China Instrument and Control Society
Instrumentation
INSTRUMENTATION, Vol. 12, No. 4, December 2025 https://doi.org/10.15878/j.instr.202500300

Article

A Memetic Algorithm for Solving UAV Routing Problems with Profits

Siliang Hua[1*], Jian Xu[1], Huiguo Zhang[1], Qian Zhang[2], Lifeng Qin[3], Lixing Hua[4]

[1] School of Electronic and Information Engineering, Suzhou University of Technology, Suzhou, Jiangsu, China

[2] School of Engineering, Liverpool John Moores University, Liverpool, United Kingdom

[3] Suzhou Pengtec Technology Co., Ltd., Suzhou, Jiangsu, China

[4] Suzhou High School of Jiangsu Province, Suzhou, Jiangsu, China

[*] Corresponding author email: huasiliang@szut.edu.cn

Abstract: This study addresses the Unmanned Aerial Vehicle routing problems with profits, which requires balancing mission profit, path efficiency, and battery health under complex constraints, particularly the nonlinear degradation of batteries. This paper proposes an enhanced memetic algorithm by integrating adaptive local search and a dynamic population management mechanism. The algorithm employs a hybrid initialization strategy to generate high-quality initial solutions. It incorporates an improved linear crossover operator to preserve beneficial path characteristics and introduces dynamically probability-controlled local search to optimize solution quality. To enhance global exploration capability, a population screening mechanism based on solution similarity and a population restart strategy simulating biological mass extinction are designed. Extensive experiments conducted on standard Tsiligirides's and Chao's datasets demonstrate the algorithm's robust performance across scenarios ranging from 21 to 66 nodes and time constraints spanning 5 to 130 minutes. The algorithm attains 95% accuracy relative to maximum total score within 30 iterations, surpassing 99% accuracy after 100 iterations. Its comprehensive performance significantly surpasses that of traditional heuristic methods. The proposed method provides an efficient and robust solution for Unmanned Aerial Vehicle routing planning under intricate constraints.

Keywords: memetic algorithm; unmanned aerial vehicle routing; orienteering problem

Citation: Siliang Hua, Jian Xu, Huiguo Zhang, Qian Zhang, Lifeng Qin, Lixing Hua. "A Memetic Algorithm for Solving UAV Routing Problems with Profits." Instrumentation 12, no.4 (December 2025). https://doi.org/10.15878/j.instr.202500300

1 Introduction

Unmanned Aerial Vehicle (UAV) route planning is a fundamental challenge in autonomous navigation and mission execution, aiming to determine optimal flight paths from start to end points under specific constraints. With the increasing application of UAVs in areas such as disaster rescue, environmental monitoring, and power line inspection, the demand for optimized route planning has become more critical than ever. However, real-world scenarios often involve multi-objective optimization challenges, considering route length, mission profit, battery endurance, and other complex constraints. Traditional single-objective optimization methods—such as shortest-path or shortest-time planning—have proven inadequate, necessitating more efficient algorithms to address multi-objective synergistic optimization under complex constraints.

This study focuses on UAV route planning problem with profits, distinct from obstacle-avoidance path planning. In disaster rescue missions, for instance, UAVs must strategically sequence visits to life-detection points within battery constraints while accounting for site priorities and maximizing coverage efficiency. Similarly, power line inspection requires prioritizing components (e. g., insulators, conductors) based on condition-criticality while ensuring optimal coverage. These applications highlight the need to balance flight distance, time, mission profits, and battery consumption. Moreover, the

nonlinear discharge characteristics of batteries impose stringent constraints on route planning algorithms to simultaneously safeguard battery health and maximize profits, ensuring safe mission completion.

Such challenges align closely with the classic Orienteering Problem (OP) [1]. As a combinatorial optimization problem inspired by orienteering sports—where participants aim to maximize scores by visiting checkpoints within time limits—OP integrates node selection and route planning. Unlike the Traveling Salesman Problem (TSP), OP does not require visiting all nodes, offering greater flexibility in resource-constrained applications.

Research on OP dates back to the 1980s, when Tsiligirides (1984) first formalized the problem[2]. Early efforts centered on exact algorithms. Fischetti et al. (1998) later proposed a branch-and-cut algorithm that significantly enhanced solving efficiency using valid inequality constraints[3]. However, the NP-hardness of OP limits the scalability of exact algorithms for large-scale problems. In recent years, metaheuristic and hybrid algorithms have emerged as leading approaches due to their ability to generate high-quality approximate solutions efficiently[4,5]. Ant Colony Optimization (ACO), simulating ant foraging behavior, has been successfully applied to OP. Xiao et al. (2022) improved ACO by dynamically updating pheromones and introducing directional functions, achieving smoother paths and higher efficiency[6]. Genetic Algorithms (GAs) and Simulated Annealing (SA) are also widely adopted. Gendreau et al. (1998) developed a GA-based heuristic with crossover and mutation operations to enhance global search and solution diversity[7]. Gao et al. (2024) proposed a multi-UAV trajectory optimization method using k-means++ task allocation and improved SA, elevating power line inspection efficiency under energy constraints[8]. Wang (2024) introduced an improved SA (I-SA) for terminal vehicle-mounted UAV logistics path planning, by modifying local disturbance mechanisms and local search strategies to optimize energy consumption and load endurance coordination[9].

Memetic Algorithms (MAs), hybrid metaheuristics combining global search (e. g., GA) and local optimization, have demonstrated exceptional performance in solving complex combinatorial problems. Montemanni and Gambardella (2009) developed an MA based on ACO for time-constrained OP variants[10]. Schilde et al. (2009) extended this to bi-objective optimization via non-dominated solution sets[11]. Kobeaga et al. (2018) further improved MA's global search capability through novel crossover and mutation operations[12].

Despite these advancements, significant gaps remain in UAV route planning, particularly in addressing the nonlinear discharge characteristics of batteries and achieving efficient and stable multi-objective optimization. To address these challenges, this paper proposes an improved Memetic Algorithm incorporating adaptive local search and dynamic population management. This approach ensures robust global search while enhancing the co-optimization of battery health and mission profits. The paper is organized as follows: Section 2 formalizes the problem, Section 3 describes the algorithm, Section 4 presents the experiments, Section 5 outlines future work, and Section 6 provides the conclusion.

2 Formulation of the Problem

The general Orienteering Problem (OP) is defined as follows[1,4,5]. Consider a node set $N=\{1,\ldots,|N|\}$, where each node $i\in N$ is associated with a non-negative score (or profit) S_i. The start and end nodes are fixed as node 1 and node $|N|$, respectively. Typically, $S_1=S_{|N|}=0$. The objective is to determine a route that visits a subset of nodes N, subject to a predefined travel time (or cost) budget T_{max}, while maximizing the total collected score. Scores are additive, and each node is visited at most once. Travel cost t_{ij} between nodes i and j is non-negative.

The OP is formalized as an integer programming model with the following decision variables: $x_{ij}=1$ if node j is visited immediately after node i, else 0. The variables u_i are used to avoid subtours and they are an indication of the relative position of node i in the route.

$$\text{Maximize} \sum_{i=2}^{|N|-1}\sum_{j=2}^{|N|} S_i x_{ij} \tag{1}$$

Objective function (1) is to maximize the total collected score.

$$\sum_{j=2}^{|N|} x_{1j} = \sum_{i=1}^{|N|-1} x_{i|N|} = 1 \tag{2}$$

Constrain (2) ensures the route starts at node 1 and ends at node $|N|$.

$$\sum_{i=1}^{|N|-1} x_{1k} = \sum_{j=2}^{|N|} x_{kj} \le 1;\ \forall k=2,\ldots,|N| \tag{3}$$

Constrain (3) maintains connectivity of the routes while guaranteeing that each node is visited at most once.

$$T_{total} = \sum_{i=1}^{|N|-1}\sum_{j=2}^{|N|} t_{ij}x_{ij} \le T_{max} \tag{4}$$

Constrain (4) ensures total travel time must not exceed T_{max}.

$$2 \le u_i \le |N|;\ \forall i=2,\ldots,|N| \tag{5}$$

$$u_i - u_j + 1 \le (|N|-1)(1-x_{ij});\ \forall i,j=2,\ldots,|N| \tag{6}$$

The combination of constraints (5) and (6) prevents subtours for each route.

To account for nonlinear battery degradation in UAVs, we introduce

- Safe Time T_{safe}: battery health remains uncompromised if $T_{total}\le T_{safe}$.
- Penalty Function: if $T_{safe}\le T_{total}\le T_{max}$, the total score is penalized by $P(T_{total}-T_{safe})\cdot H(T_{total}-T_{safe})$, where P is defined by the UAV manufacturer, usually a

non-decreasing funciton for positive arguments, for example, $P(t)=\lceil 3t \rceil$, and H is the Heaviside step function, which is $H(x)=\begin{cases} 0, & x<0 \\ 1/2, & x=0. \\ 0, & x>0 \end{cases}$

The adjusted total score becomes

$$S_{total}=\sum_{i=2}^{|N|-1}\sum_{j=2}^{|N|}S_i x_{ij}-P\left(T_{total}-T_{safe}\right)\cdot H\left(T_{total}-T_{safe}\right) \tag{7}$$

The objective is maximize S_{total} within T_{max}.

3 Algorithm Design

Genetic Algorithms (GAs) are a class of evolutionary algorithms that simulate biological evolutionary mechanisms, including selection, crossover, and mutation, to perform a global search within the solution space. Within the GA framework, potential solutions to a problem are encoded as data structures called "chromosomes". When the solution is extracted from the chromosome through a decoding process, this is referred to as indirect encoding. During algorithm execution, individuals in the population are evaluated using Equation (7), with selection operations performed based on their evaluation results. Subsequently, crossover operations enable information exchange between chromosomes, while mutation operations are introduced to preserve population diversity. Furthermore, the algorithm employs an admission condition-based population update mechanism to effectively mitigate the risk of premature convergence.

Memetic Algorithms (MAs) offer a hybrid optimization framework that integrates the global search capabilities of GA with local search techniques. As demonstrated by Prins[13] in studies on the Vehicle Routing Problem (VRP), this hybrid strategy significantly enhances optimization performance. The proposed MA enhances traditional memetic algorithms via two synergistic mechanisms: First, it unifies GA-driven global search with adaptive local optimization within a dynamic population framework. Second, it incorporates a mass-extinction-inspired population reset strategy. This dual approach achieves superior global search capability without compromising local refinement.

3.1 Basic Algorithm

The overall framework of the MA proposed in this paper is presented in Algorithm 1. This algorithm begins by using an initial feasible solution pool optimized via the Prioritized Best Insertion Algorithm (PBIA) (Section 3.3) as the initial population. Optimization occurs through iterative cycles combining crossover operations and locally enhanced searches.

Specifically, in each iteration, the algorithm first selects two parent chromosomes through a selection operator and generates child chromosomes via crossover (Section 3.4). Subsequently, it executes local search optimization on parent and child chromosomes with a dynamic probability (Section 3.5). To preserve population diversity, a chromosome-similarity-based screening mechanism is implemented: chromosomes are discarded under insufficient diversity conditions to prevent premature convergence. Optimized feasible chromosomes are then put into the population through evaluation (Section 3.2), simultaneously updating the current optimal chromosome.

Additionally, a population destruction and replenishment strategy is applied when no improvement is observed over consecutive iterations.. This strategy partially disrupts the current population and replenishes new chromosomes to enhance diversity (Section 3.6). The optimization process repeats for a preset number of iterations. Details regarding each algorithmic component are elaborated in subsequent sections.

Algorithm 1 Basic Algorithm of the Proposed MA

Input:Target function and weighted directed graph $G=(V, E)$

Output:Top-n optimal routes encoded as chromosomes

Initialize and evaluate each chromosome in population P (see Section 3.3)

$iter \leftarrow 1$

while $iter \le itermax$ **do**

Generate an empty population $P_{new} \leftarrow emptyset$

while chromosomes in P **do**

Perform chromosomes selection (see Section 3.4)

Apply LOX crossover (see Section 3.4)

if $rand(0,1) < p_m$ **then**

Apply local search (see Section 3.5)

end if

if (similarity of chromosome to $P_{new} > p_t$) **then**

Discard

else

Evaluate chromosome (see Section 3.2) and put it into P_{new}

end if

if (new best path found) **then**

$iter \leftarrow 1$

else

$iter \leftarrow iter+1$

end if

end while

$P \leftarrow P_{new}$

if $iter=\left\lceil \frac{sizeof(P)}{2} \right\rceil$ **then**

Population destruction and replenishment to P (see Section 3.6)

end if

end while

return Top-n chromosomes in P

3.2 Chromosome Encoding and Evaluation

This paper adopts a sequence-based chromosome encoding method to represent solutions for the UAV routing problem. Each chromosome is structured as an ordered sequence starting from the start node (encoded as 0) and encompassing all target points. Specifically, all points preceding end node (encoded as 1) form the actual visitation sequence executed by the UAV, while points following the end node represent the set of currently unvisited target nodes. The ordered arrangement of these unvisited nodes implicitly defines their priority for potential reinsertion into the visitation sequence during subsequent optimization processes.

As shown in Figure 1, the chromosome "0-2-3-4-5-6-1-7-8-9" indicates that the UAV departs from start node 0, sequentially visits points 2, 3, 4, 5, and 6, then reaches end node 1. Node 7, 8, and 9 remain unvisited, with their sequential order 7-8-9 indicating potential insertion priorities. This encoding method integrates both the visitation sequence and the prioritized information regarding unvisited nodes into a unified data structure. Consequently, it not only comprehensively characterizes the structural features of the solution space but also provides heuristic guidance for neighborhood search through the ordering of unvisited nodes. This approach thus achieves dual advantages in solution expression completeness and neighborhood search guidance.

0	2	3	4	5	6	1	7	8	9

Fig.1 Chromosome Encoding Method

Chromosome evaluation is based on two key metrics: the total time T_{total} and the total collection score S_{total}. The total time T_{total} is computed using Equation (4). The total collection score S_{total} is computed using Equation (7). During population maintenance, chromosomes are sorted by the (S_{total}, T_{total}) tuple as the primary key using Python's SortedContainers library, achieving $O(n \log n)$ time complexity. This leverages the SortedList data structure to enable efficient insertion and retrieval operations, ensuring the population remains permanently ordered. This structured ordering facilitates similarity computations and supports population destruction and replenishment strategy.

3.3 Population Initialization

To enhance the convergence speed of the algorithm, this paper employs a hybrid initialization strategy to construct the initial population. A small subset of chromosomes (approximately 10% of the total population size) is generated using the Prioritized Best Insertion Algorithm (PBIA), while the remaining chromosomes are randomly generated. This hybrid approach ensures both the presence of high-quality solutions within the population and sufficient genetic diversity.

The PBIA algorithm is built upon the Best Insertion Algorithm (BIA) [14], and its core workflow proceeds as follows: Given a partial solution (which may be empty) and a subset of unvisited nodes for insertion, the algorithm first evaluates the insertion cost for each candidate node z between any adjacent points i and j in the current route. The cost is calculated as c $cost_z = \frac{T_{i,z} + T_{z,j} - T_{i,j}}{S_z}$, where $T_{i,z}$, $T_{z,j}$, and $T_{i,j}$ denote travel times between corresponding nodes, and S_z represents the score of node z. The algorithm selects the feasible insertion operation with the lowest $cost_z$, choosing randomly among multiple equally optimal options if they exist. Nodes exhibiting negative $cost_z$ values are assigned lower insertion priority. The process repeats until either the total chromosome score no longer improves or all nodes are inserted.

Chromosomes generated by PBIA are feasible solutions with positive total scores, steering the population toward high-quality solution regions. Conversely, random chromosomes maintain population diversity to prevent premature convergence to local optima. When the partial solution is empty, containing only start and finish nodes, the algorithm evaluates all possible insertion positions for all unvisited nodes, resulting in a computational complexity of $O\left(\frac{n^2}{2}\right)$.

3.4 Chromosome Selection and Crossover

The chromosome selection strategy dynamically adjusts based on parent population size. When the parent population falls below 75% of the initialization size, a direct advancement method is adopted, where all parent chromosomes proceed directly to crossover operations, aiming to double population and maintain minimum population viability. When exceeding this threshold, binary tournament method is implemented: two chromosomes are randomly selected from the population and the higher total collection scores S_{total} one becomes the first parent $P1$. As a result, the population self-regulates within 75%-150% of its initial size: aggressive expansion via direct advancement at lower bounds, and controlled selection via tournaments at upper bounds. The procedure is repeated to get the second parent $P2$. For scenarios with fewer than four remaining parent chromosomes, a complete preservation method is activated to ensure all potentially high-quality chromosomes advance to subsequent optimization.

Crossover operations, which represent recombination generating new individuals, utilize a modified Linear Order Crossover (LOX) operator. While traditional LOX requires explicit start/end node encoding[13], our chromosome scheme appends unvisited nodes after the end node. To concurrently preserve high-quality parental route segments and pending nodes' priority information, we refine the standard LOX as follows: For parent chromosomes designated $P1$ and $P2$, and child chromosomes $C1$ and $C2$, Figure 2

demonstrates $C1$ construction from $P1$ as the primary chromosome. The child $C1$ first inherits $P1$'s start and end node encoding. Subsequently, a random starting position and another random segment length are determined between $P1$'s start and end nodes; this selected route segment is copied intact into the corresponding positions of $C1$. From the termination node of this copied segment, $C1$'s missing nodes are populated by cyclically scanning and incorporating nodes according to $P2$'s node sequence. This enhanced LOX preserves high-quality route structures' genetic stability, inherits heuristic information from unvisited nodes through parental node ordering, and maintains computational complexity at $O(n)$.

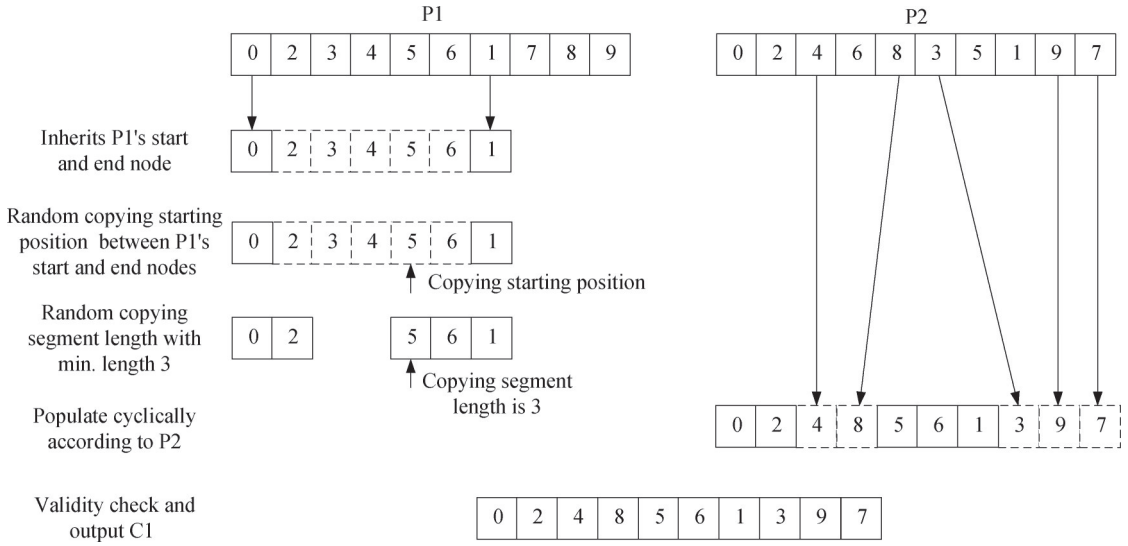

Fig.2 Sample of Chromosomes Crossover

3.5 Local Search as Mutation Operator

To fully realize the potential of the memetic algorithm, this paper employs Local Search (LS) as the mutation operator, with its execution probability denoted as p_m. The p_m value is dynamically adjusted through a two-phase strategy based on iteration progress. In the early stage, p_m decreases linearly from a preset maximum to the minimum value. In the later stage, p_m increases linearly from the minimum to the arithmetic mean of the maximum and minimum values. This dynamic adjustment balances global exploration in the initial iterations with enhanced local exploitation during convergence.

The LS process is built on three fundamental neighborhood search operations:

1.Shift

A node is selected from within the route and is relocated to a new position in the chromosome. All potential relocation positions are evaluated, and the move that maximizes S_{total} is executed.

2.Swap

Two nodes are selected, and their positions in the chromosome are exchanged. All potential exchange pairs are evaluated, and the swap that maximizes S_{total} is applied.

Note: Both the shift and swap operations require pre-evaluation of all neighborhood solutions before execution, theoretically incurring $O(n^2)$ complexity.

In our implementation, practical considerations such as the fixed total collected scores observed when shifting nodes within a route to any position outside the route, or immobile total collected scores when swapping any two nodes outside the route, enable significant reduction in computational complexity in some cases.

3.Destroy/Repair

Destroy Phase: Several nodes are randomly removed from the route and relocated to positions outside it[15].

Repair Phase: Reinsert nodes into the route using PBIA.

Prior to every LS process, a random permutation of three operations (shift, swap, destroy/repair) is generated. LS continues until three successive iterations yield no improvement.

To maintain population diversity and prevent premature convergence, the algorithm applies a chromosome-similarity-based screening mechanism after LS operations. Chromosomes with a similarity level exceeding a preset threshold compared to existing population members are discarded. The similarity threshold increases progressively with iterations, and this adjustment is performed independently from p_m. This dual-parameter adaptive mechanism ensures diversified and robust solution strategies.

3.6 Population Destruction and Replenishment

To expand the search scope and escape local optima, the algorithm simulates a "mass extinction" event whenever the number of consecutive non-improving iterations reaches $n/2$. During this process, the top-

performing chromosomes are preserved, while the bottom one-third of the population—those with the lowest scores—are discarded. To restore the population to its original size, new random chromosomes are generated. This periodic partial population destruction and replenishment diversifies the solution pool, helping to explore a broader optimization space and mitigate stagnation.

4 Numerical Results

4.1 Implementation and Benchmarks

The algorithm was implemented using Python 3.7.4 in the Visual Studio Code development environment, executed on an Intel i7-8565U processor (1.8 GHz) with 8 GB RAM under a 64-bit Windows 10 operating system. Performance evaluation utilized two classical benchmark datasets: the Tsiligirides dataset[2] and the Chao dataset[16], both widely used in comparative algorithm analysis within related research. These datasets are based on Euclidean spatial models, where vertices are represented by 2D coordinates, edge times are calculated as Euclidean distances (rounded to one decimal place), and vertex scores are derived directly from the original datasets.

The Tsiligirides dataset consists of 49 test instances with node sizes of 32, 21, and 33, and time constraints ranging from 5 to 110 minutes. These instances are distributed as follows: 18 with T_{max} between 5-85, 11 between 15-45, and 20 between 15-110. The Chao dataset includes 40 test instances with larger node sizes of 64 and 66, and time constraints spanning 5 to 130 minutes. These instances are distributed as follows: 14 with T_{max} between 15-80 and 26 between 5-130. Both datasets are publicly available athttp://www.mech.kuleuven.be/cib/op.

4.2 Parameter Setting

Considering the experimental datasets, node scales range from 21 to 66, while time constraints extend from 5 to 130 minutes. To balance solution quality and computational efficiency, the algorithm's parameters were configured as follows: the population size was set to 60, maximum iterations ranged from 30 to 200, and the local search probability was dynamically adjusted between upper and lower bounds of 0.2 and 0.01, respectively.

In real-world applications, optimal solutions are often unknown. Therefore, this study employed a variable iteration strategy to systematically evaluate algorithm convergence and stability by analyzing optimization outcomes across different iteration counts. To ensure fairness and comparability, especially since the benchmark datasets lack drone-specific constraints, the parameter T_{safe} was set equal to T_{max} during testing. This approach eliminated potential biases, enabling objective performance evaluation in standard testing environments.

4.3 Results

This study assessed 89 test instances—10 trials for each Tsiligirides 1, Tsiligirides 2, Tsiligirides 3, Chao 64, and Chao 66 datasets—recording algorithm performance at iteration counts of 30, 50, 100, and 200. Figures 3, 4, and 5 depict boxplots of total scores across iterations with maximum total score for the Tsiligirides 1, 2, and 3 datasets, respectively.

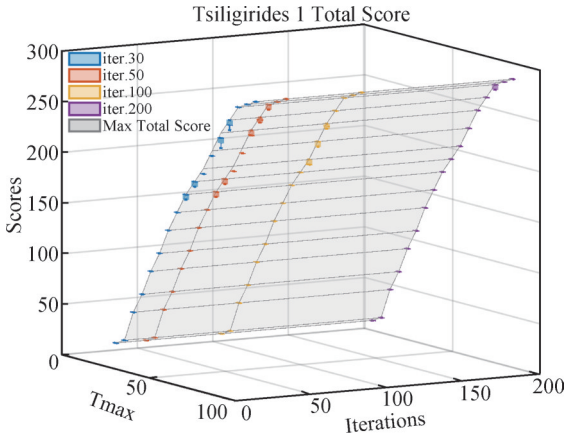

Fig.3 Total Score on Tsiligirides 1

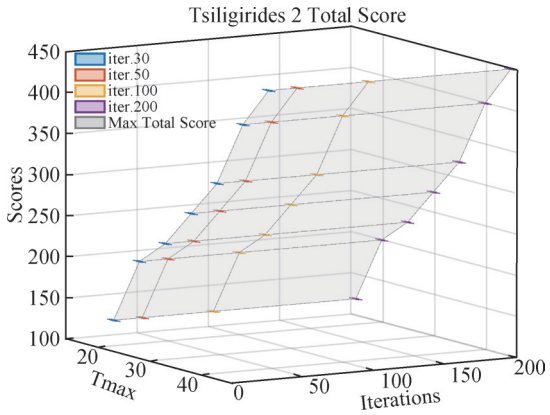

Fig.4 Total Score on Tsiligirides 2

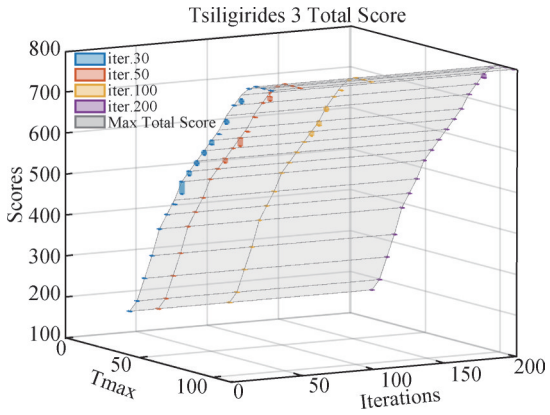

Fig.5 Total Score on Tsiligirides 3

On the Tsiligirides dataset, where the number of nodes is relatively small, experimental results show a progressive improvement in solution quality as iterations

increase. Average scores at each iteration converge toward the maximum total score. At 30 iterations, standard deviations are relatively high but decrease progressively with additional iterations, indicating the algorithm reaches a stable convergence state.

Figures 6 and 7 present boxplots of total scores for the Chao 64 and Chao 66 datasets, respectively. On the Chao dataset, which features larger node sizes, solution quality also improves with iterations. At 30 iterations, test instance scores are relatively low with high standard deviations, reflecting incomplete convergence at this stage. By 200 iterations, average scores increase significantly while standard deviations drop sharply, indicating the algorithm has reached stable convergence.

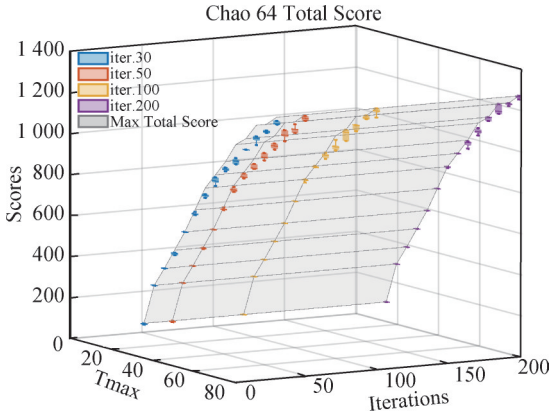

Fig.6 Total Score on Chao 64

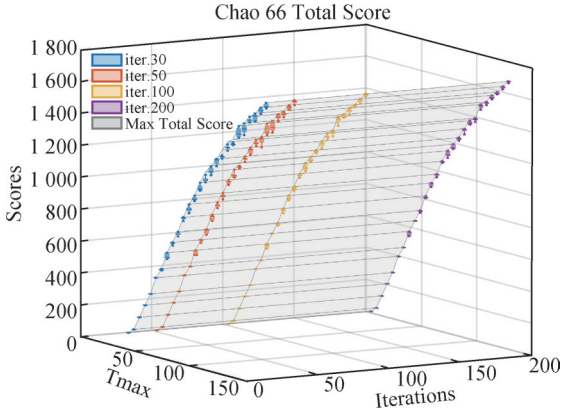

Fig.7 Total Score on Chao 66

In terms of total score accuracy, the algorithm attains 95% accuracy relative to the maximum total score across all node sizes and time constraints within 30 iterations, making it suitable for rapid assessment scenarios in drone route planning. For iteration counts exceeding 100, accuracy surpasses 99%, making the algorithm well-suited for precision-critical applications. These simulation experiments suggest that users can flexibly balance computational efficiency and solution accuracy by adjusting iteration counts to match specific application requirements.

Table 1 compares the best total scores achieved by the proposed algorithm against ARPSO[17], GLS[18], and IS-PSO[19] algorithms when tested on the Tsiligirides 2 dataset. All methods perform equally at $T_{max} \leq 20$ and $T_{max} = 40$, but this work achieves better scores at 30 and 40. This suggests the proposed method excels traditional heuristic methods.

Table 1 Comparison of Best Total Scores on Tsiligirides 2 Dataset

Tmax	ARPSO	GLS	IS-PSO	This Work
15	120	120	120	120
20	200	200	200	200
25	210	230	230	230
30	230	260	265	275
35	295	305	320	320
40	365	380	395	400
45	450	450	450	450

Figure 8 presents the optimal routes on the Tsiligirides 2 dataset under two time constraints. For $T_{max} = 30$, the total score is 275 with a total time of 29.9. For $T_{max} = 40$, the total score reaches 400 with a total time of 39.8. The node coordinates and scores, labeled in the figure, originate from the dataset file.

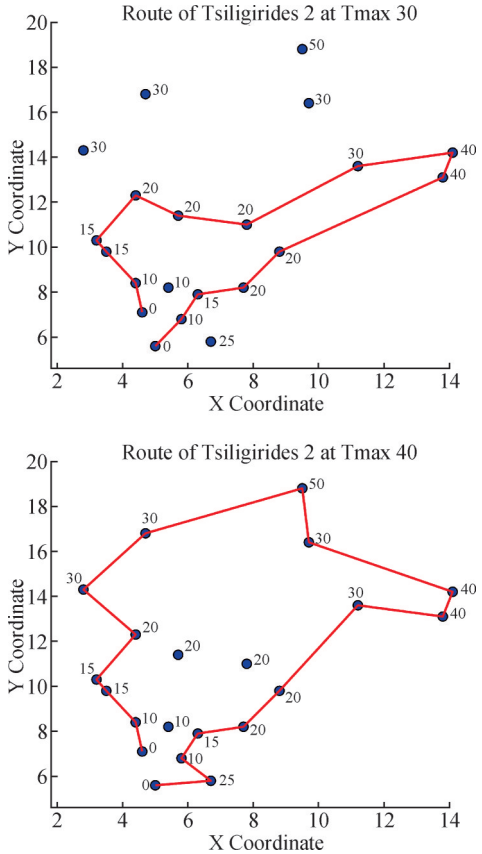

Fig.8 Route of Tsiligirides 2 at Tmax 30 and 40

Regarding computational efficiency, Figure 9 illustrates the boxplot of runtime across test instances. Results reveal a positive correlation between runtime and three factors: iteration counts, node numbers, and T_{max}.

While runtime scales linearly with iterations and T_{max}, it exhibits superlinear growth with increasing node numbers but remains significantly lower than the exponential $O\left(2^n\right)$ complexity of unoptimized approaches. This demonstrates an orders-of-magnitude improvement in computational efficiency.

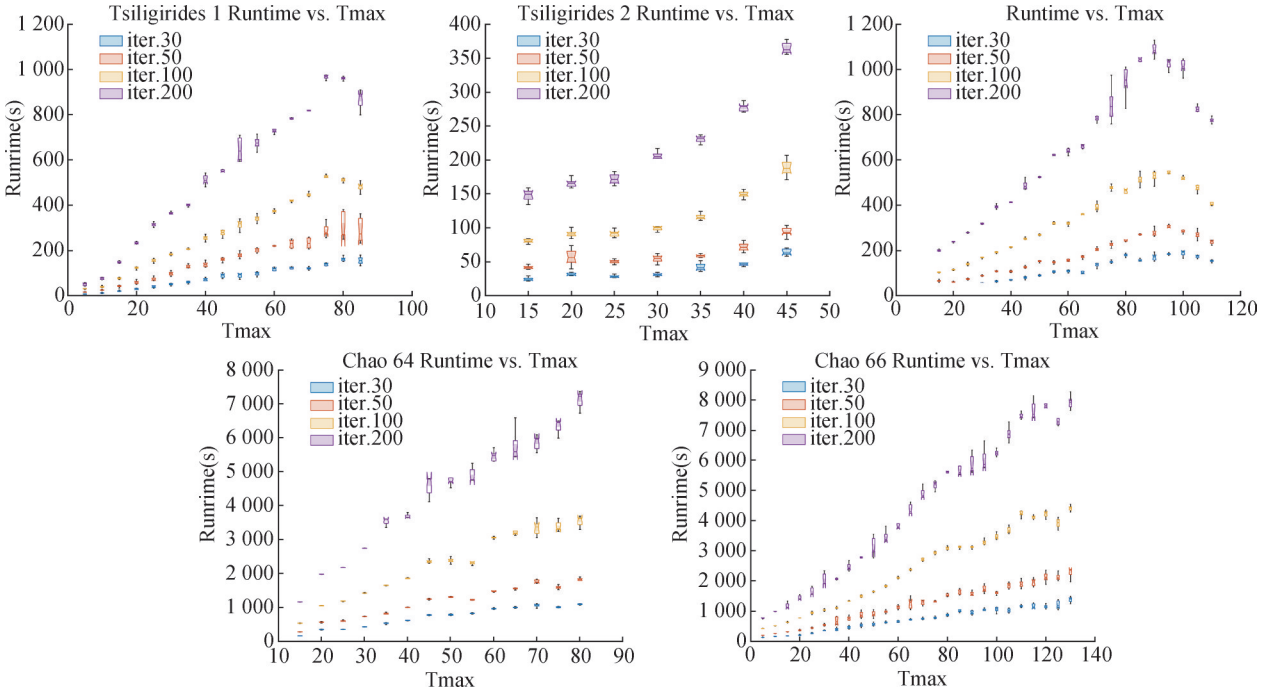

Fig.9 Runtime of Test Instances

5 Discussion

The proposed algorithm was implemented in Python, a language chosen for its flexibility and rapid prototyping capabilities. However, this choice introduced computational inefficiencies, as Python is inherently slower than compiled languages like C or C++. During experimental testing, the algorithm exhibited prolonged execution times, particularly for larger problem instances. This performance bottleneck can be attributed to Python's interpreted nature and the limitations imposed by the Global Interpreter Lock (GIL), which restricts true parallel processing. While the algorithm successfully produced feasible solutions, the trade-off between development speed and runtime efficiency became evident, especially in computationally intensive components such as local search.

A key limitation of this study is the algorithm's scalability when applied to large-scale or real-time scenarios. The runtime increased disproportionately with problem size, suggesting that further optimization is necessary for practical deployment. Additionally, the algorithm's performance was sensitive to parameter settings, such as neighborhood size and termination criteria, indicating a need for more robust adaptive tuning mechanisms. These limitations highlight the challenges of balancing solution quality, computational efficiency, and real-world applicability in heuristic-based routing algorithms.

Future research should focus on improving computational performance through platform optimization and algorithmic refinement. One promising direction is the adoption of high-performance computing techniques, such as GPU acceleration or parallel computing frameworks, to reduce execution times. Another approach involves rewriting critical modules in C or C++ and integrating them with Python using tools like Cython, thereby combining Python's ease of use with the speed of compiled languages. Additionally, exploring adaptive parameter tuning and approximation strategies could enhance the algorithm's efficiency without sacrificing solution quality. Beyond performance improvements, extending the algorithm to dynamic environments, such as real-time UAV routing with moving targets, or multi-objective optimization scenarios would broaden its applicability.

To facilitate reproducibility and encourage further development, the complete source code has been made publicly available athttps://gitee. com/hua-szut/UAV-Routing-MA. This open-access approach allows researchers to build upon the current work, whether through performance optimizations, benchmarking against existing solvers, or adaptations to related routing problems in logistics and autonomous systems.

6 Conclusion

This study developed an enhanced Memetic Algorithm for UAV routing problems with profits,

addressing real-world constraints like nonlinear battery degradation. Integrating hybrid initialization, improved crossover operators, and adaptive mechanisms including population destruction and replenishment, the algorithm achieves rapid convergence—reaching 95% solution accuracy in 30 iterations and 99% in 100 iterations. Validated across benchmark datasets (21-66 nodes, 5-130-min scenarios), it outperforms traditional methods in balancing profit, efficiency, and battery constraints. This adaptive approach provides a robust framework for UAV operations.

Author Contribution:

Siliang Hua: conceptualization, methodology and writing - original draftl; Jian Xu: software and validation; Huiguo Zhang: project administration and supervision; Qian Zhang: investigation and writing - review & editing; Lifeng Qin: funding acquisition and resources; Lixing Hua: data curation and visualization.

Acknowledgments:

The authors would like to thank Suzhou Deer-O Club for software testing and feedback suggestions in Suzhou Mass Sports Championship.

Foundation Information:

This research was funded by Ministry of Education, grant number 2024120910911.

Data Availability:

The authors declare that the datasets supporting the findings of this study are publicly accessible athttp://www. mech. kuleuven. be/cib/op. The source code developed is publicly hosted on Gitee athttps://gitee.com/hua-szut/UAV-Routing-MA.

Conflicts of Interest:

The authors declare no competing interests.

Dates:

Received 26 June 2025; Accepted 15 December 2025; Published online 31 September 2025

References

[1] Vansteenwegen, P. & Gunawan, A. (**2019**). *Orienteering Problems: Models and Algorithms for Vehicle Routing Problems with Profits.* Springer Cham. 978-3-030-29746-6

[2] Tsiligirides T. (**1984**). Heuristic Methods Applied to Orienteering [J]. *Journal of the Operational Research Society* , 35(1), 797-809. doi: 10.1057/jors.1984.162

[3] Fischetti M., González, J. & Toth, P. (**1998**). Solving the Orienteering Problem through Branch-and-Cut [J]. *INFORMS Journal on Computing* , 10(2), 133-148. doi: 10.1287/ijoc.10.2.133

[4] Vansteenwegen P., Souffriau, W. & Oudheusden, D. (**2011**). The orienteering problem: A survey [J]. *European Journal of Operational Research* , 209(1), 1-10. doi: 10.1016/j.ejor.2010.03.045

[5] Gunawan A., Lau, H. & Vansteenwegen, P. (**2016**). Orienteering Problem: A survey of recent variants, solution approaches and applications [J]. *European Journal of Operational Research* , 255(2), 315-332. doi: 10.1016/j.ejor.2016.04.059

[6] Xiao J., Yu, X. Zhou, G. et al. (**2022**). An improved ant colony algorithm for indoor AGV path planning [J]. *Chinese Journal of Scientific Instrument* , 43(03), 277-285. doi: 10.19650/j.cnki.cjsi.J2109071

[7] Gendreau M., Laporte G. & Semet F. (**1998**). A tabu search heuristic for the undirected selective travelling salesman problem [J]. *European Journal of Operational Research* , 106 (2-3), 539-545. doi: 10.1016/S0377-2217(97)00289-0

[8] Gao Y., Hu Y., Liu M., et al. (**2024**) Joint Multi-UAV Trajectory Design for Power Line Inspection [J]. *Journal of Electronics & Information Technology* , 46(5), 1958-1967. doi: 10.11999/JEIT231199

[9] Wang Y. (**2025**). Research on the logistics and distribution route planning of terminal on-board UAV based on simulated annealing algorithm [J]. *Automation & Instrumentation* , (2), 247-251. doi: 10.14016/j.cnki.1001-9227.2025.02.247

[10] Montemanni R. & Gambardella L. (**2009**). Ant colony system for team orienteering problems with time windows [J]. *Foundations of computing and Decision Sciences* , 34(4), 287-306.

[11] Schilde M., Doerner K., Hartl R.et al. (**2009**). *Metaheuristics for the bi-objective orienteering problem* [J]. Swarm Intelligence, 3(1), 179-201. doi:10.1007/s11721-009-0029-5

[12] Kobeaga G., Merino M. & Lozano J. (2018) An efficient evolutionary algorithm for the orienteering problem [J]. *Computer Operations & Research* , 90(1), 42-59. doi: 10.1016/j.cor.2017.09.003

[13] Prins C. (**2004**). A simple and effective evolutionary algorithm for the vehicle routing problem [J]. *Computer & Operations Research* , 31(12), 1985-2002. doi: 0.1016/S0305-0548(03)00158-8

[14] Souffriau W., Vansteenwegen P., Berghe G.& Oudheusden D. (2010) A path relinking approach for the team orienteering problem [J]. *Computers & Operations Research* , **37**(11) 1853-1859. doi: 10.1016/j.cor.2009.05.002

[15] Ruiz R. & Stützle T. (**2007**). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem [J]. *European Journal of Operational Research* , 177 (3), 2033-2049. doi: 10.1016/j.ejor.2005.12.009

[16] Chao I., Golden B. & Wasil E. (**1996**). A fast and effective heuristic for the orienteering problem [J]. *European Journal of Operational Research* , 88(3), 475-489. doi: 10.1016/0377-2217(95)00035-6

[17] Dallard H., Lam S., Kulturel-Konak S. (**2007**). Solving the orienteering problem using attractive and repulsive particle swarm optimization [Eds.]. *IEEE International Conference on Information Reuse and Integration Proceedings* , 12-17, 1-4244-1499-7.

[18] Vansteenwegen P., Souffriau W. et al. (**2009**). A guided local search metaheuristic for team orienteering problem [J]. *European Journal of Operational Research* , 196(1), 118-127. doi: 10.1016/j.ejor.**2008**.02.037

[19] Şevkli A. & Sevilgen F. (**2010**). StPSO: Strengthened particle swarm optimization [J]. *Turkish Journal of Electrical Engineering and Computer Sciences* , 18(6), 1095-1114. doi: 10.3906/elk-0909-18