scientific reports

Check for updates

OPEN

A spherical vector-based adaptive evolutionary particle swarm optimization for UAV path planning under threat conditions

Yanfei Liu, Hao Zhang✉, Hao Zheng, Qi Li & Qi Tian

Unmanned aerial vehicle (UAV) path planning is a constrained multi-objective optimization problem. With the increasing scale of UAV applications, finding an efficient and safe path in complex real-world environments is crucial. However, existing particle swarm optimization (PSO) algorithms struggle with these problems as they fail to consider UAV dynamics, resulting in many infeasible solutions and poor convergence to optimal solutions. To address these challenges, we propose a spherical vector-based adaptive evolutionary particle swarm optimization (SAEPSO) algorithm. This algorithm, based on spherical vectors, directly incorporates UAV dynamic constraints and introduces improved tent map and reverse learning to enhance the diversity and distribution of initial solutions. Additionally, dynamic nonlinear and adaptive factors are integrated to balance exploration and exploitation capabilities. To avoid local optima in highly complex environments, we propose an adaptive acceleration strategy for poor particles, and an evolutionary programming strategy is incorporated to further improve the optimization capability. Finally, we conducted comparative studies and in six benchmark scenarios with varying threat levels, and the results demonstrated that the proposed algorithm outperforms others in the initial solution effectiveness, the final solution accuracy, convergence stability, and scalability.

Keywords Particle swarm optimization, UAV path planning, Initialization, Adaptation, Evolutionary programming

Path planning refers to determining an optimal route from a starting point to a destination within a specific environment, using control algorithms, intelligent optimization methods, and other techniques, while adhering to various constraints to achieve the goal at minimal cost. In recent years, with the rapid development of drone technology, unmanned aerial vehicle (UAV) path planning has demonstrated great potential for applications in numerous fields. For instance, in agriculture, drones can be utilized for forest monitoring and crop irrigation[1]. They are widely applied in aerial photography and mapping in commerce. As for industry, drones are employed for data collection and targeted delivery. While in the military, drones perform tasks such as security patrols and precision strikes[2]. In practical applications, ensuring the feasibility and safety of UAV flight paths is crucial[3]. Therefore, research on UAV path planning in complex environments not only poses technical challenges but also holds immense value in various domains.

In recent years, nature-inspired path planning methods, known for handling complex constraints and possessing excellent optimization capabilities, have been widely applied in UAV path planning. Common heuristic algorithms include the A* algorithm, D* algorithm, and artificial potential field method. The A* algorithm estimates the distance from the current node to the target node using a heuristic function, thereby guiding the search direction[4]. Then the D* algorithm starts from the target node, conducting a reverse search towards the start point, and through its incremental reverse search ability, it can find adaptive optimal paths in dynamic environments[5]. The artificial potential field method (APF), on the other hand, treats the motion of the UAV as being influenced by forces within a virtual potential field, updating the position of the UAV by calculating the resultant forces to achieve path planning[6]. Metaheuristic algorithms, inspired by principles from natural or physical processes, guide the search process by adapting strategies in real-time based on gathered information, exhibiting strong adaptability and flexibility. These algorithms are generally divided into three categories: the first category comprises evolutionary algorithms inspired by natural selection and genetic

Department of Basic Courses, Xi'an Research Institute of Hi-Tech, Xi'an 710025, China. ✉email: zh_020938@163.com

inheritance, such as the differential evolution (DE) algorithm[7], cultural algorithm (CA)[8], and genetic algorithm (GA)[9]. The second category is based on physical or chemical laws found in nature, including the simulated annealing (SA) algorithm[10], spiral optimization algorithm[11], and galaxy-based search algorithm[12]. The third category, which is currently receiving significant attention, involves swarm intelligence algorithms, mimicking the spontaneous behaviors of groups in nature without supervision. For instance, inspired by the living habits of sharks, Hu et al. proposed an enhanced multi-strategy bottlenose dolphin optimizer to address the UAV path planning problem[13]. Each algorithm has its own strengths and weaknesses, and according to the No Free Lunch Theorem, no single algorithm, even an improved one, can perfectly solve all optimization problems[14]. Thus, analyzing the problem's nature and choosing an appropriate search strategy is key to successfully solving optimization challenges.

The particle swarm optimization (PSO) algorithm, introduced by Eberhart and Kennedy[15], is a swarm intelligence-based optimization method widely regarded as an ideal choice for path planning due to its simple structure and fast convergence speed[16]. The algorithm mimics the foraging behavior of bird flocks, updating positions and velocities through group information sharing and collaboration, gradually finding the optimal solution[17]. In recent years, PSO has been continuously improved in terms of search accuracy and convergence speed. For instance, Aslan et al.[18], inspired by the rapidly-exploring random tree* (RRT*) algorithm, proposed the GDRRT* algorithm based on global distance, utilizing intelligent sampling via target distance and integrating the PSO algorithm to shorten paths. However, PSO-GDRRT* struggles with local optima traps, limiting its effectiveness in complex problems. Xiang et al. combined enhanced PSO and GA to propose a hybrid optimization method for UAV path planning[19]. This method constructs initial paths by integrating Q-learning with random generation. Adaptive crossover and mutation strategies are dynamically introduced, aiding particles in escaping local optima. Although the algorithm enhances search capabilities, the uneven distribution of initial solutions may result in poor search effectiveness. Sonny et al. addressed the UAV path planning and energy consumption problem by selecting optimal target locations based on line-of-sight (LoS) probability, establishing LoS connections with users, thereby reducing energy consumption and flight time[20]. However, this algorithm does not account for UAV maneuverability, often generating infeasible solutions.

In UAV path planning within complex environments, path feasibility is closely linked to flight constraints such as flight time, distance, altitude, turning rate, climb angle variation, and safety factors. A safe path must avoid destroy zones like enemy fire coverage areas, obstacles, and no-fly zones. Although the PSO algorithm converges quickly, in environments with multiple constraints, it tends to fall into local optima, resulting in poor search accuracy. Additionally, the randomness in particle swarm initialization often leads to the generation of many infeasible solutions, further reducing the algorithm's convergence speed. The SPSO algorithm[21] introduces a spherical coordinate system, replacing the traditional Cartesian coordinate representation of the flight path with the UAV's flight attitude angles and flight distance, thereby integrating the physical information of the UAV's flight. This approach offers better constraints and greater intuitiveness. Although a substantial number of PSO variants already exist, demonstrating significant improvements in optimization capability and convergence speed, these algorithms typically rely on Cartesian coordinates to represent the UAV's position, neglecting the UAV's inherent flight characteristics. While the use of spherical coordinates for UAV path planning holds considerable potential, there remains a lack of in-depth research in this area, and optimization work based on the SPSO algorithm is still limited.

In order to fully exploit the advantages of spherical coordinates and overcome the limitations of the SPSO algorithm in optimization performance under complex environments, thereby enabling the UAV to find the optimal path in multi-obstacle environments, this paper presents further optimization based on spherical vectors, with the following key contributions:

(i) Considering the UAV's flight indices and environmental constraints, we propose a novel fitness weighting function that comprehensively evaluates the flight path length, path safety, changes in flight altitude, and angle variations, ensuring that the ultimate result is a practically feasible optimal solution.
(ii) A new algorithm named spherical vector-based adaptive evolutionary PSO (SAEPSO) is proposed. In the initialization phase, a perturbed tent map is proposed and combined with opposition-based learning to optimize the initialization. A novel adaptive factor is designed to improve the update of inertia weight and learning factors. Inspired by motion behavior, an acceleration factor is introduced into the velocity update formula. Additionally, evolutionary programming is integrated to substantially enhance the optimization capabilities.
(iii) Six scenarios of varying complexity were designed based on different levels of threat and comparative experiments with PSO, SPSO, and the latest swarm intelligence algorithms were conducted to test the initialization effectiveness, optimization performance and scalability of each algorithm. Finally, an ablation experiment was conducted to validate the effectiveness of the proposed improved modules.The remaining of this paper is organized as follows: Section "Literature review" explains the related work, introducing PSO, its improvements, and other swarm intelligence algorithms. Section "Problem formulation" provides mathematical modeling of environmental threats and constraints in UAV flight, along with an introduction to the objective function. Section "Related algorithms for UAV path planning" analyzes PSO, SPSO, and several of the latest swarm intelligence algorithms. Section "Spherical vector-based adaptive evolutionary particle swarm optimization" presents the SAEPSO algorithm and the theoretical foundation of the proposed improvements. Section "Results" designs comparative experiments and ablation studies, followed by the analysis of the experimental results. Finally, the conclusion is drawn in Section "Conclusion".

Literature review

Due to the tremendous potential of PSO in solving optimization problems, numerous researchers have focused on analyzing and improving it. However, the basic PSO suffers from premature convergence and a tendency to become trapped in local optima. To mitigate these issues and improve its effectiveness and efficiency, various modifications have been proposed by researchers. The development of PSO can generally be categorized into the following four types:

The first area is the optimization of hyperparameters. Shi and Eberhar have introduced a fuzzy system to dynamically adjust the inertia weight, enhancing the algorithm performance[22]. Compared to traditional PSO with linearly decreasing inertia weights, FAPSO showed significant performance improvements and was less sensitive to population size. An adaptive inertia weight method was proposed and benchmarked against other optimization methods by Feng et al.[23]. The results indicated that AIW-PSO outperformed others in convergence speed and optimization capability. Tanweer et al. introduced the SRPSO with self-regulating capabilities, incorporating strategies of best particle self-regulating inertia weights and self-perception for other particles[24]. Using 25 benchmark functions from CEC2005, they compared it to six advanced PSO variants, showing that SRPSO converged faster and provided superior solutions for most problems. Borowska proposed a new nonlinear dynamic inertia weight strategy, calculating new inertia weights based on the optimal and worst fitness values of particles[25]. This approach excelled in maintaining particle diversity, effectively overcoming premature convergence issues. Zhao et al. proposed an automatic parameter configuration PSO (APCPSO) algorithm, which establishes a parameter fitness evaluation criterion, allowing for periodic assessment of each parameter's fitness and dynamic adjustment of their values accordingly[26]. This method effectively addresses the challenge of parameter setting in PSO. Experimental validation on the CEC 2017 benchmark tests shows that APCPSO significantly outperforms comparative algorithms in both effectiveness and robustness.

The second area involves improvements in topology. Di Cesare et al. proposed I-PR-PSO based on a stochastic Markov chain model, using an inverse PageRank method to define an intelligent swarm topology, allowing superior particles to exert greater influence on others[27]. While more effective in achieving global optima, calculating the transition probability matrix required additional iterations. Wei et al. introduced multiple adaptive strategies for population adjustment, automatically removing weaker particles and adding stronger ones during evolution, significantly enhancing algorithm efficiency[28]. Zhang et al. introduced the ILPSO algorithm, incorporating and refining an acceleration factor to improve the convergence speed[29]. Liu et al. introduce Brownian motion and the Euler-Maruyama method to optimize particle motion pattern, maintaining the randomness of particle updates while enhancing swarm diversity[30]. A comparison with several existing PSO algorithms and artificial bee colony algorithms shows that this method is competitive in terms of accuracy.

The third area is reconstructing encoding methods. Fu et al. introduced θ-PSO by using phase angles and angular velocities to represent particle positions and movement directions, addressing high-dimensional space issues, and aiding in faster convergence[31]. Sun et al. optimized PSO using quantum mechanics concepts, resulting in QPSO. They assumed particles exhibit quantum behavior, updating positions under the influence of a quantum potential well[31,32]. Experiments on a 3DR Solo drone validated the feasibility and effectiveness of the approach. Phung and Ha encoded search trajectories as drone motion paths with MPSO, preserving crucial attributes like cognitive and social coherence of the swarm[33]. Compared with PSO variants and other metaheuristic algorithms, it showcased superiority. SPSO used spherical coordinates instead of Cartesian to represent particle positions and velocities, accurately modeling drone maneuver characteristics and efficiently constraining turn and climb angles to generate quality solutions[21].

The fourth area involves algorithm fusion. Enireddy and Kumar proposed PSO-CS algorithm to optimize the learning rate of neural networks, with experiments showing superior classification accuracy compared to unoptimized parameter settings[34]. Chen et al. integrated bee foraging learning (BFL) and PSO techniques to enhance both local and global search capabilities through three search phases: employment learning, observer learning, and scout learning[35]. During comparative experiments, BFL-PSO demonstrates high precision on the CEC2014 benchmark functions. Han et al. blended PSO with ocean predator strategies, incorporating Brownian motion and random walks to develop HMPPSO, demonstrating good performance in CEC2017 benchmarks[36]. Divasón et al. developed a novel hybrid algorithm by integrating PSO with GA, replacing particles with inferior fitness values in each iteration through selection, crossover, and mutation. This enhances solution quality, thereby accelerating the search for parsimony[37]. An improved hybrid PSO method, RGG-PSO+, is proposed by integrating the random geometric graph approach[38]. It dynamically adjusts the number of waypoints to adapt to different scenarios, enhancing the efficiency and quality of path planning. Experimental comparisons demonstrate its superior performance in convergence speed and path length.

Moreover, new swarm intelligence algorithms have also become a current research hotspot. The Grey Wolf Optimizer (GWO), first proposed by Mirjalili et al.[39], mimics the hunting behavior of grey wolves, with the leader wolves guiding the pack towards more optimal solution regions, eventually leading to the best solution. Inspired by humpback whale hunting behavior, Mirjalili and Lewis introduced the Whale Optimization Algorithm (WOA), which updates search agents' positions by simulating whales' encircling contraction, spiral updating, and prey searching strategies to find the optimal solution[40]. Heidari et al.[41], drawing from Harris hawk hunting behavior, proposed the Harris Hawk Optimization (HHO) algorithm, achieving optimal solutions through strategies that adjust between exploration, exploitation, and their transitional phases. In 2022, Xue and Shen introduced a new swarm intelligence optimization algorithm named Dung Beetle Optimization (DBO)[42], which simulates the behaviors of dung beetles in rolling, breeding, foraging, and stealing. The search agents is divided into four cooperative parts to collectively find the optimal solution.

Problem formulation

Scenario building

This study utilizes Digital Elevation Model (DEM) maps as the experimental environment to better simulate complex flight environments of UAV. One of the maps is based on Christmas Island in Australia, while the other represents a terrain with a more complex structure. After optimizing these two maps, we constructed scenarios with low, medium, and high threat levels for each. Considering the drone's diameter, each threat area is composed of three distinct cost regions. The division of one threat area is illustrated in the Fig. 1.

Assuming there are *k* danger zones, each zone is divided into three regions. The first region is the destruction zone with a radius of R_{m1}, such as enemy fire coverage or the obstacle itself. When a drone enters this area, $S_i \leq R_{m1}$, it indicates that the drone has been damaged. S_i represents the distance from the *i*-th flight segment $\overrightarrow{L'_i L'_{i+1}}$ to the center of the danger zone. Considering the radius of the drone, R_{m1} is computed as:

$$R_{m1} = D_m + U \tag{1}$$

where R_{m1} represents the effective radius of the destruction zone accounting for the drone's own radius, D_m is the radius of the original destruction zone, and *U* is the radius of the drone.

The second region is the threat zone with a radius of R_{m2}, such as enemy reconnaissance areas or the surroundings of obstacles. Due to factors of wind speed, drone positioning accuracy, and command latency in real-world conditions, if a drone enters this zone, $R_{m1} < S_i \leq R_{m2}$, it will incur a certain threat cost. By considering the risk distance *T*, R_{m2} can be calculated as:

$$\begin{cases} R_{m2} = D_m + U + T \\ T = 2 \times U \end{cases} \tag{2}$$

Finally, the last region is the safe zone, $S_i > R_{m2}$, where the drone is not exposed to any threats.

Constraint condition

The establishment of constraints in UAV path planning ensures that the drone can successfully complete its mission while maintaining its safety during flight. Besides avoiding dangers, we consider performance constraints of UAV and mission-specific requirements. Figure 2 illustrates the drone's flight status.

Flight height

The flight altitude of the *i*-th flight point H_i is restricted within a certain range. Depending on the specific drone model and flight environment, the maximum flight altitude H_{max} is limited to prevent signal loss, while the minimum flight altitude H_{min} is set to facilitate the collection of information along the flight path. These specific values are, of course, determined and adjusted based on the particular circumstances. This can be expressed as:

$$H_{min} \leq H_i \leq H_{max} \tag{3}$$

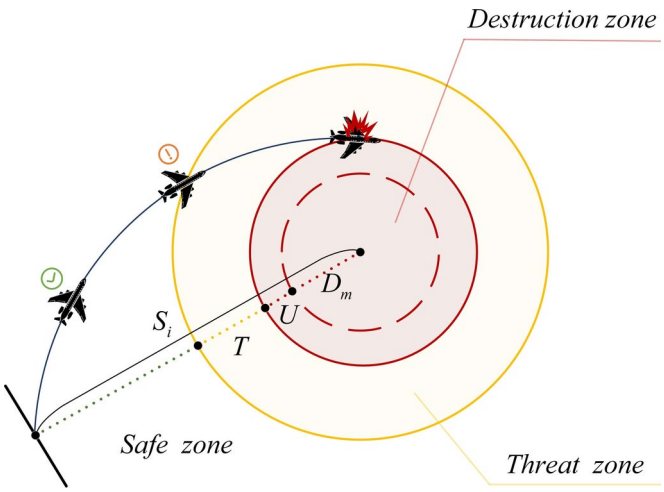


Fig. 1. Classification of the threat area.

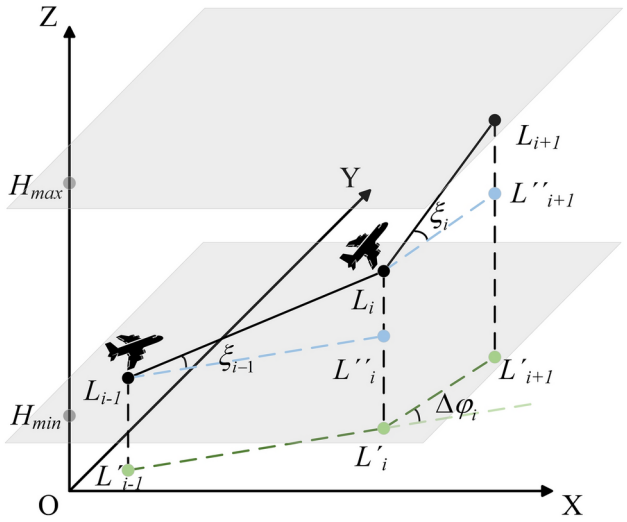


Fig. 2. Flight attitude parameters of UAV.

Turning and climbing angle

During the UAV's flight, to avoid danger zones and reach the target area with minimal cost, the turning and climbling angle will undergo adjustments. Considering the mechanical characteristics and flight inertia of the UAV, these angles must remain within their allowable maximum limits. The turning angle $\Delta\varphi_i$ refers to the change in the UAV's angle projected onto the horizontal plane *Oxy*, which means the angle from $\overrightarrow{L'_{i-1}L'_i}$ to $\overrightarrow{L'_iL'_{i+1}}$. It is given by:

$$\begin{cases} \Delta\varphi_i = \arctan\left(\dfrac{\left\|\overrightarrow{L'_{i-1}L'_i} \times \overrightarrow{L'_iL'_{i+1}}\right\|}{\overrightarrow{L'_{i-1}L'_i} \cdot \overrightarrow{L'_iL'_{i+1}}}\right) \\ \Delta\varphi_{min} \le \Delta\varphi_i \le \Delta\varphi_{max} \end{cases} \tag{4}$$

The climbing angle ξ_i represents the inclination between the UAV's trajectory and the horizontal plane. That means the angle from $\overrightarrow{L_iL_{i+1}}$ to $\overrightarrow{L'_iL'_{i+1}}$. Then the change of climbing angle $\Delta\xi_i$ can be calculated as:

$$\begin{cases} \xi_i = \arctan\left(\dfrac{z_{i+1} - z_i}{\left\|\overrightarrow{L'_iL'_{i+1}}\right\|}\right) \\ \Delta\xi_i = \xi_i - \xi_{i-1} \\ \Delta\xi_{min} \le \Delta\xi_i \le \Delta\xi_{max} \end{cases} \tag{5}$$

Fitness function

The quality of UAV flight path planning results is evaluated using a fitness function, which allows for the assessment of the algorithm's effectiveness. For problems with multiple reference criteria, Vilfredo Pareto proposed a Pareto optimal solution approach. However, this approach yields a set of results with equivalent priorities. To swiftly obtain an optimal solution and facilitate the comparison of different algorithms, we employ a weighted sum method for the reference criteria, gaining a comprehensive fitness function.

Path cost

UAV carrys a limited energy supply, which imposes a high demand for time efficiency when performing tasks such as material transport, target strikes, or area reconnaissance. Therefore, we choose to minimize the path cost F_1, defined as the total distance flown from the starting point to the target point. The path cost is obtained by calculating the Euclidean *Oxyz* distance $\left\|\overrightarrow{L'_iL'_{i+1}}\right\|$ between adjacent nodes on the flight path. The formula is expressed as:

$$\begin{aligned} F_1 &= \sum_{i=1}^{n-1} \left\|\overrightarrow{L_iL_{i+1}}\right\| = \sum_{i=1}^{n-1} \rho_i \\ &= \sum_{i=1}^{n-1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2 + (z_{i+1} - z_i)^2} \end{aligned} \tag{6}$$

where ρ_i represents the distance in spherical coordinates.

Safety cost
To ensure the UAV reaches the target area, the influence of danger zones cannot be overlooked. According to the map model we previously established, the UAV incurs varying costs $C_m(S_i)$ depending on the level of danger in different regions. The total safety cost F_2 is determined by summing the costs of all flight segments. The calculation formula is as follows:

$$\begin{cases} F_2 = \sum_{i=1}^{n-1} \sum_{m=1}^{k} C_m(S_i) \\ C_m(S_i) = \begin{cases} 0, & \text{if } S_i > R_{m2} \\ R_{m2} - S_i, & \text{if } R_{m1} < S_i \leq R_{m2} \\ \infty, & \text{if } S_i \leq R_{m1} \end{cases} \end{cases} \tag{7}$$

Height variation cost
Considering the effects of gravity, frequent changes in the UAV's flight altitude can increase energy consumption, affect the quality of the collected flight path data, and pose safety risks. Therefore, we represent the cost F_3 by calculating the height variation $|z_{i+1} - z_i|$ between adjacent nodes on the flight path. The formula for this altitude cost is as follows:

$$F_3 = \sum_{i=1}^{n-1} Mg\,|z_{i+1} - z_i| \tag{8}$$

where the parameter *M* represents the mass of the UAV, and *g* is the local gravitational acceleration.

Angle variation cost
In addition to height variations, UAV also experiences turning angles $\Delta\varphi_i$ and changes of climbing angles $\Delta\xi_i$ during flight. An excellent flight path should be smooth, without sharp turns, to minimize braking energy consumption and improve UAV's flight stability. The smoothness of the path is primarily determined by these angles, therefore the angle variation cost F_4 can be obtained from follows:

$$F_4 = \sum_{i=2}^{n-1} (a_1\,|\Delta\varphi_i| + a_2\,|\Delta\xi_i|) \tag{9}$$

where a_1 and a_2 are the penalty coefficients for the turning angle and climbing angle change, respectively.

Finally, the fitness function F_{fit} can be derived by combining the contributions from the flight path F_1, UAV safety F_2, height variation F_3 and attitude angle F_4. It is given by:

$$F_{fit} = \omega_1 F_1 + \omega_2 F_2 + \omega_3 F_3 + \omega_4 F_4 \tag{10}$$

where ω_1 to ω_4 are the weight coefficients. They reflect the relative importance of each component in fitness function. By calculating all of the contributions, we can assess the quality of the path and determine how well it meets the desired criteria for efficiency, safety, and stability.

Related algorithms for UAV path planning

The UAV path planning problem is essentially an optimization task aimed at minimizing a fitness function, with the optimal solution being found by identifying the minimum value. Heuristic, metaheuristic, and computational intelligence methods are particularly well-suited for addressing such problems, possessing the capability to find high-quality solutions among numerous feasible options. Due to its relatively fast convergence speed and high convergence precision, PSO has become one of the most popular approaches for solving UAV path planning. SPSO, based on spherical coordinates, was compared with PSO, θ-PSO, and QPSO, demonstrating the superiority of spherical vectors in addressing UAV path planning challenges. Therefore, in addition to PSO and SPSO, this section will introduce GWO, WOA, HHO and DBO, which are outstanding heuristic search algorithms that have been utilized to solve UAV path planning problems in recent years.

Particle swarm optimization

In PSO, each particle represents a potential solution, characterized by two attributes: position $X_{(\tau)}$ and velocity $V_{(\tau)}$ in the search space of τ dimensions. Since the calculation in each dimension is independent, we can use $X = (x_1, x_2, \ldots, x_n)$ to represent all nodes of a particle in τ dimension. $V = (v_1, v_2, \ldots, v_n)$ determines the

direction and distance of the particle's movement within the search space in the same dimension. During the iterative process of the algorithm, each particle updates its velocity and position based on two key factors. One is personal best position p_{best}, which encountered by the particle itself. The other is global best position g_{best}, which found by the entire swarm. It can be conducted as:

$$\begin{cases} \omega^t = \omega_{\max} - \dfrac{t}{T}\left(\omega_{\max} - \omega_{\min}\right) \\ v_i^{t+1} = \omega^t v_i^t + c_1 r_{1,i}\left(p_{best,i}^t - x_i^t\right) + c_2 r_{2,i}\left(g_{best,i}^t - x_i^t\right) \\ x_i^{t+1} = x_i^t + v_i^{t+1}, \quad i = 1, 2, \ldots, n \end{cases} \tag{11}$$

where v_i^t and x_i^t are the velocity and position of node *i* of the particle at iteration *t*.The parameter T represents the maximum number of iterations, determining how times the particles will search for the optimal solution. Increasing linearly with the iteration t, ω^t is the inertia weight that controls the influence of the previous velocity to balance the exploration and exploitation capabilities of the swarm. c_1 and c_2 are the cognitive and social coefficients, determining the learning level of the personal best and global best positions. $r_{1,i}$ and $r_{2,i}$ are random numbers uniformly distributed in the range [0, 1], aiming at adding stochasticity to the search.

For UAV path planning, the process typically occurs within a three-dimensional space, $D = 3$. The position x_i and velocity v_i of a particle can be expressed as:

$$\begin{cases} x_i = (x_{ix}, x_{iy}, x_{iz}) \\ v_i = (v_{ix}, v_{iy}, v_{iz}), \quad i = 1, 2, \ldots, n \end{cases} \tag{12}$$

x_i and v_i are iteratively calculated using the update mechanism in Eq. (11), which ultimately yields the desired result.

Spherical vector-based particle swarm optimization

SPSO utilizes a spherical coordinate system (ρ, θ, φ) instead of a Cartesian coordinate system (x, y, z) to describe the position and velocity of UAV. The three components of the spherical coordinates correspond directly to the UAV's flight distance ρ, climbing angle $(\pi/2 - \theta)$ and turning angle φ. We can use ξ to represent the climbing angle, and it can be calculated as:

$$\xi = \pi/2 - \theta \tag{13}$$

Then, a complete flight path P and the corresponding velocity ΔP can be represented as follows:

$$\begin{cases} P = (\rho_1, \xi_1, \varphi_1, \rho_2, \xi_2, \varphi_2, \ldots, \rho_n, \xi_n, \varphi_n) \\ \Delta P = (\Delta\rho_1, \Delta\xi_1, \Delta\varphi_1, \Delta\rho_2, \Delta\xi_2, \Delta\varphi_2, \ldots, \Delta\rho_n, \Delta\xi_n, \Delta\varphi_n) \end{cases} \tag{14}$$

Denoting a flight segment $(\rho_i, \xi_i, \varphi_i)$ as μ_i and velocity $(\Delta\rho_i, \Delta\xi_i, \Delta\varphi_i)$ as $\Delta\mu_i$. The update strategy of SPSO can be derived as follows:

$$\begin{cases} \Delta\mu_i^{t+1} = \omega^t \Delta\mu_i^t + c_1 r_{1,i}\left(p_{\text{best},i}^t - \mu_i^t\right) + c_2 r_{2,i}\left(g_{\text{best},i}^t - \mu_i^t\right) \\ \mu_i^{t+1} = \mu_i^t + \Delta\mu_i^{t+1}, \quad i = 1, 2, \ldots, n \end{cases} \tag{15}$$

To facilitate the computation of the fitness function, while maintaining consistency in the calculation method, the solutions should be converted from the spherical coordinate system to the Cartesian coordinate system. The conversion formulas are given by the following equations:

$$\begin{cases} x_{ix} = x_{(i-1)x} + \rho_{i-1}\cos\xi_{i-1}\cos\varphi_{i-1} \\ x_{iy} = x_{(i-1)y} + \rho_{i-1}\cos\xi_{i-1}\sin\varphi_{i-1} \\ x_{iz} = x_{(i-1)z} + \rho_{i-1}\sin\xi_{i-1} \end{cases} \tag{16}$$

Substituting (x_i, y_i, z_i) into Eq. (10), we can obtain the fitness value of SPSO. Based on this value, the local and global optimal solutions are updated. SPSO iterates continuously until the termination condition is met, with the objective of converging to the optimal solution.

Grey wolf optimizer

In GWO, search agents, representing individual wolves within the pack, are categorized into four types. The fitness of all search agents is calculated based on Eq. (10), and they are ranked accordingly. The best solution is denoted as wolf α, the second and third best solutions as wolf β and δ, respectively, the remaining search agents as wolf ϱ. They update positions guided by α, β and δ. For the UAV path planning problem involving n path nodes, the update formula for the next position $x_{j\alpha,i}^{t+1}$ of wolve *j* at node *i* under α wolf's guidance is defined by:

$$\begin{cases} x_{j\alpha,i}^{t+1} = x_{\alpha,i}^{t} - A_1 \cdot D_{\alpha,i}^{t} \\ D_{\alpha,i}^{t} = \left| C_1 \cdot x_{\alpha,i}^{t} - x_{j,i}^{t} \right|, \quad j = 1, 2, \ldots, b \end{cases} \tag{17}$$

where $D_{\alpha,i}^{t}$ represents a distance between *i*-th nodes of wolf *j* and wolf α, and *t* is the iteration count. A_1 and C_1 are constant coefficients that respectively control the convergence speed and exploration capability. A_1 and C_1 are computed as:

$$\begin{cases} C_1 = 2r_2 \\ A_1 = 2\psi \cdot r_1 - \psi \end{cases} \tag{18}$$

where r_1 and r_2 are uniformly distributed random numbers in the range [0, 1], introducing randomness to enhance exploration capability. The parameter ψ decreases linearly from 2 to 0 over the course of the iterations. The position update mechanism under the guidance of wolf β and wolf δ follows the same form as that of the wolf α, by the following equations:

$$\begin{cases} x_{j\beta,i}^{t+1} = x_{\beta,i}^{t} - A_2 \cdot D_{\beta,i}^{t} \\ D_{\beta,i}^{t} = \left| C_2 \cdot x_{\beta,i}^{t} - x_{j,i}^{t} \right| \end{cases} \quad \begin{cases} x_{j\delta,i}^{t+1} = x_{\delta,i}^{t} - A_3 \cdot D_{\delta,i}^{t} \\ D_{\delta,i}^{t} = \left| C_3 \cdot x_{\delta,i}^{t} - x_{j,i}^{t} \right| \end{cases} \tag{19}$$

Ultimately, wolf *k* hunts the target under the influence of wolf α, β and δ as follows:

$$x_{j,i}^{t+1} = \left(x_{j\alpha,i}^{t+1} + x_{j\beta,i}^{t+1} + x_{j\delta,i}^{t+1} \right) / 3 \tag{20}$$

Note that the wolf α, β and δ are not fixed; rather, they are dynamically updated throughout the iterations.

Whale optimization algorithm

WOA simulates the hunting behavior of whales, dividing the optimization process into three phases: encircling prey, bubble-net attacking and prey search. Each whale's position $X = (x_1, x_2, \ldots, x_n)$ corresponds to a candidate solution with *n* flight path nodes. During encircling prey phase, the current best search agent at node *i* is considered the position $x_{best,i}$ of the target prey, guiding other whales to encircle this target. This action can be conducted as:

$$\begin{cases} x_i^{t+1} = x_{best,i}^{t} - A \cdot D_i \\ D_i = \left| C \cdot x_{best,i}^{t} - x_i^{t} \right| \\ A = 2\psi_1 \cdot r_1 - \psi_1 \\ C = 2r_2, \quad i = 1, 2, \ldots, n \end{cases} \tag{21}$$

where D_i represents a distance between the whale and the best search agent at the *i*-th flight path node. r_1 and r_2 are random numbers within the range [0, 1]. During the iterative process, psi_1 decreases linearly from 2 to 0.

Then, WOA enters the bubble-net attacking stage, where the search agent moves along a spiral path within a shrinking circle achieved by decreasing the value of psi_1 in Eq. (21). Of course, this path is centered around the target position $x_{best,i}$. The spiral updating method can be expressed as:

$$\begin{cases} x_i^{t+1} = D_i^{*} \cdot e^{\psi_2 r_3} \cdot \cos(2\pi r_3) + x_{best,i}^{t} \\ D_i^{*} = \left| x_{best,i}^{t} - x_i^{t} \right| \end{cases} \tag{22}$$

where D_i^{*} represents the accurate distance between the *i*-th flight path node and the best search agent. ψ_2 is a constant used to define the shape of the logarithmic spiral, and r_3 is a random number within the range [−1, 1].

Finally, in addition to searching near the target position $x_{best,i}$, WOA also engages in a random search for prey. This involves randomly selecting the position $x_{rand,i}$ of a search agent as a reference to update the next position x_i^{t+1} by the following equations:

$$\begin{cases} x_i^{t+1} = x_{rand,i}^{t} - A \cdot D_i \\ D_i = \left| C \cdot x_{rand,i}^{t} - x_i^{t} \right| \end{cases} \tag{23}$$

Note that to simultaneously simulate the whale's encircling mechanism and spiral updating method, a random number ε in the range [0, 1] is introduced. Hence, the choice of strategy for a search agent is determined by $|A|$ and ε. When $\varepsilon \geq 0.5$, the spiral updating method in Eq. (22) is employed. If $\varepsilon < 0.5$ and $|A| < 1$, the encircling prey mechanism in Eq. (21) is used. When $\varepsilon < 0.5$ and $|A| \geq 1$, the prey search strategy in Eq. (23) is adopted.

Harris hawk optimization

HHO is based on the predatory behavior of Harris hawks, particularly focusing on exploring prey and surprise pounce tactics. The optimization process is divided into three phases: exploration, transition and exploitation. During the exploration phase, Harris hawks randomly perch at some locations based on two strategies while waiting to detect prey. The choice of perching strategy is determined by a chance variable ε_1, which follows a uniform distribution between 0 and 1. When $\varepsilon_1 < 0.5$, the hawk perches based on the positions of other family members or the prey. Conversely, for the condition of $\varepsilon_1 \geq 0.5$, it perches on a randomly selected tall tree. This can be expressed as:

$$x_{j,i}^{t+1} = \begin{cases} x_{rand,i}^{t} - r_1 \left| x_{rand,i}^{t} - 2r_2 \cdot x_{j,i}^{t} \right|, & \varepsilon_1 \geq 0.5 \\ \left(x_{prey,i}^{t} - x_{m,i}^{t} \right) - r_3 \left(lb_i + r_4 \left(ub_i - lb_i \right) \right), & \varepsilon_1 < 0.5 \end{cases} \tag{24}$$

where $x_{j,i}^{t+1}$, $x_{rand,i}^{t}$ and $x_{j,i}^{t}$ represent, respectively, the position of the agent *j* in the next iteration *t*, the position of a randomly selected agent from the current population, and the position of agent *j* in the *t*-th iteration, located at the *i*-th node. $x_{prey,i}^{t}$ and $x_{m,i}^{t}$ show the best search position and the average position of all agents at node *i* during iteration *t*, respectively. r_1, r_2, r_3, r_4 and ε_1 are random numbers within the range [0, 1]. lb_i and ub_i correspond to the minimum and maximum values of the search range at node *i*, respectively.

The transition phase simulates the energy decrease as the prey attempts to escape. The decision to change phases for the Harris hawk is based on the current energy level *E* of the prey,which is computed as:

$$E = 2E_0 \cdot \left(1 - \frac{t}{T} \right) \tag{25}$$

where E_0 represents the initial energy, while *T* denotes the maximum number of iterations. If the escape energy $E \geq 1$, the Harris hawks remain in exploration phase, searching for prey in different regions. When $E < 1$, Harris hawks enter exploitation phase to besiege the prey, focusing their search around the prey.

Upon entering the exploitation phase, the Harris hawks determine their strategy based on whether the prey has escaped, as indicated by chance ε_2, and the escape energy *E*. Depending on both factors, they choose from four strategies to besiege the prey. That can be defined of the form:

Case 1 When $\varepsilon_2 \geq 0.5$ and $|E| \geq 0.5$, the first strategy is given by:

$$\begin{cases} x_{j,i}^{t+1} = \left(x_{prey,i}^{t} - x_{j,i}^{t} \right) - E \left| J \cdot x_{prey,i}^{t} - x_{j,i}^{t} \right| \\ J = 2(1 - r_5) \end{cases} \tag{26}$$

where r_5 is a random number inside [0, 1], while *j* represents the degree of the prey's random jump throughout the escape process.

Case 2 When $\varepsilon_2 \geq 0.5$ and $|E| < 0.5$, the second strategy is given by:

$$x_{j,i}^{t+1} = x_{prey,i}^{t} - E \left| x_{prey,i}^{t} - x_{j,i}^{t} \right| \tag{27}$$

Case 3 When $\varepsilon_2 < 0.5$ and $|E| \geq 0.5$, the third strategy is given by:

$$x_{j,i}^{t+1} = \begin{cases} x'_{j,i}, & if\ F_{fit}\left(X'_j \right) < F_{fit}\left(X_j^t \right) \\ x''_{j,i}, & if\ F_{fit}\left(X''_j \right) < F_{fit}\left(X_j^t \right) \end{cases} \tag{28}$$

where $X'_j = \left(x'_{j,1}, x'_{j,2}, \ldots, x'_{j,n} \right)$ represents a candidate solution updated based on the soft besiege method from X_j^t, while $X''_j = \left(x''_{j,1}, x''_{j,2}, \ldots, x''_{j,n} \right)$ denotes another candidate solution updated based on the LF-based patterns from X_j^t as well. Both update approaches at the node *i* can be conducted as:

$$\begin{cases} x'_{j,i} = x_{prey,i}^{t} - E \left| J \cdot x_{prey,i}^{t} - x_{j,i}^{t} \right| \\ x''_{j,i} = x'_{j,i} + r_6 \cdot LF \end{cases} \tag{29}$$

where r_6 is a random number inside [0, 1] while LF is a random number following Levy distribution.

Case 4 When $\varepsilon_2 < 0.5$ and $|E| < 0.5$, the last strategy is given by:

$$\begin{cases} x_{j,i}^{t+1} = \begin{cases} x'_{j,i}, & if\ F_{fit}\left(X'_j\right) < F_{fit}(X_j^t) \\ x''_{j,i}, & if\ F_{fit}\left(X''_j\right) < F_{fit}\left(X_j^t\right) \end{cases} \\ x'_{j,i} = x_{prey,i}^t - E\left|J \cdot x_{prey,i}^t - x_{m,i}^t\right| \\ x''_{j,i} = x'_{j,i} + r_7 \cdot LF \end{cases} \tag{30}$$

where r_7 is a random number inside [0, 1].

Dung beetle optimizer

Inspired by the collective behavior of dung beetles, DBO categorizes search agents into four types: ball-rolling dung beetles, the brood balls, small dung beetles and thief dung beetles. Each type adopts its own strategy to update its position. The ball-rolling dung beetles operate in two modes: obstacle-free mode and obstacle mode. When the random number $\varepsilon < 0.9$, it is assumed that there are no obstacles, and the ball-rolling dung beetles navigate using the sun. The next position $x_{roll,i}$ of node *i* is computed as:

$$\begin{cases} x_{roll,i}^{t+1} = x_{roll,i}^t + \psi_a \psi_k x_{roll,i}^{t-1} + \psi_b \Delta x_{roll,i} \\ \Delta x_{roll,i} = \left|x_{roll,i}^t - x_{worst,i}^t\right| \end{cases} \tag{31}$$

where ψ_a denotes the natural influence valued at 1 or −1, while ψ_k is a constant value inside (0, 0.2], which indicates the deflection coefficient. ψ_b is also a constant value inside (0, 1). $\Delta x_{roll,i}$ and $x_{worst,i}$ represent the effect of light intensity variation and node *i* of the global worst solution, respectively.

Conversely, when $\varepsilon > 0.9$, it is assumed that they have encountered an obstacle, requiring dancing to reorient and determine the next direction by deflection angle r_θ inside [0, π]. It is provided as follows:

$$\begin{cases} x_{roll,i}^{t+1} = x_{roll,i}^t,\ if\ r_\theta = 0, \pi/2, \pi \\ x_{roll,i}^{t+1} = x_{roll,i}^t + \tan(r_\theta)\left|x_{roll,i}^t - x_{roll,i}^{t-1}\right|,\ others \end{cases} \tag{32}$$

Brood balls are laid on a suitable area by female dung beetles. A boundary selection strategy is used to simulate this area, by the following equations:

$$\begin{cases} lb_i^* = \max\left(p_{best,i}\left(1 - \frac{t}{T}\right), lb_i\right) \\ ub_i^* = \min\left(p_{best,i}\left(1 + \frac{t}{T}\right), ub_i\right) \end{cases} \tag{33}$$

where lb_i^* and ub_i^* severally represent the lower and upper bounds of the spawning area, while lb_i and ub_i correspond to the lower and upper bounds of the optimization problem, respectively. $p_{best,i}$ is node *i* of the local best position, and *T* is the maximum number of iterations. That means the spawning area is dynamically adjusted according to iteration *t*. Once that area is determined, the position of the brood ball can be calculated as:

$$x_{brood,i}^{t+1} = p_{best,i} + r_1\left(x_{brood,i}^t - lb_i^*\right) + r_2\left(x_{brood,i}^t - ub_i^*\right) \tag{34}$$

where r_1 and r_2 are random numbers within the range [0, 1].

The small dung beetles search for an optimal foraging area based on the global best solution g_{best}. This area is also dynamically updated, which can be expressed as:

$$\begin{cases} lb_i^\sharp = \max\left(g_{best,i}\left(1 - \frac{t}{T}\right), lb_i\right) \\ ub_i^\sharp = \min\left(g_{best,i}\left(1 + \frac{t}{T}\right), ub_i\right) \end{cases} \tag{35}$$

where $lb_i^\sharp$ and $ub_i^\sharp$ represent the lower and upper bounds of the optimal foraging area respectively, while $g_{best,i}$ is node *i* of the global best position. Then, the position of the small dung beetle is updated by:

$$x_{small,i}^{t+1} = x_{small,i}^t + r_3\left(x_{small,i}^t - lb_i^\sharp\right) + r_4\left(x_{small,i}^t - ub_i^\sharp\right) \tag{36}$$

where r_3 represents a random number that follows normally distributed, but r_4 is a random number inside [0, 1].

The thief dung beetles steal dung balls from other beetles. They compete for food around the best area g_{best}, as described by the follows:

$$x_{thief,i}^{t+1} = g_{best,i} + \psi_s r_5\left(\left|x_{thief,i}^t - p_{best,i}\right| + \left|x_{thief,i}^t - g_{best,i}\right|\right) \tag{37}$$

where ψ_s is a constant value, but r_5 is a random number that follows normally distributed. The four types of agents, with different roles, collaboratively search for the optimal solution during the iterations.

Spherical vector-based adaptive evolutionary particle swarm optimization

PSO is effective for solving optimization problems, and its efficiency is further enhanced by introducing spherical coordinates. However, compared to the intelligent algorithms that have emerged in recent years, it still has some limitations. Expecially, when applied to complex environments for drone optimization problems, merely using spherical vector still struggles with issues like a lack of effective initial solutions and a tendency to get trapped in local optima. These challenges lead to low algorithmic efficiency and poor solution accuracy. To fully harness the potential of the PSO algorithm and address these issues, this paper proposes a new algorithm named Spherical vector-based adaptive evolutionary particle swarm optimization(SAEPSO) based on SPSO.

Improvement of initialization

Tent map with perturbation

In all the aforementioned algorithms, the initialization process typically uses the 'rand()' function, which generates pseudo-random numbers. To improve the randomness of initial positions, this paper utilizes the tent map[43] to generate chaotic random numbers distributed in the [0, 1] range, which can be conducted as:

$$\gamma_{t+1} = f_\psi(\gamma_t) = \begin{cases} \psi\gamma_t \ , & \gamma_t < \frac{1}{2} \\ \psi(1-\gamma_t) \ , & \gamma_t \geq \frac{1}{2} \end{cases} \tag{38}$$

where ψ is a constant value that is proportional to the system's chaotic behavior, typically ranging from 0 to 2. Figure 3 illustrates the distribution of γ_t under different values of ψ.

When $\psi = 2$, the tent map becomes a central tent map, generating chaotic sequences within [0, 1], exhibiting good distribution and randomness. However, this iterative sequence has small periodic cycles. For instance, if γ_t falls within the set of 0.2, 0.4, 0.6, 0.8, it will be always within the set. Additionally, there are converging sequences, such as when γ_t = 0.25, 0.5, or 0.75, which will iterate to 0. To address these issues, this paper introduces a perturbation factor and specifically handles the case when $\gamma_t = 0.5$. This can be expressed as:

$$\gamma_{t+1} = f_\psi(\gamma_t) = \begin{cases} \psi_1\gamma_t + (2r_1 - 1)/\psi_2 \ , & 0 < \gamma_t < \frac{1}{2} \\ \psi_1(1-\gamma_t) + (2r_2 - 1)/\psi_2 \ , & \frac{1}{2} < \gamma_t < 1 \\ r_3 \ , & others \end{cases} \tag{39}$$

where r_1, r_2, r_3 and γ_0 are random numbers based on 'rand()' function, while ψ_1 and ψ_2 are key constants with values of 2 and 100, respectively. These modifications enable the system to escape periodic cycles and convergence traps, resulting in a more uniformly random distribution.

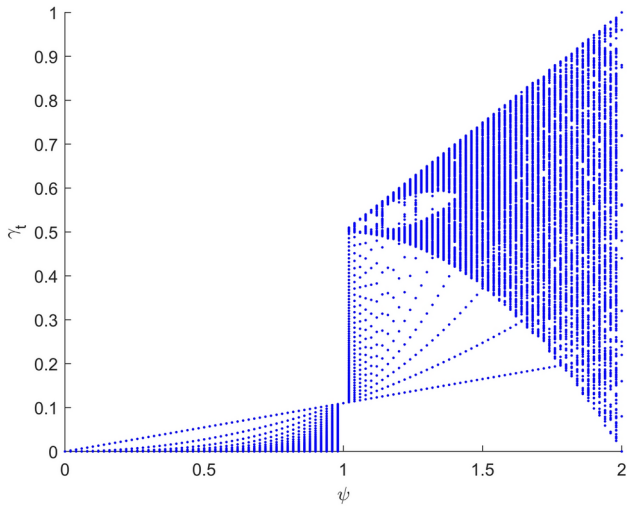


Fig. 3. Tent map under different parameters.

Opposition-based learning
In UAV path planning, the initial solutions $X_j^0 = \left(x_{j,1}^0, x_{j,2}^0, \ldots, x_{j,n}^0\right), j = 1, 2, \ldots, b$ are generally generated within upper *ul* and lower *lb* bound constraints as follows:

$$x_{j,i}^0 = lb_i + (ub_i - lb_i)\,\gamma_1 \tag{40}$$

where $gamma_1$ is a random number obtained by Eq. (39).

To achieve a broader and more uniform distribution of the initial solutions within the boundaries, the opposition-based learning[44] is employed to generate both the current solution and its opposite solution simultaneously. The opposite solution $x_{j,i}^{\prime 0}$ is calculated as:

$$x_{j,i}^{\prime 0} = lb_i + ub_i - x_{0,i} \tag{41}$$

Generating *b* solutions by Eqs. (40) and (41) respectively, a population of 2*b* agents is formed. These agents are ranked based on their fitness values, and the top *b* solutions are selected as initial solutions. This method helps to improve the quality of the initial solutions, accelerates the convergence of the algorithm, and enhances its stability.

Adaptive factor based on fitness

To extract more information from the swarm, this paper introduces a fitness-based adaptive factor *fap*, which incorporates the best agent's information and the average performance of all agents. These knowledge ensures a more precise adjustment of the agents' movement, contributing to the overall efficiency of the algorithm. The *fap* is given as follows:

$$\begin{cases} fap^t = \left(F_{fit}\left(X^t\right) - F_{fit}\left(G_{best}^t\right)\right) / \left(F_{avge}^t - F_{fit}\left(G_{best}^t\right)\right) \\ F_{avge}^t = \sum F_{fit}\left(X^t\right) / b \end{cases} \tag{42}$$

where *X* represents an individual agent, G_{best} denotes the best-performing agent, F_{avge} is the average fitness value of all agents, and *b* stands for the total number of particles in the swarm. By incorporating *fap* into inertia weight ω and velocity *v* update formula, agents can make more accurate judgments to arrive at the best solution.

Adaptive inertia weight
Inertia weight ω is a crucial parameter in PSO, reflecting a particle's ability to maintain previous velocity. It plays a key role in balancing the local and global search capabilities. A larger inertia weight ω enhances the global search capability, while a smaller ω improves the local search ability. By incorporating *fap* into the inertia weight update formula, agents can adjust their velocity accordingly. The formula is given as follows:

$$\omega^t = \omega_{\max} - (\omega_{\max} - \omega_{\min}) \cdot \frac{t}{T} \cdot fap^t \tag{43}$$

where $\omega_{\max}$ and $\omega_{\min}$ represent the maximum and minimum values of ω, respectively, while *t* is the iteration number, and *T* is the maximum number of iterations. That enables an adaptive linear transformation of the weight.

Adaptive learning factors
The learning factors in PSO adjust an agent's ability to learn from its own experience and the collective experience of the swarm, guiding itself toward the individual best position P_{best} or the global best position G_{best}. During the early stages of iteration, it's desirable for agents to have a stronger individual learning capability as c_1 to explore solutions. In the later stages, agents should possess enhanced social learning abilities as c_2 to exploit the optimal solution.

To achieve this, the learning factors are dynamically adjusted throughout the iterations, incorporating a sine factor χ to enable nonlinear variation as the iterations progress, as Fig. 4 shows. In the early stages of iteration, agents display stronger personal learning capabilities, while in the later stages, they show heightened social learning abilities. Additionally, the fitness-based adaptive factor *fap* is introduced to further enhance agent communication and collaboration, allowing them to improve self-improvement. This approach ensures a balanced exploration and exploitation process, improving the overall efficiency and accuracy of the search swarm. Learning factors, c_1 and c_2, are computed as:

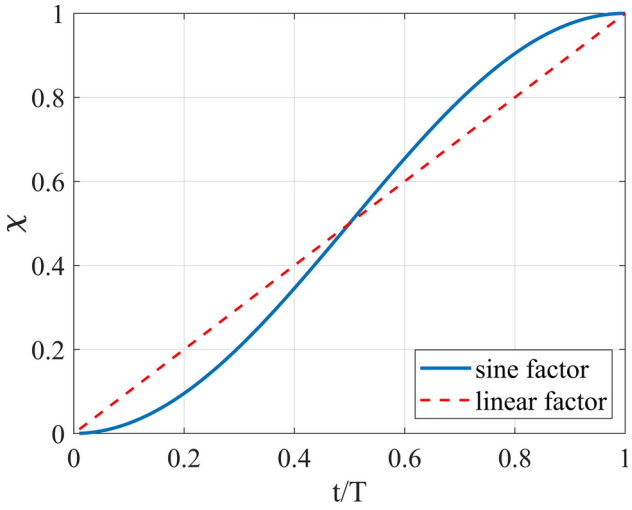


Fig. 4. Variation trends of learning factors during the iterative process.

$$\begin{cases} c_1^t & = c_{\max} - (c_{\max} - c_{\min}) \cdot \chi^t \cdot fap^t \\ c_2^t & = c_{\min} + (c_{\max} - c_{\min}) \cdot \chi^t \cdot fap^t \\ \chi^t & = \frac{1}{2} \sin\left(\frac{t}{T}\pi - \frac{\pi}{2}\right) + \frac{1}{2} \end{cases} \tag{44}$$

where $c_{\max}$ and $c_{\min}$ represent the maximum and minimum values of ω, respectively.

Acceleration factor overcoming local optima

Particles can easily get trapped in local optima, which remains a challenging problem. To overcome this, the paper introduces a dynamic acceleration factor *a*, enabling the particles to escape local optima. First, a criterion is needed to identify when a particle is stuck in a local optima. If the fitness values in three consecutive iterations are all greater than their average fitness values and the increment are both smaller than a constant ψ, we consider the particle to be trapped in a local optima. This can be expressed as:

$$\begin{cases} F\left(X^{t-1}\right) > F_{avge}^{t-1},\ F\left(X^{t}\right) > F_{avge}^{t},\ F\left(X^{t+1}\right) > F_{avge}^{t+1} \\ F\left(X^{t-1}\right) - F\left(X^{t}\right) < \psi_2,\ F\left(X^{t}\right) - F\left(X^{t+1}\right) < \psi_2 \end{cases} \tag{45}$$

where ψ_2 is a empirical parameter, used to reflect the exploration potential of a particle. The poor particles with lack potential can be filtered out by Eq. (45).

Then, we apply an acceleration factor *a* directed towards the global optima, helping the poor particle to break free from its current region. This can be expressed as:

$$a^t = r_4\left(G^t{}_{\text{best}} - X^t\right)\frac{t}{T} \cdot fap^t \tag{46}$$

where the *fap* is derived from Eq. (42). For UAV path planning in spherical coordinates, the complete update formula for a particle $P = (\mu_1, \mu_2, \ldots, \mu_n)$ is as follows:

$$\begin{cases} \Delta\mu_i^{t+1} = \omega^t \Delta\mu_i^t + c_1^t r_{1,i}\left(p^t{}_{\text{best},i} - \mu_i^t\right) + c_2^t r_{2,i}\left(g^t{}_{\text{best},i} - \mu_i^t\right) + a_i^t \\ \mu_i^{t+1} = \mu_i^t + \Delta\mu_i^{t+1}, \quad i = 1, 2, \ldots, n \end{cases} \tag{47}$$

where ω, c_1 and c_2 are the inertia weight, the individual learning factor, and the social learning factor, respectively, with adaptive propertie *fap*. As random numbers within [0,1], $r_{1,i}$ and $r_{2,i}$ introduce randomness to the individual and social learning abilities, respectively. $\Delta\mu_i$ represents the particle's velocity, while μ_i represents the position. *i* represents the *i*-th node and *t* is the iteration count. Note that, the value of *a* will be set to 0, if Eq. (45) is not met.

Require: Flight path nodes n, population size b, the maximum number of iterations T, maximum and minimum values of inertia weight ω_{max} & ω_{min}, maximum and minimum values of learning factors c_{max} & c_{min}, tent map parameter ψ_{tent}, evolutionary programming parameter ψ_{η};
Ensure: Optimized path scheme $Pathbest$, and corresponding fitness value $F_{fit}(Pathbest)$;
Generate random numbers γ based on the tent map with perturbation according to Eq.(39).
Initialize: Velocity and position of the particle $\Delta\mu$ & μ according to Eqs.(40) and (41), personal best fitness $F_{fit}(Pbest)$, global best fitness $F_{fit}(Gbest)$, evolutionary perturbation term $\eta = 1$, current generation $t = 1$;
while $t \leq T$ **do**
  for $i = 1 : b$ **do**
    Calculate the population average fitness F_{avge} and the adaptive fitness factor fap according to Eq.(42);
    Update the inertia weight ω, individual learning factor c_1, and social learning factor c_2 by Eqs.(43) and (44);
    if the particle is identified as a suboptimal one according to Ieq.(45) **then**
      Calculate the acceleration according to Eq.(46);
    else
      Set the acceleration value to 0;
    end if
    Update the velocity and position $\Delta\mu$ & μ according to Eq.(47);
    Update the evolutionary perturbation terms η_{gauss} & η_{cauchy}, and calculate the new positions μ_{gauss} & μ_{cauchy} by Gaussian mutation and Cauchy mutation according to Eqs.(48) and (49);
    if $F_{fit}(\mu) > F_{fit}(\mu_{gauss})$ **then**
      Update Gaussian position μ_{gauss} to μ;
    end if
    if $F_{fit}(\mu) > F_{fit}(\mu_{cauchy})$ **then**
      Update Cauchy position μ_{cauchy} to μ;
    end if
    if $F_{fit}(\mu) < F_{fit}(Pbest)$ **then**
      Update μ to the personal best position;
      if $F_{fit}(\mu) < F_{fit}(Gbest)$ **then**
        Update μ to the global best position;
      end if
    end if
  end for
  t = t+1;
end while
Update global best position $Gbest$ to $Pathbest$;
Return $Pathbest$ & $F_{fit}(Pathbest)$.

Algorithm 1. SAEPSO algorithm

Evolutionary selection of Particles

The evolutionary mechanism mimics the principles of natural evolution to solve optimization problems by selecting superior individuals, thereby guiding the population to evolve in a favorable direction. In this paper, Gaussian mutation operators $r_{gauss,i}$ and Cauchy mutation operators $r_{cauchy,i}$ are introduced to jointly generate new individuals to improve the whole swarm. The use of $r_{gauss,i}$, following a standard normal distribution, enhances the exploitation capability. The method of Gaussian mutation for generating new individuals is as follows:

$$\begin{cases} u^t_{gauss,i} = u^t_i + \eta^t_i r_{gauss,i} \\ \eta^t_i = \eta^{t-1}_i \exp\left[\left(\sqrt{2\sqrt{\psi}}\right)^{-1} N_0(0,1) + \left(\sqrt{2\psi}\right)^{-1} N_i(0,1)\right] \end{cases} \tag{48}$$

where $N_0(0,1)$ is a random number from a standard normal distribution with a mean of 0 and variance of 1, while $N_i(0,1)$ refers to generating a new random number from a standard normal distribution for each node. ψ is a constant set to 3 in this paper.

Under standard conditions, the Cauchy distribution exhibits a broader range compared to the normal distribution, as demonstrated in the probability density analysis in Fig. 5. Using the Cauchy mutation operators

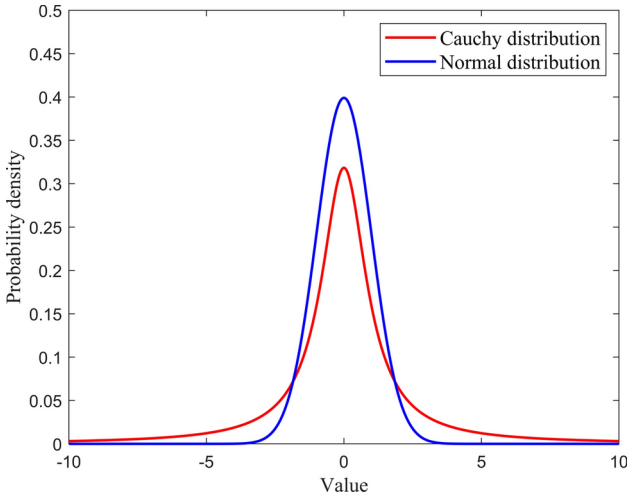


Fig. 5. Comparison of probability density functions.

enhances exploration capability. By replacing $r_{gauss,i}$ with $r_{cauchy,i}$ in Eq. (48), the new individual via Cauchy mutation can be obtained as:

$$u^t_{cauchy,i} = u^t_i + \eta^t_i r_{cauchy,i} \tag{49}$$

By comparing the fitness functions of the parent and offspring consisting of two new individuals generated by Gaussian and Cauchy mutations, the optimal individual is selected for the next iteration.

The algorithm pseudo code is shown in Algorithm 1. Once the indices and flight environment for the UAV are determined, the constant parameters required for SAEPSO are subsequently defined. Based on the improved tent map and opposition-based learning strategy, random initial solutions are generated. The adaptive factor is then utilized to optimize the inertia weight, and nonlinear variations are incorporated to improve the learning factors. Depending on the quality of the particles, a decision is made on whether to incorporate acceleration, thereby deriving optimized velocity and position update formulas, which are then employed to generate the next generation of particles. Finally, Gaussian mutation and Cauchy mutation are implemented to achieve evolutionary selection among the particles, leading to the identification of the best particle. The flowcart of SAEPSO is shown in Fig. 6.

Results

To obtain more widely reliable experimental conclusions, this section conducts a comparative experiment between SAEPSO, traditional PSO, and SPSO. Manh Duong Phungand Quang Phuc Ha confirmed that spherical coordinates can effectively improve solving UAV path planning problem[21]. To exclude the improvements brought by spherical coordinates, this paper applies them to four popular algorithms in recent years, as mentioned in section 4. Then SGWO, SWOA, SHHO, and SDBO are obtained and included in the following comparisons and experiments. The experimental environment is based on the Windows 11 [10.022631] operating system, with Matlab R2021a as the development language. The CPU is an AMD Ryzen 7 7840H with 16GB of RAM. The GPU is a NVIDIA GeForce RTX 4060 with 16GB of video memory.

Basic parameters of experiments

We used two different terrains as the base maps: one is a real digital elevation model map of an area in Christmas Island, and the other is a more complex simulated map, each set with three threat levels-low, medium, and high. On the first map, the starting point is set at (200, 100, 367), and the target position at (900, 800, 341). On the second map, the starting point is set at (50, 50, 179) and the target position at (400, 400, 236.5). Apart from these two points, another 10 flight path nodes were arranged. During the UAV flight, the maximum absolute values of the variation angles, $\Delta\varphi$ and $\Delta\xi$, were set to $\pi/2$, with a minimum flight altitude $z_{min} = 50$ and a maximum altitude of $z_{max} = 400$. The UAV has a mass of $M = 50$ and performs different tasks on the two maps, carrying loads with diameters of $U = 5$ or 10, respectively. In SAEPSO, the maximum inertia weight ω_{max} is set to 1.2, the minimum ω_{min} to 0.2, while learning factors c_{max} to 1.8, and c_{min} to 0.8. The total number of particles is $b = 200$, and the number of iterations is $t = 800$. Then, we obtained the path planning diagrams of different algorithms across six benchmarking scenarios, as shown in Figs. 7 and 8.

The second terrain area is smaller than the first one, compressing the feasible solution space and thus increasing the difficulty of the search. Figures 7 and 8 respectively display the 3D and top views of the paths generated by the intelligent algorithms. It can be observed that all algorithms are capable of producing feasible paths that meet the constraints, satisfying turn angle, climb angle variations, and flight altitude limitations, while

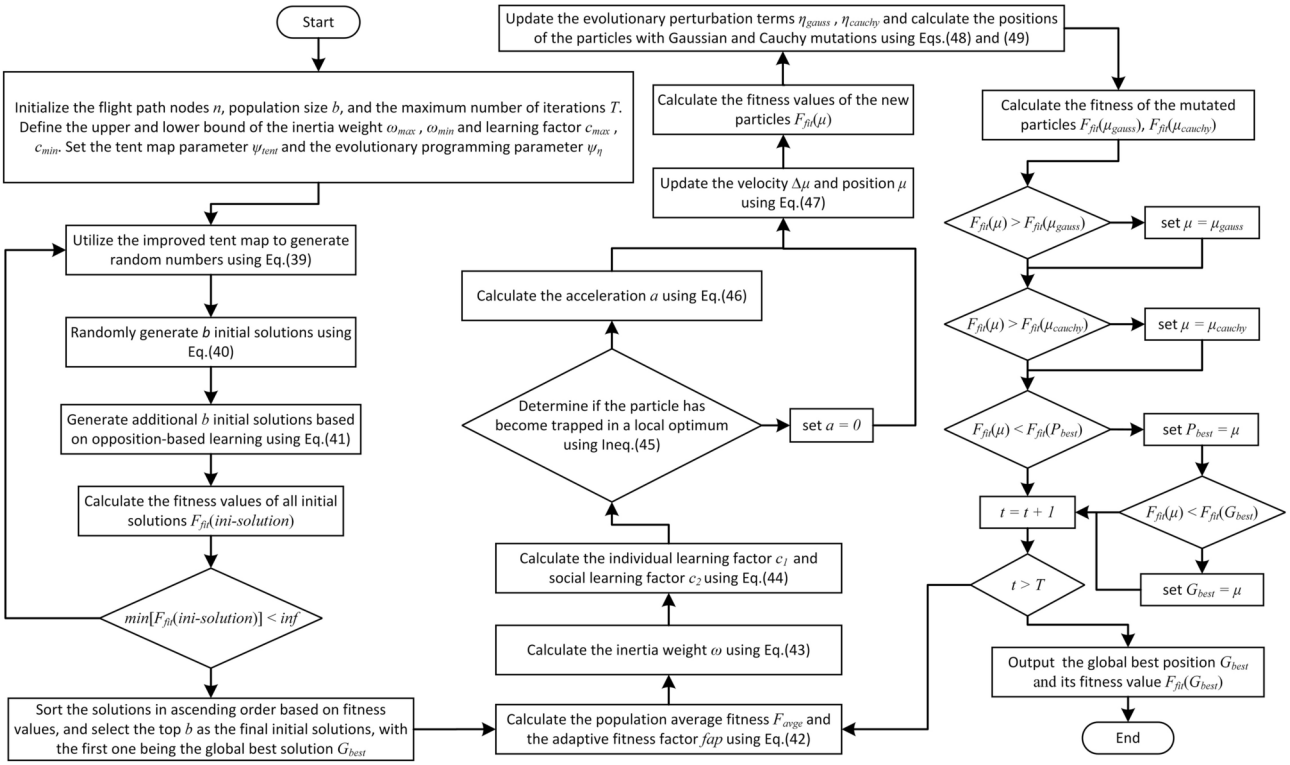


Fig. 6. The flowchart of SAEPSO.

successfully avoiding threat areas. However, due to the differences in scene complexity, the optimal solutions of the algorithms vary.

In the simpler scenarios 1 and 4, all algorithms except the traditional PSO, which tends to get trapped in local optima, managed to find relatively optimal flight paths. In the more complex scenario 2 and the more challenging scenario 3, the SPSO algorithm, using a spherical coordinate system, could directly generate solutions that comply with UAV maneuvering constraints, making it easier to find high-quality solutions. But its exploitation ability was weaker, limiting the optimization of the solution space. In scenarios 5 and 6, SDBO failed to find better solutions, mainly because of its boundary simulation strategy (female dung beetles and small dung beetles), which caused the algorithm to overly focus on local optima, neglecting other potentially better solutions in the global search space, thereby restricting its global exploration ability.

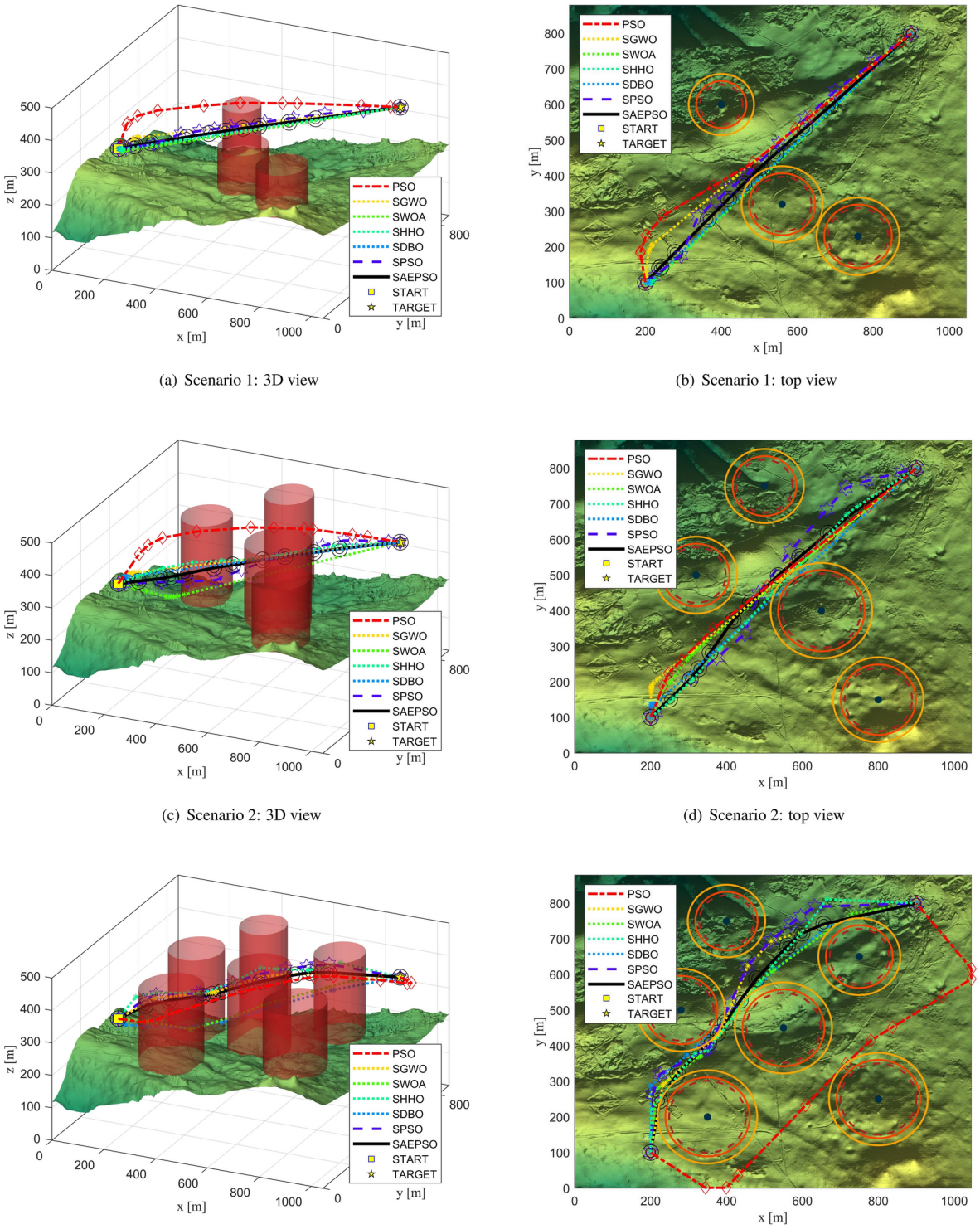
Through a visual analysis of all scenarios, the SAEPSO algorithm was able to approach the optimal solution, producing the smoothest and shortest paths. In contrast, SWOA, SHHO, SDBO, and SPSO could only converge to relatively good solutions, with significant fluctuations in solution quality. Comparatively, the PSO algorithm failed to find the highest quality solutions in most cases.

Analysis of initialization results

The initialization methods are divided into three types: the first is the traditional random initialization method based on the Cartesian coordinate system (CCS); the second is based on spherical coordinate system (SCS), which has the advantage of constraining the SPSO variables according to the UAV's dynamic constraints (such as turn angle and climb angle limits), thus reducing the search space and improving search efficiency; the third is the improved initialization method proposed in this paper, based on tent map and opposition-based learning.

To test the efficiency of generating valid solutions using the three initialization methods, we set up a swarm of 200 agents. During the initialization process, if at least one feasible solution is generated, the initialization is considered valid, and the algorithm proceeds to the next search stage. We conducted 20 experiments, recording the number of iterations required to generate the first valid solution during initialization. Subsequently, we verified the statistical results using a T-test, with a confidence level set at 0.05 (equivalent to 95% confidence). The symbol ✓ indicates that the mean of SAEPSO is statistically significantly better than the comparison algorithms, ✗ indicates the opposite result, while **O** indicates no significant difference, and – indicates not applicable data, as shown in Table 1.

Table 1 shows the maximum, minimum, average number of iterations, standard deviation, and paired sample T-test results for generating valid solutions. In terms of average iterations, the optimized initialization method proposed in this paper delivers the best performance across all scenarios. Through T-test analysis, the statistical results indicate that in scenarios 1, 2, 3, 5, and 6, SAEPSO's initialization method is significantly more efficient compared to PSO and SPSO. For the simpler scenario 4, both SPSO and SAEPSO initialization methods are able to quickly generate feasible solutions, and the T-test results indicate that the difference is not statistically



(a) Scenario 1: 3D view

(b) Scenario 1: top view

(c) Scenario 2: 3D view

(d) Scenario 2: top view

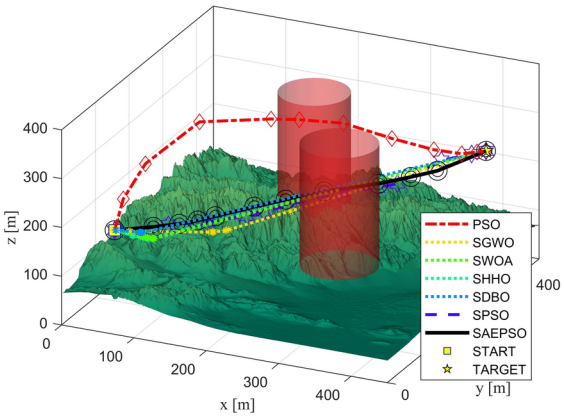
(e) Scenario 3: 3D view

(f) Scenario 3: top view

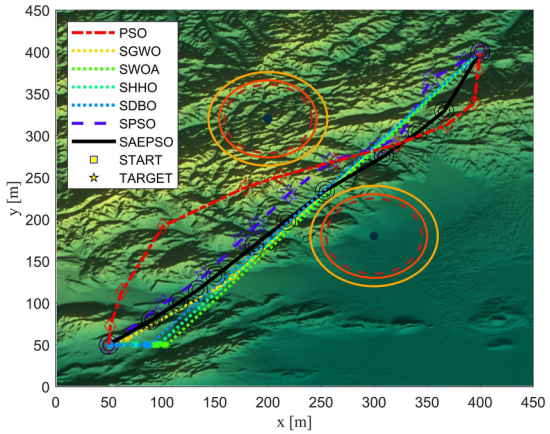
Fig. 7. Path planning solutions generated by SAEPSO and literature algorithms under three threat levels in the first terrain.

significant. However, the PSO initialization method, which does not account for UAV dynamic constraints, results in a high degree of randomness, generating numerous infeasible solutions and thus reducing efficiency.

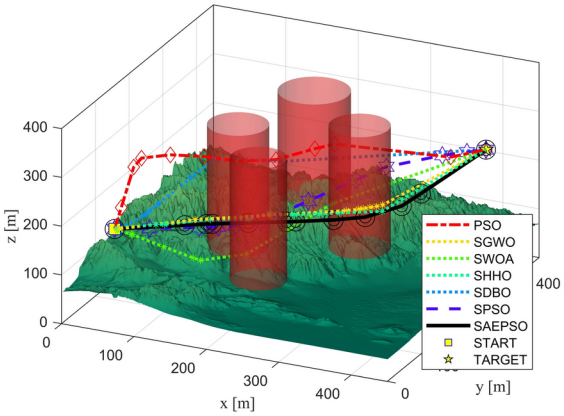
Scenarios 3 and 6 are both complex, with the main difference being the coverage area of the destruction zones near the start and end points. Since both SAEPSO and SPSO account for UAV dynamics, these algorithms tend to generate flight paths that directly aim from the start to the end point. In scenario 6, despite the higher complexity of the flight area, the algorithms can still quickly find feasible solutions. In contrast, in scenario 3,



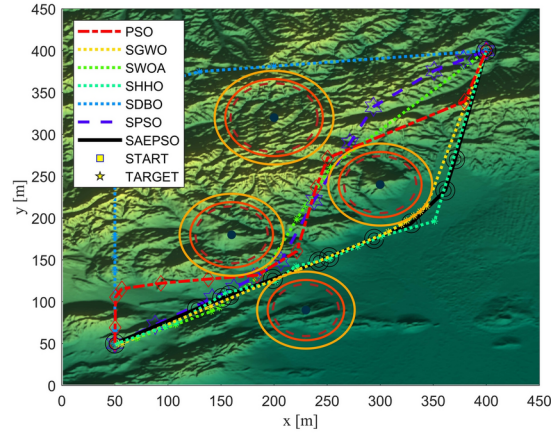
(a) Scenario 4: 3D view



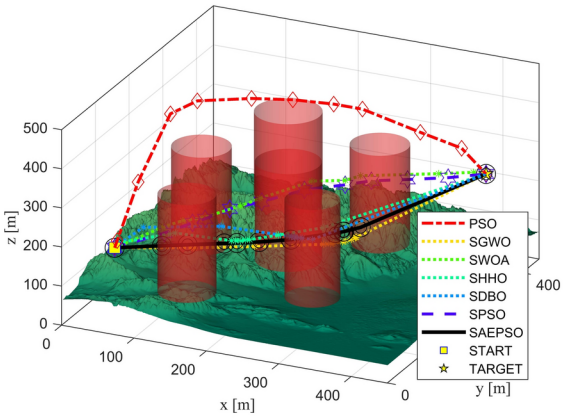
(b) Scenario 4: top view



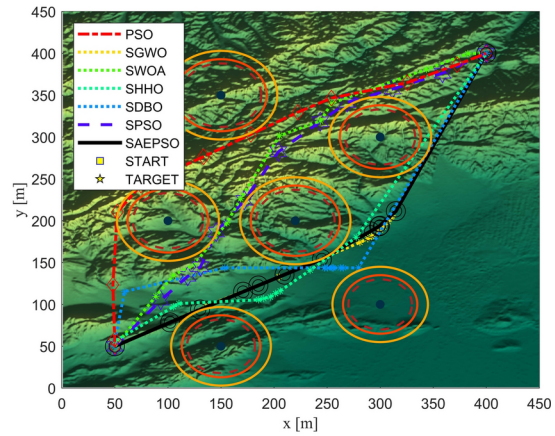
(c) Scenario 5: 3D view



(d) Scenario 5: top view



(e) Scenario 6: 3D view



(f) Scenario 6: top view

Fig. 8. Path planning solutions generated by SAEPSO and related algorithms under three threat levels in the second terrain.

the destruction zone is close to the start-end connection line and near the starting point, leading to poorer initialization performance. However, thanks to the improved tent map, the initialization distribution of SAEPSO is more uniform, allowing for a more effective global search for feasible solutions. Moreover, the opposition-based learning strategy further enhances the exploration ability of the solution space, making SAEPSO's initialization method significantly superior to that of SPSO.

Initialization [Algorithm]	Traditional method based on CCS [PSO]					Traditional method based on SCS [SGWO,SWOA,SHHO,SDBO,SPSO]					Optimization method in this paper [SAEPSO]				
Environment	Max.	Min.	Mean	Std	T-test	Max.	Min.	Mean	Std	T-test	Max.	Min.	Mean	Std	T-test
Scenario 1	28	1	10	8.78	✓	6	1	1.75	1.25	✓	1	1	**1**	0.00	–
Scenario 2	239	2	61.9	67.09	✓	9	1	3.05	2.39	✓	3	1	**1.35**	0.59	–
Scenario 3	402	48	194.25	117.37	✓	126	2	28.35	30.07	✓	31	1	**12.25**	10.12	–
Scenario 4	625	2	183.45	162.80	✓	5	1	1.75	1.07	**O**	3	1	**1.5**	0.69	–
Scenario 5	1882	22	617.2	591.06	✓	21	1	7.95	6.13	✓	5	1	**1.9**	1.12	–
Scenario 6	6555	7	1498.85	1637.48	✓	50	1	13.45	14.94	✓	8	1	**2.55**	2.19	–

Table 1. Efficiency analysis of the three initialization methods in generating valid results ($\alpha = 0.05$). Bold indicates the best result among all the algorithms or methods compared in the comparative experiments.

Initialization [Algorithm]	Traditional method based on CCS [PSO]					Traditional method based on SCS [SGWO,SWOA,SHHO,SDBO,SPSO]					[SAEPSO] Optimization method in this paper				
Environment	Max.	Min.	Mean	Std	T-test	Max.	Min.	Mean	Std	T-test	Max.	Min.	Mean	Std	T-test
Scenario 1	3	0	0.48	0.66	✓	18	4	9.58	2.70	✓	35	10	**23.62**	5.36	–
Scenario 2	2	0	0.31	0.54	✓	11	1	4.77	2.08	✓	21	5	**11.72**	3.50	–
Scenario 3	1	0	0.05	0.22	✓	2	0	0.27	0.47	✓	3	0	**0.56**	0.69	–
Scenario 4	1	0	0.06	0.24	✓	16	2	7.45	3.06	✓	30	7	**16.33**	4.51	–
Scenario 5	1	0	0.02	0.14	✓	7	0	2.16	1.22	✓	13	1	**5.01**	2.46	–
Scenario 6	1	0	0.01	0.10	✓	6	0	1.32	1.23	✓	8	0	**3.11**	1.95	–

Table 2. Quality assessment of the results generated by the three initialization methods ($\alpha = 0.05$). Bold indicates the best result among all the algorithms or methods compared in the comparative experiments.

Next, we will further verify the quality of the initialization results, which is measured by the number of valid agents. The greater the number of valid agents in the initialization results, the higher the search efficiency of the algorithm. Therefore, we conducted a single initialization experiment using 2000 agents and recorded the number of feasible agents generated in each initialization. Subsequently, using the same T-test method, we statistically analyzed the results of 100 experiments, as detailed in Table 2.

According to the statistical results in Table 2, the optimized initialization method proposed in this paper significantly outperforms the other two methods in terms of the quality of initial solutions across all scenarios. The PSO initialization method performs poorly in handling path planning problems with complex constraints, struggling to generate a sufficient number of feasible solutions. In contrast, SPSO, leveraging the characteristics of spherical vectors, is able to produce a larger number of feasible solutions, but its effectiveness remains limited in complex scenario 3 with special destruction zones. The optimized initialization method proposed in this paper demonstrates clear advantages, significantly improving the quality of initial solutions, indicating its importance in exploring the initial solution space.

Fitness analysis of the algorithms

Fitness is a key indicator for evaluating the performance of algorithms. We recorded the results of 10 runs for each algorithm across six scenarios, calculating the average, standard deviation, and paired sample T-test results, as detailed in Table 3.

From the results in Table 3, it can be seen that in the simpler scenario 4, SGWO, SWOA, SDBO, and SAEPSO all perform well, with no statistically significant differences. However, in scenarios 1, 2, 3, 5, and 6, SAEPSO shows significantly better performance than the others. Notably, in complex scenarios 2, 3, and 6, SAEPSO has the smallest standard deviation, reflecting its high stability and convergence when handling complex problems. In contrast, the SHHO algorithm shows instability, primarily because it tends to enter the exploitation phase too early when the escape energy of "prey" is low, leading to premature convergence and over-optimization issues. SGWO performs relatively well, managing to solve problems in complex scenarios effectively. Although SPSO performs ideally in simpler scenarios, it lacks the ability to deeply explore optimal solutions in more complex situations.

Figure 9, by displaying the changes in the best fitness values throughout the iteration process, provides a more detailed comparative analysis of the algorithm performance. On one hand, SAEPSO's initial solutions exhibit the best fitness in most cases, further validating the advantages and necessity of its initialization method. On the other hand, SAEPSO demonstrates exceptional exploitation capabilities in all scenarios. This is not only due to the introduction of UAV state constraints in particle variables, but also through the optimization of inertia weights and learning factors, fully utilizing swarm information. SAEPSO adaptively adjusts the balance between exploration and exploitation based on each particle's condition, introduces an acceleration factor to escape local optima, and incorporates an evolution factor to enhance population diversity, thus improving global exploration ability.

Environment	Statistics	PSO	SGWO	SWOA	SHHO	SDBO	SPSO	SAEPSO
Scenario 1	Mean	45296	26135	29947	30256	27974	24909	**22646**
	Std	9661.0	2289.7	1305.6	801.7	**501.6**	1296.6	537.7
	Time	**10.953**	11.125	11.036	27.350	11.463	19.888	53.794
	T-test	✓	✓	✓	✓	✓	✓	–
Scenario 2	Mean	48261	26662	39409	40745	39115	27416	**22634**
	Std	9112.1	2073.4	5945.9	5435.3	5643.2	3015.9	**326.3**
	Time	14.290	**13.727**	13.818	35.209	13.897	22.634	63.078
	T-test	✓	✓	✓	✓	✓	✓	–
Scenario 3	Mean	54770	40923	51765	64996	52948	37181	**30514**
	Std	6351.6	6198.9	11617.9	5797.8	9281.4	2202.4	**1183.5**
	Time	23.467	21.880	21.146	51.052	**21.012**	28.761	79.458
	T-test	✓	✓	✓	✓	✓	✓	–
Scenario 4	Mean	48211	23081	21701	25332	20845	25088	**20269**
	Std	9364.7	6611.4	4625.2	**2430.5**	3552.6	4202.7	2909.2
	Time	6.583	5.723	**5.569**	13.873	6.087	13.813	39.683
	T-test	✓	**O**	**O**	✓	**O**	✓	–
Scenario 5	Mean	52278	24932	33609	33090	37674	26515	**21853**
	Std	8942.5	**3881.0**	4591.0	5023.0	3901.6	5844.4	4216.2
	Time	19.222	8.335	**8.249**	21.432	8.559	16.692	47.248
	T-test	✓	✓	✓	✓	✓	✓	–
Scenario 6	Mean	47222	22187	33916	30721	32981	26671	**18513**
	Std	12026.0	3965.8	5903.8	4510.6	5492.0	7475.6	**759.0**
	Time	28.513	10.162	**10.143**	27.216	10.600	18.374	54.365
	T-test	✓	✓	✓	✓	✓	✓	–
Consumption	Space	1.56×10^5	6.19×10^4	6.15×10^4	$\mathbf{5.71}\times10^4$	1.57×10^5	1.59×10^5	397×10^5

Table 3. Statistical analysis of the optimal fitness values and computational complexity of SAEPSO and literature algorithms ($\alpha = 0.05$; $n = 10$). Bold indicates the best result among all the algorithms or methods compared in the comparative experiments.

Computational complexity analysis of SAEPSO

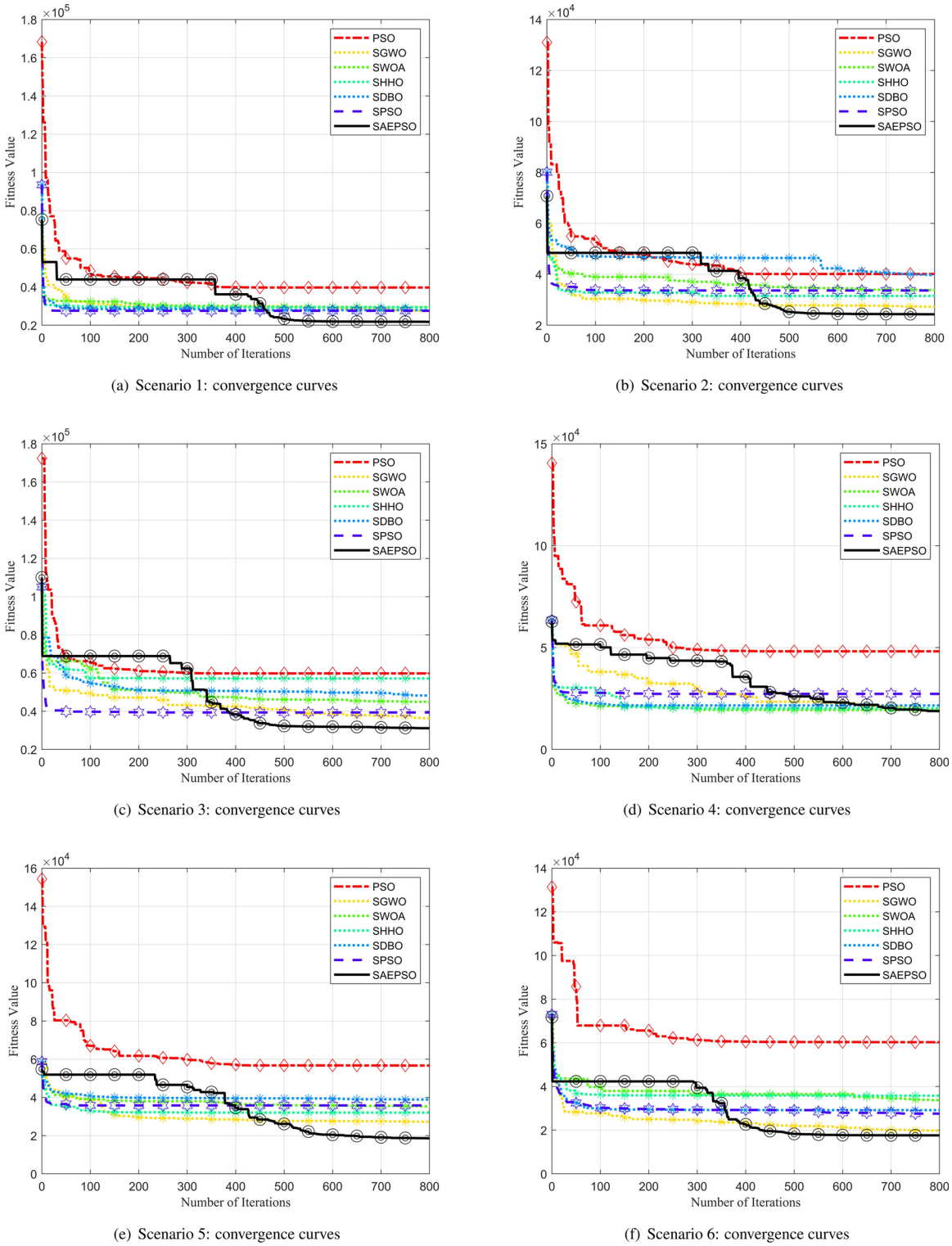
This section provides an analysis of the proposed algorithm from both time-complexity and space-complexity perspectives. Additionally, experiments were conducted in six scenarios with 10 flight nodes , and the time taken to complete SAEPSO execution, together with the storage resources consumed (with double data type), was recorded, as presented in Table 3. The time unit is seconds, and the space unit is bytes.

The primary time consumption of SAEPSO is analyzed. During the initialization phase, the improved tent map generates chaotic random numbers for each dimension of the flight nodes for every particle, which is an iterative process. The time consumed depends on the number of particles (b), nodes (n), and dimensions (τ). When calculating the fitness values of the initial solutions, spherical coordinates are used, which involves complex trigonometric calculations. During the particle update process, the core of SAEPSO lies in the iterative computation of velocity and position. Additionally, the adaptive factor requires the calculation of the average and best fitness values in each iteration, while the learning factors, which vary sinusoidally, need to be assigned to each particle. Finally, during the evolution of each particle, iterative calculations are performed involving exponential functions. Therefore, the time complexity of SAEPSO is primarily determined by the iterative computations.

The primary consumption of space resources by SAEPSO is analyzed. During the initialization phase, all the random numbers generated by tent map should be stored for later use to generate the initial solutions ($b \times n \times \tau$). Opposition-based learning additionally requires double the storage space for initial solutions, including particle positions ($2 \times b \times n \times \tau$) and fitness values ($2 \times b$). During the particles update process, it is necessary to store their current positions ($b \times n \times \tau$), velocities ($b \times n \times \tau$), fitness values (b), as well as the personal best positions ($b \times n \times \tau$) and the corresponding fitness values (b). In the particle evolution process, a temporary variable is needed to store the position of evolving particles ($b \times n \times \tau$), while two evolutionary parameters also occupy considerable storage space ($2 \times b \times n$). The data is stored in double format and the actual memory required is shown in Table 3.

Scalability analysis of the algorithms

To analyze the scalability of each algorithm, we conducted comparative experiments by setting 5 and 15 flight nodes, respectively. Based on Table 4, the traditional PSO struggles to accommodate scenarios with an increased number of path nodes, resulting in its inability to find effective solutions when refining the path ($n = 15$). Although SWOA can find feasible solutions, its convergence proves to be highly unstable, particularly in scenarios with multiple path nodes, as evidenced by the variance. SHHO demonstrates good convergence stability in the



(a) Scenario 1: convergence curves

(b) Scenario 2: convergence curves

(c) Scenario 3: convergence curves

(d) Scenario 4: convergence curves

(e) Scenario 5: convergence curves

(f) Scenario 6: convergence curves

Fig. 9. Comparison of the best fitness values between SAEPSO and literature algorithms based on convergence curves.

case of multiple path nodes, yet, as indicated by the average fitness, it struggles to escape local optima. The SAEPSO achieves the lowest average fitness and exhibits excellent convergence stability. Under T-test analysis, it significantly outperforms most of the compared algorithms, with the exception of SGWO, which is capable of finding a near-optimal solution.

Note that during the process of generating the initial solution, we employed 200 particles with a maximum of 5000 iterations. In 10 independent replications of the experiment, if the number of occurrences where a feasible

Environment	Statistics	PSO	SGWO	SWOA	SHHO	SDBO	SPSO	SAEPSO
Scenario 1 (n=5)	Mean	23949	21767	24731	24470	25209	22689	**21649**
	Std	2824.9	624.7	1292.0	1035.4	321.2	1787.0	**82.7**
	T-test	✓	**O**	✓	✓	✓	**O**	–
Scenario 1 (n=15)	Mean	74375	34661	42587	26990	28437	48245	**26695**
	Std	8397.2	5126.8	12294.8	**509.7**	4385.0	6847.0	2373.9
	T-test	✓	✓	✓	**O**	**O**	✓	–
Scenario 2 (n=5)	Mean	32661	23302	29973	32817	29932	24934	**22062**
	Std	14141.2	1507.3	8475.5	9892.2	4597.8	3715.6	**359.7**
	T-test	✓	✓	✓	✓	✓	✓	–
Scenario 2 (n=15)	Mean	–	43278	47572	31901	30488	45368	**27089**
	Std	–	11057.2	13475.2	2841.7	2617.6	6738.0	**2379.9**
	T-test	–	✓	✓	✓	✓	✓	–
Scenario 3 (n=5)	Mean	38462	32094	34746	39000	36761	30730	**27442**
	Std	5768.7	4294.0	3528.5	4687.6	6802.0	3967.2	**288.5**
	T-test	✓	✓	✓	✓	✓	✓	–
Scenario 3 (n=15)	Mean	–	46601	78334	59017	63996	50503	**42847**
	Std	–	4226.5	11573.7	**2709.0**	6443.9	6330.2	2797.3
	T-test	–	**O**	✓	✓	✓	✓	–
Scenario 4 (n=5)	Mean	29321	16889	17089	20850	18351	18126	**14823**
	Std	15650.6	7069.5	2606.0	4023.4	**1348.7**	6561.1	2440.1
	T-test	✓	**O**	✓	✓	✓	**O**	–
Scenario 4 (n=15)	Mean	–	30433	28178	23007	27265	47620	**19290**
	Std	–	4318.6	6260.6	**96.9**	5736.6	12941.5	4010.8
	T-test	–	✓	✓	✓	✓	✓	–
Scenario 5 (n=5)	Mean	25577	20708	21461	21252	24172	21428	**19431**
	Std	3495.8	1856.1	2935.8	2442.8	2498.3	2547.0	**1532.5**
	T-test	✓	✓	✓	**O**	✓	✓	–
Scenario 5 (n=15)	Mean	–	35196	45489	36526	43713	38366	**30280**
	Std	–	6187.9	10788.5	**3771.5**	5359.0	9047.1	5332.3
	T-test	–	**O**	✓	✓	✓	✓	–
Scenario 6 (n=5)	Mean	21759	17590	18688	21294	21670	18858	**16730**
	Std	5875.1	2763.2	3078.8	2712.2	2987.1	2756.3	**222.4**
	T-test	✓	**O**	**O**	✓	✓	✓	–
Scenario 6 (n=15)	Mean	–	30983	42347	34712	45608	31514	**24914**
	Std	–	3938.7	9730.4	3925.4	4098.2	5346.9	**3666.6**
	T-test	–	✓	✓	✓	✓	✓	–

Table 4. Scalability analysis of SAEPSO and lerature algorithms ($\alpha = 0.05; n = 5, 15$). Bold indicates the best result among all the algorithms or methods compared in the comparative experiments.

solution was not generated exceeds half, the algorithm is considered ineffective for this scenario, denoted by – in Table.

Ablation analysis of SAEPSO

In order to better validate the contribution of each module, we conducted ablation experiments in the six scenarios mentioned previously. We recorded the average fitness values of the algorithm with different modules incorporated and performed the following analysis.

The results are shown in Table 5. Based on the results from relatively simple scenarios 1, 2, and 4, it is evident that the improvements brought about by tent map (TM) and opposition-based learning (OP) modules are relatively smaller when compared to the more complex scenarios 3, 5, and 6. This suggests that improvements in initialization enhance the optimization capability, particularly in complex scenarios. In scenario 1, the introduction of the adaptive learning factor (ALF) led to a decline in algorithm performance. This is because the nonlinear variation of the learning factor does not suit simple scenarios well. The results in scenario 4 also showed no significant effect. However, in scenarios 5 and 6, we observed that the introduction of the nonlinear learning factor resulted in substantial improvements, as it provides stronger individual learning capabilities during the early iterations, thereby improving the ability to escape local optima. The adaptive inertia weight (AIW) and evolution modules contributed more significantly to the algorithm's performance, showing consistent improvements across all scenarios. The introduction of the acceleration factor (AF) led to a modest improvement across most scenarios. However, it resulted in a significant improvement in the more

Environment	Module	Mean	Environment	Module	Mean
Scenario 1	SPSO[21]	24909	Scenario 4	SPSO[21]	25088
	SPSO+TM	24507		SPSO+TM	24324
	SPSO+TM+OP	24185		SPSO+TM+OP	23025
	SPSO+TM+OP+AIW	23879		SPSO+TM+OP+AIW	22157
	SPSO+TM+OP+AIW+ALF	23971		SPSO+TM+OP+AIW+ALF	21930
	SPSO+TM+OP+AIW+ALF+AF	23747		SPSO+TM+OP+AIW+ALF+AF	21434
	SAEPSO	**22646**		SAEPSO	**20269**
Scenario 2	SPSO[21]	27416	Scenario 5	SPSO[21]	26515
	SPSO+TM	27042		SPSO+TM	25876
	SPSO+TM+OP	26947		SPSO+TM+OP	25005
	SPSO+TM+OP+AIW	24712		SPSO+TM+OP+AIW	24123
	SPSO+TM+OP+AIW+ALF	23783		SPSO+TM+OP+AIW+ALF	23085
	SPSO+TM+OP+AIW+ALF+AF	23377		SPSO+TM+OP+AIW+ALF+AF	22853
	SAEPSO	**22634**		SAEPSO	**21853**
Scenario 3	SPSO[21]	37181	Scenario 6	SPSO[21]	26671
	SPSO+TM	35527		SPSO+TM	24074
	SPSO+TM+OP	32197		SPSO+TM+OP	23231
	SPSO+TM+OP+AIW	31266		SPSO+TM+OP+AIW	23043
	SPSO+TM+OP+AIW+ALF	31099		SPSO+TM+OP+AIW+ALF	21279
	SPSO+TM+OP+AIW+ALF+AF	30912		SPSO+TM+OP+AIW+ALF+AF	20517
	SAEPSO	**30514**		SAEPSO	**18513**

Table 5. Ablation analysis for the improved modules of SAEPSO based on optimal fitness values ($n = 10$). Bold indicates the best result among all the algorithms or methods compared in the comparative experiments.

complex scenario 6, confirming its effectiveness in overcoming the issue of getting trapped in local optima. By progressively incorporating each module, the fitness results improved overall, and SAEPSO achieved the best results across various scenarios, validating the contributions of each module to the optimization capability.

Discussion

Through the comparative analysis of the initialization performance in Tables 1 and 2, we can clearly observe that the tent map with perturbations combined with the opposition-based learning shows significant advantages during initialization. Whether in terms of efficiency or quality, the proposed optimization method significantly outperforms others under the T-test. This is primarily due to the uniform distribution of initialized particles, and the introduction of random perturbations effectively avoids falling into cycles. Meanwhile, opposition-based learning further explores uncovered areas. However, this method requires recursively generating random numbers, leading to increased time complexity.

The analysis of Table 3 and Fig. 9 indicates that the proposed SAEPSO algorithm performs excellently across all scenarios, consistently finding paths with the smallest average fitness values. Apart from simpler cases like Scenario 4, all scenarios passed the T-test, effectively demonstrating the statistical superiority of the SAEPSO. In complex scenarios with numerous obstacles and threats, SAEPSO exhibits the smallest average standard deviation, indicating not only the generation of optimal paths but also excellent convergence stability. Table 4 demonstrates that the algorithm proposed in this paper exhibits the best path node scalability. This outstanding performance is attributed not only to improved initialization but also to the nonlinear convergence factor, which effectively balances the algorithm's exploitation and exploration capabilities. Additionally, the fitness-based adaptive factor enables particles to learn from individual best and global best particles, as well as the population's average level, adjusting their learning capabilities accordingly. Combined with the adaptive acceleration factor and integrated evolutionary programming algorithm, SAEPSO further enhances its ability to handle local optima and optimization challenges. The results of the ablation experiment, as shown in Table 5, further confirm the effectiveness of the improved modules.

However, it is worth noting that SAEPSO's relatively slower convergence at the early stages is due to the nonlinear adjustment of the learning factor-gradually increasing through a sine function. This enhances global exploration in the early iterations and boosts local exploitation in the later stages. As the number of demands increases, it becomes difficult to measure the relative importance of different optimization objectives. A single objective function may become overly complex or even inapplicable. In such cases, multi-objective optimization should be considered to accomplish the task.

Conclusion

This paper proposes a novel method for addressing the issue of UAV path planning, based on constraints from the flying environment and the drone's performance. It comprehensively considers the smoothness, safety, energy consumption, and timing of the flight path, and employs a weighted design to establish a new singular optimization objective. The initialization method, parameter updating strategy, and capability to escape local

optima and expand the solution space have been improved. Comparative experiments conducted across six benchmark scenarios of varying complexity reveal that SAEPSO consistently achieves the best initialization results and optimal paths in all cases. An analysis of sample standard deviations indicates that SAEPSO demonstrates excellent convergence stability when handling complex scenarios. Moreover, T-test results further substantiate that SAEPSO significantly outperforms other algorithms in statistical terms and demonstrates remarkable scalability. However, SAEPSO also has certain limitations, as its computational process is more complex, consuming more time and storage resources. Our future work will focus on improving the algorithm's convergence speed while considering the simulation of a more realistic UAV flight environment. Additionally, we will explore the algorithm's applicability in other fields to broaden its range of applications.

Data availability

The data presented in this study are available on request from corresponding author.

Received: 27 November 2024; Accepted: 7 January 2025
Published online: 16 January 2025

References

1. Chang, X., Jin, C. & Cheng, Y. Dynamics and advanced active disturbance rejection control of tethered UAV. *Appl. Math. Model.* **135**, 640–665 (2024).
2. Zhao, B., Huo, M., Li, Z., Yu, Z. & Qi, N. Clustering-based hyper-heuristic algorithm for multi-region coverage path planning of heterogeneous UAVs. *Neurocomputing* **610**, 128528 (2024).
3. Xie, C., Li, S., Qin, X., Fu, S. & Zhang, X. Multiple elite strategy enhanced rime algorithm for 3D UAV path planning. *Sci. Rep.* **14**, 21734 (2024).
4. Ha, I.-K. Improved a-star search algorithm for probabilistic air pollution detection using UAVs. *Sensors* **24**, 1141 (2024).
5. Suanpang, P. & Jamjuntr, P. Optimizing autonomous UAV navigation with d* algorithm for sustainable development. *Sustainability* **16**, 7867 (2024).
6. Li, X., Wang, L., Wang, H., Tao, L. & Wang, X. *A Warm-started Trajectory Planner for Fixed-wing Unmanned Aerial Vehicle Formation*. vol. 122, pp. 200–219 (Elsevier, 2023).
7. Zhang, X. & Zhang, X. UAV path planning based on hybrid differential evolution with fireworks algorithm. In *International Conference on Sensing and Imaging* 354–364 (Springer, 2022).
8. Chen, H., Wang, H. & Jiang, L. Path planning of uav based on cultural algorithm in dynamic environments. In *2016 6th International Conference on Electronics Information and Emergency Communication (ICEIEC)* 130–134 (IEEE, 2016).
9. de Moura Souza, G. & Toledo, C. F. M. Genetic algorithm applied in uav's path planning. In *2020 IEEE Congress on Evolutionary Computation (CEC)* 1–8 (IEEE, 2020).
10. Xibin, W., Guorong, Z. & Kunhu, K. Fastslam algorithm based on simulated annealing for UAV. *Proced. Eng.* **29**, 3007–3011 (2012).
11. Tamura, K. & Yasuda, K. The spiral optimization algorithm: Convergence conditions and settings. *IEEE Trans. Syst. Man Cybern. Syst.* **50**, 360–375 (2017).
12. Sardari, F. & Moghaddam, M. E. An object tracking method using modified galaxy-based search algorithm. *Swarm Evol. Comput.* **30**, 27–38 (2016).
13. Hu, G., Huang, F., Seyyedabbasi, A. & Wei, G. Enhanced multi-strategy bottlenose dolphin optimizer for UAVs path planning. *Appl. Math. Model.* **130**, 243–271 (2024).
14. Hu, G., Huang, F., Seyyedabbasi, A. & Wei, G. Enhanced multi-strategy bottlenose dolphin optimizer for UAVs path planning. *Appl. Math. Model.* **130**, 243–271 (2024).
15. Kennedy, J. & Eberhart, R. Particle swarm optimization. In *Proceedings of ICNN'95-International Conference on Neural Networks*, vol. 4, pp. 1942–1948 (IEEE, 1995).
16. Yao, J. et al. Research on hybrid strategy particle swarm optimization algorithm and its applications. *Sci. Rep.* **14**, 24928 (2024).
17. Xia, X. et al. An expanded particle swarm optimization based on multi-exemplar and forgetting ability. *Inf. Sci.* **508**, 105–120 (2020).
18. Aslan, M. F., Durdu, A. & Sabanci, K. Goal distance-based UAV path planning approach, path optimization and learning-based path estimation: Gdrrt*, pso-gdrrt* and bilstm-pso-gdrrt. *Appl. Soft Comput.* **137**, 110156 (2023).
19. Xiang, H., Liu, X., Song, X. & Zhou, W. UAV path planning based on enhanced pso-ga. In *CAAI International Conference on Artificial Intelligence* 271–282 (Springer, 2023).
20. Sonny, A., Yeduri, S. R. & Cenkeramaddi, L. R. Autonomous uav path planning using modified pso for UAV-assisted wireless networks. *IEEE Access* (2023).
21. Phung, M. D. & Ha, Q. P. Safety-enhanced UAV path planning with spherical vector-based particle swarm optimization. *Appl. Soft Comput.* **107**, 107376 (2021).
22. Shi, Y. & Eberhart, R. C. Fuzzy adaptive particle swarm optimization. In *Proceedings of the 2001 congress on evolutionary computation (IEEE Cat. No. 01TH8546)*, vol. 1, pp. 101–106 (IEEE, 2001).
23. Feng, C., Cong, S. & Feng, X. A new adaptive inertia weight strategy in particle swarm optimization. In *2007 IEEE Congress on Evolutionary Computation* 4186–4190 (IEEE, 2007).
24. Tanweer, M. R., Suresh, S. & Sundararajan, N. Self regulating particle swarm optimization algorithm. *Inf. Sci.* **294**, 182–202 (2015).
25. Borowska, B. Nonlinear inertia weight in particle swarm optimization. In *2017 12th International Scientific and Technical Conference on Computer Sciences and Information Technologies (CSIT)*, vol. 1, pp. 296–299 (IEEE, 2017).
26. Zhao, F., Ji, F., Xu, T., Zhu, N. et al. Hierarchical parallel search with automatic parameter configuration for particle swarm optimization. *Appl. Soft Comput.* 111126 (2023).
27. Di Cesare, N., Chamoret, D. & Domaszewski, M. A new hybrid PSO algorithm based on a stochastic Markov chain model. *Adv. Eng. Softw.* **90**, 127–137 (2015).
28. Wei, B. et al. Multiple adaptive strategies based particle swarm optimization algorithm. *Swarm Evol. Comput.* **57**, 100731 (2020).
29. Zhang, L., Zhang, Y. & Li, Y. Mobile robot path planning based on improved localized particle swarm optimization. *IEEE Sens. J.* **21**, 6962–6972 (2020).
30. Liu, S. et al. Brownian motion based multi-objective particle swarm optimization methodology and application in binary classification. *Appl. Soft Comput.* **157**, 111539 (2024).
31. Fu, Y., Ding, M. & Zhou, C. Phase angle-encoded and quantum-behaved particle swarm optimization applied to three-dimensional route planning for uav. *IEEE Trans. Syst. Man Cybern. Part A Syst. Humans* **42**, 511–526 (2011).
32. Sun, J., Fang, W., Wu, X., Palade, V. & Xu, W. Quantum-behaved particle swarm optimization: analysis of individual particle behavior and parameter selection. *Evol. Comput.* **20**, 349–393 (2012).
33. Phung, M. D. & Ha, Q. P. Motion-encoded particle swarm optimization for moving target search using UAVs. *Appl. Soft Comput.* **97**, 106705 (2020).

34. Enireddy, V. & Kumar, R. K. Improved cuckoo search with particle swarm optimization for classification of compressed images. *Sadhana* **40**, 2271–2285 (2015).
35. Chen, X., Tianfield, H. & Du, W. Bee-foraging learning particle swarm optimization. *Appl. Soft Comput.* **102**, 107134 (2021).
36. Han, B., Li, B. & Qin, C. A novel hybrid particle swarm optimization with marine predators. *Swarm Evol. Comput.* **83**, 101375 (2023).
37. Divasón, J., Pernia-Espinoza, A. & Martinez-de Pison, F. J. Hyb-parsimony: A hybrid approach combining particle swarm optimization and genetic algorithms to find parsimonious models in high-dimensional datasets. *Neurocomputing* **560**, 126840 (2023).
38. Liu, Y., Zhu, X., Zhang, X.-Y., Xiao, J. & Yu, X. Rgg-pso+: Random geometric graphs based particle swarm optimization method for UAV path planning. *Int. J. Comput. Intell. Syst.* **17**, 127 (2024).
39. Mirjalili, S., Mirjalili, S. M. & Lewis, A. Grey wolf optimizer. *Adv. Eng. Softw.* **69**, 46–61 (2014).
40. Mirjalili, S. & Lewis, A. The whale optimization algorithm. *Adv. Eng. Softw.* **95**, 51–67 (2016).
41. Heidari, A. A. et al. Harris hawks optimization: Algorithm and applications. *Futur. Gener. Comput. Syst.* **97**, 849–872 (2019).
42. Xue, J. & Shen, B. Dung beetle optimizer: A new meta-heuristic algorithm for global optimization. *J. Supercomput.* **79**, 7305–7336 (2023).
43. Umar, T., Nadeem, M. & Anwer, F. A new modified skew tent map and its application in pseudo-random number generator. *Comput. Stand. Interfaces* **89**, 103826 (2024).
44. Jiao, C., Yu, K. & Zhou, Q. An opposition-based learning adaptive chaotic particle swarm optimization algorithm. *J. Bion. Eng.* 1–22 (2024).

Author contributions

Yanfei Liu conceptualized the flowchart and participated in writing and revising the manuscript. Hao Zhang conceived and conducted the experiments, and participated in manuscript revisions. Hao Zheng was responsible for data analysis. Qi Li reviewed the paper and provided revision suggestions. Qi Tian was responsible for proofreading the manuscript. All authors reviewed the manuscript.

Declarations

Competing interests

The authors declare no competing interests.

Additional information

Correspondence and requests for materials should be addressed to H.Z.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.