arcs, layers and instances in each set are presented in Table 1. All these instances were created using Stallmann's generator

(Stallmann, 2012).

Table 1: Attribute of instances in Rome, North, Connect, and Uniform.

Instance set	Vertices	Arcs	Layers	Number
Rome	$[12, 102]$	$[12, 115]$	$[5, 19]$	88
North	$[30, 533]$	$[29, 603]$	$[2, 39]$	58
Connect	1000	$[2000, 5000]$	$[25, 100]$	95
Uniform	$[60, 1000]$	$[587, 9958]$	$[3, 50]$	60

The FPR algorithm was implemented in C++ on a PC running Windows 10 with an Intel Core i5-8300H CPU (2.30GHz) and 8GB RAM. Several reference heuristics and an optimization solver were used to evaluate the performance of FPR through comparisons, as shown below.

- Maximum crossing edge (MCE) heuristic proposed by Stallmann (2012).
- Strategic oscillation (SO) proposed by Martí et al. (2018).
- Tabu search (TS) proposed by Pastore et al. (2020).
- Variable depth neighborhood search (VDNS) proposed by Wu et al. (2021)
- The optimization Gurobi solver based on mathematical programming formulation.

As noted by Wu et al. (2021), the above methods were implemented (or re-implemented) on a PC with a 2.9GHz Intel Core i7 CPU. For a fair comparison, we assumed a linear relationship between CPU speed and frequency, as has been done in related studies (Peng et al., 2020a; Sun et al., 2022; Goudet et al., 2022; Wei and Hao, 2023). Thus, the CPU speed tested on our computer was slower than that in the previous study by Wu et al. (2021) owing to the smaller CPU frequency. Furthermore, we conducted 10 separate iterations of FPR on each problem instance, with a maximum time limit of

60 seconds per iteration, which is in line with the CPU times utilized by the current best-performing methods, specifically VDNS, MCE, and TS, as previously mentioned. Because the executable file of the SO does not implement multiple runs, we set the time limit of a single run to 600 seconds. For Gurobi, the time limit was set to 15 minutes. To facilitate future research and real applications, we made the benchmark instances and the source code of the proposed FPR method publicly available on the github repository [1].

4.2. Parameter tuning

Table 2: Parameter setting in FPR.

Parameter	Description	Candidate values	Final value
si	DPLS with a small number of iterations to select promising solutions in relinking operator	$(5, 10, 20)$	20
li	DPLS with a large number of iterations to improve the incumbent solution	$(20, 40, 60)$	40
ζ_{min}	Minimal perturbation strength	$(1, 2, 4)$	2
ζ_{max}	Maximal perturbation strength	$(K, K*3, \lvert V\rvert/3)$	$K*3$

Table 2 presents the parameter setting for FPR. The parameters si, li, ζ_{min}, and ζ_{max} were set using Iterated F-race (IRACE) (Birattari et al., 2010). Tuning was performed on 12 representative training instances from the four instance sets (i.e., *Rome*, $North$, *Connected*, and $Uniform$). IRACE needs a restricted range of input values for each parameter, chosen from the "Candidate values" column in Table 2. The candidate values of parameters to be selected were established from empirical judgments and extensive experimental tests. IRACE was allocated a total time budget of 100 runs of FPR, and each run was assigned a time limit of 60 seconds per instance. The optimal parameter settings recommended by IRACE are presented in Table 2 and are denoted by the *Final value*.

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

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

In the experiments with the proposed FPR algorithm, we employed 301 benchmark instances including four sets (i.e., *Rome*, $North$, *Connected*, and $Uniform$). Each instance was tested ten times using the four algorithms (MCE, TS, VDNS, and FPR) with a time limit of 60 seconds per run. The SO algorithm was run once on each instance for a total of 600 seconds. Similarly, Gurobi was run once on each instance, with a maximum time limit of 15 minutes. The results of four previous heuristics (i.e., MCE, SO, TS, and VDNS) and Gurobi were obtained from Wu et al. (2021); we ran the proposed FPR with the same run settings in line with the previous studies.

Table 3 reports the summary results of our FPR and the reference algorithms on all four instance sets. We first use f_{best} and f_{avg} to denote the best and average objective-function values of each compared algorithm for each test instance, respectively, which also correspond to the first two columns presented in Tables A1 to A11. In Table 3, the first columns F_{best} and F_{avg} present the average values of f_{best} and f_{avg}, respectively, for each algorithm on the instance set. Column $\#Min$ reports the number of instances for which the minimum objective value among all the algorithms can be obtained. Column DEV provides the average value of *dev* for each instance set.

We can observe from Table 3 that the FPR method achieves the best results among the compared algorithms in each instance set. Only Gurobi can match the results of our FPR, but with significantly longer run times. More precisely, compared with the best-performing algorithm VDNS, FPR can obtain the best solutions for 283 out of 301 instances, whereas VDNS can obtain best results for only 152 instances. In addition, FPR finds better results than VDNS in terms of F_{best} (68.53 vs. 69.73) and F_{avg} (69.68 vs. 71.50). The non-parametric Friedman test ($p - value <$ 2.2e-16) shows that FPR and other reference algorithms are fundamentally different in terms of

best results for all benchmark instances.

Table 3: Performance of FPR in comparison with the reference algorithms (i.e., MCE, SO, TS, VDNS, and Gurobi) on all the benchmark instances.

Algorithm	F_{best}	F_{avg}	#Min	DEV	p-value
Rome: small-size ($12 \leq \|V\| \leq 102$)					
Gurobi	**0.94**	–	**88**	–	NA
MCE	1.18	1.44	68	0.21	7.744e-6
SO	1.01	–	82	–	1.43e-2
TS	1.77	2.03	34	0.19	2.005e-13
VDNS	1.27	1.91	60	0.53	1.213e-7
FPR	**0.94**	**0.98**	**88**	**0.05**	
North: medium-size ($30 \leq \|V\| \leq 533$)					
Gurobi	2.86	–	50	–	5.78e-2
MCE	2.55	2.68	37	0.14	9.617e-5
SO	3.12	–	43	–	1.08e-4
TS	3.62	3.95	21	0.23	1.181e-9
VDNS	2.26	2.87	40	0.47	1.75e-3
FPR	**1.95**	**2.05**	**54**	**0.08**	
Connected: large-size ($\|V\| = 1000$)					
MCE	108.92	112.9	1	2.4	<2.2e-16
SO	125.72	–	0	–	< 2.2e-16
TS	113.78	115.56	0	**0.93**	<2.2e-16
VDNS	82.31	85.42	44	2.13	9.367e-6
FPR	**81.31**	**83.9**	**88**	1.73	
Uniform: ($60 \leq \|V\| \leq 1000$)					
MCE	230.2	232.39	0	1.36	9.486e-15
SO	232.38	–	0	–	9.486e-15
TS	225.73	228.06	0	1.3	9.486e-15
VDNS	215.47	217.88	7	1.67	3.335e-13
FPR	**211.8**	**213.31**	60	**1.14**	
#Total (301 instances)					
MCE	81.10	82.89	107	1.12	< 2.2e-16
SO	86.90	–	125	–	< 2.2e-16
TS	82.12	83.29	55	**0.65**	< 2.2e-16
VDNS	69.73	71.50	152	1.23	< 2.2e-16
FPR	**68.53**	**69.68**	**283**	0.8	

From the remaining tables (i.e., Tables A7 to A11), for both the medium-size ($North$) and small-size ($Rome$) instance sets, our FPR performs slightly better than the best-performing algorithm, VDNS, because both algorithms

can obtain optimal solutions for most instances in $North$ and $Rome$. Overall, the proposed FPR can improve the best results for 104 of the total 301 instances.

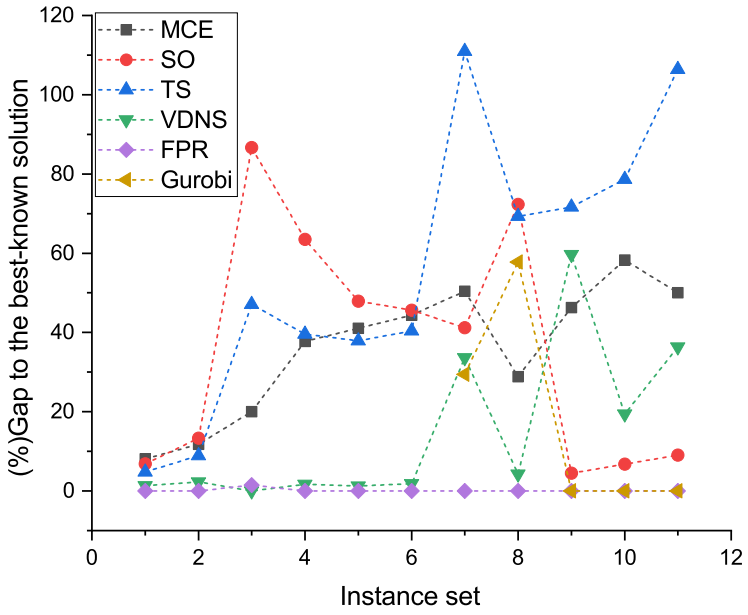
In addition, we summarize the best and average performances of the compared methods in Figure 5, where the x-axis indicates 11 instance sets presented from Tables A1 to A11, and the y-axis denotes the gap in percentage relative to the best-known result for each instance. The gap is defined as $Gap(f) = (f - f_{BK}) * 100/f_{BK}$, where f_{BK} denotes the updated best-known objective value, and f represents the objective value of the current solution obtained by the compared algorithms. Clearly, we can see from Figure 5 that in most cases, the FPR algorithm can capture the best results for both f_{best} and f_{avg} metrics. In summary, the experimental results reported above clearly show that the proposed FPR algorithm is highly competitive compared to state-of-the-art methods.

5. Analysis and Discussions

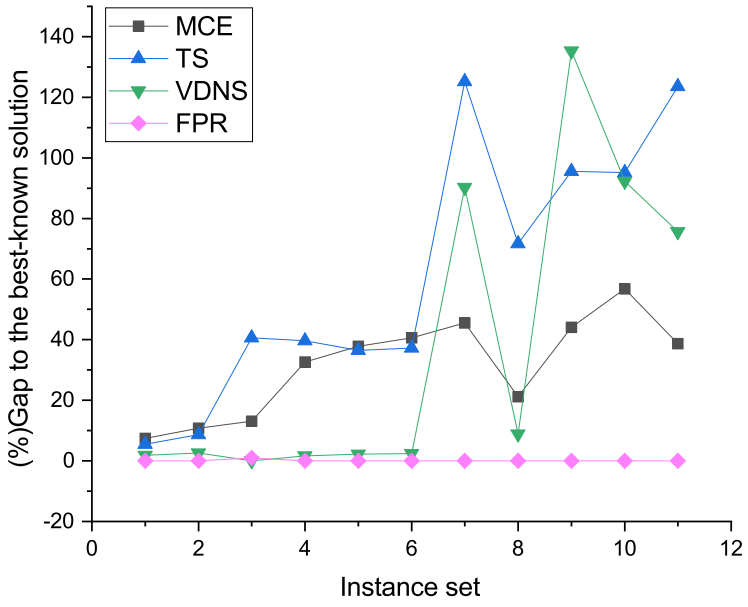
5.1. Effectiveness of the dynamic programming–based local search mechanism

In Section 3.4, we proposed a dynamic-programming mechanism to select multiple neighborhood moves simultaneously after each neighborhood evaluation. To evaluate the effectiveness of this dynamic-programming technique, we ran computational experiments to compare the performance of the FPR algorithm integrated with this mechanism to that of its variant without the dynamic-programming mechanism (FPRNDP) on all instances. More precisely, the variant FPRNDP selects only the best neighborhood move instead of a set of neighborhood moves each time in the neighborhood search phase, while keeping other FPR components unaltered.

The computational results are presented in Table 4. Column $\#Best$ reports the number of instances from which the best objective value among



(a) Best performance(f_{best}).



(b) Average performance(f_{avg}).

Fig. 5. Best and average performance of the compared algorithms tested on each instance set reported in Tables A1-A11, with each set corresponding to a table.

Table 4: Performance of FPR in comparison with the variant without dynamic programming mechanism (i.e., FPRNDP) on all the benchmark instances.

Algorithm	F_{best}	F_{avg}	#Best	DEV	p-value
Rome: small-size ($12 \leq \|V\| \leq 102$)					
FPRNDP	0.95	0.99	87	**0.05**	3.17e-1
FPR	**0.94**	**0.98**	**88**	**0.05**	
North: medium-size ($30 \leq \|V\| \leq 533$)					
FPRNDP	2.08	2.25	50	0.14	4.68e-3
FPR	**1.95**	**2.05**	**58**	**0.08**	
Connected: large-size ($\|V\| = 1000$)					
FPRNDP	81.42	84.11	55	1.84	9.82e-3
FPR	**81.31**	**83.9**	**75**	**1.73**	
Uniform: ($60 \leq \|V\| \leq 1000$)					
FPRNDP	212.02	214.96	43	1.16	3.53e-1
FPR	**211.8**	**213.31**	48	**1.14**	
#Total (301 instances)					
FPRNDP	68.64	69.96	235	0.89	5.94e-4
FPR	**68.53**	**69.68**	**269**	**0.8**	

all the algorithms can be obtained. One can observe that FPR clearly outperforms FPRNDP in terms of both the best and average results (i.e., 68.53 vs. 68.64 and 69.68 vs. 69.96, respectively). In addition, FPR can obtain better results than FPRNDP for a greater number of instances (i.e., 269 vs. 235). Although the p-values from the non-parametric Friedman test on certain instance sets (i.e., Rome and Uniform) do not permit the rejection of the null hypothesis, the p-value for the entire dataset indicates a significant difference between FPR and FPRNDP.

Furthermore, we conducted another experiment to analyze the ratio of the number of neighborhood moves with different sizes, as illustrated in Figure 6. Here, the x-axis denotes the number of layers for test instances, and the y-axis indicates the ratio of moves with different sizes. As shown in Figure 6, two, three, and four moves shown in the legend denote, respectively, two, three, and four neighborhood moves chosen to operate each time after

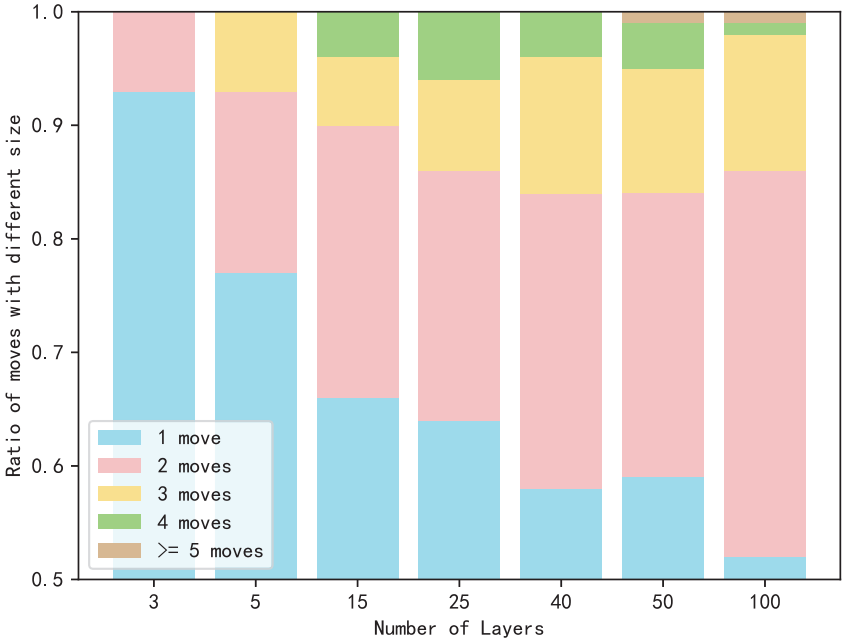


Fig. 6. The ratio of the number of neighborhood moves with different sizes.

neighborhood evaluation. In addition, ≥ 5 moves implies that more than four neighborhood moves are chosen each time. We can observe from Figure 6 that although the use of two moves or more is not as frequent for small cases (e.g., the use ratio of two moves is less than 10% for the instances with three layers), they greatly enhance the search procedure, as verified in the previous experiment (Tables A1–A11 and Table 2). Moreover, for the cases with a larger number of layers (more than 40 layers), two or more moves account for approximately (or more than) half of the rate. In other words, with the increase in the size of instances (i.e., the increase in the number of layers), the ratio of use of two moves or more noticeably increased. All the experimental results presented here demonstrate the importance of the proposed dynamic-programming strategy for selecting multiple neighborhood moves concurrently.

5.2. Importance of two-individual–based path-relinking framework

To study the impact of the proposed fast path-relinking framework based on two individuals, we compared the proposed full-featured FPR algorithm with a traditional path-relinking mechanism that manages multiple solutions (MPR) and an iterative dynamic programming–based local search managing one solution (IDPLS).

The MPR variant of the algorithm is initialized using the $InitialSolution$ procedure proposed in Section 3.3 and manages p individuals. It alternates between the dynamic programming–based local search, the relinking operator, and the updating mechanism. The updating mechanism replaces the worst individual in the set with the new offspring individual obtained if the latter is of better quality, depending on the evaluation function presented in Equation 3. The dynamic programming–based local search phase and relinking operator are the same as those in the FPR algorithm. We empirically set the size of p to 5. For the IDPLS, we iteratively ran the dynamic programming–based local search until the time limit was reached.

Table 5 summarizes the computational results obtained in terms of the best and average objective values for FPR, IDPLS, and MPR, indicating that FPR yielded the best results (i.e., 68.53, 69.68, and 261) with regard to all indicators (i.e., F_{best}, F_{avg}, and $\#Best$, respectively) for all benchmark instances.

Furthermore, the gap in the algorithm performance between IDPLS and FPR gradually increased as the size of the instances increased. Specifically, the value of IDPLS in terms of $\#Best$ is close to that of FPR (85 vs. 88) for the small-size instance set (Rome), while the $\#Best$ value of IDPLS gradually deteriorated in comparison with that of FPR (49 vs. 56 and 29 vs. 78) for the medium-size (North) and large-size (Connected) instance sets, respectively. The experimental results confirm that IDPLS yields a strong intensity of search for small-sized instances but easily falls into premature convergence for

Table 5: Comparison between FPR and the reference algorithms (i.e., IDPLS and MPR) on the all the benchmark instances.

Algorithm	F_{best}	F_{avg}	#Best	DEV	p-value
Rome: small-size ($12 \leq \|V\| \leq 102$)					
IDPLS	0.98	1.00	85	**0.03**	8.33e-1
MPR	**0.94**	**0.98**	**88**	0.05	NA
FPR	**0.94**	**0.98**	**88**	0.05	
North: medium-size ($30 \leq \|V\| \leq 533$)					
IDPLS	2.09	2.21	49	0.12	4.68e-3
MPR	**1.93**	**2.00**	**57**	**0.08**	5.63e-1
FPR	1.95	2.05	56	**0.08**	
Connected: large-size ($\|V\| = 1000$)					
IDPLS	83.09	84.59	29	1.54	1.088e-10
MPR	82.14	84.26	46	**1.51**	8.176e-5
FPR	**81.31**	**83.9**	**78**	1.73	
Uniform: ($60 \leq \|V\| \leq 1000$)					
IDPLS	212.13	**213.01**	36	**0.93**	2.74e-1
MPR	212.28	213.82	29	0.96	7.66e-3
FPR	**211.8**	213.31	**39**	1.14	
#Total (301 instances)					
IDPLS	69.20	69.88	199	**0.74**	7.098e-11
MPR	68.89	69.89	220	0.76	4.899e-6
FPR	**68.53**	**69.68**	**261**	0.8	

medium and large instances. When combined with two distinct frameworks, the FPR algorithm can outperform MPR integrated with multiple distinct frameworks. MPR and FPR reach the same performance on the Rome set, and MPR even performs slightly better than FPR on the North set. However, for all the benchmark instances, the FPR algorithm completely outperforms MPR (i.e., 68.53 vs. 68.89, 69.68 vs. 69.89, and 261 vs. 220) in terms of each indicator (i.e., F_{best}, F_{avg}, and $\#Best$). In addition, the p-values from the non-parametric Friedman test on some instance sets (i.e., Rome, North, and Uniform) do not permit the rejection of the null hypothesis, but the p-value for the entire dataset clearly indicates a significant difference between FPR and MPR (IDPLS).

All these outcomes confirm that FPR achieves a good trade-off between search intensification and diversification, which clearly contributes to obtain-

ing high-quality results.

6. Conclusion

We presented a fast path-relinking approach based on a dynamic programming–based local search to solve the min–max edge crossing problem (MMECP) in graphs. The novel and salient features of our algorithm include the use of a critical vertex–based neighborhood structure for neighborhood reduction, a dynamic programming–based local search strategy for solution improvement, and a two-individual–based path-relinking framework for search efficiency. Extensive computational experiments on publicly available benchmark instances unequivocally demonstrate that the FPR method is a highly competitive approach compared to the best-performing heuristics and the Gurobi solver. Furthermore, we conducted an in-depth analysis of the experimental results to demonstrate the effectiveness of the novel features integrated into the proposed FPR algorithm.

The main advantages of our proposed FPR method are summarized as follows: First, the fast path-relinking approach is a powerful search framework that achieves a good balance between solution quality and computational efficiency. Second, it employs a critical vertex–based neighborhood structure for neighborhood reduction. Third, the proposed local optimization procedure uses a dynamic-programming mechanism to select multiple move operators in different layers to enhance the search process.

Nonetheless, several areas require attention in future research. First, while the dynamic-programming mechanism employed in the local-search phase is effective for solving the MMECP, it would be valuable to explore its performance in solving other variants of graph drawing problems, such as arc crossing minimization in graphs (Martí, 2001). Second, the two-individual–based path-relinking search framework proposed in this study could serve as a benchmark for other population-based algorithms, including memetic

algorithms and artificial-bee-colony algorithms. Finally, since the fast path-relinking search scheme presented here is both simple and versatile, it would be interesting to apply it to other challenging combinatorial optimization problems, such as the job-shop scheduling and graph coloring problems.

Acknowledgment

We are grateful to Dr. Xinyun Wu for sharing the previous experimental results for MMECP. This research was supported by Guanghua Talent Project of Southwestern University of Finance and Economics, the Fundamental Research Funds under grant number 72201216 for the National Natural Science Foundation of China, Sichuan Sicence and Technology Program (No. 2023NSFSC1019), and Spanish Ministerio de Ciencia, Innovacióny Universidades with code PGC2018-095322-B-C21 and PID2021-125709OB-C21 (MCIU/AEI/FEDER, UE), and by the Generalitat Valenciana (CIAICO/2021/224).

References

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

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

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

Chimani, M., Gutwenger, C., Mutzel, P., Spönemann, M., Wong, H.-M., 2011. Crossing minimization and layouts of directed hypergraphs with port constraints. pp. 141–152.

Congram, R. K., Potts, C. N., van de Velde, S. L., 2002. An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. INFORMS Journal on Computing 14 (1), 52–67.

Ding, J., Lü, Z., Li, C.-M., Shen, L., Xu, L., Glover, F., 2019. A two-individual based evolutionary algorithm for the flexible job shop scheduling problem. In: Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 33. pp. 2262–2271.

Glover, F., Campos, V., Martí, R., 2021. Tabu search tutorial. a graph drawing application. TOP 29, 319–350.

Goudet, O., Grelier, C., Hao, J.-K., 2022. A deep learning guided memetic framework for graph coloring problems. Knowledge-Based Systems 258, 109986.

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.

Hu, C., Li, Y., Cheng, X., Liu, Z., 2016. A virtual dataspaces model for large-scale materials scientific data access. Future Generation Computer Systems 54, 456–468.

Kriegel, H.-P., Kröger, P., Renz, M., Schmidt, T., 2008. Hierarchical graph embedding for efficient query processing in very large traffic networks. In: Scientific and Statistical Database Management: 20th International Conference, SSDBM 2008, Hong Kong, China, July 9-11, 2008 Proceedings 20. Springer, pp. 150–167.

Laguna, M., Martí, R., 1999. Grasp and path relinking for 2-layer straight line crossing minimization. INFORMS Journal on Computing 11 (1), 44–52.

Latoza, T. D., Myers, B. A., 2011. Visualizing call graphs. In: 2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). IEEE, pp. 117–124.

Lemons, N. W., Hu, B., Hlavacek, W. S., 2011. Hierarchical graphs for rule-based modeling of biochemical systems. BMC bioinformatics 12 (1), 1–13.

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

Martí, R., 2001. Arc crossing minimization in graphs with grasp. IIE Transactions 33 (10), 913–919.

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

Mateescu, R., Dechter, R., Marinescu, R., 2008. And/or multi-valued decision diagrams (aomdds) for graphical models. Journal of Artificial Intelligence Research 33, 465–519.

Matuszewski, C., Schönfeld, R., Molitor, P., 1999. Using sifting for k-layer straightline crossing minimization. In: International Symposium on Graph Drawing. Springer, pp. 217–224.

Moalic, L., Gondran, A., 2018. Variations on memetic algorithms for graph coloring problems. Journal of Heuristics 24 (1), 1–24.

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.

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

Peng, B., Liu, D., Lü, Z., Martí, R., Ding, J., 2020a. Adaptive memory programming for the dynamic bipartite drawing problem. Information Sciences 517, 183–197.

Peng, B., Zhang, Y., Cheng, T., Lü, Z., Punnen, A. P., 2020b. A two-individual based path-relinking algorithm for the satellite broadcast scheduling problem. Knowledge-Based Systems 196, 105774.

Stallmann, M. F., 2012. A heuristic for bottleneck crossing minimization and its performance on general crossing minimization: Hypothesis and experimental study. Journal of Experimental Algorithmics (JEA) 17, 1–3.

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.

Sun, W., Hao, J.-K., Li, W., Wu, Q., 2022. An adaptive memetic algorithm for the bidirectional loop layout problem. Knowledge-Based Systems 258, 110002.

Wei, Z., Hao, J.-K., 2023. Iterated hyperplane search for the budgeted maximum coverage problem. Expert Systems with Applications 214, 119078.

Wu, X., Xiong, C., Deng, N., Xia, D., 2021. A variable depth neighborhood search algorithm for the min-max arc crossing problem. Computers & Operations Research 134, 105403.

Zehavi, M., 2022. Parameterized analysis and crossing minimization problems. Computer Science Review 45, 100490.

Appendix- computational results

The detailed computational results are given in the tables in the appendix. For Tables A1–A11, columns f_{best} and f_{avg} list the best and average objective values of each compared algorithm for each test instance, respectively. Column *dev* records the mean absolute deviation for each instance. The last row of each table gives the average result AVG for each indicator (i.e., f_{best}, f_{avg}, and dev) in the current table. Note that both Gurobi and SO methods run only once but for longer times; thus, we report only the best result in their turns. From Table A1 to Table A2, one can observe that our FPR algorithm outperforms the other best-performing methods for all the instances in the $Uniform$ set. Specifically, the FPR improves or matches the previous best results in terms of the best objective value (f_{best}) and average result (f_{avg}), compared to the other best-performing algorithms, for all instances in the $Uniform$ set. From Table A3 to Table A6 for the second instance set $Connected$, we can observe that the FPR obtains better results in terms of both f_{best} and f_{avg} for most instances in this dataset.

Table A1: Computational results on Uniform I.

Instance	MCE			SO	TS			VDNS			FPR		
	f_{best}	f_{avg}	dev	f_{best}	f_{best}	f_{avg}	dev	f_{best}	f_{avg}	dev	f_{best}	f_{avg}	dev
noug3-001	265	266.8	1.14	270	265	266.2	0.63	255	257.7	2.16	**251**	252.9	1.2
noug3-002	266	268.6	1.43	262	262	263.4	0.84	252	255	1.41	**245**	246.4	1.65
noug3-003	262	263.9	1.2	266	262	263.6	1.78	253	255	1.94	**244**	247.6	1.84
noug3-004	270	271	0.47	271	266	269.7	1.64	259	262.3	3.3	**253**	254.7	0.82
noug3-005	257	259.9	1.52	262	258	259.8	1.14	246	251.7	2.5	**243**	244.1	0.88
noug3-006	262	263.2	0.79	262	258	260.6	0.97	249	251.7	3.06	**244**	246.5	1.18
noug3-007	267	268	0.47	272	264	266	1.33	256	256.8	1.55	**250**	251.6	1.17
noug3-008	263	265.7	1.25	269	262	265.1	1.66	254	256.8	2.44	**247**	250	1.41
noug3-009	265	266.1	0.99	269	263	264.7	1.34	255	257.5	1.65	**248**	250.6	1.26
noug3-010	263	265.2	1.32	270	261	263.3	1.25	254	255.9	1.66	**249**	250.1	0.74
noug4-001	240	241.7	1.16	234	231	239.8	3.36	230	231.3	1.25	**228**	228.1	0.32
noug4-002	234	235.3	0.82	236	228	233.4	2.55	**224**	225.4	1.07	**224**	224.2	0.42
noug4-003	238	240.3	1.49	229	228	233.4	2.12	**222**	223.3	0.67	**222**	222.3	0.48
noug4-004	268	269.1	1.37	242	244	247.6	2.17	239	240.1	0.99	**238**	238.5	0.53
noug4-005	238	242.1	2.6	229	229	232.3	2.63	**221**	223.6	1.84	**221**	221.8	1.03
noug4-006	241	245.6	3.31	237	230	234.5	2.51	226	227.6	0.7	**225**	225.8	1.23
noug4-007	246	248.3	1.57	245	239	242.2	2.39	**228**	230.1	1.91	**228**	228	0
noug4-008	250	253.2	2.3	255	250	252.1	1.2	243	245.3	2.36	**242**	242.6	0.84
noug4-009	245	248.4	1.96	240	238	242.7	2.75	**230**	230.4	0.52	**230**	230.1	0.32
noug4-010	244	246.3	1.83	244	230	235.1	3.35	**224**	227.6	1.65	**224**	224.9	1.45
noug5-001	256	257.5	1.08	257	246	251.4	2.27	240	241.9	1.1	**237**	238.7	0.82
noug5-002	259	262.9	3.63	255	248	252.8	2.49	241	244.9	3.75	**240**	241	0.67
noug5-003	248	251.7	1.89	252	248	249.9	1.1	239	243.4	1.78	**236**	236.4	0.84
noug5-004	257	259.3	1.7	256	257	258.8	1.03	**242**	245.3	2.5	**242**	244.2	1.75
noug5-005	251	254	1.41	248	238	242.7	2.11	227	231.4	2.76	**223**	226.3	2.41
noug5-006	246	249.7	1.89	245	242	243.7	1.25	230	232.3	1.64	**229**	229.9	1.2
noug5-007	254	257.2	1.48	253	248	249.3	0.95	241	243.2	1.14	**231**	234.5	2.88
noug5-008	254	256.3	1.16	261	254	256.4	1.43	245	248.9	2.33	**243**	244.5	1.35
noug5-009	248	249.4	1.65	257	248	252.1	2.42	241	242.1	0.74	**238**	239.1	0.88
noug5-010	254	256.2	0.92	248	249	251.2	1.81	234	240.8	2.57	**233**	234.6	1.96
AVG	253.7	256.1	1.53	253.2	248.2	251.46	1.82	240	242.64	1.83	**236.93**	238.33	1.12

Table A2: Computational results on Uniform II.

Instance	MCE			SO	TS			VDNS			FPR		
	f_{best}	f_{avg}	dev	f_{best}	f_{best}	f_{avg}	dev	f_{best}	f_{avg}	dev	f_{best}	f_{avg}	dev
noug6-001	266	267.8	0.92	273	266	268.2	1.23	257	260.7	1.95	**253**	254.8	1.03
noug6-002	267	269.7	1.25	270	264	266.8	1.23	255	256.7	1.7	**250**	251.2	1.03
noug6-003	267	268.5	0.97	268	264	266.6	1.58	257	258.2	0.63	**249**	251.2	0.92
noug6-004	268	271.1	1.2	275	266	269.3	1.7	259	263.2	2.57	**255**	256.1	1.37
noug6-005	269	271.7	2.58	274	269	270.5	1.27	260	262.7	1.64	**255**	256.9	1.1
noug6-006	265	267.7	1.77	268	265	265.6	0.7	254	257.6	2.37	**250**	250.9	0.99
noug6-007	271	272.7	1.06	276	267	268.6	0.7	259	262.3	2.67	**253**	254.5	1.58
noug6-008	266	267.2	0.63	269	264	265.9	1.37	256	258.8	2.62	**251**	252.5	1.35
noug6-009	269	270.5	0.85	274	267	268.9	1.29	259	261.2	1.62	**253**	254.8	1.55
noug6-010	266	267.7	1.06	274	264	265.5	0.85	255	258.1	1.73	**251**	251.5	1.08
noug7-001	174	177	1.33	175	171	173.6	0.97	154	157.3	1.7	**153**	154.6	1.43
noug7-002	173	175.1	0.88	180	170	170	0	156	157.7	1.06	**152**	153.9	0.99
noug7-003	178	179.4	1.26	183	173	173	0	158	158.9	0.74	**152**	154.4	1.35
noug7-004	178	178.9	0.99	180	171	171	0	159	159.8	0.92	**154**	155.2	0.79
noug7-005	175	176.8	1.14	182	171	171	0	158	159.3	1.06	**154**	154.8	0.92
noug7-006	180	181.8	1.14	183	174	174	0	160	162.3	1.7	**156**	158.1	1.37
noug7-007	172	174.2	1.32	180	168	172.4	1.58	156	157.5	0.85	**151**	152.7	1.25
noug7-008	175	177.6	1.43	181	170	173.8	1.87	156	158.4	1.26	**153**	154.4	1.17
noug7-009	175	177.3	0.95	182	173	174.8	0.63	157	159.5	1.43	**153**	154.9	1.2
noug7-010	177	178.6	1.26	180	173	173.9	0.32	156	158.6	1.35	**155**	155.5	0.85
noug8-001	176	177.1	0.99	181	173	173	0	158	158.9	1.1	**153**	155.3	1.49
noug8-002	173	174.6	1.07	181	169	171.7	2.11	157	158	0.82	**153**	154.5	1.18
noug8-003	177	179.5	1.18	184	174	174.9	0.32	159	160	0.94	**154**	155.8	1.23
noug8-004	176	179.2	1.62	181	175	175	0	158	160.5	1.84	**155**	157	1.15
noug8-005	177	179.3	1.34	181	174	174	0	160	161.3	1.34	**155**	157.1	1.37
noug8-006	179	181.7	1.7	183	173	175.6	2.22	162	163.5	1.43	**158**	159.2	0.92
noug8-007	175	176	0.67	181	170	171.9	1.52	156	158.1	1.37	**152**	154.1	1.1
noug8-008	181	182.5	0.97	184	176	176	0	162	164.4	1.43	**158**	160.2	1.23
noug8-009	177	179	1.33	182	173	173	0	158	160.4	1.43	**155**	156.6	0.97
noug8-010	179	180.1	0.88	182	171	171	0	157	159.8	1.87	**154**	156.1	1.2
AVG	206.7	208.68	1.19	211.57	203.27	204.65	0.78	190.93	193.12	1.5	**186.67**	188.29	1.17

Table A3: Computational results on Connected I.

Instance	MCE			SO	TS			VDNS			FPR		
	f_{best}	f_{avg}	dev	f_{best}	f_{best}	f_{avg}	dev	f_{best}	f_{avg}	dev	f_{best}	f_{avg}	dev
c1000_2000_25_2_1	37	38.6	0.7	67	38	38	0	30	30.9	0.74	**29**	30.3	0.48
c1000_2000_25_2_2	35	36.5	0.97	66	39	39	0	29	29.7	0.82	**27**	29.2	0.32
c1000_2000_25_2_3	35	36.1	0.88	65	35	35	0	**29**	29.9	0.74	**29**	30.2	0.52
c1000_2000_25_4_1	44	48.7	2.58	93	58	58	0	**37**	39.2	1.23	**37**	40.6	0.63
c1000_2000_25_4_2	55	60.7	3.02	97	52	52	0	**41**	44.2	2.1	43	44.6	0.7
c1000_2000_25_4_3	47	51.6	2.67	92	56	56	0	**41**	44.2	1.99	44	46.8	0.88
c1000_2000_25_8_1	84	89.4	4.27	159	126	126	0	**77**	82.3	3.56	82	90.4	0
c1000_2000_25_8_2	79	81.4	3.2	159	120	120	0	**77**	83.2	3.97	84	87.7	0
c1000_2000_25_8_3	89	97.1	3.75	163	119	119	0	**82**	87	2.87	83	88.6	0
c1000_2000_50_2_1	19	19.3	0.48	31	22	22	0	**15**	15.8	0.63	**15**	15.5	0.67
c1000_2000_50_2_2	19	19.9	0.57	32	22	22	0	**16**	16.8	0.42	**16**	16.3	1.23
c1000_2000_50_2_3	19	19.1	0.32	31	21	21	0	**15**	15.8	0.42	**15**	15.7	1.03
c1000_2000_50_4_1	22	24.5	1.35	45	32	32	0	**21**	22.5	0.97	**21**	22.9	2.17
c1000_2000_50_4_2	26	28.5	1.72	42	33	33	0	**24**	25.1	1.2	**24**	24.9	1.26
c1000_2000_50_4_3	25	25.4	0.52	41	28	28	0	**21**	22.3	0.95	**21**	22.3	2.1
c1000_2000_50_8_1	55	57	1.89	77	84	84	0	**52**	57	3.59	**52**	56.4	5.72
c1000_2000_50_8_2	55	58.9	2.6	78	85	85	0	**54**	56.9	1.65	**54**	56.3	3.33
c1000_2000_50_8_3	59	64.6	4.3	93	91	91	0	**56**	61.7	3.06	58	63.3	2.32
c1000_2000_100_2_1	10	10.4	0.52	14	15	15	0	9	9.1	0.32	**8**	8.7	0.53
c1000_2000_100_2_2	10	10.8	0.42	14	15	15	0	**9**	9	0	**9**	9.1	0.48
c1000_2000_100_2_3	10	10.4	0.7	14	15	15	0	**8**	8.9	0.32	**8**	8.6	0.48
c1000_2000_100_4_1	**13**	16.4	1.78	20	19	20.9	0.99	**13**	13.9	1.29	**13**	13.2	0.99
c1000_2000_100_4_2	15	15.5	0.71	19	19	20	0.94	**13**	13.7	0.48	**13**	13.4	0.99
c1000_2000_100_4_3	13	13.9	0.57	19	18	19.3	0.95	13	13.9	0.57	**11**	12.1	1.16
c1000_2000_100_8_1	45	47.5	1.43	46	52	60	4.32	**41**	41.7	0.95	**41**	41	3.44
c1000_2000_100_8_2	39	41.9	1.45	42	46	49.7	1.89	35	36.3	1.57	**34**	34	1.34
c1000_2000_100_8_3	39	44.2	1.87	42	49	52.4	2.17	**32**	34	1.56	**32**	32	2.83
AVG	36.96	39.57	1.68	61.52	48.48	49.2	0.42	**32**	35	1.41	33.44	35.34	1.32

Table A4: Computational results on Connected II.

Instance	MCE			SO	TS			VDNS			FPR		
	f_{best}	f_{avg}	dev	f_{best}	f_{best}	f_{avg}	dev	f_{best}	f_{avg}	dev	f_{best}	f_{avg}	dev
c1000_3000_25_2_1	88	90.3	1.16	105	66	66	0	60	61.3	1.25	**59**	59.7	0
c1000_3000_25_2_2	88	89.4	0.97	105	65	65	0	58	59.1	1.1	**57**	58.9	0.52
c1000_3000_25_2_3	88	89.7	1.16	107	68	68	0	58	60	0.94	**57**	59.4	0.52
c1000_3000_25_4_1	120	125.3	4.08	175	106	106	0	**86**	90.9	3.28	87	92.2	0
c1000_3000_25_4_2	129	136.9	4.93	179	132	132	0	98	102.6	2.72	**94**	103.1	0.52
c1000_3000_25_4_3	113	116.1	2.13	156	123	123	0	**88**	90.4	2.59	91	94.5	0.63
c1000_3000_25_8_1	218	229.1	5.92	327	281	317.9	12.97	185	193.1	4.82	**183**	191	0.32
c1000_3000_25_8_2	232	237.6	3.44	309	273	274.2	2.53	**177**	179.2	1.62	178	181.7	1.32
c1000_3000_50_2_1	44	45.1	0.57	49	36	39.7	2.16	30	30.7	0.95	**29**	30.7	0.32
c1000_3000_50_2_2	45	45.8	0.42	51	36	36	0	**30**	31.5	0.71	**30**	31.3	0.82
c1000_3000_50_2_3	44	44.8	0.42	51	41	41	0	30	30.2	0.42	**29**	30	1.37
c1000_3000_50_4_1	59	61.5	1.78	81	56	56	0	45	48.1	1.52	**44**	47.2	1.17
c1000_3000_50_4_2	67	69.4	1.71	81	66	66	0	53	56.8	2.53	**52**	55.2	3.88
c1000_3000_50_4_3	59	61.5	1.58	77	68	68	0	**47**	48.8	1.14	**47**	49.6	4.43
c1000_3000_50_8_2	127	134.9	5.22	154	154	170.2	7.83	**102**	107.4	3.06	**102**	106.2	2.72
c1000_3000_50_8_3	157	172.9	8.92	193	203	203	0	135	138.8	3.43	**131**	137.2	5.52
c1000_3000_100_2_1	23	23.3	0.48	24	24	24	0	**17**	17	0	**17**	17	2.75
c1000_3000_100_2_2	24	24.2	0.42	24	24	24.8	0.42	18	18	0	**17**	17.6	0.95
c1000_3000_100_2_3	23	23.5	0.53	23	23	24.2	0.63	**16**	16.8	0.42	**16**	16.6	0.67
c1000_3000_100_4_1	36	42.2	2.57	41	41	45.7	2.41	**35**	35.4	0.7	**35**	35	0.47
c1000_3000_100_4_2	35	39.2	2.04	41	40	42.2	1.32	**30**	31.7	1.25	**30**	30.4	1.4
c1000_3000_100_4_3	34	37.3	2.36	37	36	37.6	1.35	29	29	0	**27**	28.2	2.82
c1000_3000_100_8_1	100	106.2	3.99	97	116	121.6	4.2	81	86.2	2.39	**80**	80.9	1.43
c1000_3000_100_8_2	89	91.4	1.26	80	94	97.2	1.69	73	74.8	2.35	**61**	62.2	2.39
c1000_3000_100_8_3	95	102.3	4.85	92	98	108.6	4.77	**73**	79.5	3.95	**73**	73.1	4.61
AVG	85.48	89.6	2.52	106.36	90.8	94.32	1.69	66.16	68.69	1.73	**65.04**	67.56	1.66

Table A5: Computational results on Connected III.

Instance	MCE			SO	TS			VDNS			FPR		
	f_{best}	f_{avg}	dev	f_{best}	f_{best}	f_{avg}	dev	f_{best}	f_{avg}	dev	f_{best}	f_{avg}	dev
c1000_4000_25_2_1	132	133.6	1.17	147	108	108	0	93	94.8	1.48	**90**	92.5	0.48
c1000_4000_25_2_2	135	136.7	1.34	147	100	100	0	93	94.7	1.16	**91**	92.1	0.97
c1000_4000_25_2_3	129	131.3	1.06	141	101	101	0	91	92.7	1.16	**89**	91.2	0.67
c1000_4000_25_4_1	220	226.2	4.05	256	196	197.6	1.84	**149**	157.1	5.07	154	159	0.32
c1000_4000_25_4_2	230	244.4	8.42	260	236	236	0	**163**	168.7	4.76	165	169.3	0.88
c1000_4000_25_4_3	208	218.8	5.03	243	205	205	0	148	157.6	5.6	**147**	160.9	0.52
c1000_4000_25_8_1	392	402.8	9.2	474	519	521.7	0.95	302	312.5	5.17	**295**	300	0
c1000_4000_25_8_2	367	375.3	4.72	434	438	439.7	2.11	**263**	270.6	3.72	264	267.7	0
c1000_4000_50_2_1	65	66.5	0.71	70	58	58	0	**46**	47.9	1.2	**46**	47.9	0
c1000_4000_50_2_2	66	66.7	0.82	70	60	60	0	48	48.7	0.48	**47**	48.1	1.51
c1000_4000_50_2_3	65	65.3	0.48	69	54	54	0	46	47.2	0.92	**44**	46.3	0.74
c1000_4000_50_4_1	107	112.4	3.06	116	97	97	0	**75**	77.3	1.34	76	78.1	1.48
c1000_4000_50_4_2	123	128.3	3.56	126	122	122	0	**87**	90.3	2.06	88	90.1	3.46
c1000_4000_50_4_3	118	126.8	3.82	124	103	103	0	91	93	1.41	**90**	92.3	3.97
c1000_4000_50_8_3	284	295	5.1	296	297	297	0	219	223.6	4.06	**210**	216.5	5.65
c1000_4000_100_2_1	34	34.7	0.67	33	33	33.6	0.52	**25**	26	0.47	**25**	25.3	4
c1000_4000_100_2_2	34	34.9	0.57	34	34	34.1	0.32	27	27.9	0.74	**26**	27.4	2.79
c1000_4000_100_2_3	33	33.6	0.52	33	32	32.8	0.42	26	26.1	0.32	**25**	25.7	1.6
c1000_4000_100_4_1	86	89.2	1.99	78	78	84.4	3.17	69	70.9	1.1	**68**	68.1	0.88
c1000_4000_100_4_2	64	66.3	1.34	63	60	63	1.41	**46**	47.2	1.03	**46**	47.1	1.16
c1000_4000_100_4_3	65	66.4	0.84	62	57	61.5	2.12	47	47.9	1.45	**46**	46.6	1.29
c1000_4000_100_8_1	178	181.2	2.74	168	177	192.3	7.69	**139**	146.6	5.91	**139**	139	2.28
c1000_4000_100_8_2	148	150.8	1.55	137	144	149.3	2.67	121	126	3.43	**118**	118	1.64
c1000_4000_100_8_3	164	166.5	2.12	145	165	168.7	2.06	136	141.1	3.9	**130**	130	4.48
AVG	143.63	148.07	2.7	155.25	144.75	146.65	1.05	106.25	109.85	2.41	**104.96**	107.47	1.7

Table A6: Computational results on Connected IV.

Instance	MCE			SO	TS			VDNS			FPR		
	f_{best}	f_{avg}	dev	f_{best}	f_{best}	f_{avg}	dev	f_{best}	f_{avg}	dev	f_{best}	f_{avg}	dev
c1000_5000_25_2_1	172	174.8	1.48	187	138	139	1.33	130	132.5	1.78	**126**	127.5	0.42
c1000_5000_25_2_2	172	175	1.33	190	138	138	0	127	130.2	2.53	**125**	126.4	0.7
c1000_5000_25_2_3	172	173.1	0.88	183	139	139	0	129	131.1	1.45	**124**	126.4	0.53
c1000_5000_25_4_1	317	330.7	8.18	355	308	308.6	0.52	**221**	230.5	5.46	223	227.6	0
c1000_5000_25_4_2	367	382.1	6.76	360	340	340	0	248	259	5.79	**247**	260.8	1.52
c1000_5000_25_4_3	313	328.8	6.8	335	315	315	0	222	232.6	8.41	**220**	225.9	0.79
c1000_5000_25_8_1	569	580.5	6.24	602	684	684	0	432	441.8	9.64	**416**	425	1.43
c1000_5000_25_8_2	543	556.8	7.41	581	645	645	0	384	402.2	8.3	**381**	393	0.97
c1000_5000_50_2_2	84	85.1	0.74	88	78	78	0	65	66.1	0.88	**63**	64.6	1.43
c1000_5000_50_2_3	84	84.8	0.79	87	75	75	0	65	65.4	0.52	**62**	63.4	2.72
c1000_5000_50_4_1	173	176.3	2.16	172	159	159	0	119	122.5	2.64	**116**	120.8	5.63
c1000_5000_50_4_2	176	182.1	4.2	172	157	157	0	119	120.9	2.33	**117**	122.4	3.67
c1000_5000_50_4_3	189	192.9	1.66	176	175	175	0	128	133.2	3.88	**126**	130.9	6.5
c1000_5000_100_2_1	43	43.3	0.48	42	41	41.4	0.52	35	35.4	0.52	**34**	34.2	8.62
c1000_5000_100_2_2	44	44.8	0.92	44	42	42.7	0.48	36	37.4	0.7	**35**	35.6	0.97
c1000_5000_100_2_3	43	43.8	0.42	42	42	42.8	0.42	36	36.9	0.74	**35**	35.5	0.7
c1000_5000_100_4_1	118	120.2	2.25	109	110	118.2	3.99	**95**	99.2	4.1	**95**	95	2.86
c1000_5000_100_4_2	95	95.9	0.88	85	84	86.8	1.48	67	68.9	1.52	**65**	66.9	3.47
c1000_5000_100_4_3	91	92.8	1.14	87	86	87.8	1.14	67	70	2.45	**66**	66.8	3.31
AVG	198.16	203.36	2.88	205.11	197.68	198.54	0.52	143.42	148.2	3.35	**140.84**	144.67	2.43

Table A7: Computational results on North I.

Instance	Gurobi	MCE			SO	TS			VDNS			FPR		
		f_{best}	f_{avg}	dev	f_{best}	f_{best}	f_{avg}	dev	f_{best}	f_{avg}	dev	f_{best}	f_{avg}	dev
north.30.29.7	**0**	**0**	0	0	**0**	**0**	0	0	**0**	0.4	0.7	**0**	0	0
north.30.29.11	**0**	**0**	0	0	**0**	**0**	0	0	**0**	1.9	0.88	**0**	0	0
north.30.29.12	**0**	**0**	0	0	**0**	**0**	0	0	**0**	0	0	**0**	0	0
north.30.33.5	**0**	**0**	0.5	0.53	**0**	**0**	0.9	0.32	**0**	2.4	1.35	**0**	0	0
north.30.33.19	**1**	**1**	1.4	0.52	**1**	**1**	1.4	0.52	**1**	2.1	0.57	**1**	1	0
north.30.34.20	**6**	8	8	0	**6**	7	7.5	0.53	**6**	6	0	**6**	6	0
north.30.35.4	**1**	**1**	1	0	**1**	**1**	1	0	**1**	1.8	0.79	**1**	1	0
north.30.35.14	**1**	**1**	1	0	**1**	2	2	0	**1**	1.9	0.57	**1**	1	0
north.30.35.18	**1**	2	3.9	0.74	**1**	2	2.3	0.48	**1**	2.1	0.52	**1**	1	0
north.30.37.9	**2**	**2**	2	0	**2**	**2**	2	0	**2**	2	0	**2**	2	0
north.30.37.10	**2**	**2**	2	0	**2**	**2**	2	0	**2**	2	0	**2**	2	0
north.30.39.17	**1**	2	2	0	**1**	2	2.9	0.32	2	3.2	1.23	**1**	1	0
north.30.40.13	**2**	**2**	2.8	0.63	**2**	3	3.6	0.52	3	5.1	0.88	**2**	2	0
north.30.40.21	**0**	**0**	0	0	**0**	**0**	0.1	0.32	**0**	1.3	1.7	**0**	0	0
north.30.41.3	**3**	**3**	3	0	**3**	**3**	3.4	0.52	**3**	3.8	0.42	**3**	3	0
north.30.43.1	**1**	**1**	1.1	0.32	**1**	3	3	0	2	2.5	0.71	**1**	1.1	0.18
north.30.44.15	**1**	2	2	0	2	3	3.8	0.42	**1**	1.6	0.52	**1**	1.5	0.5
north.30.45.2	**1**	2	2	0	**1**	3	3.6	0.52	2	2.4	0.7	**1**	1.1	0.18
north.30.45.8	**1**	2	2	0	**1**	3	3.6	0.52	2	3	0.47	**1**	1	0
north.30.55.16	**1**	2	2	0	2	4	4	0	2	2.8	0.42	**1**	1	0
north.30.62.6	**1**	2	2	0	3	6	6	0	**1**	1.9	0	2	2	0
north.40.39.1	**0**	**0**	0	0	**0**	**0**	0	0	**0**	0.4	0.52	**0**	0	0.56
north.40.39.9	**0**	**0**	0	0	**0**	**0**	0	0	**0**	0	0	**0**	0	0
north.40.39.11	**0**	**0**	0	0	**0**	**0**	0	0	**0**	0	0	**0**	0	0
north.40.39.12	**0**	3	3	0	**0**	**0**	0.5	0.53	**0**	0.6	0.52	**0**	0	0
north.40.46.6	**0**	1	1	0	**0**	2	2	0	1	2.1	0.57	**0**	0	0
north.40.47.3	**0**	1	1	0	**0**	2	2	0	2	2.5	0.53	**0**	0	0
north.40.48.5	**0**	1	1.1	0.32	**0**	2	2	0	1	2.4	0.84	**0**	0	0
north.40.49.2	**0**	1	1	0	1	2	2.7	0.48	2	2.1	0.32	**0**	0.2	0
north.40.49.4	**0**	**0**	0.1	0.32	**0**	2	2.5	0.53	2	2.2	0.42	**0**	0	0
north.40.49.14	**1**	**1**	1.4	0.52	**1**	3	3.1	0.32	**1**	1.9	0.32	**1**	1	0.32
north.40.54.8	**0**	**0**	0.4	0.52	**0**	2	2.9	0.32	2	3.2	0.79	**0**	0	0
north.40.56.13	**1**	2	2.4	0.52	2	4	4	0	**1**	2	0.67	**1**	1	0
north.40.60.16	**0**	1	1	0	**0**	2	2.7	0.48	1	1.5	0.53	**0**	0	0
north.40.72.10	**4**	**4**	4.2	0.42	**4**	6	6.1	0.32	**4**	4.2	0.42	**4**	4	0
north.40.73.7	**1**	2	2	0	4	5	5	0	2	2	0	2	2	0
north.40.131.15	24	**8**	8.8	0.63	20	14	14	0	**8**	9.2	0.79	9	9.7	0
AVG	1.54	1.62	1.79	0.16	1.68	2.51	2.77	0.21	1.59	2.34	0.53	**1.19**	1.23	0.05

Table A8: Computational results on North II.

Instance	Gurobi	MCE			SO	TS			VDNS			FPR		
		f_{best}	f_{avg}	dev	f_{best}	f_{best}	f_{avg}	dev	f_{best}	f_{avg}	dev	f_{best}	f_{avg}	dev
north.45.45.7	**5**	**5**	5.1	0.32	**5**	**5**	5.4	0.52	**5**	5.1	0.32	**5**	5	0
north.45.46.5	**0**	1	1	0	**0**	1	1.9	0.32	1	1.8	0.42	**0**	0	0
north.45.47.2	**0**	**0**	0	0	**0**	**0**	0	0	**0**	0	0	**0**	0	0
north.45.47.3	**0**	**0**	0	0	**0**	**0**	0	0	**0**	0	0	**0**	0	0
north.45.56.4	**1**	2	2	0	**1**	3	3	0	2	2	0	**1**	1	0
north.45.57.6	**10**	**10**	10	0	**10**	**10**	10	0	**10**	12.6	2.12	**10**	10	0
north.45.59.1	**2**	3	3.1	0.32	3	4	5.6	0.7	**2**	2.8	0.42	**2**	2	0
north.50.49.2	**0**	**0**	0	0	**0**	**0**	0	0	1	1.5	0.53	**0**	0	0
north.50.69.3	**4**	**4**	4.1	0.32	**4**	5	5.1	0.32	**4**	4.8	0.42	**4**	4	0
north.50.75.1	**11**	27	27	0	**11**	**11**	11.3	0.48	**11**	11	0	**11**	11	0
north.55.63.1	**4**	**4**	4	0	**4**	**4**	4.9	0.32	**4**	4	0	**4**	4	0.48
north.55.65.5	**2**	**2**	2	0	**2**	3	3.9	0.32	3	3.6	0.7	**2**	2.1	0.18
north.55.72.8	**3**	**3**	3	0	**3**	4	4	0	**3**	3	0	**3**	3	0.32
north.55.82.7	**3**	**3**	3.5	0.71	**3**	6	6	0	**3**	3	0	**3**	3	0.5
north.55.105.2	7	**2**	2	0	9	6	6.9	0.74	**2**	2.1	0.32	**2**	2.8	0.48
north.55.105.3	7	**2**	2	0	8	7	7	0	**2**	2.1	0.32	**2**	2.5	0.42
north.55.105.4	4	**2**	2	0	8	7	7.9	0.32	**2**	2.2	0.42	**2**	2.4	0.48
north.55.105.10	4	**2**	2	0	8	7	7.9	0.32	**2**	2.1	0.32	**2**	2.6	0
north.55.105.11	6	**2**	2	0	10	6	7.5	0.97	**2**	2.1	0.32	**2**	2.9	0.18
north.55.111.9	14	**7**	7.2	0.42	14	13	13	0	**7**	7.3	0.48	**7**	7.3	0
north.55.130.6	22	7	7.1	0.32	16	15	15	0	**6**	6.9	0.32	7	7.8	0
AVG	5.19	4.19	4.24	0.11	5.67	5.57	6.01	0.25	3.43	3.81	0.35	**3.29**	3.5	0.14

Table A9: Computational results on Rome I.

Instance	Gurobi	MCE			SO	TS			VDNS			FPR		
		f_{best}	f_{avg}	dev	f_{best}	f_{best}	f_{avg}	dev	f_{best}	f_{avg}	dev	f_{best}	f_{avg}	dev
rome.10.10.43	**0**	**0**	0.5	0.53	**0**	**0**	0	0	**0**	0.6	0.52	**0**	0	0
rome.10.10.54	**0**	**0**	0	0	**0**	**0**	0	0	**0**	0.7	0.48	**0**	0	0
rome.11.12.65	**0**	**0**	0	0	**0**	**0**	0	0	**0**	0.8	0.42	**0**	0	0
rome.11.17.88	**1**	**1**	1.5	0.53	**1**	**1**	1.8	0.42	2	3	0.67	**1**	1	0
rome.12.11.10	**0**	**0**	0	0	**0**	**0**	0	0	**0**	0	0	**0**	0	0
rome.12.11.32	**0**	**0**	0	0	**0**	**0**	0	0	**0**	0.5	0.53	**0**	0	0
rome.12.13.81	**0**	**0**	0	0	**0**	**0**	0.3	0.48	1	1.3	0.48	**0**	0	0
rome.13.17.80	**1**	**1**	1	0	**1**	**1**	1	0	**1**	1.7	0.67	**1**	1	0
rome.13.19.87	**2**	**2**	2.4	0.52	**2**	**2**	2	0	3	3.6	0.52	**2**	2	0
rome.15.15.85	**0**	**0**	0	0	**0**	**0**	0	0	**0**	0	0	**0**	0	0
rome.15.18.75	**0**	**0**	0.9	0.32	**0**	1	1	0	1	1.3	0.48	**0**	0	0
rome.15.24.83	**1**	**1**	1.7	0.48	**1**	3	3	0	**1**	1.7	0.48	**1**	1	0
rome.16.17.21	**1**	**1**	1	0	**1**	**1**	1	0	**1**	1.5	0.53	**1**	1	0
rome.18.26.86	**1**	**1**	1.4	0.52	**1**	2	2.5	0.53	2	2.4	0.52	**1**	1.1	0.18
rome.20.21.5	**1**	**1**	1	0	**1**	**1**	1	0	**1**	1.5	0.71	**1**	1	0
rome.20.25.9	**1**	**1**	1	0	**1**	2	2	0	**1**	1.9	0.32	**1**	1	0
rome.20.26.78	**1**	2	2	0	**1**	2	2	0	2	2.2	0.42	**1**	1.2	0.32
rome.21.21.15	**0**	**0**	0	0	**0**	**0**	0.8	0.42	**0**	1.2	0.63	**0**	0	0
rome.21.24.40	**0**	1	1	0	**0**	1	1	0	1	1.4	0.52	**0**	0	0
rome.21.24.44	**0**	1	1	0	**0**	1	1.8	0.42	1	1.3	0.48	**0**	0	0
rome.21.24.46	**1**	**1**	1	0	**1**	**1**	1	0	**1**	2	0.47	**1**	1	0
rome.21.24.79	**1**	**1**	1	0	**1**	**1**	1	0	**1**	1.8	0.63	**1**	1	0
rome.21.26.11	**1**	**1**	1	0	**1**	2	2	0	**1**	2.2	0.63	**1**	1	0
rome.21.26.24	**1**	**1**	1	0	**1**	**1**	1.2	0.42	2	2.1	0.32	**1**	1	0
rome.21.27.20	**1**	**1**	1.5	0.53	**1**	2	2	0	2	2.6	0.84	**1**	1	0
rome.21.33.41	**1**	**1**	1.9	0.32	2	3	3.7	0.48	2	2	0	**1**	1	0
rome.21.36.25	**2**	**2**	2.6	0.52	**2**	3	3.9	0.32	**2**	2	0	**2**	2	0
AVG	**0.67**	0.78	0.98	0.16	0.7	1.15	1.33	0.13	1.07	1.6	0.45	**0.67**	0.68	0.02

Table A10: Computational results on Rome II.

Instance	Gurobi	MCE			SO	TS			VDNS			FPR		
		f_{best}	f_{avg}	dev	f_{best}	f_{best}	f_{avg}	dev	f_{best}	f_{avg}	dev	f_{best}	f_{avg}	dev
rome.22.23.28	**0**	**0**	0	0	**0**	**0**	0	0	**0**	0.8	0.92	**0**	0	0
rome.22.24.55	**1**	**1**	1	0	**1**	**1**	1	0	**1**	1.5	0.71	**1**	1	0
rome.22.26.73	**1**	**1**	1.8	0.42	**1**	2	2	0	2	2.3	0.48	**1**	1	0
rome.22.27.52	**1**	**1**	1.8	0.42	**1**	2	2	0	**1**	2.3	0.67	**1**	1	0
rome.22.30.14	**1**	**1**	1.9	0.32	**1**	2	2.8	0.42	**1**	2.1	0.57	**1**	1	0
rome.22.31.34	**1**	**1**	1.1	0.32	**1**	3	3	0	2	2.8	0.63	**1**	1	0
rome.23.24.63	**1**	**1**	1	0	**1**	**1**	1	0	**1**	1.3	0.48	**1**	1	0
rome.23.24.76	**0**	1	1	0	**0**	**0**	0.3	0.48	1	1.5	0.53	**0**	0	0
rome.23.25.70	**1**	**1**	1	0	**1**	**1**	1	0	**1**	1.2	0.42	**1**	1	0
rome.23.26.19	**1**	2	2	0	**1**	2	2	0	**1**	1.7	0.67	**1**	1	0
rome.23.30.82	**1**	2	2	0	**1**	2	2.1	0.32	2	2.3	0.48	**1**	1	0
rome.24.25.47	**1**	**1**	1	0	**1**	**1**	1	0	**1**	1.1	0.32	**1**	1	0
rome.24.28.27	**2**	**2**	2.1	0.32	**2**	**2**	2	0	**2**	2.6	0.52	**2**	2	0
rome.24.32.50	**1**	**1**	1.7	0.48	**1**	**1**	1.9	0.32	2	2.7	0.48	**1**	1	0
rome.24.33.7	**1**	2	2	0	**1**	3	3.1	0.32	**1**	2.7	0.95	**1**	1	0
rome.25.37.51	**3**	**3**	3.9	0.57	4	5	5	0	4	4.1	0.32	**3**	3	0
rome.26.30.2	**0**	1	1.6	0.52	**0**	1	1.4	0.52	**0**	2.2	0.92	**0**	0	0
rome.26.31.61	**1**	**1**	1.6	0.7	**1**	2	2	0	**1**	1.6	0.97	**1**	1.3	0.54
rome.26.38.57	**2**	**2**	2.3	0.48	**2**	4	4	0	**2**	2.6	0.52	**2**	2	0
rome.27.28.62	**0**	1	1	0	**0**	1	1	0	**0**	1.1	0.74	**0**	0	0
rome.27.37.68	**1**	3	3.8	0.42	2	3	3.5	0.53	**1**	2.4	0.7	**1**	1	0
rome.28.29.12	**1**	**1**	1	0	**1**	**1**	1.5	0.53	**1**	2.1	0.99	**1**	1	0
rome.28.31.1	**1**	**1**	1	0	**1**	2	2	0	**1**	1.6	1.07	**1**	1	0
rome.28.32.71	**1**	**1**	1.2	0.42	**1**	2	2	0	**1**	2.1	0.57	**1**	1	0
rome.28.34.29	**1**	**1**	1	0	**1**	2	2	0	**1**	2.1	0.74	**1**	1	0
rome.28.35.36	**1**	2	2	0	**1**	3	3.2	0.42	**1**	2.1	0.74	**1**	1	0
rome.28.38.4	**2**	3	3	0	**2**	3	3.1	0.32	**2**	2	0	**2**	2	0
rome.29.28.35	**0**	**0**	0.3	0.48	**0**	**0**	0.2	0.42	**0**	1.2	0.63	**0**	0	0
rome.29.30.31	**1**	**1**	1.1	0.32	**1**	**1**	1.2	0.42	**1**	1.9	0.57	**1**	1	0
rome.29.32.45	**1**	**1**	1.2	0.42	**1**	**1**	1.8	0.42	**1**	2.1	0.57	**1**	1	0
rome.29.40.69	**2**	**2**	3	0.67	**2**	3	3.8	0.42	**2**	2	0	**2**	2	0
AVG	**1.03**	1.35	1.63	0.23	1.1	1.84	2.03	0.19	1.23	2	0.61	**1.03**	1.04	0.02

Table A11: Computational results on Rome III.

Instance	Gurobi	MCE			SO	TS			VDNS			FPR		
		f_{best}	f_{avg}	dev	f_{best}	f_{best}	f_{avg}	dev	f_{best}	f_{avg}	dev	f_{best}	f_{avg}	dev
rome.30.31.3	**0**	**0**	0.2	0.42	**0**	1	1	0	1	1.9	0.57	**0**	0	0
rome.30.31.53	**0**	1	1	0	**0**	1	1	0	1	1.1	0.32	**0**	0	0
rome.30.32.48	**1**	**1**	1.8	0.42	**1**	**1**	1.6	0.52	**1**	1.4	0.97	**1**	1	0
rome.30.33.60	**1**	**1**	1	0	**1**	2	2	0	**1**	1.8	0.63	**1**	1	0
rome.30.35.64	**2**	**2**	2.4	0.52	**2**	3	3	0	**2**	2	0	**2**	2	0
rome.30.35.74	**0**	1	1	0	**0**	1	1.8	0.42	1	1.8	0.42	**0**	0.4	0.48
rome.30.36.77	**2**	**2**	2	0	**2**	**2**	2.9	0.32	**2**	2.3	0.48	**2**	2	0
rome.30.36.84	**1**	2	2	0	**1**	2	2.6	0.52	2	2.9	0.57	**1**	1	0
rome.30.40.23	**2**	**2**	2	0	**2**	**2**	2.4	0.52	**2**	2.9	0.57	**2**	2	0
rome.30.40.59	**1**	**1**	1.9	0.32	2	3	3.9	0.32	**1**	1.9	0.32	**1**	1.3	0.42
rome.31.34.38	**0**	1	1	0	**0**	2	2	0	2	2.2	0.42	**0**	0.6	0.48
rome.31.36.18	**2**	**2**	2	0	**2**	**2**	2.7	0.48	**2**	2.7	0.48	**2**	2	0
rome.31.37.33	**1**	**1**	1.9	0.32	**1**	2	2.9	0.32	**1**	2	0.47	**1**	1	0
rome.31.40.49	**2**	3	3	0	**2**	3	3	0	**2**	2.2	0.42	**2**	2.1	0.18
rome.31.40.56	**1**	**1**	1.7	0.48	**1**	2	2.8	0.42	**1**	1.4	0.52	**1**	1	0
rome.32.37.13	**1**	**1**	1	0	**1**	**1**	1.8	0.42	**1**	1.9	0.57	**1**	1.1	0.18
rome.33.34.42	**1**	**1**	1.1	0.32	**1**	**1**	1	0	**1**	2.4	0.7	**1**	1	0
rome.33.37.30	**1**	**1**	1.6	0.52	**1**	2	2.4	0.52	2	3.5	1.18	**1**	1	0
rome.33.38.39	**1**	**1**	1.7	0.48	**1**	2	2	0	2	2.2	0.42	**1**	1	0
rome.33.42.66	**1**	**1**	1.6	0.52	**1**	2	2.9	0.32	**1**	1.5	0.53	**1**	1	0
rome.35.42.16	**1**	**1**	1.8	0.42	**1**	2	2.8	0.42	2	2.4	0.52	**1**	1	0
rome.35.42.67	**2**	**2**	2.3	0.48	**2**	3	3	0	**2**	3.7	1.16	**2**	2	0
rome.38.48.26	**1**	2	2	0	**1**	3	3	0	2	2	0	**1**	1	0
rome.40.42.8	**1**	**1**	1	0	**1**	2	2	0	2	2.2	0.42	**1**	1	0
rome.40.49.22	**1**	2	2	0	2	4	4	0	**1**	1.3	0.48	**1**	1	0
rome.41.45.72	**1**	2	2.3	0.67	**1**	2	2.9	0.32	2	2.7	0.82	**1**	1.4	0.48
rome.41.54.37	**1**	**1**	1.1	0.32	**1**	4	4	0	**1**	1	0	**1**	1	0
rome.43.58.6	**2**	**2**	2	0	**2**	4	4.3	0.48	**2**	2.2	0.42	**2**	2	0
rome.48.58.17	**1**	**1**	1.3	0.48	**1**	3	3.8	0.42	**1**	1.6	0.52	**1**	1.4	0.48
rome.49.62.58	**1**	**1**	1.9	0.57	2	4	4.3	0.48	**1**	1.6	0.52	**1**	1.4	0.64
AVG	**1.1**	1.37	1.65	0.24	1.2	2.27	2.66	0.24	1.5	2.09	0.51	**1.1**	1.19	0.11