An intensification-driven search algorithm for the family traveling salesman problem with incompatibility constraints

Zequn Wei [a], Jin-Kao Hao [b], Jintong Ren [c,*], Qinghua Wu [d], Eduardo Rodriguez-Tello [e]

[a] *School of Economics and Management, Beijing University of Posts and Telecommunications, 100876 Beijing, China*

[b] *LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France*

[c] *Business School, Hohai University, 210098 Nanjing, China*

[d] *School of Management, Huazhong University of Science and Technology, No. 1037, Luoyu Road, Wuhan, China*

[e] *Cinvestav Tamaulipas, Km. 5.5 Carretera Victoria - Soto La Marina, 87130 Victoria Tamps., Mexico*

Accepted to **Knowledge-Based Systems**, Aug. 2024

Abstract

The family traveling salesman problem with incompatibility constraints (FTSP-IC) is a variant of the well-known traveling salesman problem. Given a set of candidate nodes divided into several subsets (families), the FTSP-IC is to find several routes such that the sum of their total traveling distance is minimized, while ensuring a predetermined number of nodes from each family is visited and satisfying the incompatibility constraints. The FTSP-IC has a number of real-life applications, yet it is challenging to solve the problem due to its NP-hard nature. In this work, we introduce a competitive intensification-driven search algorithm for solving this relevant problem. The proposed algorithm significantly intensifies the search by performing extensive searches in the nearby area of discovered local optima. Computational results on 63 benchmark instances from the literature show that our algorithm is able to improve 29 best-know solutions (new upper bounds) and match all the remaining 34 proven optimal solutions. The impacts of the key components of the algorithm on its performance are experimentally analyzed.

Keywords: Family traveling salesman problem; Incompatibility constraints; Intensification-driven search; Heuristics; Combinatorial optimization.

1 Introduction

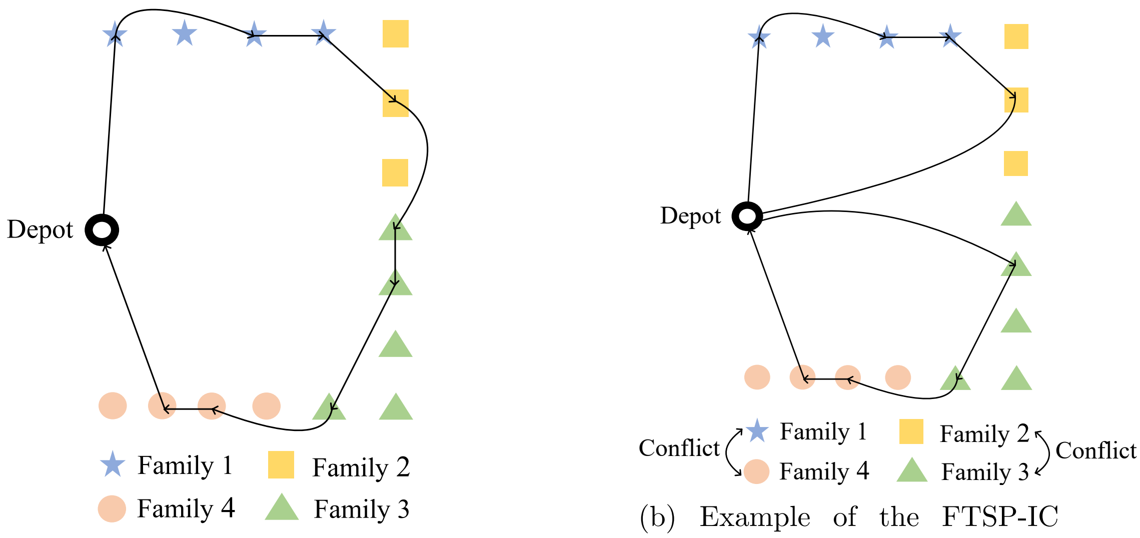
As a generalization of the conventional traveling salesman problem (TSP), the family traveling salesman problem (FTSP) (Morán-Mirabal et al., 2014) is defined as follows. Given a complete and directed graph $G(V, E)$, $V = \{0\} \cup V_c$ is the set of nodes, where 0 denotes the depot and V_c is the set of customers. E is the edge set in which each edge $(i, j) \in E$ is assigned a cost (distance) d_{ij}. The customer set $V_c = \{1, 2, ..., n\}$ is divided into L subsets (families), i.e., $\{F_1, F_2, ..., F_L\}$ $(\bigcup_{l=1}^{L} F_l = V_c$ and $F_{l_1} \cap F_{l_2} = \emptyset$, $l_1 \neq l_2$, $\forall l_1, l_2 \in \{1, 2, ..., L\})$. Herein, $\mathcal{F}(i) \in \{1, 2, ..., L\}$ is the family of the customer i. The salesman starts from the depot 0, and selects at least h_l $(0 \leq h_l \leq |F_l|, \forall l \in \{1, 2, ..., L\})$ different nodes from the family F_l to visit. The goal is to find a route that minimizes the total traveling distance. As an illustrative example, Figure 1(a) shows four families where each family contains several customers. The number of customers to visit in families 1 to 4 is 3, 1, 3, and 2, respectively. Then, the traveling salesman needs to choose a shortest route that starts and ends at the depot node, while meeting the requirement to visit the given number of customers in each family. Figure 1(a) shows a feasible FTSP solution for this example.

In this study, we address the family traveling salesman problem with incompatibility constraints (FTSP-IC) as introduced by Bernardino and Paias (2022). This problem involves managing incompatibility conflicts between different families, which may naturally lead to more than one route in a feasible solution. Starting from a central depot, the salesman must choose at least h_l customers from each family F_l, where $0 \leq h_l \leq |F_l|$ for all $l \in \{1, 2, ..., L\}$. The selected customers are then organized into multiple routes designed to cover all chosen customers with the minimum total traveling distance. Due to incompatibility constraints, nodes (representing customers) belonging to incompatible families cannot be visited within the same route. Therefore, the salesman needs to plan more than one route that minimize total travel distance while adhering to these constraints. To model these constraints, FTSP-IC introduces an $L \times L$ matrix M_F, where $M_F[l_1][l_2] = true$ indicates that the customers of the family l_1 and l_2 can be visited together in a route. The presence of incompatibility constraints inherently allows for multiple routes in a feasible solution, rendering FTSP-IC more complex compared to the standard FTSP.

Figure 1(b) shows an illustrative example of the FTSP-IC. In this case, there are incompatibility constraints between families 1 and 4 as well as families 2 and 3, which means that the conflicting families cannot be visited in the

* Corresponding author.
Email address: jintong.ren@hhu.edu.cn (Jintong Ren).

same route. Therefore, a feasible solution of the FTSP could be infeasible for the FTSP-IC due to the incompatibility constraints. A feasible FTSP-IC solution containing two routes is shown in Figure 1(b). Because the solution of the FTSP-IC contains multiple routes, its solution representation is a two-dimensional array, where each row of the array represents a route.



(a) Example of the FTSP model

(b) Example of the FTSP-IC model

Fig. 1. Illustrative examples of the FTSP model and the FTSP-IC model.

Like other TSP models (Chisman, 1975; Jünger et al., 1995; Baker, 1983; Anily and Mosheiov, 1994; Gendreau et al., 1996; Feillet et al., 2005; Bektas, 2006; Li et al., 2014; Agatz et al., 2018; Zhu et al., 2022; Liu et al., 2024), the FTSP-IC is widely used in practical applications. For example, we consider the scenario in the filed of supply chain distribution. Suppose that a supply chain network is divided into several regions, each region containing a number of warehouses. Due to geographical limitations, business requirements or operational policy differences, the delivery man needs to visit a specified number of warehouses in each region. Additionally, there are some conflicting constraints between different regions, meaning that the delivery man cannot visit two conflicting regions in the same route. The goal of this problem is to complete the delivery tasks in the supply chain while satisfying the conflicting constraints and minimizing the total traveling distance of the delivery routes. This problem is equivalent to the FTSP-IC when a family corresponds to a region of the supply chain and a node of the family indicates a warehouse of the region. Another example is the problem of planning the route of a patroller or inspector. Suppose they need to visit a certain number of points within a number of regions to be inspected. Due to factors such as human resources, timeliness and regional representativeness, they only need to visit certain nodes within designated areas to understand the overall situation of the entire region. This allows them to cover more areas within the limited patrol time. There are conflicting constraints between some regions due to location or security factors. The objective of the problem is to complete the inspection task with the shortest route while satisfying the constraints. The FTSP-IC model can be used to conveniently represent this scenario, where each family corresponds to a check region and each node corresponds to a check point.

Due to its relevance, the FTSP has drawn growing interest in recent years, which is related to the FTSP-IC studied in this work. Morán-Mirabal et al. (2014) introduced an integer programming model and benchmark instances of the FTSP for the first time and successfully solved small-size instances using the CPLEX solver. They also designed two randomized heuristics: genetic algorithm (GA) and a greedy randomized adaptive search procedure (GRASP) to find good near-optimal solutions of the FTSP. Bernardino and Paias (2018) proposed a number of compact and non-compact formulations of the FTSP for the first time and designed an iterative local search algorithm (ILS) for solving the FTSP effectively. Pop et al. (2018) decomposed FTSP into two subproblems that can be solved separately, thus obtaining several competitive results. The first macro-level subproblem is solved by the GA to determine the tours for visiting the families and the second micro-level subproblem is solved optimally by the Concorde TSP solver (Applegate et al. (2006)) to find the minimum-cost tour. Later, Bernardino and Paias (2021) proposed three novel heuristic approaches to obtain better upper bounds of the FTSP, i.e., a hybrid algorithm that integrates a branch-and-cut algorithm with a local search procedure, an easily implementable GA and an enhanced ILS algorithm.

The above studies provide effective solution methods for solving the FTSP, however, only few algorithms have been introduced for the FTSP-IC. Bernardino and Paias (2022) introduced the compact and non-compact models of this problem for the first time and generated the benchmark instances of the FTSP-IC according to the instances of FTSP. They designed a branch-and-cut algorithm to solve small-size instances optimally and developed two heuristic approaches for large-size instances, i.e., an ant colony optimization (ACO) algorithm and an ILS algorithm. These two state-of-the-art algorithms will be used as the reference algorithms for our comparative experiments.

In this work, we contribute to the advancement of solving the FTSP-IC by introducing a novel intensification-driven search algorithm (IDSA). The main contributions are summarized as follows.

- From the perspective of algorithm design, the IDSA integrates a variable neighborhood search procedure and a dedicated perturbation procedure to strengthen search ability. During the search, the surrounding area of each local optimum will be carefully examined to avoid missing nearby high-quality solutions. The proposed algorithm will conditionally update the center of the search area throughout the search procedure to guide the search towards regions of potential interest.

- From the perspective of experimental results, we report 29 new upper bounds from the 63 benchmark instances in the literature, while matching all the known optimal solutions for the remaining 34 instances. These bounds can serve as references for future research into the FTSP-IC. Moreover, we provide for the first time an instance space analysis of FTSP-IC to observe the algorithmic performance across different areas of the instance space. Furthermore, we will make the code of our IDSA publicly available, providing support for future research on the FTSP-IC and its related real-life applications.

The rest of the paper is organized as follows. We present the proposed algorithm in Section 2. Followed by that, we describe the experimental settings and results in Section 3. In Section 4, an analysis of the key components of the algorithm is given. Conclusions and perspectives are provided in the last section.

2 Intensification-driven search algorithm for the FTSP-IC

The intensification-driven search algorithm (IDSA) proposed in this work is inspired by distance-guided local search (DGLS) (Porumbel and Hao, 2020; Ding et al., 2017). Basically, the DGLS framework is dedicated to enhance the local search (LS) capacity by intensifying the search around known local optima. Through iterative launches of LS procedures within a specified sphere radius, DGLS constructs a tree-like search trajectory instead of a continuous path, reducing the possibility of missing nearby promising solutions. DGLS demonstrates flexibility by enabling the selection and utilization of specific distance-based techniques that have proven to be highly effective for some combinatorial optimization problems, such as the graph coloring problem (Porumbel and Hao, 2020), the capacitated arc routing problem (Porumbel and Hao, 2020), the traveling repairman problem with profit (Ren et al., 2022) and the nearest neighbor search problem (Xu et al., 2021).

The flow chart of our IDSA framework is presented in Figure 2. Basically, IDSA starts from an `InitialSolution` procedure and then iteratively operates between a `LocalOptimization` procedure and a `Perturbation` procedure to find local optima. During the search process, the search area is updated according to the solution quality and the search radius. Finally, IDSA terminates when the stop condition is reached.

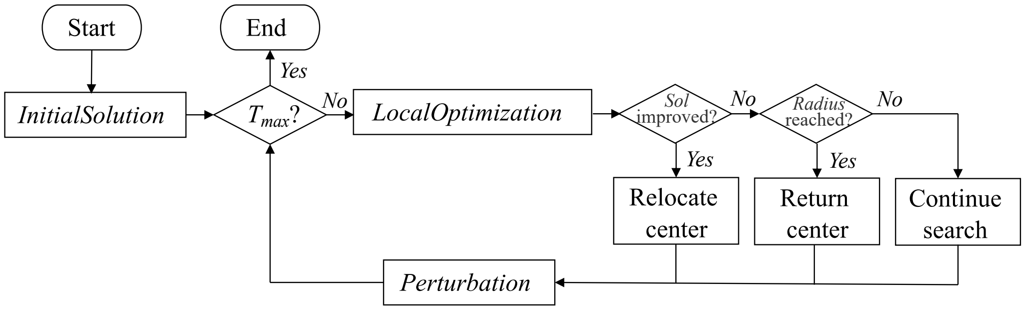


Fig. 2. Flow chart of the IDSA framework.

2.1 *Main framework*

The main framework of our IDSA is shown in Algorithm 1. IDSA starts with several initialization operations (lines 3-5), and then enters the main `while` loop (lines 6-18). During each iteration of the `while` loop, a variable neighborhood search procedure (VNS) is first adopted to obtain the local optimum φ (line 7). Then, the solutions φ, φ^*, and the distance counter Ct will be updated conditionally (lines 8-16). Specifically, three cases are considered: (1) φ^* and Ct will be updated when a better solution φ is found (lines 8-10); (2) Ct will be reinitialized to 0 and φ be updated by φ^* when no better solution is found and the search goes beyond the search sphere (lines 11-13); (3) otherwise, Ct will be updated by 1, which means that the search will continue inside the search sphere (lines 14-16). Then, the random perturbation procedure is employed to drive the search to the new area around the known local optimum (line 17). The above process is repeated until a pre-determined cut-off time is attained and the algorithm returns the best solution φ^* found during the search.

From the view of search space, IDSA always starts from a 'centering solution' (the best-found solution during the search procedure) as the starting solution, and performs the local search procedure multiple times (recorded as Ct, which is used to describe the distance between the current solution and the 'centering solution') to intensify the search. When the 'counter' Ct reaches 'R' (line 11 in Algorithm 1), the algorithm returns to 'centering solution' (line 13 in Algorithm 1) and restarts the search. The `Perturbation` employed random perturbation method, which ensures that the search trajectory will be different for each time search (starting from the 'centering solution'). In this way, the search is restricted to an area (like a sphere of radius 'R') and generates a tree-like trajectory to achieve a detailed search.

2.2 *Greedy initialization procedure*

Algorithm 1 Intensification-driven search algorithm for the FTSP-IC (IDSA)

Input: Input graph $G(V, E)$, radius R of the search sphere, evaluation function f, neighborhoods N_1 to N_8 (see Section 2.3) and cut-off time T_{max}.
Output: Best found solution φ^*.
/* `GreedyIniSol` is used to generate an initial solution. See Section 2.2. */
$\varphi \leftarrow$ `GreedyIniSol`(G)
$\varphi^* \leftarrow \varphi$
$Ct \leftarrow 0$ /* Ct is the distance counter. */
while T_{max} is not reached **do**
    /* `VNS is used to perform the local refinement. See Section 2.3.`*/
    $\varphi \leftarrow$ `VNS`(φ, N_{1-8})
    if $f(\varphi) < f(\varphi^*)$ **then**
        $Ct \leftarrow 0$
        $\varphi^* \leftarrow \varphi$
    else if $Ct \geq R$ **then**
        $Ct \leftarrow 0$
        $\varphi \leftarrow \varphi^*$
    else
        $Ct \leftarrow Ct + 1$
    end if
    /* `RandomPerturb` is used to perturb the local optimum. See Section 2.4.*/
    $\varphi \leftarrow$ `RandomPerturb`(φ)
end while
return φ^*

To obtain an initial solution of good quality, we employ a greedy randomized construction procedure, whose pseudo-code is presented in Algorithm 2. At first, a vertex list V_r is initialized by the nodes from all families (line 3) and a vector γ is generated to store the number of nodes for each family in the solution (line 4). Starting from an empty solution φ, the algorithm constructs the first route and sets the first position of the route to the depot (lines 5-7). Then, the initialization procedure iteratively adds one vertex to φ in a greedy randomized way (lines 8-25). At each iteration, we first filter the nodes incompatible to the current route k to obtain a node set V_a, and construct a candidate set V_b by selecting the $min(p, |V_a|)$ [1] closest nodes with respect to the previous node $\varphi(k, q-1)$ (lines 9-10), where the parameter p is empirically set to 5. If set V_b is not empty, we carry out the add operation based on the current route by randomly choosing a node v from V_b, adding it to the partial solution $\varphi(k, q)$ and removing it from V_r (lines 12-14). After that, the next position on this route is considered (the counter q is increased by 1) and the number of nodes for the family of v is updated (lines 15-16). All the remaining

[1] The size of V_a can be smaller than p. For the purpose of rigorous description, the notation $min(p, |V_a|)$ is introduced to indicate that the maximal size of candidate set V_b is p.

Algorithm 2 Greedy initialization procedure (`GreedyIniSol`)

1: **Input**: Input graph $G(V, E)$ and the maximal size p of the candidate subset V_b.
2: **Output**: Current solution φ.
3: $V_r \leftarrow F_1 \cup F_2 \cup ... \cup F_L$
4: /* $\gamma[l]$ depicts the number of nodes for the family l in the solution.*/
$\gamma \leftarrow [0, 0, ..., 0]$
5: $k \leftarrow 1$
6: $q \leftarrow 1$
7: /*φ is a solution with multiple permutations, where $\varphi(k, q)$ denotes the node on the position q in the k-th route.*/
$\varphi(k, 0) \leftarrow 0$
8: **repeat**
9: $V_a \leftarrow$ subset of V_r with the nodes which are compatible with the route k
10: /* $min(p, |V_a|)$ denotes the smaller value between p and $|V_a|$ */
$V_b \leftarrow$ subset of V_a with $min(p, |V_a|)$ nodes which have the minimum distance respect to the previous visited node $\varphi(p, q-1)$
11: **if** $|V_b| > 0$ **then**
12: $v \leftarrow$ randomly select one node from V_b
13: $\varphi(k, q) \leftarrow v$
14: $V_r \leftarrow V_r \setminus \{v\}$
15: $q \leftarrow q + 1$
16: $\gamma[\mathcal{F}(v)] \leftarrow \gamma[\mathcal{F}(v)] + 1$
/* h_l is the minimal number of nodes to visit for the family l, $\mathcal{F}(v)$ is the family for the node v, and $\gamma[\mathcal{F}(v)]$ depicts the number of visited nodes in family $\mathcal{F}(v)$. A feasible solution requires at least h_l nodes for family l. */
17: **if** $\gamma[\mathcal{F}(v)] = h_l$ **then**
18: $V_r \leftarrow V_r \setminus \{i : i \in V_r \cap F_{\mathcal{F}(v)}\}$
19: **end if**
20: **else**
21: $k \leftarrow k + 1$
22: $q \leftarrow 1$
23: $\varphi(k, 0) \leftarrow 0$
24: **end if**
25: **until** $V_r = \emptyset$
26: **return** φ

nodes coming from $F_{\mathcal{F}(v)}$ will be removed when the salesman visits enough nodes of the family $\mathcal{F}(v)$ (lines 17-19). If V_b is empty, a new route starting at the depot is created (lines 20-24). These steps are repeated until V_r becomes empty (line 25), and a complete feasible solution φ is returned (line 26).

2.3 Variable neighborhood search procedure

Given a solution φ, the `VNS` procedure is employed to discover a local optimum, which is used as the center of a new sphere that is intensively

Algorithm 3 Variable Neighborhood Search (`VNS`)

1: **Input**: Evaluation function f, current solution φ and neighborhoods N_1 to N_8.
2: **Output**: Local best solution φ'.
3: **repeat**
4: $\quad \varphi' \leftarrow \varphi$
5: $\quad$ /* Construct the neighborhood set NL */
$\quad NL \leftarrow \{N_1, N_2, N_3, N_4, N_5, N_6, N_7, N_8\}$
6: $\quad$ **while** $NL \neq \emptyset$ **do**
7: $\qquad$ Randomly select a neighborhood $N \in NL$
8: $\qquad NL \leftarrow NL \setminus \{N\}$
9: $\qquad \varphi \leftarrow$ `LocalSearch`(φ, N)
10: $\quad$ **end while**
11: **until** $f(\varphi) \geq f(\varphi')$
12: **return** φ'

examined by IDSA. As presented in Algorithm 3, the `VNS` procedure iteratively improves the current solution φ by performing local descent with eight different neighborhoods until no better solution φ' can be found. Specifically, the local best solution φ' is first updated by φ (line 4). After creating the set NL containing all the defined neighborhoods (line 5), the algorithm enters a `while` loop to perform the local optimization iteratively (lines 6-10). During each loop, a neighborhood N is randomly selected from the set NL (line 7) and then removed from NL (line 8). Then the descent local search procedure with the first improvement strategy is invoked to improve the current solution φ within the neighborhood N (line 9). The `while` loop terminates when the set NL is empty. Finally, the `VNS` procedure returns the local best solution φ' found so far.

The `VNS` procedure relies on three sets of neighborhoods induced by eight move operators. The first set I is composed of four move operators, which change the orders of the nodes in one route as follows.

- *Swap* (N_1): Exchange the positions of two nodes in one route.
- *Insert* (N_2): Remove one node from its position and insert it between two adjacent nodes in the same route.
- 2-*opt* (N_3): Remove two non-adjacent edges in the same route and replace them with two new edges in the same route.
- *Block-Insert* (N_4): Remove a block of h ($h = 2, 3$) successive nodes from their positions and insert this block between two adjacent nodes in the same route. Figure 3 shows an illustrative example demonstrating the *Block-Insert* operator, where a block containing two consecutive nodes B and C is moved.

The second set II of two move operators is designed for changing nodes between different routes with respect to the incompatibility constraints.

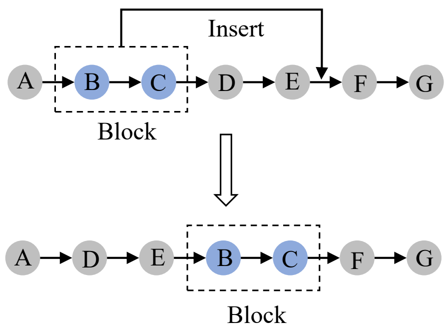


Fig. 3. Illustration of the *Block-Insert* operator: the block with two nodes (B and C) is removed from the original position and inserted between the node E and F without changing the order in the block.

- *Inter-Swap* (N_5): Exchange the positions of two nodes from two different routes by respecting the incompatibility constraints.
- *Inter-Insert* (N_6): Remove one node from its position and insert it between two adjacent nodes in a different route by respecting the incompatibility constraints.

The third set III of two move operators works as follows.

- *Switch* (N_7): Switch a visited node and a non-visited node from the same family.
- *DropAdd* (N_8): Remove a visited node and insert a non-visited node from the same family to any position by respecting the incompatibility constraints.

In this work, the local search procedure is integrated with the multiple neighborhoods and the first-improvement strategy, which can be simply summarized as follows. For each neighborhood, the algorithm evaluates neighboring solutions one by one until an improving solution is encountered or all the neighboring solutions in this neighborhood are evaluated. The current solution is replaced with the first-encountered improving solution (first-improvement strategy). Then, the algorithm switches to another neigborhood and repeats the same procedure. The local search procedure sequentially explores the above neighborhoods in a random order to search for high-quality local optimal solutions. Note that the *Switch* move operator has already been proposed in Bernardino and Paias (2022), while *DropAdd* is a new move operator specially designed for the FTSP-IC. In fact, the *Switch* operator is a subset of the *DropAdd* operator. The influence of these two key move operators will be analyzed in Section 4.2.

2.4 Random perturbation procedure

Algorithm 4 Random perturbation procedure (`RandomPerturb`)

1: **Input**: Evaluation function f, the strength k_1 of $Insert$ operation, the strength k_2 of *DropAdd* operation and current solution φ.

2: **Output**: Perturbed solution φ.

3: $Cp \leftarrow 0$ /* Cp is the perturbation counter. */

4: **while** $Cp < k_1$ **do**

5: $\quad \varphi \leftarrow Insert(\varphi)$

6: $\quad Cp \leftarrow Cp + 1$

7: **end while**

8: $Cp \leftarrow 0$

9: **while** $Cp < k_2$ **do**

10: $\quad \varphi \leftarrow DropAdd(\varphi)$

11: $\quad Cp \leftarrow Cp + 1$

12: **end while**

13: **return** φ

To help IDSA escape from local optimum traps, a random perturbation procedure, depicted in Algorithm 4, involving two simple operations is adopted. At first, we apply k_1 times the $Insert$ operation to change the input solution, where k_1 is a parameter that indicates the strength of the $Insert$ operation. Specifically, a perturbation step removes a random node and re-inserts the node into a random position between two adjacent nodes in the same route (lines 3-7). After that, the *DropAdd* operator is adopted by randomly removing a node from the solution and inserting a non-visited node from the same family to that random position. The *DropAdd* operation is executed k_2 times where k_2 is a parameter that indicates the strength of the *DropAdd* operation (lines 8-12). At last, the perturbed solution φ is returned (line 13). The value of the parameters k_1 and k_2 are given in Section 3.2. Note that when we insert a non-visited node to the solution, the incompatibility constraints are satisfied to ensure the feasibility of the new obtained solution.

2.5 Discussion

The proposed algorithm IDSA differs from the reference algorithm ILS (Bernardino and Paias, 2022) mainly in two aspects. On the one hand, the proposed algorithm adopted an intensification-driven framework, which can significantly intensify the search by performing extensive searches in the nearby area of discovered local optima. On the other hand, three new neighborhood operators are introduced ($Insert$, *Block-*$Insert$ and *Drop-Add*) to enhance the search ability of the proposed algorithm in the local search procedure. The important role of each component is revealed in Section 4 by extensive experiments.

It is worth noting that the proposed algorithm IDSA in this work is different from the algorithm in Ren et al. (2022) in terms of the detailed components. The problem studied in Ren et al. (2022) is the traveling repairman problem with profits (TRPP), which is totally different from the FTSP-IC in the representation of the solution, the objective function as well as the constraints. TRPP's solution comprises a single route, while FTSP-IC needs to find a solution with multiple routes. TRPP's objective function is to maximize the collecting profits (related to the cumulative traveling distance), but FTSP-IC is to minimize the traveling distance. FTSP-IC considers the incompatibility constraints on one route, which is not included in TRPP. These differences lead to a more complex construction heuristic (considering the incompatibility constraints and the solution formulation) and more types of neighborhoods of VNS (considering the operations among the routes). In summary, these two algorithms for solving TRPP and FTSP-IC are quite different in the detailed components.

For the computational complexity, each neighboring solution stated in Section 2.3 could be evaluated in $O(1)$ by only calculating the changing part of the fitness function. Therefore, the computational complexity of evaluating the solutions in each neighborhood is $O(n^2)$ (n is the number of nodes) and the complexity of each iteration is also $O(n^2)$. The complexity of the whole algorithm depends on the iterations of execution, which could be controlled by an input parameter. Considering the notation 'iteration' may vary significantly between different algorithms, we adopt the cutoff-time as the stopping condition in experiments for a fair comparison.

3 Computational results and comparisons

This section is devoted to evaluating the performance and drawing comparisons between IDSA and the state-of-the-art algorithms for the FTSP-IC. The benchmark instances and the source code of IDSA will be publicly available at `https://github.com/Zequn-Wei/FTSP-IC_IDSA`.

3.1 Benchmark Instances

The 63 FTSP-IC benchmark instances used in this paper were originally proposed in Bernardino and Paias (2022), which are derived from the instances of the FTSP introduced in Morán-Mirabal et al. (2014). These benchmark instances are divided into seven groups according to the number of nodes ranging from 14 to 1002. Each group of instances is further distinguished based on the number of nodes to be visited per family and conflict density for incompatible families. For example, Gn_x_d designates that a FTSP-IC instance with type G and n nodes (including the depot point), category x of nodes to be visited per family and conflict density of d, where $d \in \{0.30, 0.60, 0.90\}$. Specifically, d is given by $d = 2\lambda/\mu(\mu - 1)$ where λ is the number of incompatible family pairs and μ is the number of families.

3.2 Experimental settings

Reference algorithms. We adopt the two state-of-the-art FTSP-IC algorithms proposed in Bernardino and Paias (2022), i.e., the ACO and the ILS. Considering that the experimental results for these two algorithms were obtained under different termination conditions, including the maximum number of iterations and the time limit, we select the best result reported for each instance as the reference result. Moreover, we also include the best lower bound (LB) reported in Bernardino and Paias (2022).

Computing platform. The source code of our IDSA is implemented in C++ and compiled using the g++ compiler with the -O3 option. All experiments were executed in multiple threads and performed on an Intel Xeon 6148 processor (2.40 GHz CPU) under the Linux operating system. Note that all the results of the reference algorithms were obtained on an AMD Ryzen5 2600 processor (3.40 GHz CPU).

Stopping condition. To ensure a fair comparison, we adopt the cut-off time reported in Bernardino and Paias (2022) as the stopping condition for our IDSA. The detailed cut-off time T_{max} of each instances is presented in Table 2. Note that no cut-off time is provided in Bernardino and Paias (2022) for three groups of small-size instances, i.e., $burma14$, $bayg29$ and $att48$. Thus, we simply set T_{max} to a small value of 5 seconds for these instances. Moreover, each instance was solved independently 5 times with different random seeds, which is the same as the settings used for the reference algorithms.

Parameter setting. The proposed IDSA requires three parameters whose values were determined automatically by the 'IRACE' tool (López-Ibáñez et al., 2016), i.e., the radius R of the search sphere, the perturbation strength k_1 of the *Insert* operator and the perturbation strength k_2 of the *DropAdd* operator. The 'IRACE' tuning experiment was carried out on six representative benchmark instances with the same cut-off time as the time used by IDSA. The range of candidate parameter values for the 'IRACE' tool and the final values are shown in the last two columns of Table 1.

Table 1
Parameter settings of the IDSA.

Parameter	Description	Type	'IRACE' Range	Final value
R	Radius of the search sphere	Integer	$[0, 100]$	70
k_1	Perturbation strength of *Insert*	Integer	$[10, 50, 100, 300, 500, 1000]$	300
k_2	Perturbation strength of *DropAdd*	Integer	$[0, 50]$	3

3.3 Experimental results

The computational results from our IDSA along with those attained by each reference algorithm are reported in Table 2. Column 1 shows the name of each instance and Column 2 represents the lower bounds (LB). Column 3 shows the best-known values (BKV) of each instance reported in the literature while the asterisk (*) indicates a known proven optimal value. Columns 4-9 report the best results (f_{best}) and average results (f_{avg}) achieved by the two reference algorithms (ILS and ACO) and our IDSA. We also report the average run times t_{avg} to obtain the f_{best} value of IDSA in column 10. The last column shows the gap between the best results of IDSA and BKV, where $Gap = (f_{best} - BKV)/BKV$. Moreover, we also present the average value #Avg of each column, the p-values obtained from the Wilcoxon tests between IDSA and each reference algorithm, and the comparative statistical results in the last three rows of the table. Specifically, W/M/F indicates the number of instances for which IDSA performs better (W), equally well (M) and worse (F) compared to each reference algorithm. Furthermore, the best results of each instance are highlighted in bold.

From Table 2, we can observe that our IDSA dominates the two reference algorithms by achieving better or equal results for all the 63 benchmark instances without exception. Specifically, IDSA is able to obtain 42 and 38 better f_{best} values compared to ILS and ACO, respectively, while matching all the remaining results. When comparing with the BKV values, IDSA achieves all the 34 known−optimal results and improves the remaining 29 best-known results of the literature (see the negative Gaps in the last column). Moreover, the small p-values ($\ll 0.05$) indicate that the differences between IDSA and each reference algorithm are statistically significant.

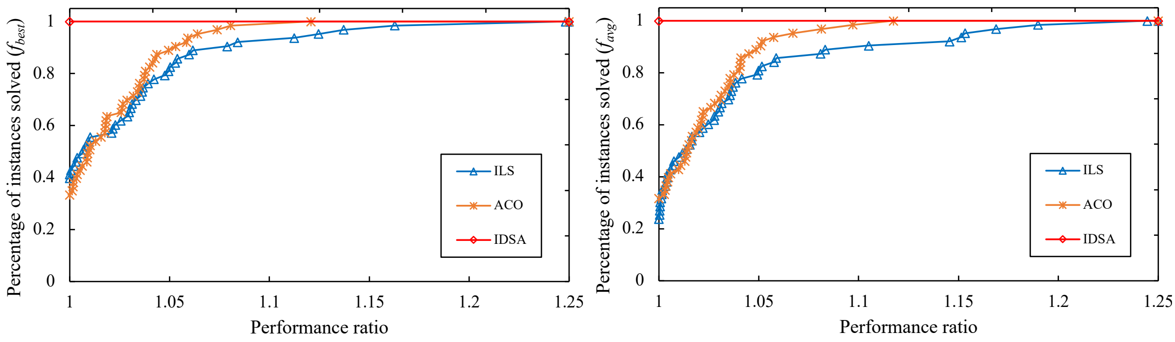
To complete the performance assessment of the compared algorithms, we present the performance profiles (see Dolan and Moré (2002) for more details) on the 63 benchmark instances. Given a set $\mathcal{P}$ of instances to be tested and a set $\mathcal{A}$ of compared algorithms. Since the FTSP-IC studied in this work is a minimization problem, we define the performance ratio by $r_{g,a} = \frac{f_{g,a}}{min\{f_{g,a}:a\in\mathcal{A}\}}$, where $f_{g,a}$ represents the objective values of f_{best} or f_{avg}

Table 2
Experimental results of the IDSA and the reference algorithms, the maximal running time (in seconds) for each instance is listed in the column $T_{max}(s)$, each instance is executed independently 5 times with different seeds, in multiple threads, and on an Intel Xeon 6148 processor (2.40 GHz CPU) under the Linux operating system.

Instance	LB	BKV	ILS		ACO		IDSA				
			f_{best}	f_{avg}	f_{best}	f_{avg}	f_{best}	f_{avg}	$t_{avg}(s)$	$T_{max}(s)$	*Gap*
burma14_1_0.30	17.0	17.00*	**17.00**	**17.00**	**17.00**	**17.00**	**17.00**	**17.00**	0.00	5.00	0.00 %
burma14_1_0.60	20.47	20.47*	**20.47**	**20.47**	**20.47**	**20.47**	**20.47**	**20.47**	0.00	5.00	0.00 %
burma14_1_0.90	21.57	21.57*	**21.57**	**21.57**	**21.57**	**21.57**	**21.57**	**21.57**	0.00	5.00	0.00 %
burma14_2_0.30	28.84	28.84*	**28.84**	**28.84**	**28.84**	**28.84**	**28.84**	**28.84**	0.00	5.00	0.00 %
burma14_2_0.60	34.35	34.35*	**34.35**	**34.35**	**34.35**	**34.35**	**34.35**	**34.35**	0.00	5.00	0.00 %
burma14_2_0.90	37.78	37.78*	**37.78**	**37.78**	**37.78**	**37.78**	**37.78**	**37.78**	0.00	5.00	0.00 %
burma14_3_0.30	13.2	13.20*	**13.20**	**13.20**	**13.20**	**13.20**	**13.20**	**13.20**	0.00	5.00	0.00 %
burma14_3_0.60	13.8	13.80*	**13.80**	**13.80**	**13.80**	**13.80**	**13.80**	**13.80**	0.00	5.00	0.00 %
burma14_3_0.90	14.9	14.90*	**14.90**	**14.90**	**14.90**	**14.90**	**14.90**	**14.90**	0.00	5.00	0.00 %
bayg29_1_0.30	5657.45	5657.45*	**5657.45**	**5657.45**	**5657.45**	5750.80	**5657.45**	**5657.45**	0.00	5.00	0.00 %
bayg29_1_0.60	7560.39	7560.39*	**7560.39**	**7560.39**	**7560.39**	**7560.39**	**7560.39**	**7560.39**	0.00	5.00	0.00 %
bayg29_1_0.90	9795.97	9795.97*	**9795.97**	**9795.97**	**9795.97**	**9795.97**	**9795.97**	**9795.97**	0.00	5.00	0.00 %
bayg29_2_0.30	6917.82	6917.82*	**6917.82**	**6917.82**	**6917.82**	**6917.82**	**6917.82**	**6917.82**	0.01	5.00	0.00 %
bayg29_2_0.60	8269.98	8269.98*	**8269.98**	**8269.98**	**8269.98**	8275.10	**8269.98**	**8269.98**	0.00	5.00	0.00 %
bayg29_2_0.90	9678.84	9678.84*	**9678.84**	**9678.84**	**9678.84**	**9678.84**	**9678.84**	**9678.84**	0.00	5.00	0.00 %
bayg29_3_0.30	7608.14	7608.14*	**7608.14**	**7608.14**	**7608.14**	7640.30	**7608.14**	**7608.14**	0.00	5.00	0.00 %
bayg29_3_0.60	8516.86	8516.86*	**8516.86**	**8516.86**	8548.59	8567.40	**8516.86**	**8516.86**	0.00	5.00	0.00 %
bayg29_3_0.90	10556.8	10556.80*	**10556.80**	**10556.80**	**10556.80**	10563.86	**10556.80**	**10556.80**	0.00	5.00	0.00 %
att48_1_0.30	37531.2	37531.20*	37858.40	37903.22	37876.70	38154.30	**37531.20**	**37531.20**	0.00	5.00	0.00 %
att48_1_0.60	45476.2	45476.20*	45627.00	45679.36	**45476.20**	45670.26	**45476.20**	**45476.20**	0.02	5.00	0.00 %
att48_1_0.90	60267.7	60267.70*	60367.50	60427.02	60302.00	60681.00	**60267.70**	**60267.70**	0.01	5.00	0.00 %
att48_2_0.30	31659.1	31659.10*	31859.00	32071.20	31660.20	31704.86	**31659.10**	**31659.10**	0.03	5.00	0.00 %
att48_2_0.60	33752.2	33752.20*	33789.90	34409.52	**33752.20**	33803.96	**33752.20**	**33752.20**	0.72	5.00	0.00 %
att48_2_0.90	40444.7	40444.70*	41197.90	41197.90	**40444.70**	40461.74	**40444.70**	**40444.70**	0.00	5.00	0.00 %
att48_3_0.30	14358.0	14358.00*	**14358.00**	**14358.00**	**14358.00**	14363.24	**14358.00**	**14358.00**	0.00	5.00	0.00 %
att48_3_0.60	16397.2	16397.20*	**16397.20**	16449.28	**16397.20**	**16397.20**	**16397.20**	**16397.20**	0.00	5.00	0.00 %
att48_3_0.90	20066.3	20066.30*	**20066.30**	**20066.30**	**20066.30**	**20066.30**	**20066.30**	**20066.30**	0.00	5.00	0.00 %
bier127_1_0.30	42968.1	46080.50	46080.50	46362.76	46299.00	46635.66	**45267.90**	**45368.08**	6.71	10.00	-1.76 %
bier127_1_0.60	47482.2	47482.20*	47963.70	48135.50	47625.00	48140.44	**47482.20**	**47482.20**	0.01	7.00	0.00 %
bier127_1_0.90	67336.9	67336.90*	67686.60	67735.82	**67336.90**	67560.22	**67336.90**	**67336.90**	0.00	6.00	0.00 %
bier127_2_0.30	113553.0	121356.00	121356.00	121783.40	121973.00	122223.20	**119484.00**	**119790.40**	7.78	13.00	-1.54 %
bier127_2_0.60	124486.0	124486.00*	128809.00	129485.60	127168.00	129090.20	**124486.00**	**124486.00**	4.53	10.00	0.00 %
bier127_2_0.90	157638.0	157638.00*	157915.00	158198.00	157966.00	158254.40	**157638.00**	**157638.00**	1.48	7.00	0.00 %
bier127_3_0.30	55369.9	55369.90*	55867.10	57864.12	56788.30	57066.40	**55369.90**	**55369.90**	2.15	8.00	0.00 %
bier127_3_0.60	62061.1	62061.10*	62586.60	62947.78	62570.90	63019.88	**62061.10**	**62061.10**	1.65	6.00	0.00 %
bier127_3_0.90	76174.7	76174.70*	76463.30	76536.16	**76174.70**	76265.94	**76174.70**	**76174.70**	0.29	5.00	0.00 %
a280_1_0.30	2249.73	3039.67	3039.67	3103.64	3071.44	3099.91	**2985.35**	**3025.20**	66.14	112.00	-1.79 %
a280_1_0.60	2986.27	3826.12	3826.12	3849.54	3831.76	3836.09	**3700.53**	**3733.25**	58.30	74.00	-3.28 %
a280_1_0.90	5444.79	5605.77	5665.02	5737.80	5605.77	5623.84	**5567.46**	**5567.55**	20.26	51.00	-0.68 %
a280_2_0.30	2029.25	2874.99	2874.99	2882.85	2903.70	2936.92	**2786.26**	**2851.99**	60.69	93.00	-3.09 %
a280_2_0.60	2767.25	3695.29	3740.03	3744.75	3695.29	3725.43	**3583.62**	**3597.53**	41.19	62.00	-3.02 %
a280_2_0.90	5254.75	5390.06	5548.48	5551.81	5390.06	5399.40	**5335.25**	**5335.34**	25.32	41.00	-1.02 %
a280_3_0.30	1866.72	2762.10	2764.53	2772.91	2762.10	2788.57	**2665.36**	**2677.52**	45.88	82.00	-3.50 %
a280_3_0.60	2528.23	3510.15	3565.11	3580.22	3510.15	3546.83	**3385.88**	**3439.06**	40.36	57.00	-3.54 %
a280_3_0.90	5256.26	5385.75	5422.34	5440.38	5385.75	5392.64	**5352.99**	**5352.99**	11.54	38.00	-0.61 %
gr666_1_0.30	1568.46	2603.54	2603.54	2688.85	2820.96	2997.12	**2508.31**	**2564.41**	709.84	836.00	-3.66 %
gr666_1_0.60	2224.41	3786.99	3786.99	3807.12	3830.89	3888.75	**3608.44**	**3703.69**	371.09	500.00	-4.71 %
gr666_1_0.90	4218.3	5014.68	5114.51	5126.12	5014.68	5081.20	**4763.10**	**4804.21**	250.43	294.00	-5.02 %
gr666_2_0.30	1310.14	2337.92	2337.92	2464.30	2382.80	2573.36	**2208.56**	**2246.32**	634.71	779.00	-5.53 %
gr666_2_0.60	1975.66	3334.47	3334.47	3368.83	3345.10	3367.39	**3085.92**	**3115.50**	404.34	490.00	-7.45 %
gr666_2_0.90	3754.53	4440.32	4440.32	4494.25	4474.41	4494.23	**4263.31**	**4273.69**	269.73	292.00	-3.99 %
gr666_3_0.30	1269.32	2421.06	2421.06	2468.26	2456.77	2543.20	**2160.31**	**2208.94**	583.93	787.00	-10.77%
gr666_3_0.60	1834.92	3251.59	3251.59	3297.32	3253.26	3301.22	**3069.83**	**3117.99**	432.86	497.00	-5.59 %
gr666_3_0.90	3732.65	4381.93	4433.85	4457.32	4381.93	4453.71	**4252.08**	**4303.64**	208.77	286.00	-2.96 %
pr1002_1_0.30	NA	259286.00	259286.00	269518.20	298480.00	306177.80	**256723.00**	**265505.20**	3540.28	4253.00	-0.99 %
pr1002_1_0.60	NA	384544.00	384544.00	388290.00	392447.00	398616.80	**374619.00**	**379872.40**	1999.88	2552.00	-2.58 %
pr1002_1_0.90	NA	551845.00	551845.00	554247.80	554164.00	556249.40	**533217.00**	**535695.60**	1303.39	1519.00	-3.38 %
pr1002_2_0.30	NA	275893.00	275893.00	290893.80	298346.00	333587.80	**268200.00**	**280370.00**	3245.61	4695.00	-2.79 %
pr1002_2_0.60	NA	393510.00	393510.00	398189.20	404173.00	430939.20	**383491.00**	**390004.00**	2303.96	2744.00	-2.55 %
pr1002_2_0.90	NA	568551.00	568551.00	569907.80	572606.00	574746.80	**554279.00**	**562341.00**	1345.32	1646.00	-2.51 %
pr1002_3_0.30	NA	255858.00	255858.00	261017.20	300204.00	309114.40	**240475.00**	**248360.00**	3004.28	3985.00	-6.01 %
pr1002_3_0.60	NA	358443.00	358443.00	365692.60	369860.00	387871.20	**352176.00**	**358067.20**	2426.14	2533.00	-1.75 %
pr1002_3_0.90	NA	538003.00	542094.00	543124.60	538003.00	544730.20	**522460.00**	**525672.00**	1451.72	1576.00	-2.89 %
#Avg	-	75972.98	76177.90	77080.83	78340.17	80247.54	**74300.64**	**75273.50**	394.94	493.35	-1.57 %
p-value	-	-	1.65×10^{-8}	-	7.74×10^{-8}	-	-	-	-	-	-
W/M/F	-	-	42/21/0	43/20/0	38/25/0	48/15/0	-	-	-	-	-

for instance g obtained from algorithm a. Then, we obtain the performance profiles shown in Figure 4 based on the performance ratio $r_{g,a}$ of both f_{best} and f_{avg}. The X-*axis* indicates the performance ratio and the Y-*axis* represents the proportion of the number of instances in which algorithm a attains the best values ($min\{f_{g,a} : a \in \mathcal{A}\}$), with a maximum value of 1.

From Figure 4, we can clearly observe that our IDSA can achieve all the best values of the set $\mathcal{A}$ of tested algorithms, while the reference algorithms fail on more than 60% of instances according to both f_{best} and f_{avg} indicators. These outcomes provide further evidences on the dominance of IDSA over the reference algorithms.



(a) f_{best} indicators (b) f_{avg} indicators

Fig. 4. Performance profiles of the compared algorithms.

4 Analysis

In this section, we analyzed two important components of IDSA as well as the instance space to observe their influence on the performance of our algorithm. The experiments are mainly based on 36 large-size benchmark instances, whose optimal results are still unknown in most cases.

4.1 Influence of the intensification-driven framework

The intensification-driven framework has significant influences in guiding the search towards nearby promising areas. To see its impacts on the performance of IDSA, we create an IDSA variant (denoted by *IDSA-noInten*) by disabling the intensification-driven part. Thus, the *IDSA-noInten* variant can be regarded as an ILS algorithm with the VNS procedure introduced in Section 3. Then we run IDSA and *IDSA-noInten* with the default experimental settings introduced in Section 3.2.

The computational results are shown in Table 3. Column 1 gives the names of the 36 instances tested. The remaining columns present the same performance indicators as shown in Table 2 and the standard deviations *Std* over 5 runs. Moreover, we also show #Avg, *p-value* and W/M/F in the last three rows of Table 3.

Table 3
Comparison between the IDSA and the *IDSA-noInten* variant.

Instance	*IDSA-noInten*				IDSA				*Gap*
	f_{best}	f_{avg}	$t_{avg}(s)$	*Std*	f_{best}	f_{avg}	$t_{avg}(s)$	*Std*	
bier127_1_0.30	45349.10	45396.32	3.84	91.05	**45267.90**	**45368.08**	6.71	108.43	-0.18 %
bier127_1_0.60	**47482.20**	**47482.20**	0.01	0.00	**47482.20**	**47482.20**	0.01	0.00	0.00 %
bier127_1_0.90	**67336.90**	**67336.90**	0.00	0.00	**67336.90**	**67336.90**	0.00	0.00	0.00 %
bier127_2_0.30	**119399.00**	120225.20	4.75	424.54	119484.00	**119790.40**	7.78	231.89	0.07 %
bier127_2_0.60	**124486.00**	125020.80	4.09	402.84	**124486.00**	**124486.00**	4.53	0.00	0.00 %
bier127_2_0.90	**157638.00**	**157638.00**	1.60	0.00	**157638.00**	**157638.00**	1.48	0.00	0.00 %
bier127_3_0.30	**55369.90**	**55369.90**	1.96	0.00	**55369.90**	**55369.90**	2.15	0.00	0.00 %
bier127_3_0.60	**62061.10**	**62061.10**	3.02	0.00	**62061.10**	**62061.10**	1.65	0.00	0.00 %
bier127_3_0.90	**76174.70**	**76174.70**	0.24	0.00	**76174.70**	**76174.70**	0.29	0.00	0.00 %
a280_1_0.30	2992.22	3033.12	47.15	31.70	**2985.35**	**3025.20**	66.14	34.67	-0.23 %
a280_1_0.60	3715.56	3750.89	37.12	19.22	**3700.53**	**3733.25**	58.30	21.36	-0.40 %
a280_1_0.90	5568.24	5568.73	30.23	0.34	**5567.46**	**5567.55**	20.26	0.18	-0.01 %
a280_2_0.30	2819.56	2873.90	34.88	29.23	**2786.26**	**2851.99**	60.69	45.34	-1.18 %
a280_2_0.60	3590.88	3612.70	32.15	19.46	**3583.62**	**3597.53**	41.19	24.41	-0.20 %
a280_2_0.90	**5335.25**	5336.01	16.01	0.65	**5335.25**	**5335.34**	25.32	0.19	0.00 %
a280_3_0.30	2683.25	2691.00	34.28	5.25	**2665.36**	**2677.52**	45.88	8.56	-0.67 %
a280_3_0.60	**3385.88**	**3395.30**	28.74	15.63	**3385.88**	3439.06	40.36	64.47	0.00 %
a280_3_0.90	**5352.99**	5353.08	5.87	0.19	**5352.99**	**5352.99**	11.54	0.00	0.00 %
gr666_1_0.30	2545.93	2591.01	426.35	41.55	**2508.31**	**2564.41**	709.84	59.26	-1.48 %
gr666_1_0.60	3703.47	3733.04	220.86	46.68	**3608.44**	**3703.69**	371.09	72.31	-2.57 %
gr666_1_0.90	4814.83	4842.42	182.87	28.44	**4763.10**	**4804.21**	250.43	24.74	-1.07 %
gr666_2_0.30	2277.81	2300.47	297.43	30.98	**2208.56**	**2246.32**	634.71	37.50	-3.04 %
gr666_2_0.60	3175.69	3202.48	213.29	23.10	**3085.92**	**3115.50**	404.34	23.01	-2.83 %
gr666_2_0.90	4314.11	4345.03	131.34	23.12	**4263.31**	**4273.69**	269.73	16.19	-1.18 %
gr666_3_0.30	2172.64	2228.63	418.00	48.95	**2160.31**	**2208.94**	583.93	35.81	-0.57 %
gr666_3_0.60	3090.40	3142.02	344.90	30.56	**3069.83**	**3117.99**	432.86	44.02	-0.67 %
gr666_3_0.90	4343.56	4367.92	191.63	16.00	**4252.08**	**4303.64**	208.77	28.54	-2.11 %
pr1002_1_0.30	264968.00	270789.80	2991.25	3231.15	**256723.00**	**265505.20**	3540.28	6300.79	-3.11 %
pr1002_1_0.60	377990.00	385452.20	1228.20	5643.38	**374619.00**	**379872.40**	1999.88	3633.14	-0.89 %
pr1002_1_0.90	**531952.00**	**534978.80**	979.71	2224.79	533217.00	535695.60	1303.39	3541.47	0.24 %
pr1002_2_0.30	280319.00	284656.60	2279.81	2750.07	**268200.00**	**280370.00**	3245.61	9837.24	-4.32 %
pr1002_2_0.60	397063.00	400991.00	1671.96	3612.74	**383491.00**	**390004.00**	2303.96	5534.29	-3.42 %
pr1002_2_0.90	555637.00	562918.80	628.83	4565.40	**554279.00**	**562341.00**	1345.32	4392.40	-0.24 %
pr1002_3_0.30	247625.00	252589.40	1139.89	3500.46	**240475.00**	**248360.00**	3004.28	8420.00	-2.89 %
pr1002_3_0.60	359301.00	368417.60	1091.08	5603.69	**352176.00**	**358067.20**	2426.14	4259.62	-1.98 %
pr1002_3_0.90	524196.00	528161.00	1129.00	2306.16	**522460.00**	**525672.00**	1451.72	3315.86	-0.33 %
Average	121117.50	122556.34	440.34	965.76	119617.31	121319.82	691.13	1392.10	-0.98 %
p-value	2.68×10^{-4}	-	-	-	-	-	-	-	-
W/M/F	24/10/2	28/6/2	-	-	-	-	-	-	-

From Table 3 we can observe that IDSA significantly outperforms the *IDSA-noInten* variant by obtaining 24 better f_{best} results and 28 better f_{avg} results, while matching most of the remaining results. A significant difference between IDSA and *IDSA-noInten* is indicated by the small *p-value* derived from the Wilcoxon test. These outcomes provide evidence for the advantages of the intensification-driven framework for solving the FSTP-IC.

4.2 Influence of the Switch and DropAdd move operators

The proposed IDSA relies on three sets of move operators to achieve the VNS procedure 3. Our preliminary experiments show that all these neighborhoods have an impact on the algorithm's performance. Here we analyze the *DropAdd* and *Switch* move operators of set III specifically designed for the FTSP-IC. For this experiment, we create two variants by disabling the *DropAdd* (denoted by *IDSA-noDropAdd*) operator and the *Switch* (denoted by *IDSA-noSwi*) operator of IDSA, respectively. Then, we run the two variants with the same settings as in Section 4.1. The computational results are presented in Table 4 (for *IDSA-noDropAdd*) and Table 5 (for *IDSA-noSwi*) with the same performance indicators as in Table 3.

Table 4
Comparison between the IDSA and the *IDSA-noDropAdd* variant.

Instance	*IDSA-noDropAdd*				IDSA				*Gap*
	f_{best}	f_{avg}	$t_{avg}(s)$	*Std*	f_{best}	f_{avg}	$t_{avg}(s)$	*Std*	
bier127_1_0.30	45352.50	45676.94	8.13	263.09	**45267.90**	**45368.08**	6.71	108.43	-0.19 %
bier127_1_0.60	**47482.20**	**47482.20**	3.78	0.00	**47482.20**	**47482.20**	0.01	0.00	0.00 %
bier127_1_0.90	**67336.90**	**67336.90**	0.96	0.00	**67336.90**	**67336.90**	0.00	0.00	0.00 %
bier127_2_0.30	120975.00	121485.60	10.49	687.66	**119484.00**	**119790.40**	7.78	231.89	-1.23 %
bier127_2_0.60	126284.00	126956.40	7.41	597.90	**124486.00**	**124486.00**	4.53	0.00	-1.42 %
bier127_2_0.90	157837.00	157912.20	4.99	77.92	**157638.00**	**157638.00**	1.48	0.00	-0.13 %
bier127_3_0.30	**55369.90**	55638.52	2.73	198.56	**55369.90**	**55369.90**	2.15	0.00	0.00 %
bier127_3_0.60	**62061.10**	62076.28	2.74	15.48	**62061.10**	**62061.10**	1.65	0.00	0.00 %
bier127_3_0.90	**76174.70**	76196.80	2.26	23.52	**76174.70**	**76174.70**	0.29	0.00	0.00 %
a280_1_0.30	3054.29	3108.26	98.85	41.74	**2985.35**	**3025.20**	66.14	34.67	-2.26 %
a280_1_0.60	3751.60	3783.94	57.18	18.02	**3700.53**	**3733.25**	58.30	21.36	-1.36 %
a280_1_0.90	5579.45	5582.60	41.11	2.93	**5567.46**	**5567.55**	20.26	0.18	-0.21 %
a280_2_0.30	2914.53	2927.63	74.63	11.48	**2786.26**	**2851.99**	60.69	45.34	-4.40 %
a280_2_0.60	3674.17	3681.64	50.16	8.70	**3583.62**	**3597.53**	41.19	24.41	-2.46 %
a280_2_0.90	5343.63	5349.63	32.60	5.10	**5335.25**	**5335.34**	25.32	0.19	-0.16 %
a280_3_0.30	2683.94	2715.29	70.16	28.10	**2665.36**	**2677.52**	45.88	8.56	-0.69 %
a280_3_0.60	3529.15	3538.62	41.66	6.00	**3385.88**	**3439.06**	40.36	64.47	-4.06 %
a280_3_0.90	5353.92	5362.73	25.85	8.07	**5352.99**	**5352.99**	11.54	0.00	-0.02 %
gr666_1_0.30	2733.80	2921.59	723.99	103.96	**2508.31**	**2564.41**	709.84	59.26	-8.25 %
gr666_1_0.60	3829.69	3848.69	467.21	10.43	**3608.44**	**3703.69**	371.09	72.31	-5.78 %
gr666_1_0.90	4783.02	4820.00	222.94	38.22	**4763.10**	**4804.21**	250.43	24.74	-0.42 %
gr666_2_0.30	2463.38	2531.69	655.26	64.92	**2208.56**	**2246.32**	634.71	37.50	-10.34%
gr666_2_0.60	3376.25	3451.11	361.93	39.35	**3085.92**	**3115.50**	404.34	23.01	-8.60 %
gr666_2_0.90	4324.32	4372.05	266.16	27.90	**4263.31**	**4273.69**	269.73	16.19	-1.41 %
gr666_3_0.30	2322.47	2413.95	723.18	84.27	**2160.31**	**2208.94**	583.93	35.81	-6.98 %
gr666_3_0.60	3399.76	3404.99	354.78	6.50	**3069.83**	**3117.99**	432.86	44.02	-9.70 %
gr666_3_0.90	4326.25	4399.18	263.63	62.07	**4252.08**	**4303.64**	208.77	28.54	-1.71 %
pr1002_1_0.30	269830.00	276416.60	3540.03	4819.41	**256723.00**	**265505.20**	3540.28	6300.79	-4.86 %
pr1002_1_0.60	394011.00	412308.20	2215.07	11021.18	**374619.00**	**379872.40**	1999.88	3633.14	-4.92 %
pr1002_1_0.90	535326.00	539544.60	1027.62	4193.81	**533217.00**	**535695.60**	1303.39	3541.47	-0.39 %
pr1002_2_0.30	287559.00	292608.40	4100.73	3834.61	**268200.00**	**280370.00**	3245.61	9837.24	-6.73 %
pr1002_2_0.60	410180.00	420893.00	2299.00	8463.10	**383491.00**	**390004.00**	2303.96	5534.29	-6.51 %
pr1002_2_0.90	556429.00	567989.40	1367.98	6259.89	**554279.00**	**562341.00**	1345.32	4392.40	-0.39 %
pr1002_3_0.30	271413.00	280566.40	3262.08	6299.90	**240475.00**	**248360.00**	3004.28	8420.00	-11.40%
pr1002_3_0.60	356892.00	374421.40	2290.08	13063.04	**352176.00**	**358067.20**	2426.14	4259.62	-1.32 %
pr1002_3_0.90	526688.00	532896.80	1315.36	3605.63	**522460.00**	**525672.00**	1451.72	3315.86	-0.80 %
Average	123184.58	125739.45	722.02	1777.57	119617.31	121319.82	691.13	1392.10	-3.03 %
p-value	1.17×10^{-6}	-	-	-	-	-	-	-	-
W/M/F	31/5/0	34/2/0	-	-	-	-	-	-	-

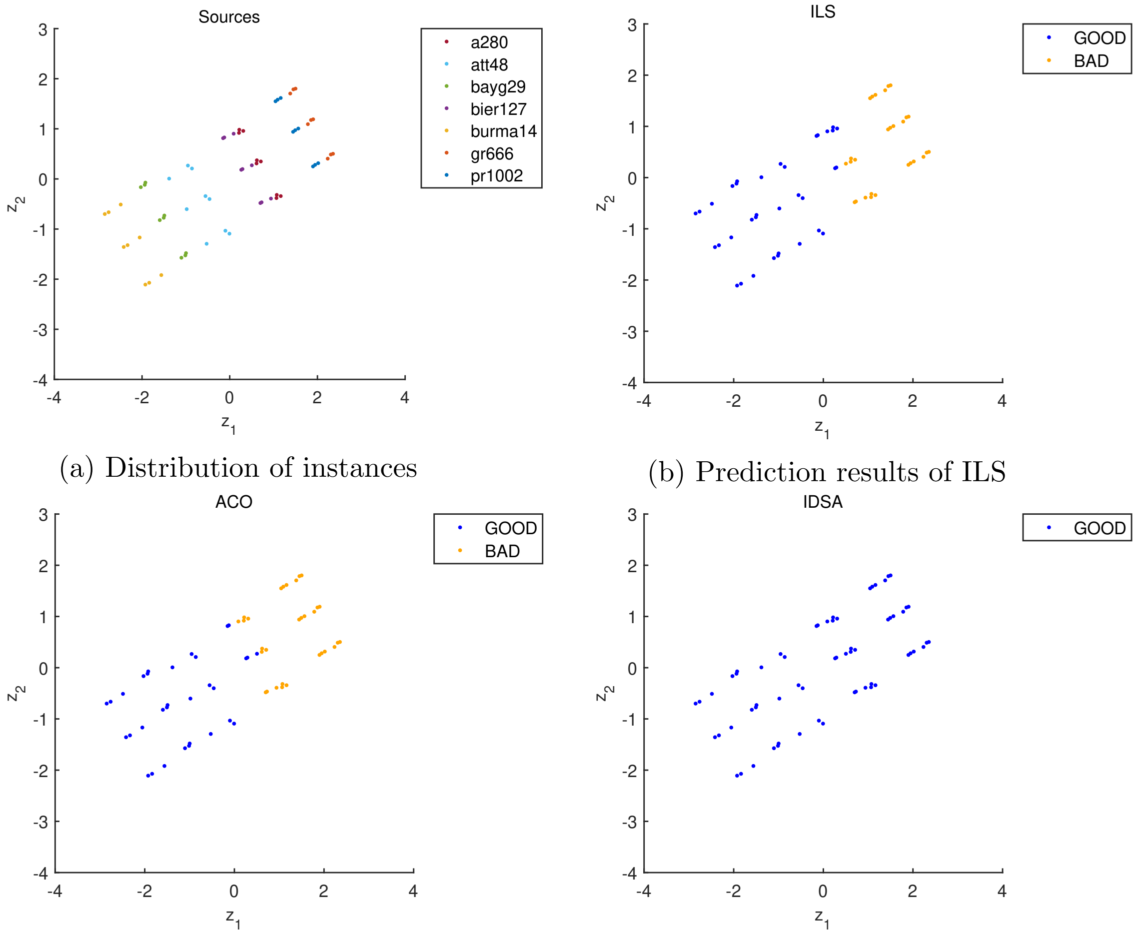
From Table 4, we can clearly observe that our IDSA dominates *IDSA-noDropAdd* by reporting better or equal results for all the 36 instances without exception. Moreover, the small values of t_{avg} and *Std* of IDSA indicate that IDSA is more efficient and robust than *IDSA-noDropAdd*. From Table 5, the small #Avg values of f_{best} from IDSA show that its overall performance is slightly better than the *IDSA-noSwi* variant. However, the similar #Avg values of f_{avg} and t_{avg} indicate that there is little difference in stability and efficiency between IDSA and *IDSA-noSwi*. Moreover, the p-values of Table 5 also display that there are no significant differences in performance between *IDSA* and *IDSA-noSwi*. Note that *IDSA-noSwi* has better performance on several large-size instances, it can be seen as an alternative algorithm of the proposed *IDSA*. In this work, we adopted the *Switch* (N_7) in our IDSA to achieve better or equally good results than the reference algorithms (Bernardino and Paias, 2022). These outcomes show that both the *DropAdd* and *Switch* operators contribute to improving the performance of IDSA, especially the *Drop-Add* operator has a greater impact on the proposed algorithm.

Table 5
Comparison between the IDSA and the *IDSA-noSwi* variant.

Instance	*IDSA-noSwi*				IDSA				*Gap*
	f_{best}	f_{avg}	$t_{avg}(s)$	*Std*	f_{best}	f_{avg}	$t_{avg}(s)$	*Std*	
bier127_1_0.30	45322.50	45424.22	2.27	124.58	**45267.90**	**45368.08**	6.71	108.43	-0.12 %
bier127_1_0.60	**47482.20**	**47482.20**	0.10	0.00	**47482.20**	**47482.20**	0.01	0.00	0.00 %
bier127_1_0.90	**67336.90**	**67336.90**	0.00	0.00	**67336.90**	**67336.90**	0.00	0.00	0.00 %
bier127_2_0.30	**119300.00**	119981.20	4.28	521.62	119484.00	**119790.40**	7.78	231.89	0.15 %
bier127_2_0.60	**124486.00**	125008.00	8.79	871.01	**124486.00**	**124486.00**	4.53	0.00	0.00 %
bier127_2_0.90	**157638.00**	**157638.00**	2.00	0.00	**157638.00**	**157638.00**	1.48	0.00	0.00 %
bier127_3_0.30	**55369.90**	**55369.90**	1.69	0.00	**55369.90**	**55369.90**	2.15	0.00	0.00 %
bier127_3_0.60	**62061.10**	**62061.10**	0.53	0.00	**62061.10**	**62061.10**	1.65	0.00	0.00 %
bier127_3_0.90	**76174.70**	**76174.70**	0.06	0.00	**76174.70**	**76174.70**	0.29	0.00	0.00 %
a280_1_0.30	**2943.94**	**2978.77**	87.22	24.89	2985.35	3025.20	66.14	34.67	1.41 %
a280_1_0.60	**3694.77**	**3719.89**	60.65	22.26	3700.53	3733.25	58.30	21.36	0.16 %
a280_1_0.90	**5567.00**	5567.64	17.18	0.55	5567.46	**5567.55**	20.26	0.18	0.01 %
a280_2_0.30	**2785.75**	**2831.98**	73.43	50.22	2786.26	2851.99	60.69	45.34	0.02 %
a280_2_0.60	**3581.21**	3621.24	50.64	31.23	3583.62	**3597.53**	41.19	24.41	0.07 %
a280_2_0.90	**5335.25**	5335.85	22.27	0.76	**5335.25**	**5335.34**	25.32	0.19	0.00 %
a280_3_0.30	2665.82	**2671.36**	61.47	7.19	**2665.36**	2677.52	45.88	8.56	-0.02 %
a280_3_0.60	**3385.88**	3439.83	32.97	66.18	**3385.88**	**3439.06**	40.36	64.47	0.00 %
a280_3_0.90	**5352.99**	**5352.99**	5.78	0.00	**5352.99**	**5352.99**	11.54	0.00	0.00 %
gr666_1_0.30	**2439.14**	**2553.58**	699.27	106.69	2508.31	2564.41	709.84	59.26	2.84 %
gr666_1_0.60	**3547.79**	**3658.09**	430.29	74.01	3608.44	3703.69	371.09	72.31	1.71 %
gr666_1_0.90	**4755.12**	4819.90	222.78	37.08	4763.10	**4804.21**	250.43	24.74	0.17 %
gr666_2_0.30	2243.07	2268.67	628.18	29.60	**2208.56**	**2246.32**	634.71	37.50	-1.54 %
gr666_2_0.60	3100.47	3149.91	407.53	28.48	**3085.92**	**3115.50**	404.34	23.01	-0.47 %
gr666_2_0.90	4269.44	4316.68	260.33	37.82	**4263.31**	**4273.69**	269.73	16.19	-0.14 %
gr666_3_0.30	**2145.77**	**2180.94**	731.65	36.46	2160.31	2208.94	583.93	35.81	0.68 %
gr666_3_0.60	**3053.42**	**3095.86**	398.40	34.58	3069.83	3117.99	432.86	44.02	0.54 %
gr666_3_0.90	4306.74	4329.90	242.87	22.17	**4252.08**	**4303.64**	208.77	28.54	-1.27 %
pr1002_1_0.30	**244302.00**	**258472.20**	3772.22	8074.39	256723.00	265505.20	3540.28	6300.79	5.08 %
pr1002_1_0.60	**366229.00**	**371403.00**	2186.79	5690.52	374619.00	379872.40	1999.88	3633.14	2.29 %
pr1002_1_0.90	**530569.00**	**534631.80**	1224.71	4709.06	533217.00	535695.60	1303.39	3541.47	0.50 %
pr1002_2_0.30	279763.00	282850.80	3267.27	3768.64	**268200.00**	**280370.00**	3245.61	9837.24	-4.13 %
pr1002_2_0.60	395684.00	402682.20	2290.28	5556.88	**383491.00**	**390004.00**	2303.96	5534.29	-3.08 %
pr1002_2_0.90	562468.00	564171.60	935.67	1279.53	**554279.00**	**562341.00**	1345.32	4392.40	-1.46 %
pr1002_3_0.30	240952.00	**245307.80**	2910.73	3531.62	**240475.00**	248360.00	3004.28	8420.00	-0.20 %
pr1002_3_0.60	359073.00	360325.20	2286.01	847.59	**352176.00**	**358067.20**	2426.14	4259.62	-1.92 %
pr1002_3_0.90	522826.00	**525126.40**	1453.56	2579.96	**522460.00**	525672.00	1451.72	3315.86	-0.07 %
Average	120061.41	121315.01	688.33	1060.15	119617.31	121319.82	691.13	1392.10	0.03 %
p-value	8.39×10^{-1}	-	-	-	-	-	-	-	-
W/M/F	12/10/14	16/7/13	-	-	-	-	-	-	-

4.3 Instance space analysis

To further get some insights of the performance of IDSA along with the reference algorithms on the benchmark instances of different features, we present an instance space analysis (ISA) (Smith-Miles et al., 2014). Specifically, ISA allows us to understand the strengths and weaknesses of the compared algorithms over the areas of the instance space. This experiment was carried out on the recently developed toolkit MATILDA (Muñoz and Smith-Miles, 2020; Smith-Miles and Muñoz, 2023), which has been successfully adopted to analyze various of combinatorial optimization problems, such as the variants of the knapsack problem (Smith-Miles et al., 2021; Wei et al., 2023), the max flow problem (Alipour et al., 2023), the clustering problem (Fernandes et al., 2021) and the personnel scheduling problem (Kletzander et al., 2021). The MATILDA framework provides a comprehensive approach to evaluating algorithm performance using instance space analysis. By employing a support vector machine (SVM) classification model, MATILDA predicts whether an algorithm performs well or poorly on the 2D instance space. This method allows for a detailed and intuitive comparison of algorithms, highlighting their strengths and weaknesses across the instance space.



(a) Distribution of instances

(b) Prediction results of ILS

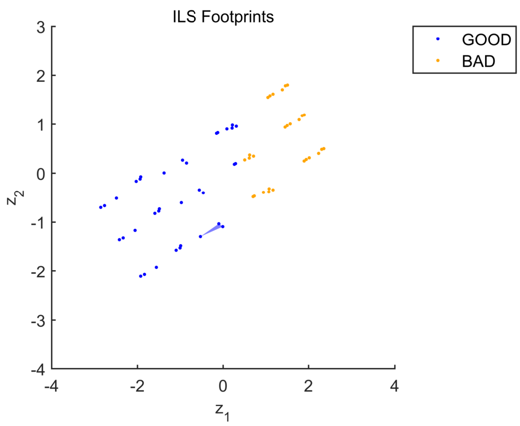
(c) Prediction results of ACO

(d) Prediction results of IDSA

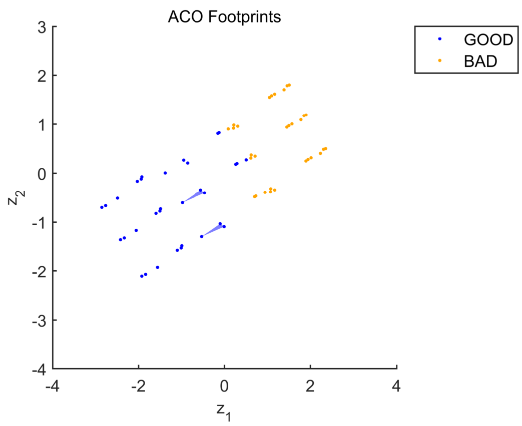
Fig. 5. Distributions of benchmark instances (subfigure (a)) and prediction results for SVM model on the compared algorithms (subfigures (b) to (d)).

For the experiment, we selected eight representative instance features of FSTP-IC, including the conflict density d of incompatible families as introduced in Section 3.1. We primarily used the default parameter settings of MATILDA, while setting the threshold parameter 'opts.perf.epsilon' to 0.01.

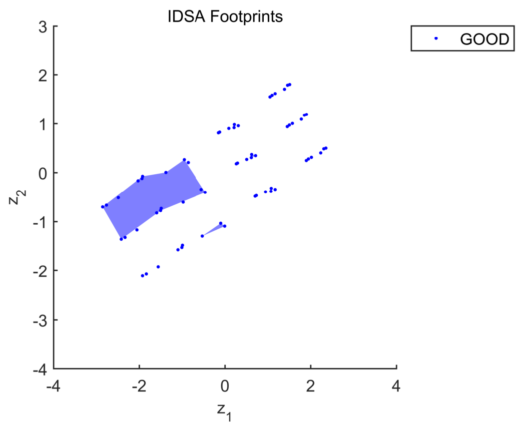
Figure 5 displays the distributions of instances and prediction results obtained from the SVM model. The X and Y axes in the figure represent the linear combination of the selected features. In Figure 5(b) to 5(d), the indicators of 'GOOD' and 'BAD' reveal where each compared algorithm can achieve good or bad performance on the benchmark instances. Specifically, an algorithm is defined as 'GOOD' if the SVM predicts its performance to be relatively better than that of the compared algorithms (Smith-Miles and Muñoz, 2023). From Figure 5(a), we can clearly see that different types of instances (distinguished by color) are clustered in different space. When considering the prediction results given by the SVM model as shown in Figure 5(b) to 5(d), we can observe that IDSA demonstrates a high



(a) Predicted footprint of ILS



(b) Predicted footprint of ACO



(c) Predicted footprint of IDSA

Fig. 6. Predicted footprints of the compared algorithms.

competitiveness on all the tested instances, while ILS and ACO show obvious weakness on three groups of instances, i.e., a280, gr666 and pr1002. These prediction results again confirm the superiority of the proposed IDSA, which is consistent with the experimental results of Section 3.3.

Furthermore, we show in Figure 6 the predicted footprints of the compared algorithms obtained from MATILDA. Basically, the footprint represents the regions within the instance space where an algorithm is expected to perform well. The larger the shaded footprint area in Figure 6, the more instances on which an algorithm can achieve good performance. By visualizing these areas, we can objectively evaluate the strengths and weaknesses of the compared algorithms and understand where each algorithm performs well or struggles. From Figure 6, we can observe that the predicted footprints of IDSA are significantly larger than those of the two reference algorithms, indicating its superior performance on the instance space induced by the 63 benchmark instances.

5 Conclusions and perspectives

The family traveling salesman problem with incompatibility constraints (FTSP-IC) studied in this work is a variant of the conventional traveling salesman problem with numerous relevant applications. This work is devoted to advancing the state-of-the-art algorithms for solving the FTSP-IC. We present the intensification-driven search algorithm that achieves extensive exploitation around local optimal solutions through a distance guided local search framework.

Computational results indicate that the proposed IDSA is highly competitive comparing with the reference algorithms. In particular, IDSA is able to achieve all the best-known results reported in literature (including 34 proven optimal solutions) and discover new upper bounds for 29 instances. These bounds hold potential to play a valuable role in future research on the FTSP-IC. The algorithm and its code can be used to solve practical problems related to the FTSP-IC.

For future work, there are at least two possible directions. First, the *IDSA-noSwi* variant discovered some better results than IDSA on the large-size instances (see Table 5), indicating that the performance of the proposed algorithm could be further improved. We will seek out more powerful local search techniques to further improve the algorithm's performance on large-size instances. Second, the IDSA algorithm is a relatively general framework, thus it would be interesting to explore its application to other problems related to the FTSP-IC.

Acknowledgments

We are grateful to the reviewers for their valuable comments and suggestions, which helped us to improve the paper. This work was partially supported by the National Natural Science Foundation Program of China (Grant No. 72301036, 72122006) and the Fundamental Research Funds for the Central Universities (Grant No. 423184). Support from the High-performance Computing Platform of Beijing University of Posts and Telecommunications is also acknowledged.

References

Agatz, N., Bouman, P., Schmidt, M., 2018. Optimization approaches for the traveling salesman problem with drone. Transportation Science 52, 965–981.

Alipour, H., Muñoz, M.A., Smith-Miles, K., 2023. Enhanced instance space analysis for the maximum flow problem. European Journal of Operational Research 304, 411–428.

Anily, S., Mosheiov, G., 1994. The traveling salesman problem with delivery and backhauls. Operations Research Letters 16, 11–18.

Applegate, D., Bixby, R., Chvatal, V., Cook, W., 2006. Concorde tsp solver. Aavailable at `http://www.math.uwaterloo.ca/tsp/concorde/index.html`.

Baker, E.K., 1983. An exact algorithm for the time-constrained traveling salesman problem. Operations Research 31, 938–945.

Bektas, T., 2006. The multiple traveling salesman problem: an overview of formulations and solution procedures. Omega 34, 209–219.

Bernardino, R., Paias, A., 2018. Solving the family traveling salesman problem. European Journal of Operational Research 267, 453–466.

Bernardino, R., Paias, A., 2021. Heuristic approaches for the family traveling salesman problem. International Transactions in Operational Research 28, 262–295.

Bernardino, R., Paias, A., 2022. The family traveling salesman problem with incompatibility constraints. Networks 79, 47–82.

Chisman, J.A., 1975. The clustered traveling salesman problem. Computers & Operations Research 2, 115–119.

Ding, J., Lü, Z., Zhou, T., Xu, L., 2017. A quality and distance guided hybrid algorithm for the vertex separator problem. Computers & Operations Research 78, 255–266.

Dolan, E.D., Moré, J.J., 2002. Benchmarking optimization software with performance profiles. Mathematical Programming 91, 201–213.

Feillet, D., Dejax, P., Gendreau, M., 2005. Traveling salesman problems with profits. Transportation Science 39, 188–205.

Fernandes, L.H.d.S., Lorena, A.C., Smith-Miles, K., 2021. Towards understanding clustering problems and algorithms: an instance space analysis. Algorithms 14, 95.

Gendreau, M., Hertz, A., Laporte, G., 1996. The traveling salesman problem with backhauls. Computers & Operations Research 23, 501–508.

Jünger, M., Reinelt, G., Rinaldi, G., 1995. The traveling salesman problem. Handbooks in Operations Research and Management Science 7, 225–330.

Kletzander, L., Musliu, N., Smith-Miles, K., 2021. Instance space analysis for a personnel scheduling problem. Annals of Mathematics and Artificial Intelligence 89, 617–637.

Li, J., Zhou, M., Sun, Q., Dai, X., Yu, X., 2014. Colored traveling salesman problem. IEEE Transactions on Cybernetics 45, 2390–2401.

Liu, S., Yan, X., Jin, Y., 2024. An edge-aware graph autoencoder trained on scale-imbalanced data for traveling salesman problems. Knowledge-Based Systems 291, 111559.

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

Morán-Mirabal, L., González-Velarde, J., Resende, M.G., 2014. Randomized heuristics for the family traveling salesperson problem. International Transactions in Operational Research 21, 41–57.

Muñoz, M.A., Smith-Miles, K., 2020. Instance space analysis: A toolkit for the assessment of algorithmic power. Source code is available at `https://github.com/andremun/InstanceSpace`.

Pop, P., Matei, O., Pintea, C., 2018. A two-level diploid genetic based algorithm for solving the family traveling salesman problem, in: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 340–346.

Porumbel, D., Hao, J.K., 2020. Distance-guided local search. Journal of Heuristics 26, 711–741.

Ren, J., Hao, J.K., Wu, F., Fu, Z.H., 2022. Intensification-driven local search for the traveling repairman problem with profits. Expert Systems with Applications 202, 117072.

Smith-Miles, K., Baatar, D., Wreford, B., Lewis, R., 2014. Towards objective measures of algorithm performance across instance space. Computers & Operations Research 45, 12–24.

Smith-Miles, K., Christiansen, J., Muñoz, M.A., 2021. Revisiting where are the hard knapsack problems? via instance space analysis. Computers & Operations Research 128, 105184.

Smith-Miles, K., Muñoz, M.A., 2023. Instance space analysis for algorithm testing: Methodology and software tools. ACM Computing Surveys 55, 1–31.

Wei, Z., Hao, J.K., Ren, J., Glover, F., 2023. Responsive strategic oscillation for solving the disjunctively constrained knapsack problem. European Journal of Operational Research 309, 993–1009.

Xu, X., Wang, M., Wang, Y., Ma, D., 2021. Two-stage routing with optimized guided search and greedy algorithm on proximity graph. Knowledge-Based Systems 229, 107305.

Zhu, Y., Chen, Y., Fu, Z.H., 2022. Knowledge-guided two-stage memetic search for the pickup and delivery traveling salesman problem with fifo loading. Knowledge-Based Systems 242, 108332.