symmetry

MDPI

Article

Five-Element Cycle Optimization Algorithm Based on an Integrated Mutation Operator for the Traveling Thief Problem

Yue Xiang [1], Jingjing Guo [2], Zhengyan Mao [3], Chao Jiang [4] and Mandan Liu [1,*]

1 Key Laboratory of Smart Manufacturing in Energy Chemical Process Ministry of Education, East China University of Science and Technology, Shanghai 200237, China; yue.xiang@mail.ecust.edu.cn
2 Department of Aerospace Science and Technology, Space Engineering University, Beijing 101416, China; jingjing.guo@mail.ecust.edu.cn or gjingjing0606@163.com
3 Shanghai Key Laboratory of Computer Software Testing & Evaluating, Shanghai Development Center of Computer Software Technology, Shanghai 201112, China; maozy@sscenter.sh.cn
4 Faculty of Information Technology, Beijing Key Laboratory of Computational Intelligence and Intelligent System, Engineering Research Center of Digital Community, Ministry of Education, Beijing Artificial Intelligence Institute and Beijing Laboratory for Intelligent Environmental Protection, Beijing University of Technology, Beijing 100124, China; chaojiang@mail.ecust.edu.cn
* Correspondence: liumandan@ecust.edu.cn

Abstract: This paper presents a novel algorithm named Five-element Cycle Integrated Mutation Optimization (FECOIMO) for solving the Traveling Thief Problem (TTP). The algorithm introduces a five-element cycle structure that integrates various mutation operations to enhance both global exploration and local exploitation capabilities. In experiments, FECOIMO was extensively tested on 39 TTP instances of varying scales and compared with five common metaheuristic algorithms: Enhanced Simulated Annealing (ESA), Improved Grey Wolf Optimization Algorithm (IGWO), Improved Whale Optimization Algorithm (IWOA), Genetic Algorithm (GA), and Profit-Guided Coordination Heuristic (PGCH). The experimental results demonstrate that FECOIMO outperforms the other algorithms across all instances, particularly excelling in large-scale instances. The results of the Friedman test show that FECOIMO significantly outperforms other algorithms in terms of average solution, maximum solution, and solution standard deviation. Additionally, although FECOIMO has a longer execution time, its complexity is comparable to that of other algorithms, and the additional computational overhead in solving complex optimization problems translates into better solutions. Therefore, FECOIMO has proven its effectiveness and robustness in handling complex combinatorial optimization problems.

Keywords: combinatorial optimization; five-element cycle optimization; integrated mutation operator; heuristic operator; traveling thief problem

Citation: Xiang, Y.; Guo, J.; Mao, Z.; Jiang, C.; Liu, M.. Five-Element Cycle Optimization Algorithm Based on an Integrated Mutation Operator for the Traveling Thief Problem. *Symmetry* **2024**, *16*, 1153. https://doi.org/10.3390/sym16091153

Academic Editors: Hsien-Chung Wu and Theodore E. Simos

Received: 25 July 2024
Revised: 26 August 2024
Accepted: 2 September 2024
Published: 4 September 2024

1. Introduction

To address the increasing complexity of real-world optimization challenges, the Traveling Thief Problem (TTP) was introduced as a benchmark that integrates two classical combinatorial optimization problems: the Traveling Salesman Problem (TSP) and the Knapsack Problem (KP) [1]. The TTP presents a scenario in which a thief must traverse a series of cities, returning to the starting point while selectively picking valuable items to maximize profit. The challenge lies in the limited capacity of the knapsack and the time-based rental cost, creating a unique interdependence between the TSP and KP. This study aims to develop an optimal travel plan and item selection strategy to maximize the thief's total benefit.

The TTP combines the path optimization of the TSP with the item selection of the KP, creating a complex problem where solving one subproblem directly influences the outcome of the other. The complexity of TTP arises not only from the NP-hard nature of both the

TSP and KP [2], but also from the intricate interdependence between these two problems, making the overall problem even more challenging.

Metaheuristic algorithms have emerged as powerful tools for addressing complex challenges like the TTP. These algorithms incorporate randomness and heuristic rules to explore global optimal solutions. Metaheuristic algorithms can be broadly categorized based on their operational principles and application scenarios:

- Evolutionary Algorithms (EAs), Including Genetic Algorithms (GA) and Differential Evolution (DE): These algorithms generate new solutions by simulating natural selection. Through operations like selection, crossover, and mutation, they are widely used in path optimization and resource allocation [3].
- Swarm Intelligence Algorithms, such as Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO): These algorithms solve combinatorial optimization problems by mimicking the collective behavior of swarms. They have shown excellent performance in solving TSP and other path optimization problems [4].
- Simulated Annealing (SA): Inspired by the physical annealing process, this algorithm gradually reduces randomness to help the search process converge to a global optimum [5]. SA has been widely applied in solving combinatorial optimization problems.
- Tabu Search (TS): This algorithm maintains a "tabu list" of previously visited solutions to avoid cycling, helping to escape local optima. It is particularly effective in path optimization and scheduling problems [6].
- Hybrid Metaheuristics: These combine two or more metaheuristic algorithms to enhance efficiency and precision. For instance, the hybrid of Genetic Algorithms with Tabu Search (GATS) combines the global search capability of GA with the local search capability of TS, showing outstanding performance in complex problems [7].

The TSP and KP have been extensively studied, with various algorithms developed to address their complexities. Significant progress in TSP has been achieved through heuristic algorithms like ACO and GA, which have demonstrated excellent performance in path optimization [3,4]. Similarly, KP has been effectively tackled using dynamic programming and greedy algorithms, proving effective in resource allocation [8,9].

Building on these foundational problems, the TTP not only inherits their complexities but also introduces a new layer of interdependence between path selection and item selection, making it a more challenging and comprehensive optimization problem [10]. Consequently, developing new approaches to manage the interaction between path and item selection in TTP is particularly important.

While TTP shares some similarities with the Capacitated Vehicle Routing Problem (CVRP) in optimizing routes under capacity constraints, there are fundamental differences in their structures. CVRP focuses primarily on optimizing vehicle routes within capacity limits to service customers, whereas TTP adds the complexity of item selection, where the choice of items directly affects overall route efficiency. This interdependence between path selection and item selection makes TTP a more comprehensive and challenging problem [11].

Researchers have proposed various methods to solve TTP instances. Evolutionary approaches, such as GA and Hybrid Genetic Algorithms combined with Tabu Search (GATS), have been widely applied to tackle TTP [12,13]. Ant Colony Algorithms [14] and Hybrid Ant Colony Algorithms [15] have also been employed. Additionally, modified Artificial Bee Colony Algorithms [16] and innovative Differential Evolution Algorithms [17] have shown significant advantages. Typically, these algorithms determine the tour plan first, followed by the item selection plan. However, in some studies [18,19], an enhanced Simulated Annealing Algorithm was used to determine the item selection plan first, followed by the tour plan. These studies emphasize the importance of integrating multiple optimization methods in solving complex TTP problems.

In recent years, heuristic methods for TTP have gained widespread attention, leading to many innovative studies. For instance, Mei et al. [20] introduced Cooperative Coevolution (CC) and the Memetic Algorithm (MA) to solve TTP. CC decomposes TTP into two independent subproblems, solving them separately, while MA addresses TTP as a unified problem, emphasizing the interdependence between the subproblems. Their research highlights the importance of considering the interaction between subproblems in the optimization process, a conclusion further validated by Bonyadi et al. [21]. Bonyadi and colleagues proposed the CoSolver algorithm, which also focuses on this interdependence, and developed the Density-based Heuristic (DH). Building on these frameworks, El Yafrani et al. [22] proposed the CS2SA* and CS2SA-R algorithms, combining the 2-OPT steepest ascent hill-climbing heuristic with Simulated Annealing.

Additionally, Polyakovskiy et al. [10] initially proposed simple heuristic methods, marking the first attempt to address TTP. Building on this foundation, Mei et al. [23] developed an efficient Memetic Algorithm that incorporates a two-stage local search, demonstrating effectiveness in solving large-scale TTP benchmark instances. Mei et al. [24] further advanced the automatic evolution of effective item selection heuristics through Genetic Programming (GP). Martins et al. [25] proposed an Estimation of Distribution Algorithm (EDA)-based heuristic selection method in a hyper-heuristic context, while El Yafrani et al. [26] applied low-level hyper-heuristics to small and mid-sized TTP instances. These studies expanded the solution space for TTP and validated the effectiveness of combining local search heuristics with other methods [27–30]. Comprehensive comparisons were conducted in the experimental section, and the outcomes were contrasted with those of the proposed algorithm.

The application of mutation operators in optimization algorithms, particularly for solving TSP and other combinatorial optimization problems, has been extensively studied. These operators introduce randomness and diversity into the search process, effectively preventing the algorithm from getting trapped in local optima [31]. In the context of TTP, the application of mutation operators has proven crucial, especially in multiobjective optimization problems [32].

To address these complex optimization challenges, the Five-element Cycle Optimization algorithm (FECO) was introduced [33]. FECO is a heuristic algorithm based on the traditional Chinese Five Elements theory, simulating the generative and restrictive relationships among the five elements to create a dynamically balanced optimization model. Over time, FECO has evolved to tackle more challenging multiobjective optimization problems. For instance, the Local Search-Based Many-Objective FECO (LSMaOFECO) [34] significantly improves FECO's effectiveness in high-dimensional objective spaces. Additionally, in cold chain logistics, the Dual-Mode Updated FECO (FECO-DMUI) [35] optimizes delivery routes by balancing costs and customer satisfaction, demonstrating superior performance compared with traditional algorithms.

Despite its versatility and robustness in addressing complex optimization problems, FECO's application to TTP remains underexplored, particularly in scenarios involving highly interdependent objectives. This study aims to bridge this gap by applying an enhanced version of FECO to TTP and integrating mutation operators to further improve solution quality and convergence speed.

In this paper, the Five-element Cycle Optimization algorithm based on Integrated Mutation Operator (FECOIMO) is proposed and applied to solve TTP. FECOIMO enhances the basic FECO method by incorporating various mutation operators to prevent premature convergence in complex combinatorial optimization problems. Our goal is to validate the effectiveness of FECOIMO in solving TTP instances of different scales and demonstrate its superiority by comparing it with other well-established algorithms.

The remainder of this paper is structured as follows: Section 2 provides an overview of the TTP, while Section 3 delves into the principles underlying FECO. In Section 4, the application of FECOIMO in solving TTP instances is elucidated. Subsequently, Section 5 presents the experimental results and analysis derived from our approach. Finally, Section 6 offers concluding remarks and outlines potential avenues for future research.

2. The Traveling Thief Problem (TTP)

The Traveling Thief Problem (TTP) is a unique combination of two classic combinatorial optimization problems: the Traveling Salesman Problem (TSP) and the Knapsack Problem (KP). TTP presents a scenario where a thief must traverse a set of n cities exactly once, starting and ending at the same city—a structure similar to TSP. However, unlike the traditional TSP, during each visit to a city, the thief has the option to steal items from a pool of m available choices, placing them in a knapsack that has a finite capacity—this aspect resembles KP.

The decision variables in TTP consist of two main components: the tour plan, denoted by $\mathbf{x} = [x_1,\ x_2,\ \cdots, x_p,\ \cdots,\ x_n,\ x_1]$, representing a permutation of the n cities including the starting city x_1, and the picking plan, denoted by $\mathbf{z} = [z_1,\ z_2,\ \cdots, z_t,\ \cdots,\ z_m]$, where $z_t =\ x_p$ indicates that item t is picked in city x_p, and $z_t = 0$ signifies that item t remains unpicked ($p = 1,\ 2,\ \cdots,\ n;\ t = 1,\ 2,\ \cdots,\ m$).

The thief's speed decreases linearly with the cumulative weight of the items in the knapsack. Thus, the departure speed of the thief from city x_p can be expressed as follows:

$$v_{x_p} = v_{max} - (v_{max} - v_{min})\frac{W_{x_p}}{Q} \tag{1}$$

where W_{x_p} ($0 \le W_{x_p} \le Q$) represents the total weight of items carried when departing city x_p. When the knapsack is empty ($W_{x_p} = 0$), the thief's speed is at its maximum, whereas when the knapsack reaches full capacity ($W_{x_p} = Q$), the speed decreases to its minimum.

To compute W_{x_p}, the variable $y_{x_p t}$ ($x_p \in \mathbf{x},\ t = 1,\ 2,\ \cdots,\ m$) is defined as follows:

$$y_{x_p t} = \begin{cases} 1 & z_t = x_p \\ 0 & z_t = 0 \end{cases} \tag{2}$$

where $y_{x_p t} = 1$ indicates that item t is picked in city x_p, while $y_{x_p t} = 0$ indicates it remains unpicked. Thus, W_{x_p} is calculated as:

$$W_{x_p} = \sum_{k=1}^{p-1} W_{x_k} + \sum_{t=1}^{m} w_t y_{x_p t} \tag{3}$$

The distances between the n cities are represented by the symmetric distance matrix $\mathbf{d}_{n\times n}$, where $d_{x_p,\, x_{p+1}}$ denotes the distance from city x_p to x_{p+1}, with $d_{x_p,\, x_{p+1}} = d_{x_{p+1},\, x_p}$. When $p = n,\ p+1$ is replaced by 1 to account for the cyclic nature of the tour.

Additionally, the distribution of the m items across the cities is governed by the availability matrix $\boldsymbol{a}_{m\times n}$, a binary matrix where $a(t, x_p) = 1$ indicates that item t is present at city x_p, and $a(t, x_p) = 0$ signifies its absence. Each item t has a value denoted by b_t and a weight denoted by w_t, while the knapsack's capacity is denoted by Q.

The total time spent by the thief on the tour can be calculated as follows:

$$f(\mathbf{x}, \mathbf{z}) = \sum_{p=1}^{n-1} \frac{d_{x_p,\, x_{p+1}}}{v_{x_p}} + \frac{d_{x_n,\, x_1}}{v_{x_n}} \tag{4}$$

The aggregate value of all picked items can be expressed as follows:

$$g(\mathbf{x}, \mathbf{z}) = \sum_{p=1}^{n} \sum_{t=1}^{m} b_t y_{x_p t} \tag{5}$$

This paper focuses on solving the single-objective version of TTP, where the primary goal is to develop an optimal tour plan, $\mathbf{x}$, along with an optimal picking plan, $\mathbf{z}$, to maximize the thief's total benefit. This benefit comprises two components: the total value of the stolen items and the corresponding knapsack rental costs. The objective function is to balance maximizing the value of the stolen items while minimizing the associated rental costs, thereby ensuring the thief attains the highest possible net benefit. The objective function of TTP can be succinctly represented by Equation (6):

$$max \quad G(\mathbf{x},\, \mathbf{z}) = g(\mathbf{x},\, \mathbf{z}) - Rf(\mathbf{x},\, \mathbf{z}) \tag{6}$$

where $G(\mathbf{x},\, \mathbf{z})$ denotes the total benefit accrued by the thief, $g(\mathbf{x},\, \mathbf{z})$ represents the aggregated value of all picked items. R signifies the rental cost of the knapsack per unit of time, and $f(\mathbf{x},\, \mathbf{z})$ corresponds to the duration of the thief's tour.

Subject to:

$$W_{x_n} \leq Q \tag{7}$$

This constraint ensures that the total weight of the items picked by the thief does not exceed the capacity of the knapsack.

3. Five-Element Cycle Optimization Algorithm (FECO)

Optimization algorithms are crucial tools for solving complex combinatorial problems. In this context, the Five-element Cycle Optimization algorithm (FECO) emerges as a unique and effective approach, inspired by ancient Chinese philosophy. FECO leverages the Five-element Cycle Model (FECM), which models the dynamic interactions among different elements, to guide the search for optimal solutions. This section delves into the underlying principles of the FECM and explains how it forms the foundation for the FECO, enabling it to address challenging optimization tasks like the Traveling Salesman Problem (TSP) and other multiobjective problems.

3.1. The Five-Element Cycle Model

The Five-element Cycle Model (FECM) [33] is rooted in the ancient Chinese philosophy of the Five Elements (Wu Xing), which represents the dynamic interactions among the elements of metal, wood, water, fire, and earth. These elements interact through generating (promoting) and restricting (inhibiting) forces, forming a balanced and cyclical relationship. The generating interaction can be likened to a mother nurturing her child, ensuring its growth, while the restricting interaction is similar to a grandparent disciplining a grandchild, maintaining order within the system.

As shown in Figure 1, the outer circle symbolizes generating interactions, while the inner circle represents restricting interactions. For example, wood receives generating force from water and is constrained by metal. At the same time, wood exerts a generating force on fire and restricts earth.

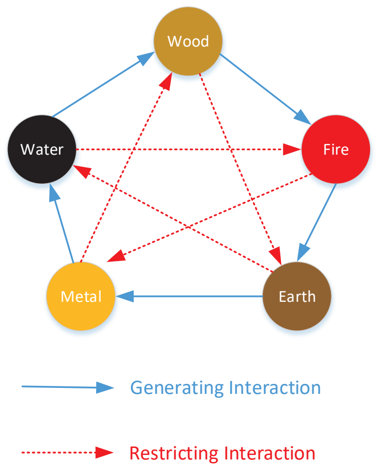


Figure 1. The generating and restricting interaction among five elements.

In FECM, these interactions are mathematically modeled to capture the dynamics of these relationships. The mass of each element, denoted as M_i^k, where i represents the element and k the time step, evolves based on the forces exerted by other elements. The force F_i^k acting on element i at time k is calculated considering the generating and restricting influences from other elements, as described by Equation (8).

$$\begin{cases} F_1^k = \ln\left[\frac{M_5^k}{M_1^k}\right] - \ln\left[\frac{M_4^k}{M_1^k}\right] - \ln\left[\frac{M_1^k}{M_2^k}\right] - \ln\left[\frac{M_1^k}{M_3^k}\right] \\ F_2^k = \ln\left[\frac{M_1^k}{M_2^k}\right] - \ln\left[\frac{M_5^k}{M_2^k}\right] - \ln\left[\frac{M_2^k}{M_3^k}\right] - \ln\left[\frac{M_2^k}{M_4^k}\right] \\ F_3^k = \ln\left[\frac{M_2^k}{M_3^k}\right] - \ln\left[\frac{M_1^k}{M_3^k}\right] - \ln\left[\frac{M_3^k}{M_4^k}\right] - \ln\left[\frac{M_3^k}{M_5^k}\right] \\ F_4^k = \ln\left[\frac{M_3^k}{M_4^k}\right] - \ln\left[\frac{M_2^k}{M_4^k}\right] - \ln\left[\frac{M_4^k}{M_5^k}\right] - \ln\left[\frac{M_4^k}{M_1^k}\right] \\ F_5^k = \ln\left[\frac{M_4^k}{M_5^k}\right] - \ln\left[\frac{M_3^k}{M_5^k}\right] - \ln\left[\frac{M_5^k}{M_1^k}\right] - \ln\left[\frac{M_5^k}{M_2^k}\right] \end{cases} \tag{8}$$

where M_1^k, M_2^k, M_3^k, M_4^k, and M_5^k represent the masses of metal, water, wood, fire, and earth, respectively, at time k.

Each element F_i^k $(i = 1, 2, \cdots, 5)$ in the cycle is influenced by four distinct segments:

- Parent element generation force: The first segment in the equation, $\ln\left[\frac{M_{i-1}^k}{M_i^k}\right]$, represents the force generated by the parent element on the current element. This force is positive if the mass of the parent element M_{i-1}^k is greater than the mass of the current element M_i^k, implying effective generation. Conversely, if the current element's mass is greater, this force diminishes, indicating a weaker generation effect.
- Grandparent element inhibition force: The second segment, $-\ln\left[\frac{M_{i-2}^k}{M_i^k}\right]$, represents the inhibitory force exerted by the grandparent element on the current element. This segment is negative, indicating that as the mass of the grandparent element M_{i-2}^k increases relative to the current element M_i^k, the inhibitory effect strengthens, reducing the current element's growth or influence.
- Child element generation force: The third segment, $-\ln\left[\frac{M_i^k}{M_{i+1}^k}\right]$, denotes the generation force that the current element exerts on its child element. A smaller mass of the current element M_i^k compared with its child M_{i+1}^k implies a weaker generation force, as the element's ability to generate the next element in the cycle is diminished.
- Grandchild element inhibition force: The fourth segment, $-\ln\left[\frac{M_i^k}{M_{i+2}^k}\right]$, describes the inhibition force that the current element exerts on its grandchild element. Similar to the child element generation force, inhibition becomes stronger as the mass of the current element decreases relative to the grandchild M_{i+2}^k.

To illustrate, consider the wood element (i.e., $i = 3$). Wood receives generating force from water (element $i = 2$) and is inhibited by metal (element $i = 1$). Simultaneously, wood exerts a generation force on fire (element $i = 4$) and an inhibitory force on earth (element $i = 5$). The first term $\ln\left[\frac{M_2^k}{M_3^k}\right]$ reflects water's positive influence on Wood, as water nourishes wood. The second term $-\ln\left[\frac{M_1^k}{M_3^k}\right]$ illustrates metal's inhibitory effect on wood. The third and fourth terms represent Wood's influence on fire and earth, respectively, showing how wood interacts with its descendants in the cycle.

From the above analysis, it is evident that the forces acting on each element in the Five-element Cycle are intricately tied to the masses of the elements themselves. The greater the mass difference between related elements, the stronger the force exerted, whether generative or inhibitive. This relationship between mass and force is crucial to understanding how the elements dynamically interact within the cycle, providing insight into their balance and overall system behavior.

Generalizing to a case with L elements in each cycle, the FECM can be formulated as follows:

$$\begin{cases} F_i^k = & \omega_{gp} \ln\left[\frac{M_{i-1}^k}{M_i^k}\right] - \omega_{rp} \ln\left[\frac{M_{i-2}^k}{M_i^k}\right] - \omega_{ga} \ln\left[\frac{M_i^k}{M_{i+1}^k}\right] - \omega_{ra} \ln\left[\frac{M_i^k}{M_{i+2}^k}\right] \\ M_i^{k+1} = & M_i^k \cdot \frac{2}{1+\exp(-F_i^k)} \end{cases} \tag{9}$$

where i ranges from 1 to L. When $i = 1,\ i-1$ is replaced by L; when $i = 2,\ i-2$ is replaced by L; when $i = L,\ i+1$ is replaced by 1; and when $i = L-1,\ i+2$ is replaced by 1. The weight coefficients $\omega_{gp},\ \omega_{rp},\ \omega_{ga}$ and ω_{ra} control the strength of these interactions and are typically set to 1, reflecting equal emphasis on all interaction types.

The FECM is not only a conceptual model but also forms the basis for FECO, which has been effectively applied to solve complex combinatorial optimization problems, such as the Traveling Salesman Problem (TSP) [33]. Additionally, FECO has evolved to tackle more challenging multiobjective optimization problems. For instance, the Local Search-Based Many-Objective FECO (LSMaOFECO) [34] significantly improves FECO's effectiveness in high-dimensional objective spaces. By modeling the interactions among elements as iterative processes, FECO leverages the balance of generating and restricting forces to guide the search for optimal solutions.

This model offers a unique perspective on optimization, enabling a dynamic balance between exploration and exploitation in the search space, mirroring the harmonious interplay found in nature.

3.2. The Five-Element Cycle Optimization

Building on FECM, FECO has been proposed and effectively applied to solve TSP [33]. FECO is a nature-inspired metaheuristic algorithm, drawing on the traditional Chinese philosophy of the Five Elements theory. Metaheuristic algorithms are commonly used to solve complex optimization problems, characterized by their use of randomness and heuristic rules to search for global optimal solutions [36]. In FECO, the population in the optimization algorithm corresponds to the elements within the Five-element Cycle framework, with individuals representing each element. The element space is illustrated in Figure 2. It shows that a population of size N is divided into q cycles, each containing L elements, i.e., $N = q \times L$. FECO operates as an iterative algorithm, where $\mathbf{e}_{ij}^k$ denotes the i-th element within the j-th cycle at the k-th iteration ($i = 1, 2, \ldots, L; j = 1, 2, \ldots, q$). Here, M_{ij}^k represents the mass of $\mathbf{e}_{ij}^k$, and F_{ij}^k signifies the force exerted by $\mathbf{e}_{ij}^k$.

When applying FECO to solve optimization problems, an element corresponds to a solution. The objective function, or a related variant, can be used as the mass M of the element. Based on the relationship between M and F, F can be used to evaluate the objective function. For instance, in the case of minimizing an objective function, when the objective function value is used as the mass of the element, a larger F implies a smaller corresponding M and hence a smaller objective value.

Consequently, when solving the TSP, solutions where $F > 0$ are considered superior and require no further updating, while solutions where $F \leq 0$ are deemed inferior and are subject to update operations [33]. Through this distinctive mechanism of FECO, in conjunction with appropriate operators, effective resolution of the optimization problem can be achieved.

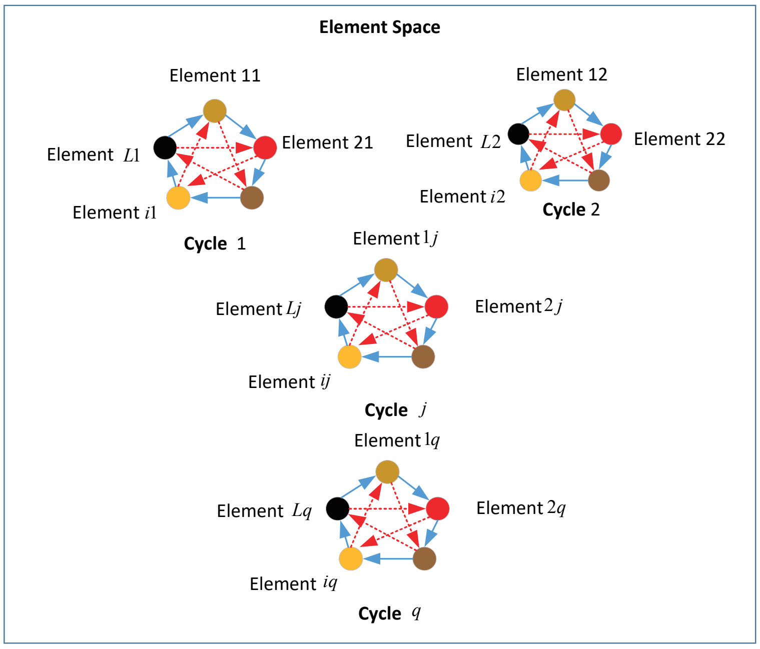


Figure 2. Element space.

4. Five-Element Cycle Optimization Algorithm Based on Integrated Mutation Operator for the TTP

In this section, the Five-element Cycle Optimization algorithm based on the Integrated Mutation Operator (FECOIMO) is presented, specifically designed to address the complex Traveling Thief Problem (TTP). The TTP poses significant challenges due to its dual-component structure, which involves optimizing both a tour plan and a picking plan. FECOIMO combines the principles of the Five-element Cycle Model with advanced mutation operators to efficiently explore the solution space and enhance the quality of the solutions. The algorithm dynamically adjusts its operations based on the current state of the solution, ensuring a balanced approach between exploration and exploitation. The following subsections detail the expression of solutions, initial solution generation, force calculation, integrated mutation operator, heuristic operator, element update process, and the overall implementation of FECOIMO.

4.1. Expression of Solution and the Mass of the Element

This paper introduces the Five-element Cycle Optimization algorithm based on the Integrated Mutation Operator (FECOIMO). When applying FECOIMO to address the Traveling Thief Problem (TTP), $\mathbf{e}_{ij}^{k}$, represents a feasible solution for TTP. Given that the decision variable of TTP comprises two distinct components–a tour plan $\mathbf{x}_{ij}^{k}$ and a picking plan $\mathbf{z}_{ij}^{k}$–the solution can be expressed as $\mathbf{e}_{ij}^{k} = [\mathbf{x}_{ij}^{k}\,, \mathbf{z}_{ij}^{k}]$.

The tour plan $\mathbf{x}_{ij}^{k}$ encompasses n decision variables, corresponding to the number of cities, n:

$$\mathbf{x}_{ij}^{k} = [x_1,\ x_2,\ \cdots,\ x_n,\ x_1]^k \tag{10}$$

The picking plan $\mathbf{z}_{ij}^{k}$ comprises m decision variables, reflecting the number of items, m:

$$\mathbf{z}_{ij}^{k} = [z_1,\ z_2,\ \cdots,\ z_m]^k \tag{11}$$

In solving the TTP using FECOIMO, the objective function aims to maximize the benefit G. Leveraging the relationship between mass M and force F, the mass of the element is formulated according to Equation (12), as designed in this paper.

$$M_{ij}^k = \max_{ij}(G(\mathbf{x}_{ij}^k, \mathbf{z}_{ij}^k)) - G(\mathbf{x}_{ij}^k, \mathbf{z}_{ij}^k) + 1 \tag{12}$$

In Equation (12), $G(\mathbf{x}_{ij}^k, \mathbf{z}_{ij}^k)$ denotes the benefit of element $\mathbf{e}_{ij}^k$, while $\max_{ij}(G(\mathbf{x}_{ij}^k, \mathbf{z}_{ij}^k))$ represents the maximum benefit among all elements at the k-th iteration. To ensure the validity of Equation (9), Equation (12) has been adjusted by adding 1, ensuring that $M_{ij}^k > 0$. Notably, a smaller value of M_{ij}^k indicates closer proximity to the optimal solution. Consequently, a larger value of F_{ij}^k suggests that the i-th element in the j-th cycle may represent a promising solution. Elements are thus evaluated based on F_{ij}^k according to FECO.

Based on the relationship among FECM, FECOIMO, and TTP, as shown in Table 1, FECOIMO is specifically designed to solve the TTP.

Table 1. Relationship among FECM, FECOIMO, and TTP.

FECM	FECOIMO	TTP
Elements $\mathbf{x}_i^k (i = 1, 2, \cdots, L)$	Elements $\mathbf{e}_{ij}^k$ $(i = 1, 2, \cdots, L; j = 1, 2, \cdots, q)$	Solution (Tour plan, picking plan)$\mathbf{x}_{ij}^k, \mathbf{z}_{ij}^k$
Mass of elements M_i^k	Mass of elements M_{ij}^k	Variant of objective function (The total benefit G)
Force exerted on elements F_i^k	Force exerted on elements F_{ij}^k	Variables estimating the quality of solutions

4.2. Initial Solutions Generation

The process of constructing the initial solution $\mathbf{e}_{ij}^0$ $(i = 1, 2, \cdots, L; j = 1, 2, \cdots, q)$ involves generating both an initial tour plan and an initial picking plan. To generate the initial tour plan, a random tour $\mathbf{x}_{ij}^0 = [x_1, x_2, \cdots, x_p, \cdots, x_n, x_1]^0$ is first created. Following this, a greedy operator is applied to refine the tour and approximate an optimal solution. The greedy operator iteratively adjusts the positions of remaining cities x_p $(p = 4, \cdots, n)$ by selecting the position that minimizes the overall tour length. The pseudocode for the greedy operator is provided in Algorithm 1. The goal is to construct a tour where each city's position is determined based on a local optimization criterion, typical of a greedy algorithm.

Algorithm 1 Greedy operator

```
Require: $n$, $\mathbf{d}_{n\times n}$ (distance matrix)
Ensure: $\mathbf{x}_{ij}^0 = [x_1, x_2, \cdots, x_p, \cdots, x_n, x_1]^0$
1: Randomly select three cities to generate a tour $\mathbf{x}_{ij}^0$, $\mathbf{x}_{ij}^0 = [x_1, x_2, x_3]^0$
2: for $p = 4 \to n$ do
3:     Find the optimal position for $x_p$ by minimizing the additional distance
4:     Insert $x_p$ into the tour $\mathbf{x}_{ij}^0$ at the position that results in the shortest distance
5: end for
6: Return $\mathbf{x}_{ij}^0 = [x_1, x_2, \cdots, x_n, x_1]^0$
```

The initial picking plan $\mathbf{z}_{ij}^0 = [z_1, z_2, \cdots, z_m]^0$ is then determined based on the generated initial tour plan $\mathbf{x}_{ij}^0$. In the initial picking plan, each item is assigned to a city where it can be collected. A greedy approach can be applied here to maximize the value-to-weight ratio of the items selected while respecting the knapsack's capacity. If the total weight exceeds the knapsack's capacity, items are removed, prioritizing those with the lowest value-to-weight ratio. The detailed steps for generating the initial picking plan are provided in Algorithm 2.

Algorithm 2 Initial picking plan

```
Require: $Q,\ \boldsymbol{a}_{m\times n},\ \mathbf{x}^0_{ij},\ w_t(\ t=\ 1,\ 2,\ \cdots,\ m),$
Ensure: $\mathbf{z}^0_{ij} =\ [z_1,\ z_2,\ \cdots,\ z_t,\ \cdots,\ z_m]^0$
 1: $restofWeight \leftarrow Q,\ \boldsymbol{avc}_m \leftarrow \emptyset$
 2: for $t = 1 \rightarrow m$ do
 3:     for $p = 1 \rightarrow n$ do
 4:         if $a(t, x_p) = 1$ then
 5:             $\boldsymbol{avc}(t) = [\boldsymbol{avc}(t)\ x_p]$
 6:         end if
 7:     end for
 8:     Randomly select one city $x_{rp}$ from $\boldsymbol{avc}(t)$
 9:     $z_t \leftarrow x_{rp}$
10:     $restofWeight =\ restofWeight - w_t$
11: end for
12: while $restofWeight < 0$ do
13:     Remove item $t$ with the lowest value-to-weight ratio
14:     $z_t \leftarrow 0$
15: end while
16: Return $\mathbf{z}^0_{ij} = [z_1,\ z_2,\ \cdots,\ z_m]^0$
```

4.3. Force Calculation

In FECOIMO, the force exerted on each element is calculated using the following formula:

$$F^k_{ij} = \ln\left[\frac{M^k_{(i-1)j}}{M^k_{ij}}\right] - \ln\left[\frac{M^k_{(i-2)j}}{M^k_{ij}}\right] - \ln\left[\frac{M^k_{ij}}{M^k_{(i+1)j}}\right] - \ln\left[\frac{M^k_{ij}}{M^k_{(i+2)j}}\right] \quad (13)$$

where F^k_{ij} represents the force exerted on the i-th element by other elements within the j-th cycle at the k-th iteration.

4.4. Integrated Mutation Operator

The mutation operator plays a critical role in optimization algorithms by randomly altering the genes of individuals. This mechanism enables the algorithm to explore additional solutions within the search space, thereby reducing the risk of getting trapped in local optima. Additionally, the mutation operator introduces novel individuals, which enhances the diversity of the population and effectively mitigates premature convergence. By incorporating the mutation operator, the algorithm can escape local optimal solutions, thus improving its global search capability. The randomness introduced by the mutation operator also helps in escaping local optima and accelerating the algorithm's convergence speed.

In this study, three distinct mutation operators are selected to update the tour plan of elements: the flip mutation operator (f_{flip}), the insert mutation operator (f_{insert}), and the move mutation operator (f_{move}). Each operator has unique characteristics and impacts on the optimization process.

4.4.1. Flip Mutation Operator (f_{flip})

The flip mutation operator alters a chromosome by flipping a gene, effectively rearranging the gene order and generating new solutions. This operator primarily fosters population diversity, facilitates the escape from local optima, and enhances the exploitation of the global optimum. By changing the orientation of specific genes, it introduces variations that help explore new regions of the search space.

The implementation process of $f_{flip}(\mathbf{x}_{ij}^k)$ is outlined as follows:

- Copy $\mathbf{x}_{ij}^k$ to $\mathbf{x}_{ij}^{k+1}$;
- Randomly select two numbers p_1 and p_2, where $p_1,\ p_2 \in \{1,\ 2,\ \cdots, n\}$ and $p_1 < p_2$;
- Flip the sequence $[x_{p_1},\ \cdots,\ x_{p_2}]$ in $\mathbf{x}_{ij}^{k+1}$;
- Return $\mathbf{x}_{ij}^{k+1}$.

An example of the f_{flip} process is shown in Figure 3. The red-colored numbers in the figure represent the selected sequence that are flipped during the process, i.e., $[x_{p_1},\ \cdots,\ x_{p_2}]$.

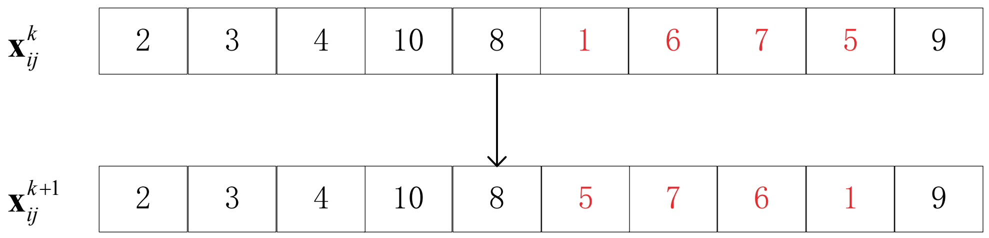


Figure 3. The process of f_{flip}, $n = 10,\ p_1 = 6,\ p_2 = 9$.

4.4.2. Insert Mutation Operator (f_{insert})

The insert mutation operator mutates a chromosome by relocating a gene from one position to another within the chromosome. This reshuffling produces novel solutions and focuses on enhancing the local search capability of the algorithm. By enabling the population to better adapt to local optimal solutions, this operator ensures that the search process can effectively exploit the local regions of the search space.

The implementation process of $f_{insert}(\mathbf{x}_{ij}^k)$ is outlined as follows:

- Randomly select two numbers p_3 and p_4, where $p_3,\ p_4 \in \{1,\ 2,\ \cdots,\ n\}$ and $p_3 < p_4$;
- Remove $[x_{p_3}, \cdots, x_{p_4}]$ from $\mathbf{x}_{ij}^k = [x_1,\ x_2, \cdots,\ x_n\ ]^k$ and generate a new tour $\mathbf{x}_{ij}^{k+1} = [x_1,\ x_2,\ \cdots,\ x_{(n-p_4+p_3-1)}]$;
- Randomly select a number p_5, where $p_5 \in \{1,\ 2,\ \cdots,\ n - p_4 + p_3 - 1\}$;
- Insert the sequence $[x_{p_3}, \cdots, x_{p_4}]$ into $\mathbf{x}_{ij}^{k+1}$ starting from the p_5-th position;
- Return $\mathbf{x}_{ij}^{k+1}$.

An example of the f_{insert} process is shown in Figure 4. The red-colored numbers in the figure represent the selected sequence that are inserted during the process, i.e., $[x_{p_3},\ \cdots,\ x_{p_4}]$.

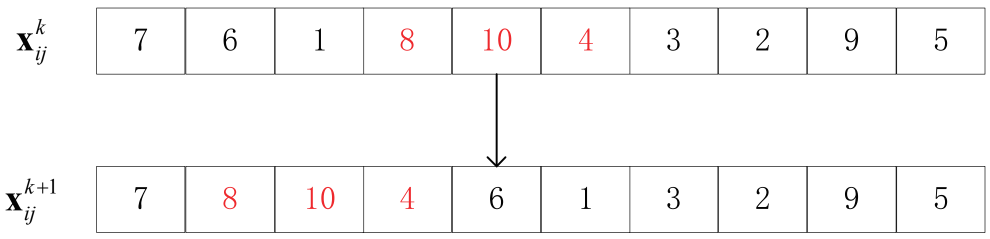


Figure 4. The process of f_{insert}, $n = 10,\ p_3 = 4,\ p_4 = 6,\ p_5 = 2$.

4.4.3. Move Mutation Operator (f_{move})

The move mutation operator displaces a gene segment either left or right within the chromosome, altering its structure and generating new solutions. This operator is designed to diversify local chromosome structures, thereby facilitating broader exploration of the solution space. By modifying the positions of gene segments, it helps discover new configurations that might lead to better solutions.

The implementation process of $f_{move}(\mathbf{x}_{ij}^k)$ is outlined as follows:

- Copy $\mathbf{x}_{ij}^k$ to $\mathbf{x}_{ij}^{k+1}$;
- Randomly select a number p_6, where $p_6 \in \{2, \cdots, n\}$;
- Swap the segments $[x_1, x_2, \cdots, x_{p_6-1}]$ and the tour $[x_{p_6}, x_{p_6+1}, \cdots, x_n]$ in $\mathbf{x}_{ij}^{k+1}$;
- Return $\mathbf{x}_{ij}^{k+1}$.

An example of the f_{move} process is shown in Figure 5. The red-colored numbers in the figure represent the selected sequence that are moved during the process, i.e., $[x_1, x_2, \cdots, x_{p_6-1}]$.

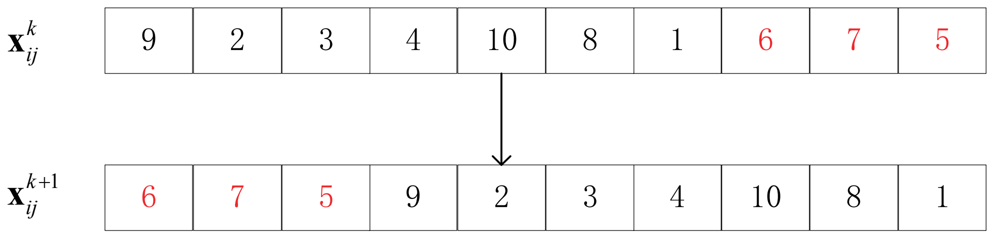


Figure 5. The process of f_{move}, $n = 10$, $p_6 = 8$.

Although these three operators share the commonality of being mutation operators, their specific operations and mutation effects differ. The concurrent utilization of these operators enhances population diversity and ensures a thorough exploration of the solution space. The flip mutation operator contributes to global exploration by introducing significant changes, the insert mutation operator improves local adaptation by fine-tuning gene positions, and the move mutation operator provides structural diversity to prevent premature convergence. Together, these operators synergize to create a robust optimization process capable of efficiently finding high-quality solutions.

4.4.4. The Calculation Process of the Effect of Mutation Operators

In this study, the usage probabilities of each mutation operator are dynamically adjusted based on their optimization effects. The process involves the following steps:

(1) Initial probability assignment:
Each mutation operator is initially assigned the same usage probability for evolving the individuals in the population.

(2) Optimization effect calculation:
For each generation, the optimization effect of each mutation operator is determined by identifying the individual with the greatest improvement due to that operator. Specifically, for each operator f_m ($f_m = f_{flip}, f_{insert}, f_{move}$), the effect is calculated as follows:

$$\Delta G_{f_m}(k) = \max_{ij} \left(G(\mathbf{x}_{ij}^k, \mathbf{z}_{ij}^k) - G(f_m(\mathbf{x}_{ij}^k, \mathbf{z}_{ij}^k)) \right) \tag{14}$$

where $G(\mathbf{x}, \mathbf{z})$ and $G(f_m(\mathbf{x}, \mathbf{z}))$ represent the objective function values before and after applying the mutation operator f_m.

(3) Average effect over generations:
The optimization effects of each mutation operator are averaged over all generations to determine their overall effectiveness for each TTP instance. For a given operator f_m, the average effect is computed as:

$$E_{f_m} = \frac{1}{k_{\max}} \sum_{k=1}^{k_{\max}} \Delta G_{f_m}(k) \tag{15}$$

where k_{max} is the maximum iteration.

(4) Summing effects across TTP instance:

To generalize the effectiveness of each mutation operator, the average effects E_{f_m} are summed across different test instances. The cumulative effect for each operator is given by:

$$S_{f_m} = \sum_{c=1}^{C} E_{f_m}(c) \tag{16}$$

where C is the total number of TTP instances that need to be resolved.

(5) Determining usage probabilities:
The final step involves determining the usage probability of each mutation operator based on its cumulative effect. The probability p_{f_m} for each operator f_m is calculated as:

$$p_{f_m} = \frac{S_{f_m}}{\sum\limits_{m} S_{f_m}} \tag{17}$$

In this approach, multiple mutation operators are integrated by assigning different probabilities to each operator. This ensures that the operators are selected based on their assigned likelihoods, allowing for a balanced application of each operator during the optimization process. The pseudocode for this integration is outlined in Algorithm 3.

Algorithm 3 Integrated mutation operator

```
Require: $\mathbf{x}_{ij}^{k}$, $r_1$, $r_2$
Ensure: $\mathbf{x}_{ij}^{k+1}$
1: if $rand \le r_1$ then
2:     $\mathbf{x}_{ij}^{k+1} \Leftarrow f_{flip}(\mathbf{x}_{ij}^{k})$
3: else if $r_1 < rand \le r_2$ then
4:     $\mathbf{x}_{ij}^{k+1} \Leftarrow f_{insert}(\mathbf{x}_{ij}^{k})$
5: else
6:     $\mathbf{x}_{ij}^{k+1} \Leftarrow f_{move}(\mathbf{x}_{ij}^{k})$
7: end if
8: Return $\mathbf{x}_{ij}^{k+1}$
```

In Algorithm 3, r_1 is set to the probability $p_{f_{flip}}$ for the flip mutation operator, r_2 is set to the sum of probabilities $p_{f_{flip}} + p_{f_{insert}}$ for the flip and insert mutation operators.

By following this process, the selection of a mutation operator for each individual in the population is probabilistically determined. This method allows for dynamic and balanced integration of multiple mutation operators, enhancing the exploration and exploitation capabilities of the optimization algorithm. The probabilities $p_{f_{flip}}$ (where $p_{f_{insert}} = r_2 - r_1$), $p_{f_{insert}}$ (where $p_{f_{insert}} = r_2 - r_1$), and $p_{f_{move}}$ (where $p_{f_{move}} = 1 - r_2$) are assigned based on the optimization effects of each operator, ensuring that more effective operators are applied more frequently.

4.4.5. Conclusion of the Integrated Mutation Operator

A key enhancement in the FECOIMO algorithm over the original FECM is the dynamic adjustment of mutation operator usage based on their effectiveness during the optimization process. By evaluating the contribution of each mutation operator throughout the evolutionary process, FECOIMO can assign usage probabilities that prioritize more effective strategies. This adaptive approach ensures that the algorithm can efficiently explore and exploit the solution space, increasing the likelihood of finding the global optimum.

Rather than relying on a single, static mutation strategy, this probabilistic adjustment allows the algorithm to balance exploration and exploitation dynamically. This flexibility is crucial in navigating complex optimization problems and represents a significant improvement that enhances the overall performance of the FECOIMO algorithm.

4.5. Heuristic Operator

Heuristic operators are strategies or rules employed during problem-solving to guide the search process. They are designed based on specific knowledge and experience within the problem domain to help algorithms avoid local optima, accelerate the search process, reduce the search space, and thus more efficiently explore the solution space. Heuristic operators aim to find high-quality solutions, improve the efficiency and performance of algorithms, and identify optimal solutions within a reasonable timeframe.

In this paper, when optimizing the picking plan ($\mathbf{z}_{ij}^{k}$) in the TTP, the thief's speed is directly affected by the weight of the items carried. Heavier items will slow down the thief's travel speed. Additionally, since each item is available in multiple cities, denoted as $\mathbf{avc}(t)$ ($t = 1, 2, \cdots, m$), for each item t, the last city x_b on the optimized route ($\mathbf{x}_{ij}^{k+1}$) that contains the item t is identified. By picking item t from x_b, the value of the item is ensured, and the travel time due to picking item t is minimized, thereby maximizing the thief's overall profit. Therefore, this paper designs the heuristic operator (f_{HO}) to optimize the picking plan ($\mathbf{z}_{ij}^{k}$).

The heuristic operator (f_{HO}) is designed to optimize the picking plan in the following steps:

(1) Adjustment of picked items:
Initially, all previously picked items in $\mathbf{z}_{ij}^{k}$ are adjusted to be picked from the last city x_b on the $\mathbf{x}_{ij}^{k+1}$ that contains the item t.

(2) Random removal of an item:
To increase algorithm diversity and generate a broader range of solutions, one of the already picked items z_{rt} is randomly removed from the knapsack with a probability of $p_{rt} = 0.5$.

(3) Calculation of remaining capacity:
The remaining capacity of the knapsack is then calculated, denoted as $restofWeight$.

(4) Selection of unpicked items:
Within the constraint of the knapsack's remaining capacity, unpicked items are selected based on their value-to-weight ratio $r_{vw}(t)$ in descending order. These items are also chosen from the last city x_b on the $\mathbf{x}_{ij}^{k+1}$ that contains the item t.

$$r_{vw}(t) = \frac{b_t}{w_t} \tag{18}$$

Overall, the heuristic operator enhances the algorithm's ability to explore and exploit the solution space effectively, leading to better optimization performance in solving the $\mathbf{z}_{ij}^{k}$ of TTP.

The following pseudocode in Algorithm 4 illustrates the implementation of the heuristic operator for optimizing the $\mathbf{z}_{ij}^{k}$.

Algorithm 4 Heuristic operator for the picking plan

```
Require: $\mathbf{x}_{ij}^{k+1}$, $\mathbf{z}_{ij}^{k}$, $Q$, $\boldsymbol{a}_{m\times n}$, $b_t$, $w_t$, $(t = 1, 2, \cdots, m)$
Ensure: $\mathbf{z}_{ij}^{k+1}$
 1: Initialize $\mathbf{z}_{ij}^{k+1} \leftarrow \mathbf{z}_{ij}^{k}$
 2: for each item $t$ in $\mathbf{z}_{ij}^{k+1}$ do
 3:     Find the last city $x_b$ on $\mathbf{x}_{ij}^{k+1}$ that contains item $t$
 4:     Update $\mathbf{z}_{ij}^{k+1}$ to pick item $t$ from city $x_b$
 5: end for
 6: if $rand < p_{rt}$ then
 7:     Select a random item $rt$ from $\mathbf{z}_{ij}^{k+1}$ that $z_{rt} \neq 0$
 8:     Remove item $rt$ from $\mathbf{z}_{ij}^{k+1}$: $z_{rt} \leftarrow 0$
 9: end if
10: Calculate the remaining capacity of the knapsack: $restofWeight$
11: Sort unpicked items by their $r_{vw}$ in descending order
12: for $t' = 1 \rightarrow$ size(unpicked items) do
13:     if $w_{t'} \leq restofWeight \wedge rand < p_{rt}$ then
14:         Add item $t'$ to $\mathbf{z}_{ij}^{k+1}$ from city $x_b$
15:         Update the $restofWeight$
16:     end if
17: end for
18: Return $\mathbf{z}_{ij}^{k+1}$
```

4.6. Update the Elements

As an iterative algorithm, the performance of FECOIMO in solving the TTP heavily relies on the updating process of its elements. In FECOIMO, the evaluation of elements is based on the force F of each element, which is closely related to the objective function G. Specifically, if $F_{ij}^{k} > 0$, the element $\mathbf{e}_{ij}^{k}$ is considered a good solution and should be retained. As the objective function value of an element approaches that of the optimal element in the population, the mass value M_{ij}^{k}, as defined, decreases, leading to a correspondingly larger force value F_{ij}^{k}. Conversely, if $F_{ij}^{k} \leq 0$, the element $\mathbf{e}_{ij}^{k}$ needs to be updated.

New elements are generated in the vicinity of the optimal element within the j-th cycle $\mathbf{e}_{oj}^{k}$ and the global optimal element $\mathbf{e}_{best}^{k}$ to replace it. The probability of updating $\mathbf{e}_{oj}^{k}$ is p_s, and the probability of updating $\mathbf{e}_{best}^{k}$ is $1 - p_s$. Through conditional checks and random selection of mutation targets, the algorithm achieves a balance between exploitation (utilizing the currently known best solutions) and exploration (seeking new solutions). Specifically, when the cycle's best element is chosen for mutation, the algorithm focuses more on local search; when the global best element is chosen, it emphasizes global search. This balance helps to improve the overall performance of the algorithm. The pseudocode for the element $\mathbf{e}_{ij}^{k}$ updating process is shown in Algorithm 5.

Algorithm 5 The update of the element $\mathbf{e}_{ij}^{k}$

```
Require: $\mathbf{x}_{ij}^{k}$, $\mathbf{z}_{ij}^{k}$, $\mathbf{e}_{oj}^{k} = [\mathbf{x}_{oj}^{k}, \mathbf{z}_{oj}^{k}]$, $\mathbf{e}_{best}^{k} = [\mathbf{x}_{best}^{k}, \mathbf{z}_{best}^{k}]$, $p_s$
Ensure: $\mathbf{x}_{ij}^{k+1}$, $\mathbf{z}_{ij}^{k+1}$
 1: if $F_{ij}^{k} > 0$ then
 2:     $(\mathbf{x}_{ij}^{k+1}, \mathbf{z}_{ij}^{k+1}) \Leftarrow (\mathbf{x}_{ij}^{k}, \mathbf{z}_{ij}^{k})$
 3: else
 4:     if $rand < p_s$ then
 5:         $\mathbf{x}_{ij}^{k+1} \Leftarrow f_{IMO}(\mathbf{x}_{oj}^{k}))$
 6:         $\mathbf{z}_{ij}^{k+1} \Leftarrow f_{HO}(\mathbf{x}_{ij}^{k+1}, \mathbf{z}_{oj}^{k})$
 7:     else
 8:         $\mathbf{x}_{ij}^{k+1} \Leftarrow f_{IMO}(\mathbf{x}_{best}^{k})$
 9:         $\mathbf{z}_{ij}^{k+1} \Leftarrow f_{HO}(\mathbf{x}_{ij}^{k+1}, \mathbf{z}_{best}^{k})$
10:     end if
11: end if
12: Return $\mathbf{x}_{ij}^{k+1}$, $\mathbf{z}_{ij}^{k+1}$
```

4.7. Implementation of FECOIMO

The flowchart of the FECOIMO algorithm for solving the TTP is depicted in Figure 6.

The process begins with setting the algorithm parameters and generating the initial population. The initial objective function values G_{ij}^{k}, the masses M_{ij}^{k}, and the forces F_{ij}^{k} of the elements are then calculated. The optimal element within the j-th cycle $\mathbf{e}_{oj}^{k}$ and the current optimal element $\mathbf{e}_{best}^{k}$ are identified.

During the k-th generation, the algorithm determines how to update the elements based on the calculated forces. Specifically, the elements are updated using different strategies depending on whether the force is nonpositive and based on a probabilistic decision.

The mutation operator (f_{IMO}) is applied to update the elements. The selection of the specific mutation operator ($f_{flip}, f_{insert}, f_{move}$) is based on random probabilities. If the random probability p_{IMO} is less than or equal to a predefined threshold r_1, the f_{flip} operator is used. If p_{IMO} lies between r_1 and r_2, the f_{insert} operator is used. Otherwise, the f_{move} operator is applied.

When the tour plan $\mathbf{x}_{ij}^{k}$ has been updated, the heuristic operator f_{HO} is then used to update another decision variable, the picking plan $\mathbf{z}_{ij}^{k}$.

After updating the elements, the objective function values and masses are recalculated. The current optimal element is updated accordingly. This process is repeated until the termination criteria are met ($k > k_{max}$), at which point the algorithm stops and outputs the optimal solution ($\mathbf{x}_{best}^{k}, \mathbf{z}_{best}^{k}$).

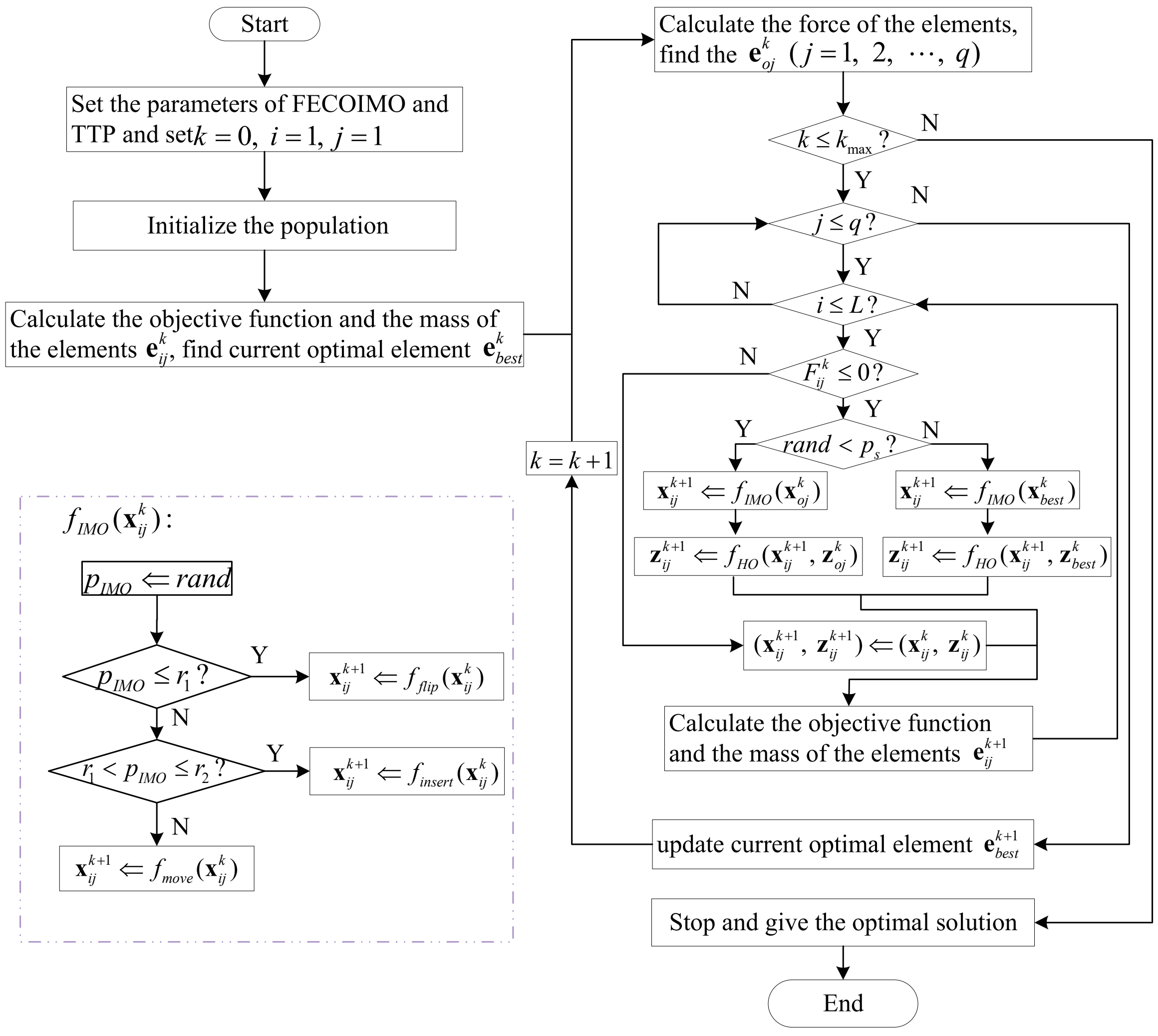

Start
Set the parameters of FECOIMO and TTP and set $k=0,\ i=1,\ j=1$
Initialize the population
Calculate the objective function and the mass of the elements $\mathbf{e}_{ij}^k$, find current optimal element $\mathbf{e}_{best}^k$
Calculate the force of the elements, find the $\mathbf{e}_{oj}^k$ $(j=1,\ 2,\ \cdots,\ q)$
$k \le k_{\max}$?
$j \le q$?
$i \le L$?
$F_{ij}^k \le 0$?
$rand < p_s$?
$\mathbf{x}_{ij}^{k+1} \Leftarrow f_{IMO}(\mathbf{x}_{oj}^k)$
$\mathbf{x}_{ij}^{k+1} \Leftarrow f_{IMO}(\mathbf{x}_{best}^k)$
$\mathbf{z}_{ij}^{k+1} \Leftarrow f_{HO}(\mathbf{x}_{ij}^{k+1}, \mathbf{z}_{oj}^k)$
$\mathbf{z}_{ij}^{k+1} \Leftarrow f_{HO}(\mathbf{x}_{ij}^{k+1}, \mathbf{z}_{best}^k)$
$(\mathbf{x}_{ij}^{k+1}, \mathbf{z}_{ij}^{k+1}) \Leftarrow (\mathbf{x}_{ij}^k, \mathbf{z}_{ij}^k)$
Calculate the objective function and the mass of the elements $\mathbf{e}_{ij}^{k+1}$
update current optimal element $\mathbf{e}_{best}^{k+1}$
Stop and give the optimal solution
End
$k = k+1$
Y
N
$f_{IMO}(\mathbf{x}_{ij}^k)$:
$p_{IMO} \Leftarrow rand$
$p_{IMO} \le r_1$?
$\mathbf{x}_{ij}^{k+1} \Leftarrow f_{flip}(\mathbf{x}_{ij}^k)$
$r_1 < p_{IMO} \le r_2$?
$\mathbf{x}_{ij}^{k+1} \Leftarrow f_{insert}(\mathbf{x}_{ij}^k)$
$\mathbf{x}_{ij}^{k+1} \Leftarrow f_{move}(\mathbf{x}_{ij}^k)$

Figure 6. The flowchart of Five-element Cycle Optimization algorithm based on Integrated Mutation Operator (FECOIMO).

5. Experimental Results and Analysis

This section presents a comprehensive analysis of the experimental results obtained by applying the proposed Five-Element Cyclic Integrated Mutation Optimization (FECOIMO) algorithm to the Traveling Thief Problem (TTP). The experiments are designed to validate the effectiveness of FECOIMO in solving a variety of TTP instances, ranging from small to large scales, and to compare its performance with that of five well-known metaheuristic algorithms. The analysis covers several aspects, including the operation environment and TTP instances used, the determination of maximum iteration for each instance, the effects of different mutation operators, the determination of update probability, and a comparative analysis of FECOIMO against other algorithms in terms of performance, convergence behavior, and execution time.

5.1. Operation Environment and Traveling Thief Problem (TTP) Instances

The experiments conducted in this study were implemented using MATLAB R2018a and were run on a system equipped with a 2.4 GHz Intel Xeon-E5645 processor, 32 GB of RAM, and running Windows 10. The dataset utilized in the experiments consists of a representative subset of the TTP instances generated by Bonyadi et al. [1].

The naming convention for the TTP instances follows the pattern n-m-ID-τ, where n represents the number of cities, m denotes the number of items, ID is the instance identity, and τ indicates the tightness of the capacity constraint (i.e., the ratio of the knapsack capacity to the total weight of the items). The values for n are chosen as $10,\ 20,\ 50$, , and 100, with corresponding values for m as described in the paper. The values for n are chosen as 10, 20, 50, and 100, with corresponding values for m as described in the paper. Specifically, for $n = 10,\ m = 10,\ 15$; for $n = 20,\ m = 10,\ 20,\ 30$; for $n = 50,\ m = 15,\ 25,\ 50,\ 75$; and for $n = 100,\ m = 10,\ 25,\ 50,\ 100$. The instances considered in this paper correspond to $ID = 1$, and three values of τ are selected: $25,\ 50$ and 75. This selection results in a total of 39 TTP instances, encompassing various scales, to comprehensively validate the performance of the FECOIMO.

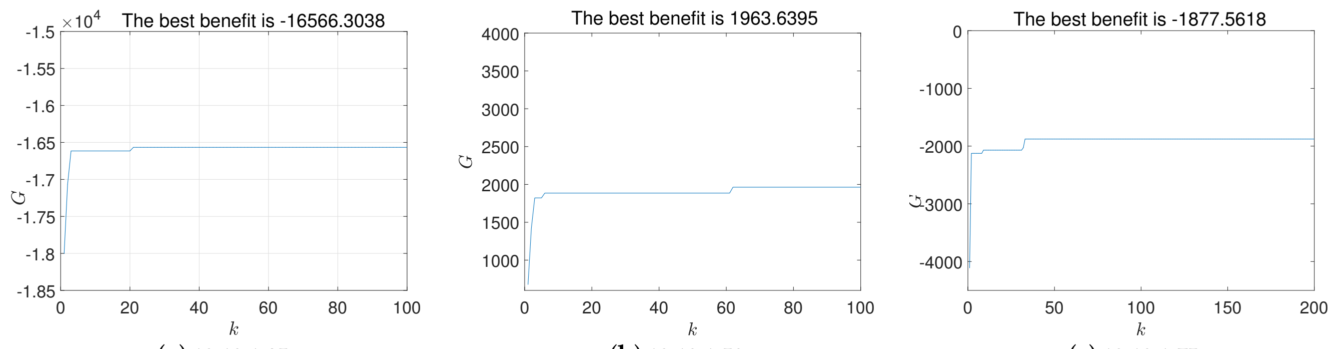
The parameters of the proposed FECOIMO include the number of cycles q, the number of elements in each cycle L, the maximum iteration k_{max}, the update probability p_s, and parameters r_1 and r_2. Based on the findings in reference [33], $q = 20$ and $L = 5$ were set. The remaining parameters were fine-tuned through experimentation.

5.2. Determination of the Maximum Iteration Corresponding to Each TTP Instance

Due to the varying number of cities (n) and items (m) in each TTP instance, the corresponding maximum iteration count (k_{max}) also differs. The determination of k_{max} is based on the evolutionary process of FECOIMO when solving each instance. Specifically, k_{max} is identified as the iteration at which the objective function value converges.

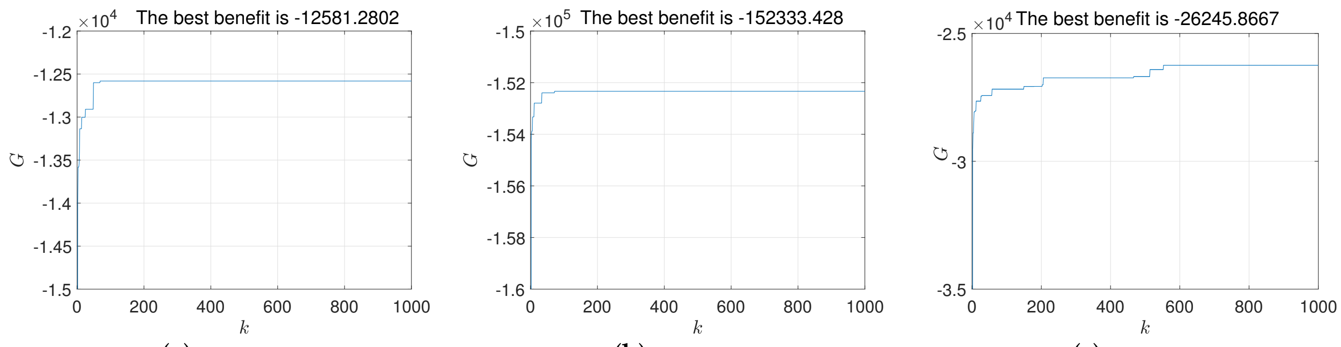
For brevity, Figures 7–9 display convergence curves for a subset of 9 out of the 39 instances. These figures visually represent the convergence process for solving small ($n = 10,\ m = 10$), medium ($n = 50,\ m = 50$), and large ($n = 100,\ m = 100$) TTP instances. Each subplot within the graphs illustrates the convergence for instances of different types (i.e., different τ) but of the same scale.

Observing the convergence curves, it becomes evident that the number of iterations required for convergence primarily depends on n and m, with larger values leading to larger k_{max}. Based on the convergence behavior, suitable values for k_{max} are determined for each instance, as outlined in Table 2.



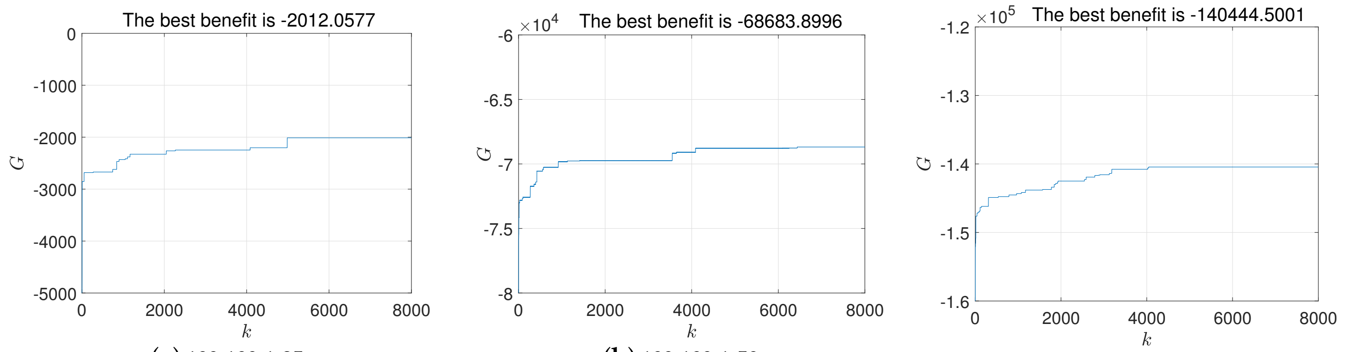
(**a**) 10-10-1-25 (**b**) 10-10-1-50 (**c**) 10-10-1-75

Figure 7. Convergence curves of the Traveling Thief Problem (TTP) instances with $n = 10,\ m = 10$.



(**a**) 50-25-1-25 (**b**) 50-25-1-50 (**c**) 50-25-1-75

Figure 8. Convergence curves of the TTP instances with $n = 50,\ m = 25$.



(a) 100-100-1-25 **(b)** 100-100-1-50 **(c)** 100-100-1-75

Figure 9. Convergence curves of the TTP instances with $n = 100$, $m = 100$.

Table 2. Iteration numbers of each instance.

Instance	k_{max}	Instance	k_{max}	Instance	k_{max}
10-10-1-25	100	10-10-1-50	100	10-10-1-75	100
10-15-1-25	200	10-15-1-50	200	10-15-1-75	200
20-10-1-25	300	20-10-1-50	300	20-10-1-75	300
20-20-1-25	300	20-20-1-50	300	20-20-1-75	400
20-30-1-25	400	20-30-1-50	600	20-30-1-75	600
50-15-1-25	400	50-15-1-50	600	50-15-1-75	600
50-25-1-25	800	50-25-1-50	800	50-25-1-75	800
50-50-1-25	2000	50-50-1-50	2000	50-50-1-75	2000
50-75-1-25	5000	50-75-1-50	5000	50-75-1-75	5000
100-10-1-25	4000	100-10-1-50	4000	100-10-1-75	6000
100-25-1-25	5000	100-25-1-50	5000	100-25-1-75	5000
100-50-1-25	5000	100-50-1-50	5000	100-50-1-75	5000
100-100-1-25	6000	100-100-1-50	6000	100-100-1-75	6000

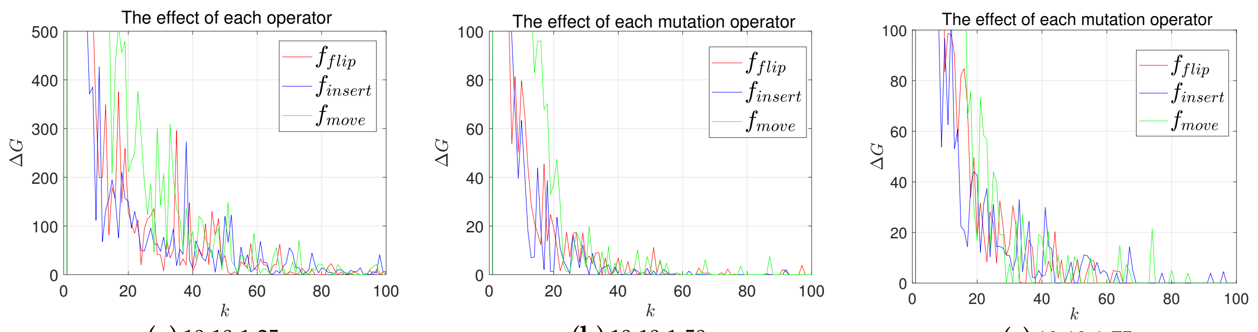
5.3. Determination of the Integrated Mutation Operator

In this paper, the effect of each operator on each instance is calculated according to the method introduced in Section 4.4.4, and the probability of each mutation operator is thereby determined to integrate the operators. Initially, the parameters are set as $r_1 = 1/3$ and $r_2 = 2/3$ to ensure equal use probability for all three mutation operators. The maximum iteration count k_{max} is determined based on Table 2. To mitigate experiment contingency, the FECOIMO is independently run 30 times on each instance, and the final effect of each operator is averaged over these 30 runs.

The Figures 10–12 illustrate the average effect of three mutation operators (f_{flip}, f_{insert}, f_{move}) during the evolution process across nine representative TTP instances out of a total of 39 instances. These nine instances are selected to represent different scales, including small-scale (e.g., $n = 10$, $m = 10$), medium-scale (e.g., $n = 50$, $m = 50$), and large-scale instances (e.g., $n = 100$, $m = 100$). Each graph shows the average effect of the mutation operators over 30 independent runs of FECOIMO, highlighting the contribution of each operator throughout the evolutionary process.

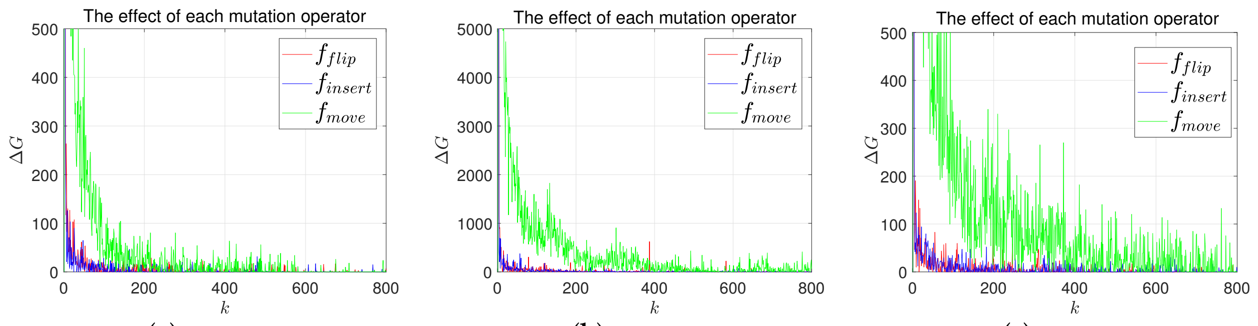
It can be concluded that the f_{flip} and the f_{insert} have comparable effects, while the f_{move} performs better than both, especially as the scale of the instance increases. In smaller-scale TTP instances (Figure 10), the effects of the f_{flip} and the f_{insert} operators are similar, while the f_{move} operator shows a significantly better effect. In medium-scale TTP instances (Figure 11), the f_{move} operator performs significantly better than the f_{flip} and the f_{insert} operators, demonstrating stronger optimization capability. In large-scale TTP instances (Figure 12), the effect of the f_{move} is the most prominent, with its optimization capability far exceeding the other two operators, especially in large-scale problems. The f_{move} shows stronger optimization capability across different scales of TTP instances, particularly in large-scale problems. This may be due to the f_{move} operator's ability to more effectively adjust the structure of the solution, thus finding better solutions in more complex search spaces. This finding indicates that considering the roles and advantages of different operators is crucial for improving the overall performance of the algorithm in the design and optimization process.

Table 3 provides a quantitative analysis of the effect of each mutation operator across all 39 instances. Notably, f_{flip} and f_{insert} have comparable effects, while f_{move} becomes more influential as the instance scale increases. For instance, at a scale of $n = 10,\ m = 10$, f_{move} is twice as effective as f_{flip}, whereas for $n = 100,\ m = 100$, its effectiveness is nearly quadruple. This underscores the importance of f_{move} in exploring solutions within a larger search space. From the results calculated in Table 3, the usage probabilities of each operator in f_{IMO} are $p_{f_{flip}} = 0.233,\ p_{f_{insert}} = 0.233,\ p_{f_{move}} = 0.547$. Therefore, $r_1 = 0.233,\ r_2 = 0.453$.



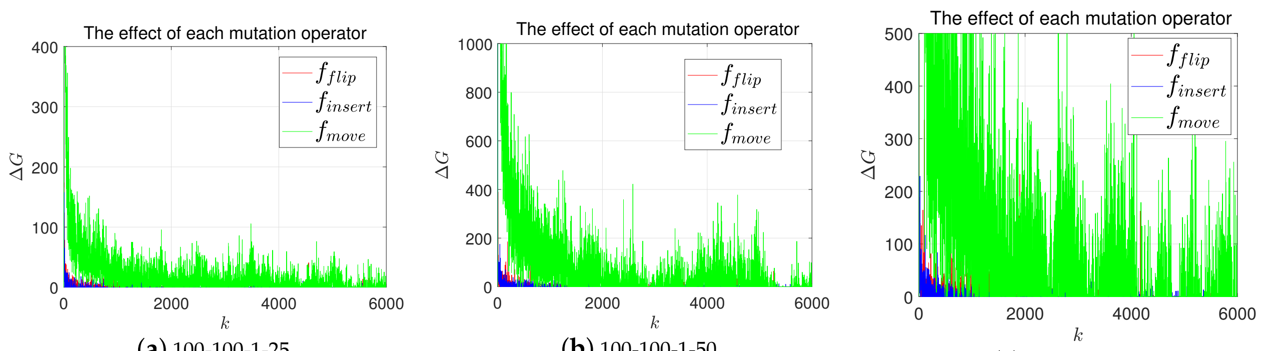
(**a**) 10-10-1-25 (**b**) 10-10-1-50 (**c**) 10-10-1-75

Figure 10. The average effect of each mutation operator over 30 runs on the instance with n = 10, m = 10.



(**a**) 50-25-1-25 (**b**) 50-25-1-50 (**c**) 50-25-1-75

Figure 11. The average effect of each mutation operator over 30 runs on the instance with $n = 50,\ m = 25$.



(**a**) 100-100-1-25 (**b**) 100-100-1-50 (**c**) 100-100-1-75

Figure 12. The average effect of each mutation operator over 30 runs on the instance with $n = 100,\ m = 100$.

Table 3. The effect of each mutation operator in each instance.

Instance \ Mutation Operator	$E_{f_{flip}}$	$E_{f_{insert}}$	$E_{f_{move}}$
10-10-1-25	234.10	220.46	447.58
10-10-1-50	35.49	31.47	62.30
10-10-1-75	90.53	87.01	141.97
10-15-1-25	24.08	24.76	41.94
10-15-1-50	57.72	53.80	76.42
10-15-1-75	153.01	158.01	220.68
20-10-1-25	8.19	6.35	17.30
20-10-1-50	11.39	9.18	22.26
20-10-1-75	31.33	29.86	66.82
20-20-1-25	36.09	32.75	70.10
20-20-1-50	60.59	52.91	114.23
20-20-1-75	230.15	207.86	403.16
20-30-1-25	46.35	39.83	84.80
20-30-1-50	39.49	35.96	69.75
20-30-1-75	95.82	87.98	182.88
50-15-1-25	15.30	12.23	40.00
50-15-1-50	19.87	17.87	52.31
50-15-1-75	76.98	71.08	203.02
50-25-1-25	36.72	35.22	89.44
50-25-1-50	289.70	264.99	810.68
50-25-1-75	80.98	80.17	221.98
50-50-1-25	27.65	26.28	70.62
50-50-1-50	115.24	112.25	291.09
50-50-1-75	219.30	209.42	556.14
50-75-1-25	23.59	23.87	62.95
50-75-1-50	15.14	14.09	36.47
50-75-1-75	8.68	8.19	22.76
100-10-1-25	1.33	1.03	5.23
100-10-1-50	2.42	2.18	8.89
100-10-1-75	3.89	3.25	11.54
100-25-1-25	7.51	7.13	26.66
100-25-1-50	7.13	6.41	24.23
100-25-1-75	30.67	29.41	118.83
100-50-1-25	31.41	32.62	161.57
100-50-1-50	15.49	14.91	67.90
100-50-1-75	23.21	22.94	90.79
100-100-1-25	53.95	51.88	189.37
100-100-1-50	28.64	28.30	119.95
100-100-1-75	53.95	51.88	189.37
S_{f_m}	2343.06	2205.80	5493.99
p_{f_m}	0.233	0.220	0.547

5.4. Determination of Update Probability

When employing FECOIMO to update elements, the parameter p_s plays a crucial role in determining the probability of updating $\mathbf{e}^k_{best}$ and $\mathbf{e}^k_{oj}$. As delineated in Algorithm 5, the probability of selecting $\mathbf{e}^k_{oj}$ is represented by p_s, while the probability of selecting $\mathbf{e}^k_{best}$ is $(1 - p_s)$. Similarly, our proposed algorithm conducts parameter experiments, where only the value of p_s is varied for comparison experiments. The parameter p_s ranges from 0 to 1, with $p_s = 0$ signifying mutation solely on $\mathbf{e}^k_{best}$, and $p_s = 1$ indicating mutation exclusively on $\mathbf{e}^k_{oj}$.

To ensure experimental reliability, each instance is independently executed 30 times. Tables 4 and 5 present the mean benefits and Friedman test ranks from 30 independent runs of FECOIMO with different p_s values. The Friedman test ranks [37] indicate the performance of the algorithm under various p_s settings, with a lower rank indicating better overall performance.

Based on these results, the p-value from the Friedman test is $4.37E - 44$, indicating significant differences in performance between different p_s values. And the optimal value of p_s is determined to be 0.9, as it has the lowest Friedman rank (2.36) and the highest final rank (1), indicating the best overall performance of the algorithm.

This observation underscores the efficacy of prioritizing the update of the optimal solution within the cycle in most scenarios, complemented by occasional updates of the current optimal solution. This dual-update strategy enhances the algorithm's global search capability by facilitating synchronous exploration of all cycles, thereby improving search efficiency. Notably, updating the current optimal solution mitigates the risk of algorithmic convergence to local optima. The synergistic integration of these update methods substantially enhances the algorithm's search prowess.

Table 4. Mean of benefits and Friedman test ranks from 30 independent runs of FECOMO (I).

p_s Instance	$p_s = 0$	$p_s = 0.1$	$p_s = 0.2$	$p_s = 0.3$	$p_s = 0.4$	$p_s = 0.5$
10-10-1-25	-1.67×10^4	-1.67×10^4	-1.67×10^4	-1.67×10^4	-1.67×10^4	-1.67×10^4
10-10-1-50	1.95×10^3	1.93×10^3	1.92×10^3	1.95×10^3	1.94×10^3	1.94×10^3
10-10-1-75	-1.93×10^3	-1.93×10^3	-1.93×10^3	-1.92×10^3	-1.90×10^3	-1.90×10^3
10-15-1-25	3.81×10^2	3.83×10^2	3.81×10^2	3.82×10^2	3.81×10^2	3.79×10^2
10-15-1-50	1.18×10^3	1.16×10^3	1.14×10^3	1.14×10^3	1.18×10^3	1.17×10^3
10-15-1-75	-6.40×10^3	-6.36×10^3	-6.37×10^3	-6.37×10^3	-6.37×10^3	-6.38×10^3
20-10-1-25	7.02×10^2	6.96×10^2	6.98×10^2	7.11×10^2	7.07×10^2	6.98×10^2
20-10-1-50	1.98×10^3	1.98×10^3	1.98×10^3	2.01×10^3	2.00×10^3	1.99×10^3
20-10-1-75	-1.64×10^3	-1.67×10^3	-1.65×10^3	-1.63×10^3	-1.66×10^3	-1.64×10^3
20-20-1-25	-1.82×10^3	-1.78×10^3	-1.78×10^3	-1.73×10^3	-1.72×10^3	-1.75×10^3
20-20-1-50	-1.99×10^3	-1.97×10^3	-1.93×10^3	-1.89×10^3	-1.95×10^3	-1.91×10^3
20-20-1-75	-4.47×10^4	-4.46×10^4	-4.42×10^4	-4.43×10^4	-4.42×10^4	-4.44×10^4
20-30-1-25	-1.49×10^3	-1.49×10^3	-1.49×10^3	-1.45×10^3	-1.46×10^3	-1.52×10^3
20-30-1-50	-1.10×10^3	-9.82×10^2	-1.04×10^3	-9.51×10^2	-8.58×10^2	-8.49×10^2
20-30-1-75	-1.79×10^4	-1.80×10^4	-1.78×10^4	-1.79×10^4	-1.79×10^4	-1.78×10^4
50-15-1-25	-1.46×10^3	-1.44×10^3	-1.41×10^3	-1.45×10^3	-1.45×10^3	-1.39×10^3
50-15-1-50	-1.71×10^3	-1.61×10^3	-1.74×10^3	-1.66×10^3	-1.66×10^3	-1.64×10^3

Table 4. *Cont.*

Instance \ p_s	$p_s = 0$	$p_s = 0.1$	$p_s = 0.2$	$p_s = 0.3$	$p_s = 0.4$	$p_s = 0.5$
50-15-1-75	-2.45×10^4	-2.45×10^4	-2.42×10^4	-2.42×10^4	-2.40×10^4	-2.41×10^4
50-25-1-25	-1.28×10^4	-1.28×10^4	-1.27×10^4	-1.27×10^4	-1.28×10^4	-1.27×10^4
50-25-1-50	-1.57×10^5	-1.57×10^5	-1.55×10^5	-1.54×10^5	-1.55×10^5	-1.54×10^5
50-25-1-75	-2.76×10^4	-2.81×10^4	-2.76×10^4	-2.73×10^4	-2.75×10^4	-2.73×10^4
50-50-1-25	-2.15×10^4	-2.13×10^4	-2.14×10^4	-2.15×10^4	-2.12×10^4	-2.12×10^4
50-50-1-50	-1.27×10^5	-1.28×10^5	-1.28×10^5	-1.28×10^5	-1.28×10^5	-1.27×10^5
50-50-1-75	-2.64×10^5	-2.61×10^5	-2.59×10^5	-2.60×10^5	-2.60×10^5	-2.59×10^5
50-75-1-25	-6.06×10^4	-6.01×10^4	-6.00×10^4	-5.97×10^4	-5.92×10^4	-5.92×10^4
50-75-1-50	-1.37×10^4	-1.32×10^4	-1.32×10^4	-1.30×10^4	-1.29×10^4	-1.30×10^4
50-75-1-75	1.48×10^4	1.50×10^4	1.51×10^4	1.51×10^4	1.51×10^4	1.53×10^4
100-10-1-25	-1.60×10^3	-1.61×10^3	-1.60×10^3	-1.59×10^3	-1.62×10^3	-1.61×10^3
100-10-1-50	-2.17×10^3	-2.21×10^3	-2.23×10^3	-2.21×10^3	-2.16×10^3	-2.17×10^3
100-10-1-75	-1.06×10^4	-1.04×10^4	-1.04×10^4	-1.04×10^4	-1.03×10^4	-1.04×10^4
100-25-1-25	-1.89×10^4	-1.85×10^4	-1.86×10^4	-1.85×10^4	-1.84×10^4	-1.83×10^4
100-25-1-50	-1.37×10^4	-1.35×10^4	-1.34×10^4	-1.35×10^4	-1.34×10^4	-1.33×10^4
100-25-1-75	-8.56×10^4	-8.56×10^4	-8.52×10^4	-8.46×10^4	-8.42×10^4	-8.42×10^4
100-50-1-25	-9.41×10^4	-9.29×10^4	-9.35×10^4	-9.27×10^4	-9.22×10^4	-9.22×10^4
100-50-1-50	-3.07×10^4	-3.04×10^4	-2.98×10^4	-2.95×10^4	-2.95×10^4	-2.92×10^4
100-50-1-75	-5.10×10^4	-5.05×10^4	-4.96×10^4	-5.00×10^4	-4.99×10^4	-4.92×10^4
100-100-1-25	-2.28×10^3	-2.22×10^3	-1.97×10^3	-2.22×10^3	-2.04×10^3	-2.23×10^3
100-100-1-50	-7.29×10^4	-7.22×10^4	-7.18×10^4	-7.17×10^4	-7.14×10^4	-7.12×10^4
100-100-1-75	-1.50×10^5	-1.50×10^5	-1.48×10^5	-1.47×10^5	-1.48×10^5	-1.46×10^5
Friedman rank	9.64	9.17	8.47	7.24	6.81	6.51
Final rank	11	10	9	8	7	6

Table 5. Mean of benefits and Friedman test ranks from 30 independent runs of FECOMO (II).

Instance \ p_s	$p_s = 0.6$	$p_s = 0.7$	$p_s = 0.8$	$p_s = 0.9$	$p_s = 1$
10-10-1-25	-1.67×10^4	-1.66×10^4	-1.66×10^4	-1.66×10^4	-1.66×10^4
10-10-1-50	1.94×10^3	1.94×10^3	1.94×10^3	1.95×10^3	1.95×10^3
10-10-1-75	-1.90×10^3	-1.88×10^3	-1.90×10^3	-1.88×10^3	-1.88×10^3
10-15-1-25	3.81×10^2	3.81×10^2	3.83×10^2	3.84×10^2	3.83×10^2
10-15-1-50	1.14×10^3	1.18×10^3	1.18×10^3	1.20×10^3	1.23×10^3
10-15-1-75	-6.36×10^3	-6.36×10^3	-6.38×10^3	-6.34×10^3	-6.37×10^3
20-10-1-25	7.04×10^2	7.07×10^2	7.08×10^2	7.14×10^2	7.13×10^2
20-10-1-50	2.02×10^3	2.01×10^3	2.01×10^3	2.02×10^3	2.02×10^3
20-10-1-75	-1.64×10^3	-1.64×10^3	-1.63×10^3	-1.63×10^3	-1.61×10^3

Table 5. *Cont.*

p_s / Instance	$p_s = 0.6$	$p_s = 0.7$	$p_s = 0.8$	$p_s = 0.9$	$p_s = 1$
20-20-1-25	-1.75×10^3	-1.72×10^3	-1.74×10^3	-1.69×10^3	-1.72×10^3
20-20-1-50	-1.87×10^3	-1.91×10^3	-1.89×10^3	-1.93×10^3	-1.92×10^3
20-20-1-75	-4.41×10^4	-4.42×10^4	-4.39×10^4	-4.39×10^4	-4.39×10^4
20-30-1-25	-1.41×10^3	-1.51×10^3	-1.51×10^3	-1.46×10^3	-1.46×10^3
20-30-1-50	-9.37×10^2	-9.04×10^2	-8.47×10^2	-8.53×10^2	-7.32×10^2
20-30-1-75	-1.78×10^4	-1.77×10^4	-1.77×10^4	-1.77×10^4	-1.76×10^4
50-15-1-25	-1.37×10^3	-1.35×10^3	-1.37×10^3	-1.35×10^3	-1.38×10^3
50-15-1-50	-1.60×10^3	-1.66×10^3	-1.61×10^3	-1.58×10^3	-1.57×10^3
50-15-1-75	-2.40×10^4	-2.40×10^4	-2.39×10^4	-2.39×10^4	-2.38×10^4
50-25-1-25	-1.27×10^4	-1.26×10^4	-1.26×10^4	-1.25×10^4	-1.26×10^4
50-25-1-50	-1.54×10^5	-1.54×10^5	-1.54×10^5	-1.53×10^5	-1.53×10^5
50-25-1-75	-2.73×10^4	-2.72×10^4	-2.72×10^4	-2.71×10^4	-2.70×10^4
50-50-1-25	-2.14×10^4	-2.12×10^4	-2.12×10^4	-2.12×10^4	-2.10×10^4
50-50-1-50	-1.27×10^5	-1.27×10^5	-1.27×10^5	-1.27×10^5	-1.26×10^5
50-50-1-75	-2.58×10^5	-2.57×10^5	-2.58×10^5	-2.58×10^5	-2.57×10^5
50-75-1-25	-5.92×10^4	-5.93×10^4	-5.90×10^4	-5.88×10^4	-5.87×10^4
50-75-1-50	-1.27×10^4	-1.28×10^4	-1.26×10^4	-1.25×10^4	-1.24×10^4
50-75-1-75	1.53×10^4	1.54×10^4	1.54×10^4	1.54×10^4	1.55×10^4
100-10-1-25	-1.58×10^3	-1.60×10^3	-1.57×10^3	-1.59×10^3	-1.58×10^3
100-10-1-50	-2.21×10^3	-2.20×10^3	-2.15×10^3	-2.13×10^3	-2.16×10^3
100-10-1-75	-1.02×10^4	-1.03×10^4	-1.02×10^4	-1.01×10^4	-1.03×10^4
100-25-1-25	-1.83×10^4	-1.83×10^4	-1.82×10^4	-1.83×10^4	-1.83×10^4
100-25-1-50	-1.32×10^4	-1.32×10^4	-1.32×10^4	-1.32×10^4	-1.33×10^4
100-25-1-75	-8.45×10^4	-8.42×10^4	-8.41×10^4	-8.38×10^4	-8.43×10^4
100-50-1-25	-9.14×10^4	-9.20×10^4	-9.21×10^4	-9.21×10^4	-9.18×10^4
100-50-1-50	-2.92×10^4	-2.90×10^4	-2.92×10^4	-2.92×10^4	-2.92×10^4
100-50-1-75	-4.95×10^4	-4.93×10^4	-4.94×10^4	-4.90×10^4	-4.89×10^4
100-100-1-25	-2.08×10^3	-2.18×10^3	-1.92×10^3	-1.87×10^3	-1.97×10^3
100-100-1-50	-7.12×10^4	-7.07×10^4	-7.06×10^4	-7.11×10^4	-7.04×10^4
100-100-1-75	-1.45×10^5	-1.45×10^5	-1.45×10^5	-1.46×10^5	-1.46×10^5
Friedman rank	4.99	4.56	3.60	2.36	2.64
Final rank	5	4	3	1	2

5.5. Comparative Analysis of Algorithms

To validate the effectiveness of the proposed FECOIMO algorithm in solving the TTP, it was compared against five other metaheuristic algorithms: Enhanced Simulated Annealing (ESA) [18], Improved Grey Wolf Optimization Algorithm (IGWO) [38], Improved Whale Optimization Algorithm (IWOA) [39], Genetic Algorithm (GA) [40], and Profit Guided Coordination Heuristic (PGCH) [41]. To ensure a fair comparison, the parameter settings of these algorithms were aligned with those of FECOIMO wherever similar parameters were involved, while other parameters were set according to their original proposals. Each algorithm was independently executed 30 times across 39 TTP instances. Their performance was evaluated based on execution time, solution quality, and statistical significance using the Friedman test.

5.5.1. Performance Comparison

Tables 6–8 show the mean, maximum, and standard deviations of the objective function values obtained from 30 independent runs of the six algorithms on 39 TTP instances. These tables provide insights into the performance of each algorithm on different-sized TTP instances. At the bottom of each table, the Friedman test ranks are displayed, offering a statistical comparison of the algorithms' performances.

It can be observed that FECOIMO consistently ranks first in the Friedman test, indicating superior performance across all instances. Specifically, the p-values of 2.89×10^{-38}, 7.17×10^{-37}, and 5.37×10^{-35} suggest significant differences among the six algorithms. Moreover, as the problem instance size increases, FECOIMO's performance becomes noticeably superior compared with the other algorithms. This demonstrates the effectiveness of the operators designed specifically for FECOIMO in addressing larger instances, whereas the simpler mechanisms employed by the other algorithms likely contribute to their inability to find optimal solutions for larger instances.

It can be observed that FECOIMO consistently ranks first in the Friedman test, indicating superior performance across all instances. Specifically, the p-values of 2.89×10^{-38}, 7.17×10^{-37}, and 5.37×10^{-35} suggest significant differences among the six algorithms. Moreover, as the problem instance size increases, FECOIMO's performance becomes noticeably superior compared with the other algorithms. This demonstrates the effectiveness of the operators designed specifically for FECOIMO in addressing larger instances, whereas the simpler mechanisms employed by the other algorithms likely contribute to their inability to find optimal solutions for larger instances.

Given that the p-values in Tables 6–8 are p-value = 2.89×10^{-38}, p-value = 7.17×10^{-37}, and p-value = 5.37×10^{-35}, respectively, it is clear that there are significant differences among the six algorithms. To further identify the specific differences between each pair of algorithms, multiple comparison tests were applied. Tables 9–11 show the adjusted p-values from these multiple comparison tests. The detailed comparisons reveal significant differences between specific pairs of algorithms. For example, in Table 9, the p-value for the comparison between GA and FECOIMO is 6.35×10^{-5}. In these tables, p-values below 0.05 indicate that the differences between the corresponding row and column algorithms are statistically significant.

The differences between ESA and FECOIMO are not significant, as the p-values in Tables 9–11 are greater than 0.05. This indicates that there are no significant differences between ESA and FECOIMO in terms of the mean, variance, and maximum results. Although FECOIMO and Enhanced Simulated Annealing (ESA) do not show significant differences in certain statistical metrics, this actually underscores the strength of FECOIMO. ESA is a mature and well-established algorithm, particularly effective in path optimization problems. However, FECOIMO not only matches ESA's performance but also demonstrates advantages in balancing exploration and exploitation and avoiding premature convergence. This suggests that FECOIMO is highly adaptable and robust in handling complex optimization problems.

From Tables 9–11, it can be observed that the differences in the mean results between GA and ESA, IWOA, and PGCH are not significant. This indicates that their overall performance is comparable, even though their performance might vary in specific instances or under certain conditions. Similarly, the differences in the maximum results between GA and ESA, IWOA, and PGCH are also not significant, suggesting that their capabilities in finding the optimal solutions are similar. The final optimal solutions they find are very close. Additionally, the differences in the variance of results between IWOA, IGWO, and PGCH are not significant, indicating that these three algorithms have similar stability and consistency. The degree of fluctuation in the quality of solutions they obtain across different runs is comparable.

Table 6. Mean of benefits and Friedman test ranks from 30 independent runs of the six algorithms (p-value = 2.89×10^{-38}).

Algorithm Instance	GA	ESA	IWOA	IGWO	PGCH	FECOIMO
10-10-1-25	-1.78×10^{4}	-1.72×10^{4}	-1.84×10^{4}	-2.58×10^{4}	-2.39×10^{4}	-1.66×10^{4}
10-10-1-50	1.62×10^{3}	1.73×10^{3}	1.23×10^{3}	1.12×10^{2}	-1.13×10^{2}	1.95×10^{3}
10-10-1-75	-2.58×10^{3}	-2.85×10^{3}	-3.69×10^{3}	-6.02×10^{3}	-4.62×10^{3}	-1.88×10^{3}
10-15-1-25	-1.21×10^{1}	1.59×10^{2}	-8.89×10^{2}	-2.42×10^{3}	-1.78×10^{3}	3.84×10^{2}
10-15-1-50	7.00×10^{2}	7.93×10^{2}	-4.42×10^{2}	-2.59×10^{3}	-1.64×10^{3}	1.20×10^{3}
10-15-1-75	-6.94×10^{3}	-7.65×10^{3}	-1.07×10^{4}	-1.79×10^{4}	-1.29×10^{4}	-6.34×10^{3}
20-10-1-25	5.01×10^{2}	6.12×10^{2}	-9.08×10^{2}	-1.64×10^{3}	-1.73×10^{3}	7.14×10^{2}
20-10-1-50	1.64×10^{3}	1.85×10^{3}	4.95×10^{2}	-1.25×10^{3}	-2.04×10^{3}	2.02×10^{3}
20-10-1-75	-2.34×10^{3}	-2.06×10^{3}	-8.58×10^{3}	-1.51×10^{4}	-1.21×10^{4}	-1.63×10^{3}
20-20-1-25	-2.68×10^{3}	-2.33×10^{3}	-8.93×10^{3}	-1.28×10^{4}	-1.22×10^{4}	-1.69×10^{3}
20-20-1-50	-3.68×10^{3}	-2.71×10^{3}	-1.49×10^{4}	-2.46×10^{4}	-1.80×10^{4}	-1.93×10^{3}
20-20-1-75	-5.03×10^{4}	-4.73×10^{4}	-1.01×10^{5}	-1.57×10^{5}	-1.00×10^{5}	-4.39×10^{4}
20-30-1-25	-2.81×10^{3}	-2.49×10^{3}	-1.16×10^{4}	-2.01×10^{4}	-1.77×10^{4}	-1.46×10^{3}
20-30-1-50	-2.77×10^{3}	-2.74×10^{3}	-1.44×10^{4}	-2.82×10^{4}	-2.14×10^{4}	-8.53×10^{2}
20-30-1-75	-2.34×10^{4}	-2.10×10^{4}	-5.49×10^{4}	-9.59×10^{4}	-6.14×10^{4}	-1.77×10^{4}
50-15-1-25	-3.16×10^{3}	-1.55×10^{3}	-1.41×10^{4}	-1.59×10^{4}	-1.54×10^{4}	-1.35×10^{3}
50-15-1-50	-5.29×10^{3}	-2.22×10^{3}	-2.81×10^{4}	-3.25×10^{4}	-2.71×10^{4}	-1.58×10^{3}
50-15-1-75	-3.91×10^{4}	-2.65×10^{4}	-1.43×10^{5}	-1.61×10^{5}	-1.14×10^{5}	-2.39×10^{4}
50-25-1-25	-1.75×10^{4}	-1.37×10^{4}	-7.61×10^{4}	-8.71×10^{4}	-7.92×10^{4}	-1.25×10^{4}
50-25-1-50	-2.13×10^{5}	-1.64×10^{5}	-7.31×10^{5}	-8.42×10^{5}	-6.20×10^{5}	-1.53×10^{5}
50-25-1-75	-4.36×10^{4}	-3.13×10^{4}	-1.90×10^{5}	-2.22×10^{5}	-1.50×10^{5}	-2.71×10^{4}
50-50-1-25	-2.81×10^{4}	-2.31×10^{4}	-1.47×10^{5}	-1.70×10^{5}	-1.68×10^{5}	-2.12×10^{4}
50-50-1-50	-1.59×10^{5}	-1.36×10^{5}	-7.05×10^{5}	-8.52×10^{5}	-6.01×10^{5}	-1.27×10^{5}
50-50-1-75	-3.38×10^{5}	-2.79×10^{5}	-1.58×10^{5}	-2.05×10^{5}	-1.22×10^{5}	-2.58×10^{5}
50-75-1-25	-7.42×10^{4}	-6.57×10^{4}	-3.54×10^{5}	-4.12×10^{5}	-3.98×10^{5}	-5.88×10^{4}
50-75-1-50	-2.26×10^{4}	-1.67×10^{4}	-2.07×10^{5}	-2.55×10^{5}	-2.02×10^{5}	-1.25×10^{4}
50-75-1-75	8.33×10^{3}	9.51×10^{3}	-1.07×10^{5}	-1.61×10^{5}	-1.03×10^{5}	1.54×10^{4}
100-10-1-25	-6.83×10^{3}	-1.76×10^{3}	-2.13×10^{4}	-2.36×10^{4}	-2.31×10^{4}	-1.59×10^{3}
100-10-1-50	-1.26×10^{4}	-2.46×10^{3}	-3.49×10^{4}	-3.71×10^{4}	-3.37×10^{4}	-2.13×10^{3}
100-10-1-75	-3.98×10^{4}	-1.12×10^{4}	-9.42×10^{4}	-9.78×10^{4}	-7.93×10^{4}	-1.01×10^{4}
100-25-1-25	-3.95×10^{4}	-1.97×10^{4}	-1.55×10^{5}	-1.61×10^{5}	-1.42×10^{5}	-1.83×10^{4}
100-25-1-50	-3.94×10^{4}	-1.46×10^{4}	-1.52×10^{5}	-1.58×10^{5}	-1.21×10^{5}	-1.32×10^{4}
100-25-1-75	-2.25×10^{5}	-9.02×10^{4}	-8.25×10^{5}	-8.41×10^{5}	-5.64×10^{5}	-8.38×10^{4}
100-50-1-25	-1.41×10^{5}	-9.78×10^{4}	-8.30×10^{5}	-8.57×10^{5}	-7.49×10^{5}	-9.21×10^{4}
100-50-1-50	-5.87×10^{4}	-3.29×10^{4}	-4.36×10^{5}	-4.70×10^{5}	-3.11×10^{5}	-2.92×10^{4}
100-50-1-75	-1.11×10^{5}	-5.57×10^{4}	-6.68×10^{5}	-7.17×10^{5}	-4.37×10^{5}	-4.90×10^{4}
100-100-1-25	-1.25×10^{4}	-2.64×10^{3}	-1.80×10^{5}	-1.84×10^{5}	-1.82×10^{5}	-1.87×10^{3}
100-100-1-50	-1.35×10^{5}	-7.78×10^{4}	-1.03×10^{5}	-1.11×10^{5}	-7.71×10^{5}	-7.11×10^{4}
100-100-1-75	-2.55×10^{5}	-1.66×10^{5}	-1.95×10^{5}	-2.21×10^{5}	-1.30×10^{5}	-1.46×10^{5}
Friedman rank	2.95	2.05	4.49	5.92	4.59	1.00
Final rank	3	2	4	6	5	1

These observations collectively suggest that while the classic optimization algorithms—GA, ESA, IWOA, IGWO, and PGCH—show similarities in specific performance metrics when applied to TTP instances of varying scales, FECOIMO stands out as the more effective solution. The design of FECOIMO's updating operators, tailored to the characteristics of the TTP instances, enhances its applicability and effectiveness. Therefore, while the classical algorithms exhibit comparable performance in certain aspects, FECOIMO's superior adaptability to the problem characteristics makes it more effective across different scales of TTP instances. This tailored design of FECOIMO allows it to handle the complexities and nuances of the TTP more efficiently, thereby yielding better overall performance.

Table 7. Maximum of benefits and Friedman test ranks from 30 independent runs of the six algorithms (p-value = 7.17×10^{-37}).

Algorithm / **Instance**	**GA**	**ESA**	**IWOA**	**IGWO**	**PGCH**	**FECOIMO**
10-10-1-25	-1.66×10^4	-1.66×10^4	-1.66×10^4	-2.01×10^4	-1.92×10^4	-1.66×10^4
10-10-1-50	1.96×10^3	1.95×10^3	1.68×10^3	6.93×10^2	8.86×10^2	1.96×10^3
10-10-1-75	-1.88×10^3	-2.35×10^3	-2.48×10^3	-4.64×10^3	-4.15×10^3	-1.88×10^3
10-15-1-25	3.77×10^2	3.35×10^2	-2.18×10^2	-1.60×10^3	-1.38×10^3	3.89×10^2
10-15-1-50	1.21×10^3	1.30×10^3	1.96×10^2	-1.26×10^3	-5.87×10^1	1.30×10^3
10-15-1-75	-6.26×10^3	-6.61×10^3	-8.73×10^3	-1.44×10^4	-1.10×10^4	-6.26×10^3
20-10-1-25	7.23×10^2	7.23×10^2	-3.54×10^2	-1.24×10^3	-1.03×10^3	7.23×10^2
20-10-1-50	1.99×10^3	2.02×10^3	1.27×10^3	-1.14×10^2	-4.75×10^2	2.06×10^3
20-10-1-75	-1.60×10^3	-1.60×10^3	-4.16×10^3	-1.37×10^4	-1.06×10^4	-1.60×10^3
20-20-1-25	-1.78×10^3	-1.88×10^3	-4.07×10^3	-1.10×10^4	-9.98×10^3	-1.58×10^3
20-20-1-50	-2.03×10^3	-1.88×10^3	-1.06×10^4	-2.19×10^4	-1.59×10^4	-1.69×10^3
20-20-1-75	-4.60×10^4	-4.48×10^4	-7.69×10^4	-1.42×10^5	-8.85×10^4	-4.35×10^4
20-30-1-25	-1.66×10^3	-1.77×10^3	-7.64×10^3	-1.73×10^4	-1.56×10^4	-1.22×10^3
20-30-1-50	-7.71×10^2	-9.90×10^2	-7.05×10^3	-1.87×10^4	-1.75×10^4	-3.37×10^2
20-30-1-75	-1.98×10^4	-1.84×10^4	-4.03×10^4	-8.23×10^4	-5.43×10^4	-1.73×10^4
50-15-1-25	-2.33×10^3	-1.30×10^3	-1.20×10^4	-1.48×10^4	-1.40×10^4	-1.25×10^3
50-15-1-50	-3.62×10^3	-1.79×10^3	-2.26×10^4	-2.80×10^4	-2.51×10^4	-1.30×10^3
50-15-1-75	-3.05×10^4	-2.45×10^4	-1.13×10^5	-1.39×10^5	-1.03×10^5	-2.33×10^4
50-25-1-25	-1.46×10^4	-1.25×10^4	-6.09×10^4	-7.76×10^4	-7.13×10^4	-1.20×10^4
50-25-1-50	-1.77×10^5	-1.53×10^5	-6.09×10^5	-7.68×10^5	-5.56×10^5	-1.51×10^5
50-25-1-75	-3.20×10^4	-2.83×10^4	-1.59×10^5	-2.02×10^5	-1.38×10^5	-2.63×10^4
50-50-1-25	-2.45×10^4	-2.16×10^4	-1.18×10^5	-1.49×10^5	-1.51×10^5	-2.03×10^4
50-50-1-50	-1.35×10^5	-1.29×10^5	-5.63×10^5	-7.83×10^5	-5.39×10^5	-1.24×10^5
50-50-1-75	-2.88×10^5	-2.61×10^5	-1.24×10^5	-1.76×10^5	-1.10×10^5	-2.54×10^5
50-75-1-25	-6.53×10^4	-6.05×10^4	-3.09×10^5	-3.75×10^5	-3.61×10^5	-5.70×10^4
50-75-1-50	-1.85×10^4	-1.48×10^4	-1.56×10^5	-2.30×10^5	-1.79×10^5	-1.15×10^4
50-75-1-75	1.25×10^4	1.19×10^4	-7.47×10^4	-1.48×10^5	-9.18×10^4	1.59×10^4
100-10-1-25	-5.11×10^3	-1.62×10^3	-1.95×10^4	-2.20×10^4	-2.18×10^4	-1.44×10^3
100-10-1-50	-1.13×10^4	-2.25×10^3	-3.18×10^4	-3.44×10^4	-3.20×10^4	-1.94×10^3
100-10-1-75	-3.58×10^4	-1.03×10^4	-8.64×10^4	-9.14×10^4	-7.55×10^4	-9.83×10^3
100-25-1-25	-2.86×10^4	-1.83×10^4	-1.33×10^5	-1.49×10^5	-1.26×10^5	-1.73×10^4
100-25-1-50	-3.36×10^4	-1.35×10^4	-1.33×10^5	-1.42×10^5	-1.15×10^5	-1.26×10^4
100-25-1-75	-1.93×10^5	-8.53×10^4	-7.76×10^5	-7.72×10^5	-5.24×10^5	-8.19×10^4
100-50-1-25	-1.18×10^5	-9.40×10^4	-7.46×10^5	-7.59×10^5	-6.74×10^5	-8.83×10^4
100-50-1-50	-4.63×10^4	-3.00×10^4	-3.85×10^5	-4.13×10^5	-2.80×10^5	-2.75×10^4
100-50-1-75	-8.93×10^4	-5.03×10^4	-5.37×10^5	-6.58×10^5	-3.98×10^5	-4.61×10^4
100-100-1-25	-7.38×10^3	-1.71×10^3	-1.59×10^5	-1.61×10^5	-1.62×10^5	-1.03×10^3
100-100-1-50	-1.14×10^5	-7.05×10^4	-9.56×10^5	-1.05×10^5	-6.91×10^5	-6.89×10^4
100-100-1-75	-2.18×10^5	-1.56×10^5	-1.77×10^5	-2.00×10^5	-1.16×10^5	-1.42×10^5
Friedman rank	2.69	2.21	4.35	5.90	4.72	1.14
Final rank	3	2	4	6	5	1

Table 8. SD of benefits and Friedman test ranks from 30 independent runs of the six algorithms (p-value = 5.37×10^{-35}).

Algorithm / Instance	GA	ESA	IWOA	IGWO	PGCH	FECOIMO
10-10-1-25	9.34×10^2	4.91×10^2	1.04×10^3	2.22×10^3	2.60×10^3	1.11×10^{-11}
10-10-1-50	2.53×10^2	1.46×10^2	2.61×10^2	2.38×10^2	5.64×10^2	1.75×10^1
10-10-1-75	3.84×10^2	3.15×10^2	6.93×10^2	6.28×10^2	3.27×10^2	1.39×10^{-12}
10-15-1-25	2.22×10^2	1.10×10^2	4.23×10^2	3.65×10^2	2.52×10^2	6.03
10-15-1-50	3.17×10^2	3.08×10^2	3.53×10^2	6.58×10^2	6.55×10^2	1.01×10^2
10-15-1-75	6.77×10^2	6.24×10^2	1.30×10^3	1.93×10^3	8.89×10^2	4.61×10^1
20-10-1-25	1.43×10^2	7.60×10^1	2.54×10^2	1.85×10^2	3.83×10^2	2.43×10^1
20-10-1-50	2.29×10^2	9.00×10^1	3.49×10^2	3.47×10^2	7.68×10^2	4.32×10^1
20-10-1-75	4.45×10^2	4.28×10^2	1.80×10^3	8.06×10^2	6.21×10^2	5.33×10^1
20-20-1-25	4.79×10^2	2.95×10^2	1.71×10^3	7.29×10^2	1.26×10^3	9.07×10^1
20-20-1-50	1.03×10^3	5.02×10^2	2.68×10^3	1.44×10^3	9.89×10^2	1.87×10^2
20-20-1-75	2.40×10^3	2.15×10^3	1.44×10^4	8.58×10^3	5.41×10^3	3.57×10^2
20-30-1-25	7.96×10^2	5.46×10^2	2.29×10^3	1.57×10^3	1.25×10^3	1.30×10^2
20-30-1-50	1.02×10^3	8.11×10^2	2.90×10^3	3.26×10^3	2.55×10^3	1.92×10^2
20-30-1-75	1.84×10^3	1.42×10^3	8.31×10^3	6.02×10^3	3.21×10^3	3.13×10^2
50-15-1-25	5.68×10^2	1.38×10^2	1.08×10^3	5.03×10^2	5.44×10^2	9.69×10^1
50-15-1-50	1.23×10^3	2.35×10^2	2.30×10^3	1.64×10^3	9.41×10^2	1.54×10^2
50-15-1-75	6.68×10^3	1.15×10^3	1.02×10^4	7.92×10^3	5.31×10^3	4.30×10^2
50-25-1-25	1.26×10^3	7.87×10^2	5.59×10^3	3.92×10^3	6.76×10^3	1.85×10^2
50-25-1-50	2.39×10^4	5.24×10^3	5.22×10^4	3.64×10^4	3.07×10^4	1.69×10^3
50-25-1-75	6.22×10^3	1.74×10^3	1.88×10^4	1.04×10^4	6.29×10^3	6.47×10^2
50-50-1-25	2.33×10^3	1.05×10^3	1.02×10^4	8.28×10^3	1.02×10^4	3.73×10^2
50-50-1-50	1.51×10^4	4.85×10^3	6.06×10^4	3.64×10^4	2.73×10^4	1.63×10^3
50-50-1-75	2.99×10^4	1.05×10^4	1.77×10^5	8.36×10^4	6.14×10^4	2.31×10^3
50-75-1-25	5.82×10^3	2.85×10^3	2.51×10^4	1.97×10^4	2.14×10^4	1.35×10^3
50-75-1-50	2.42×10^3	1.30×10^3	2.52×10^4	1.20×10^4	1.56×10^4	6.87×10^2
50-75-1-75	1.81×10^3	1.22×10^3	1.47×10^4	8.24×10^3	3.38×10^3	3.97×10^2
100-10-1-25	7.17×10^2	1.14×10^2	1.09×10^3	4.85×10^2	4.76×10^2	5.97×10^1
100-10-1-50	6.91×10^2	1.44×10^2	1.61×10^3	9.86×10^2	7.41×10^2	1.20×10^2
100-10-1-75	1.67×10^3	3.99×10^2	3.46×10^3	3.51×10^3	1.92×10^3	1.95×10^2
100-25-1-25	5.51×10^3	6.56×10^2	7.62×10^3	5.93×10^3	8.93×10^3	4.48×10^2
100-25-1-50	2.93×10^3	5.60×10^2	6.12×10^3	6.57×10^3	3.64×10^3	3.44×10^2
100-25-1-75	1.52×10^4	2.79×10^3	2.78×10^4	3.39×10^4	1.55×10^4	1.10×10^3
100-50-1-25	1.48×10^4	1.98×10^3	2.88×10^4	3.30×10^4	5.63×10^4	1.75×10^3
100-50-1-50	5.81×10^3	1.39×10^3	1.93×10^4	1.93×10^4	1.39×10^4	9.23×10^2
100-50-1-75	1.25×10^4	2.34×10^3	3.66×10^4	2.51×10^4	1.32×10^4	1.20×10^3
100-100-1-25	2.44×10^3	5.29×10^2	8.54×10^3	7.21×10^3	1.05×10^4	3.46×10^2
100-100-1-50	1.32×10^4	4.10×10^3	4.63×10^4	3.74×10^4	3.64×10^4	1.22×10^3
100-100-1-75	2.20×10^4	5.95×10^3	1.15×10^5	8.17×10^4	4.04×10^4	2.08×10^3
Friedman rank	3.23	2.00	5.54	4.81	4.42	1.00
Final rank	3	2	6	5	4	1

Table 9. Adjusted *p*-values from multiple comparison tests following Friedman's test for the six algorithms on the mean of benefits.

	GA	ESA	IWOA	IGWO	PGCH
GA					
ESA	5.12×10^{-1}				
IWOA	4.23×10^{-3}	1.34×10^{-7}			
IGWO	3.31×10^{-11}	9.46×10^{-19}	1.05×10^{-2}		
PGCH	1.61×10^{-3}	3.11×10^{-8}	1.00	2.47×10^{-2}	
FECOIMO	6.35×10^{-5}	1.96×10^{-1}	2.78×10^{-15}	4.87×10^{-30}	3.58×10^{-16}

Table 10. Adjusted *p*-values from multiple comparison tests following Friedman's test for the six algorithms on the maximum of benefits.

	GA	ESA	IWOA	IGWO	PGCH
GA					
ESA	1.00				
IWOA	1.25×10^{-3}	5.23×10^{-6}			
IGWO	3.60×10^{-13}	2.32×10^{-17}	3.34×10^{-3}		
PGCH	2.15×10^{-5}	3.36×10^{-8}	1.00	7.51×10^{-2}	
FECOIMO	3.34×10^{-3}	1.70×10^{-1}	3.60×10^{-13}	1.59×10^{-28}	2.57×10^{-16}

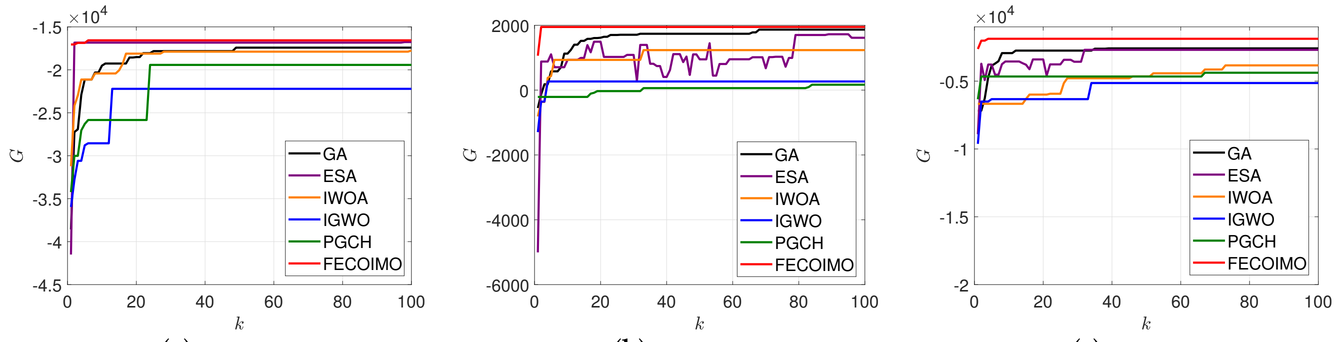
Table 11. Adjusted *p*-values from multiple comparison tests following Friedman's test for the six algorithms on the SD of benefits.

	GA	ESA	IWOA	IGWO	PGCH
GA					
ESA	5.47×10^{-2}				
IWOA	7.51×10^{-7}	9.55×10^{-16}			
IGWO	2.93×10^{-3}	4.96×10^{-10}	1.00		
PGCH	7.29×10^{-2}	1.56×10^{-7}	1.26×10^{-1}	1.00	
FECOIMO	2.05×10^{-6}	2.73×10^{-1}	1.22×10^{-25}	3.57×10^{-18}	9.27×10^{-15}

5.5.2. Convergence Comparison

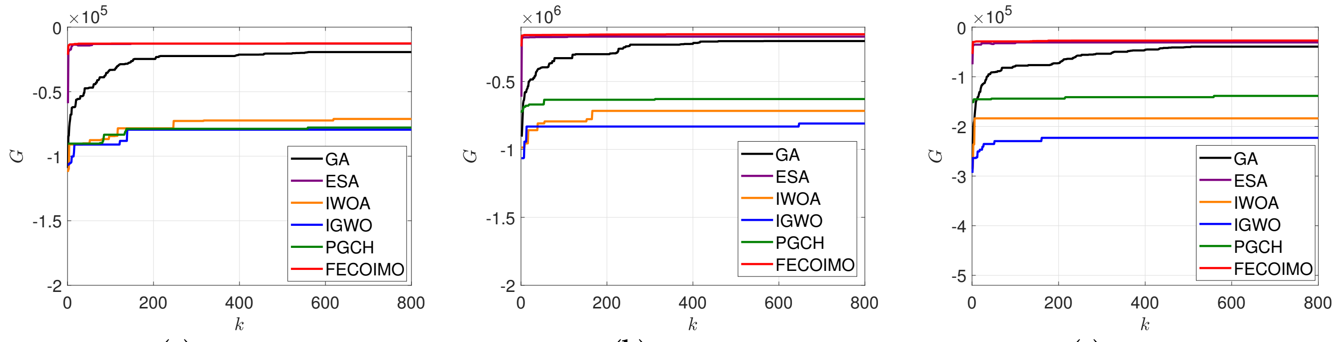
Figures 13–15 present the convergence curves of the six algorithms on nine representative instances from the 39 TTP instances, covering various scales. These figures illustrate the differences in convergence behavior among the algorithms, providing insight into their performance characteristics. It is important to note that the starting points of these curves differ because the initial populations for each algorithm are randomly generated. As a result, the algorithms were run independently, leading to different initial conditions for each algorithm.

For smaller-scale instances (e.g., $n = 10$, $m = 10$ in Figure 13), FECOIMO consistently achieves better convergence compared with the other algorithms. GA and ESA show slower convergence rates, while IWOA and IGWO perform relatively well in the initial stages but are eventually surpassed by FECOIMO. In medium and large-scale instances shown in Figures 14 and 15, FECOIMO continues to outperform the others, demonstrating faster convergence and higher-quality solutions.



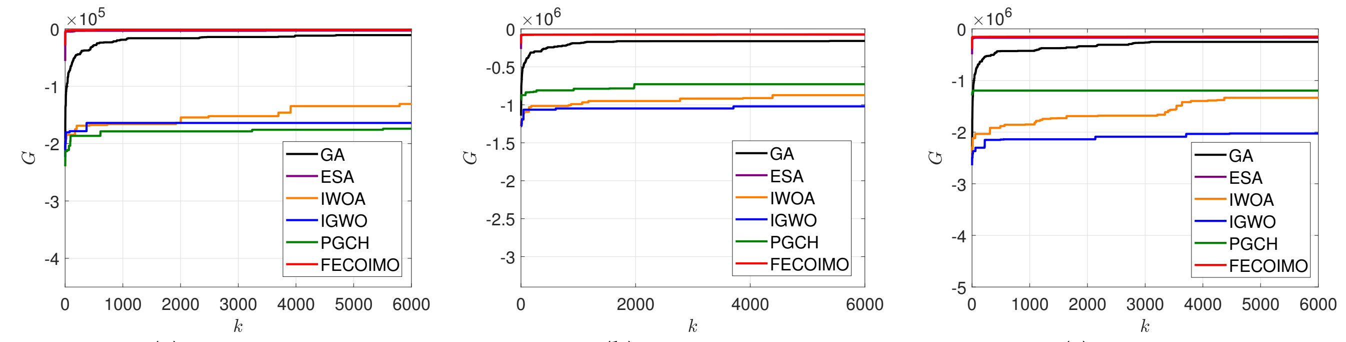
(**a**) 10-10-1-25 (**b**) 10-10-1-50 (**c**) 10-10-1-75

Figure 13. Convergence comparison of different algorithms on the instances with $n = 10,\ m = 10$.



(**a**) 50-25-1-25 (**b**) 50-25-1-50 (**c**) 50-25-1-75

Figure 14. Convergence comparison of different algorithms on the instances with $n = 50,\ m = 25$.



(**a**) 100-100-1-25 (**b**) 100-100-1-50 (**c**) 100-100-1-75

Figure 15. Convergence comparison of different algorithms on the instances with $n = 100,\ m = 100$.

5.5.3. Execution Time and Complexity Comparison

In the comparison of algorithm complexity and the time taken for a single run of each algorithm across different problem scales, it is observed that while the algorithms have similar theoretical complexities, their actual execution times show significant differences.

The FECOIMO algorithm proposed in this paper demonstrates unique performance characteristics in terms of both time complexity and execution time. Regarding time complexity, as shown in Table 12, FECOIMO is characterized by a complexity of $O(k_{max} * L * q * (n + m))$, where L and q represent the number of elements in each cycle and cycles, respectively. This allows FECOIMO to conduct a more comprehensive search and optimization of the solution space through its complex five-element cycle structure. Although FECOIMO's time complexity is similar to other algorithms like GA, IWOA, and PGCH, its structure is more intricate, especially when handling large-scale problems, resulting in a somewhat higher computational complexity.

In terms of execution time shown in Table 13, FECOIMO exhibits significantly longer run times across various instances compared with other algorithms. This can be attributed to the additional computational steps and deeper exploration of the solution space inherent in FECOIMO. While these extra computations increase execution time, they also enhance the algorithm's capability to solve large-scale complex problems. In other words, the additional computational overhead in FECOIMO translates into better solutions, which is particularly crucial when dealing with complex combinatorial optimization problems.

Overall, despite the increased execution time, FECOIMO's superior performance on complex problems justifies this additional time investment. Through comparison, it is evident that FECOIMO's design achieves a new level of balance between computational complexity and solution accuracy, providing an effective approach to solving complex optimization problems.

Table 12. The comparison of time complexity of each algorithm.

Algorithm	Time Complexity
GA	$O(k_{max} * N * (n + m))$
ESA	$O(k_{max} * (n + m))$
IWOA	$O(k_{max} * N * (n + m))$
IGWO	$O(k_{max} * N * (n * m + log(N))$
PGCH	$O(k_{max} * N * (n + m))$
FECOIMO	$O(k_{max} * L * q * (n + m))$

Table 13. The execution time for a single run of each algorithm (s).

Instance \ Algorithm	GA	ESA	IWOA	IGWO	PGCH	FECOIMO
10-10-1-25	2.02	2.24	1.62	1.55	1.87	3.49
10-10-1-50	2.04	3.30	1.51	1.56	1.94	3.87
10-10-1-75	2.03	2.15	1.49	1.54	1.77	3.92
50-25-1-25	2.29×10^1	3.41×10^1	1.66×10^1	1.67×10^1	1.88×10^1	4.20×10^1
50-25-1-50	2.22×10^1	1.65×10^1	1.64×10^1	1.71×10^1	1.85×10^1	6.10×10^1
50-25-1-75	2.33×10^1	1.64×10^1	1.65×10^1	1.69×10^1	1.83×10^1	7.12×10^1
100-100-1-25	2.67×10^2	1.17×10^2	2.27×10^2	2.25×10^2	3.23×10^2	7.56×10^2
100-100-1-50	2.59×10^2	1.07×10^2	2.37×10^2	2.37×10^2	3.32×10^2	1.43×10^3
100-100-1-75	2.75×10^2	1.22×10^2	2.47×10^2	2.50×10^2	3.27×10^2	1.72×10^3

6. Conclusions and Future Work

In this paper, we introduced the Five-element Cycle Integrated Mutation Optimization algorithm (FECOIMO), designed specifically to address the complexities of the Traveling Thief Problem (TTP). By integrating the Five-element Cycle Model with a tailored set of mutation and heuristic operators, FECOIMO effectively manages the dual challenges of optimizing both the tour and picking plans across 39 TTP instances of varying scales and complexities. The algorithm's iterative approach, enhanced by the integration of specialized mutation operators, significantly improves its ability to explore the search space and avoid premature convergence, thereby yielding superior solutions.

The efficacy of FECOIMO was rigorously validated through extensive comparative experiments against five other state-of-the-art metaheuristic algorithms. The results clearly demonstrate FECOIMO's superior performance, particularly in larger and more complex TTP instances, where it consistently outperformed the alternatives in terms of solution quality and robustness. These findings underscore FECOIMO's capability to address the diverse challenges posed by TTP instances comprehensively.

However, this study also has its limitations. Notably, the role of each mutation operator varies depending on the scale of the problem instances, suggesting that further refinement could involve adapting these operators more precisely to different problem sizes. Future research should explore these differential effects and consider the development of alternative mutation operators that can further enhance the algorithm's performance. Additionally, integrating these strategies with other advanced optimization techniques could lead to the creation of even more robust and versatile algorithms.

In conclusion, the FECOIMO algorithm represents a significant advancement in solving the TTP by effectively combining mutation operators and heuristic strategies to address this complex combinatorial optimization problem. Future work will focus on refining these strategies, extending their application to other complex optimization challenges, and continuing to build on the algorithm's practical and theoretical contributions to the field.

Author Contributions: Conceptualization, Y.X.; methodology, Y.X. and M.L.; software, Y.X.; formal analysis, Y.X. and J.G.; investigation, Y.X.; data curation, Y.X. and Z.M.; writing—original draft preparation, Y.X.; writing—review and editing, J.G., Z.M. and C.J.; supervision, M.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Fundamental Research Funds for the Central Universities, Grant No. 222201917006, under the affiliation of the Key Laboratory of Smart Manufacturing in Energy Chemical Process (East China University of Science and Technology), Ministry of Education.

Data Availability Statement: The data presented in this study are available at https://cs.adelaide.edu.au/~optlog/research/combinatorial.php (accessed on 24 July 2024), reference [1].

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Bonyadi, M.R.; Michalewicz, Z.; Barone, L. The travelling thief problem: The first step in the transition from theoretical problems to realistic problems. In Proceedings of the 2013 IEEE Congress on Evolutionary Computation, Cancun, Mexico, 20–23 June 2013; pp. 1037–1044.
2. Garey, M.R.; Johnson, D.S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*; W. H. Freeman and Company: San Francisco, CA, USA, 1979.
3. Holland, J.H. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*; MIT Press: Cambridge, MA, USA, 1992.
4. Dorigo, M.; Maniezzo, V.; Colorni, A. Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Trans. Syst. Man, Cybern. Part B Cybern.* **1996**, *26*, 29–41. [CrossRef] [PubMed]
5. Kirkpatrick, S.; Gelatt, C.D.; Vecchi, M.P. Optimization by Simulated Annealing. *Science* **1983**, *220*, 671–680. [CrossRef] [PubMed]
6. Glover, F. Tabu Search—Part I. *ORSA J. Comput.* **1989**, *1*, 190–206. [CrossRef]
7. Talbi, E.G. Combining Metaheuristics with Mathematical Programming, Constraint Programming and Machine Learning. In Proceedings of the MIC'2001—4th Metaheuristics International Conference, Porto, Portugal, 16–20 July 2002; pp. 189–194.
8. Dantzig, G.B. Discrete-Variable Extremum Problems. *Oper. Res.* **1957**, *5*, 266–288. [CrossRef]
9. Kellerer, H.; Pferschy, U.; Pisinger, D. *Knapsack Problems*; Springer: Berlin/Heidelberg, Germany, 2004.
10. Polyakovskiy, S.; Bonyadi, M.R.; Wagner, M.; Michalewicz, Z.; Neumann, F. A comprehensive benchmark set and heuristics for the traveling thief problem. In Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, Vancouver, BC, Canada, 12–16 July 2014; pp. 477–484.
11. Golden, B.L.; Raghavan, S.; Wasil, E.A., Eds. *The Capacitated Vehicle Routing Problem*; Operations Research/Computer Science Interfaces Series; Springer: Berlin/Heidelberg, Germany, 2008; Volume 43.
12. Moeini, M.; Schermer, D.; Wendt, O. A hybrid evolutionary approach for solving the traveling thief problem. In Proceedings of the International Conference on Computational Science and Its Applications, Trieste, Italy, 3–6 July 2017; Springer: Berlin/Heidelberg, Germany, 2017; pp. 652–668.
13. Vieira, D.K.; Soares, G.L.; Vasconcelos, J.A.; Mendes, M.H. A genetic algorithm for multi-component optimization problems: The case of the travelling thief problem. In Proceedings of the European Conference on Evolutionary Computation in Combinatorial Optimization, Amsterdam, The Netherlands, 19–21 April 2017; Springer: Berlin/Heidelberg, Germany, 2017; pp. 18–29.
14. Wagner, M. Stealing items more efficiently with ants: A swarm intelligence approach to the travelling thief problem. In Proceedings of the International Conference on Swarm Intelligence, Brussels, Belgium, 7–9 September 2016; Springer: Berlin/Heidelberg, Germany, 2016; pp. 273–281.
15. Zouari, W.; Alaya, I.; Tagina, M. A new hybrid ant colony algorithms for the traveling thief problem. In Proceedings of the Genetic and Evolutionary Computation Conference Companion, Prague, Czech Republic, 13–17 July 2019; pp. 95–96.

16. Alharbi, S.T. The design and development of a modified artificial bee colony approach for the traveling thief problem. *Int. J. Appl. Evol. Comput. (IJAEC)* **2018**, *9*, 32–47. [CrossRef]
17. Ali, I.M.; Essam, D.; Kasmarik, K. Differential Evolution Algorithm for Multiple Inter-dependent Components Traveling Thief Problem. In Proceedings of the 2020 IEEE Congress on Evolutionary Computation (CEC), Glasgow, UK, 19–24 July 2020; pp. 1–8.
18. Ali, H.; Rafique, M.Z.; Sarfraz, M.S.; Malik, M.S.A.; Alqahtani, M.A.; Alqurni, J.S. A novel approach for solving travelling thief problem using enhanced simulated annealing. *PeerJ Comput. Sci.* **2021**, *7*, e377. [CrossRef] [PubMed]
19. Zhang, Z.; Yang, L.; Kang, P.; Jia, X.; Zhang, W. Solving the Traveling Thief Problem Based on Item Selection Weight and Reverse-Order Allocation. *IEEE Access* **2021**, *9*, 54056–54066. [CrossRef]
20. Mei, Y.; Li, X.; Yao, X. On investigation of interdependence between sub-problems of the travelling thief problem. *Soft Comput.* **2016**, *20*, 157–172. [CrossRef]
21. Bonyadi, M.R.; Michalewicz, Z.; Przybylek, M.R.; Wierzbicki, A. Socially inspired algorithms for the travelling thief problem. In Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, Vancouver, BC, Canada, 12–16 July 2014; pp. 421–428.
22. El Yafrani, M.; Ahiod, B. Efficiently solving the Traveling Thief Problem using hill climbing and simulated annealing. *Inf. Sci.* **2018**, *432*, 231–244. [CrossRef]
23. Mei, Y.; Li, X.; Yao, X. Improving efficiency of heuristics for the large scale traveling thief problem. In Proceedings of the Asia-Pacific Conference on Simulated Evolution and Learning, Dunedin, New Zealand, 15–18 December 2014; Springer: Berlin/Heidelberg, Germany, 2014; pp. 631–643.
24. Mei, Y.; Li, X.; Salim, F.; Yao, X. Heuristic evolution with genetic programming for traveling thief problem. In Proceedings of the 2015 IEEE Congress on Evolutionary Computation (CEC), Sendai, Japan, 25–28 May 2015; pp. 2753–2760.
25. Martins, M.S.; El Yafrani, M.; Delgado, M.R.; Wagner, M.; Ahiod, B.; Lüders, R. HSEDA: A heuristic selection approach based on estimation of distribution algorithm for the travelling thief problem. In Proceedings of the Genetic and Evolutionary Computation Conference, Berlin, Germany, 15–17 July 2017; pp. 361–368.
26. El Yafrani, M.; Martins, M.; Wagner, M.; Ahiod, B.; Delgado, M.; Lüders, R. A hyperheuristic approach based on low-level heuristics for the travelling thief problem. *Genet. Program. Evolvable Mach.* **2018**, *19*, 121–150. [CrossRef]
27. El Yafrani, M.; Ahiod, B. A local search based approach for solving the Travelling Thief Problem: The pros and cons. *Appl. Soft Comput.* **2017**, *52*, 795–804. [CrossRef]
28. Maity, A.; Das, S. Efficient hybrid local search heuristics for solving the travelling thief problem. *Appl. Soft Comput.* **2020**, *93*, 106284. [CrossRef]
29. El Yafrani, M.; Ahiod, B. Cosolver2B: An efficient local search heuristic for the travelling thief problem. In Proceedings of the 2015 IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA), Marrakech, Morocco, 17–20 November 2015; pp. 1–5.
30. Yafrani, M.E.; Martins, M.S.; Krari, M.E.; Wagner, M.; Delgado, M.R.; Ahiod, B.; Lüders, R. A fitness landscape analysis of the travelling thief problem. In Proceedings of the Genetic and Evolutionary Computation Conference, Kyoto, Japan, 15–19 July 2018; pp. 277–284.
31. Eiben, A.E.; Smith, J.E. *Introduction to Evolutionary Computing*; Natural Computing Series; Springer: Berlin/Heidelberg, Germany, 2003.
32. Deb, K.; Pratap, A.; Agarwal, S.; Meyarivan, T. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* **2002**, *6*, 182–197. [CrossRef]
33. Liu, M. Five-elements cycle optimization algorithm for the travelling salesman problem. In Proceedings of the 2017 18th International Conference on Advanced Robotics (ICAR), Hong Kong, China, 10–12 July 2017; pp. 595–601.
34. Mao, Z.; Liu, M. A local search-based many-objective five-element cycle optimization algorithm. *Swarm Evol. Comput.* **2022**, *68*, 101009. [CrossRef]
35. Jing, R.; Yue, X.; Mandan, L. Multi-Objective Cold Chain Distribution Based on Dual-Mode Updated Five-Element Cycle Algorithm. *J. East China Univ. Sci. Technol.* **2023**, *49*, 236–246.
36. Talbi, E.G. *Metaheuristics: From Design to Implementation*; John Wiley & Sons: Hoboken, NJ, USA, 2009.
37. Derrac, J.; García, S.; Molina, D.; Herrera, F. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm Evol. Comput.* **2011**, *1*, 3–18. [CrossRef]
38. Liu, J.; Wei, X.; Huang, H. An improved grey wolf optimization algorithm and its application in path planning. *IEEE Access* **2021**, *9*, 121944–121956. [CrossRef]
39. Mostafa Bozorgi, S.; Yazdani, S. IWOA: An improved whale optimization algorithm for optimization problems. *J. Comput. Des. Eng.* **2019**, *6*, 243–259. [CrossRef]
40. Mathew, T.V. Genetic algorithm. *Rep. Submitt. Iit Bombay* **2012**, *53*, 1–15.
41. Namazi, M.; Newton, M.; Sattar, A.; Sanderson, C. A profit guided coordination heuristic for travelling thief problems. In Proceedings of the International Symposium on Combinatorial Search, Napa, CA, USA, 16–17 July 2019; Volume 10, pp. 140–144.