Transformer-Augmented MCTS for Aircraft Landing Problem

Jie Hu , Shuai Zhang , Xiaorong Feng * , Xinglong Wang

Posted Date: 11 March 2026

doi: 10.20944/preprints202603.0874.v1

Keywords: aircraft landing problem; time window constraints; dynamic programming; deep reinforcement learning; Monte Carlo tree search

 doi:10.20944/preprints202603.0874.v1

Article

Transformer-Augmented MCTS for Aircraft Landing Problem

Jie Hu [1,2], Shuai Zhang [3] , Xiaorong Feng [3,*] and Xinglong Wang [3]

[1] The 28th Research Institute of China Electronics Technology Group Corporation, Nanjing 210007, China
[2] State Key Laboratory of Air Traffic Management System, Nanjing 210007, China
[3] School of Air Traffic Management, Civil Aviation University of China, Tianjing 300300, China
* Correspondence: xrfeng@cauc.edu.cn

Abstract

The Aircraft Landing Problem (ALP) poses significant challenges for traditional Monte Carlo Tree Search (MCTS) due to its vast search space and reliance on inefficient random simulations. To overcome these limitations, this paper proposes a novel **Transformer-Augmented Monte Carlo Tree Search (TMCTS)** algorithm. Our approach integrates a reinforcement learning framework that incorporates key operational constraints, including wake turbulence separation and time windows, and employs a cost function aimed at minimizing both delay time and fuel consumption. A core innovation is the replacement of the conventional random simulation phase in MCTS with a **Transformer-based value predictor**. This leverages the Transformer's superior capability in sequence modeling and capturing global dependencies among flights, thereby dramatically accelerating search convergence. Specifically, we design a **two-head Transformer network** (comprising policy and value heads) to provide informed prior knowledge, which effectively guides the selection and expansion steps of the MCTS tree. The model is trained within an **Actor-Critic framework**, utilizing behavior cloning for pre-training followed by reinforcement learning for fine-tuning. Experimental evaluations on the standard OR-Library benchmark demonstrate that our TMCTS method significantly **reduces scheduling deviation** compared to state-of-the-art baselines (including FCFS, DPALO+GA, DPALO+PSO, and CPLEX). Moreover, it achieves a **93.7% reduction in computation time** relative to the CPLEX method, highlighting its superior efficiency and practical applicability for real-time scheduling.

Keywords: aircraft landing problem; time window constraints; dynamic programming; deep reinforcement learning; Monte Carlo tree search

1. Introduction

In 2024, China's civil aviation industry achieved a total transport turnover of 1,485.17 billion ton-kilometers, 730 million passenger trips, and 8.9816 million tons of cargo and mail volume, representing year-on-year increases of 25.0%, 17.9%, and 22.1%, respectively. These figures reflect steady growth in the industry's transportation operations [1]. During the same year, Chinese passenger airlines operated a cumulative total of 5.0031 million flights, of which 4.3577 million were punctual, resulting in an average flight punctuality rate of 87.1%. This decline in punctuality underscores the pressing need to enhance runway throughput and scheduling efficiency, as runway resource constraints are identified as a primary cause of flight delays at hub airports [2].

To mitigate the risks posed by wake turbulence during approach and landing phases, aircraft must maintain a minimum separation distance from preceding flights. The International Civil Aviation Organization (ICAO) categorizes aircraft into three classes based on maximum takeoff weight (MTOW):

- Light (L): MTOW less than 7 tons.
- Medium (M): MTOW between 7 and 136 tons.

 doi:10.20944/preprints202603.0874.v1

- Heavy (H): MTOW greater than 136 tons.

The wake vortices generated by a leading aircraft can significantly affect the takeoff and landing performance of following aircraft. Therefore, during the approach phase, the required separation between aircraft must satisfy both the wake turbulence separation minima and the standard aircraft safety separation requirements. In scenarios where wake turbulence separation is not explicitly required, adherence to standard safety separation alone is sufficient.

In 2010, Balakrishnan [3] reformulated the wake separation distance criteria into equivalent wake time separation intervals, as detailed in Table 1. This conversion facilitates integration into runway scheduling models.

The core optimization problem addressed in this paper is to determine an optimal aircraft sequencing schedule. This schedule must:

1. Satisfy all prescribed wake time separation constraints.
2. Minimize the total completion time for takeoff and landing operations.
3. Reduce associated landing costs.

Table 1. Tailrace time interval.

The aircraft ahead	The aircraft behind		
	Heavy	Medium	Light
Heavy	96	157	196
Medium	60	69	131
Light	60	69	82

According to references [4,5], the aircraft landing problem (ALP) is subject to two primary constraints:

1.Time Window Constraint: Each flight must land within a specified time interval bounded by an Earliest Landing Time (ELT) and a Latest Landing Time (LLT).

2.Wake Separation Constraint: Successive landings must adhere to the minimum wake turbulence separation intervals.

The Earliest Landing Time (ELT) is defined as the earliest possible touchdown time achievable when the aircraft operates at its maximum permissible airspeed. Conversely, the Latest Landing Time (LLT) represents the maximum allowable time an aircraft can remain airborne, typically involving holding patterns, before it must land.

Deviations from an ideal schedule incur operational costs. Landing earlier than planned can increase airport congestion and air traffic controller workload, leading to higher associated labor costs. Landing later typically results in increased fuel consumption due to extended flight time. Consequently, for each flight i, there exists an Optimal Landing Time (T_i) that minimizes the total operational cost. The cost function for flight i is typically modeled as a piecewise linear function:

$$C_i(t_i) = \begin{cases} g_i \times (T_i - t_i) & \text{if } t_\pi^i < T_\pi^i \text{ (early landing)} \\ 0 & \text{if } t_\pi^i = T_\pi^i \\ h_i \times (t_i - T_i) & \text{if } t_\pi^i > T_\pi^i \text{ (late landing)} \end{cases} \quad (1)$$

where t_i is the actual landing time, E_i and L_i are the earliest and latest landing times, and g_i and h_i are the unit costs for earliness and tardiness, respectively [6]. This creates a V-shaped convex cost profile centered at T_i, as illustrated in Figure 1.

 doi:10.20944/preprints202603.0874.v1

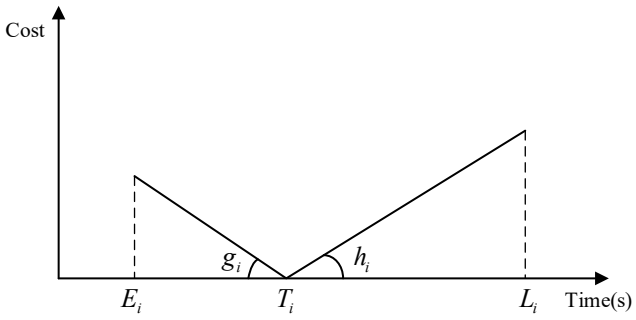

Figure 1. Plot of flight landing time versus cost.

The ALP is a well-known NP-hard combinatorial optimization problem characterized by multiple operational constraints and strict time-sensitivity requirements. Over the past two decades, extensive research has been conducted, leading to the development of various models and solution methodologies. Beasley [7,8] established a foundational mixed-integer programming (MIP) model for the ALP and created the widely-used OR-Library benchmark dataset. This dataset formally defines the key temporal parameters: Earliest Landing Time (ELT), Optimal Landing Time (OLT), and Latest Landing Time (LLT), enabling the optimization of objectives for different stakeholders.

To tackle the computational complexity, numerous heuristic algorithms have been proposed. H. Pinol [9] introduced a population-based heuristic, demonstrating its capability on a large-scale instance with 500 aircraft and 5 runways, though its performance degraded with over 50 aircraft. Sheng-Peng Yu [10] developed a cellular automata-based optimization algorithm, which achieved optimal solutions for small-scale cases (test cases 1-7) and near-optimal solutions for large-scale datasets from the OR-Library. Gui D et al. [11] proposed a novel meta-heuristic for the ALP, integrating tabu search to improve solution quality and a rolling time horizon method to enhance computational speed. Daniel A et al. [12] presented a fast greedy heuristic, the Closest Aircraft Sequence with Time Windows (CAS-TW), designed to minimize total delay while respecting operational constraints through a discretization strategy.

Recognizing the multi-faceted nature of the problem, several studies have incorporated multiple objectives. Chen Xiaodao et al. [13] formulated the problem as an integer linear program and transformed it into a multi-objective optimization, proposing an uncertainty-aware scheduling algorithm. Zhang Junfeng et al. [14] constructed a multi-objective ranking and scheduling model for a Parallel Machine Scheduling (PMS) analogy and proposed a multi-objective Imperial Competition Algorithm (ICA), showing significant improvement over First-Come-First-Served (FCFS). Wang Jianzhong et al. [15] proposed a multi-objective time index model, using a genetic algorithm to minimize overall delay and controller workload by weighting different objectives. Chen K et al. [16] established a multi-objective model for the complex interdependent runway ASP (ASP-CIR) and developed an Improved Multi-Objective Restarted Variable Neighborhood Search (IMORVNS) algorithm.

Recent work has increasingly focused on fairness and integrated arrival-departure operations. Liu Jixin et al. [17] addressed four combined operational scenarios, introducing fairness metrics—"equalized delay time for arrivals" and "equalized satisfaction for departures"—and established a two-level planning model to balance fairness and efficiency. Zhou Dingkai et al. [18] applied reinforcement learning to a static terminal area sequencing model, incorporating "flight satisfaction" to enhance fairness in resource allocation between airports and airlines.

To address large solution spaces and real-time requirements, advanced computational methods have been employed. Kang Rui [19] integrated an EoR (Earliest on Route) approach with sequencing strategies and proposed an S-shaped function-based Adaptive Particle Swarm Optimization (SA-PSO) algorithm to minimize flight delay. Zhang C et al. [20] introduced two prescription analysis methods—Estimation-Trial-Optimization (ETO) and Estimation-Trial-Distributionally Robust Optimization (ETDRO)—to improve the efficiency and robustness of sequencing under uncertainty.

 doi:10.20944/preprints202603.0874.v1

Dönmez K et al. [21] applied a Multi-Criteria Decision-Making (MCDM) approach, using the MEREC method for criteria weighting and the MARCOS technique to rank solutions, aiming to find a single objective representing an optimal compromise for all stakeholders. Feng Xiaorong et al. [6] developed a time-window-constrained optimization model and proposed a compact subsequence-based algorithm. This method demonstrated high accuracy and speed for small datasets but exhibited lower computational efficiency on large-scale problems.

Although the aforementioned methods have achieved satisfactory performance in addressing the ALP, their high computational complexity often leads to substantial processing time when handling large-scale instances. This limitation poses significant challenges for real-time scheduling and dynamic adjustments in practical operational environments. Consequently, improving algorithmic efficiency and reducing computational overhead remain critical and unresolved issues in this field of research.

In 2006, French computer scientist Coulom [22] first proposed Monte Carlo Tree Search (MCTS), a heuristic search algorithm designed for optimal decision-making in complex scenarios. MCTS efficiently approximates optimal policies through iterative random sampling and tree structure optimization, without requiring complete prior knowledge of the state space. Initially gaining prominence in artificial intelligence, MCTS has achieved groundbreaking success in domains such as the game of Go [23]. A notable milestone was reached in 2016 when AlphaGo, leveraging MCTS, defeated world champion Lee Sedol 4–1, marking the first time an AI system surpassed top human performance in Go.

Beyond game playing, MCTS has been effectively adapted to various optimization and scheduling problems. For instance, Song Wenshuai et al. [24] addressed the dynamic adjustment of reservoir surface sorting strategies under changing environmental conditions by proposing a method based on deep Monte Carlo tree search, which integrates deep neural networks for prior action probability prediction and policy evaluation. In railway operations, Wang Guanghui et al. [25] introduced a hybrid reinforcement learning–Monte Carlo tree search approach for route optimization, employing a breadth-first followed by depth-first selection strategy to balance global and local benefits. For military logistics, Peng Jian et al. [26] designed a heuristic algorithm incorporating MCTS to optimize task sequences for fleet support, using Monte Carlo trees to record evaluation outcomes and guide subsequent search directions. Similarly, Yu Ze et al. [27] applied an improved MCTS method to unmanned aerial vehicle (UAV) task planning in complex three-dimensional environments, introducing normalized reward ranges to handle parameter uncertainty and balance exploration versus exploitation. Pang et al. [28] proposed a machine learning-enhanced traveling salesman problem framework aimed at reducing total landing time compared to the FCFS approach.

For ALP, existing Arrival Management (AMAN) systems primarily achieve time sequencing through terminal area spatial path expansion and employ speed management when sufficient planning scope is available. In practice, it typically requires air traffic controllers (ATCs) to perform unified, globally optimized sequencing and scheduling of multiple incoming flights when they enter the radar range (radar visibility) of the airport ATC system, assigning each flight a landing time and corresponding time slot. The most widely used strategy for the ALP problem is the FCFS rule, which schedules aircraft based on their order of entry into the radar range. Current mainstream algorithms predominantly rely on heuristics [7–12,29]. There exists a demand and gap for developing aircraft landing scheduling algorithms based on deep reinforcement learning to minimize landing costs. Therefore, this paper proposes a Transformer-Augmented Monte Carlo Tree Search (TMCTS) algorithm to solve the flight sequencing scheduling problem. By considering constraints such as wake separation standards and time windows, combined with estimated arrival times in the terminal area, the algorithm aims to minimize the total cost of the flight sequence. The TMCTS method is employed to calculate the optimal landing sequence and landing time for each flight, The main contributions of this work are threefold:

1.Model Formulation: A reinforcement learning-based optimization model is established for aircraft sequencing, framing the scheduling task as a sequential decision-making process.

 doi:10.20944/preprints202603.0874.v1

2.Algorithm Design: A novel flight scheduling methodology is proposed, which integrates the structural strengths of the Transformer architecture with the strategic search capability of MCTS.

3.Architecture Innovation: A specialized two-head output module is designed, comprising a policy head, a value head, and a dedicated first-aircraft head, to effectively capture the unique dependencies and priority constraints inherent in flight scheduling.

Achieve more precise flight sequencing and scheduling through deep learning models. The proposed TMCTS algorithm operates as a deterministic search method, demonstrating strong robustness and stability in handling large-scale scheduling instances.We employed the Airland test case from the O-R library benchmark to train the model and validate the effectiveness and efficiency of our proposed algorithm. Results were compared against FCFS, heuristic algorithms (GA, PSO, CAO), and a commercial mixed-integer programming (MIP) solver (CPLEX).

The remainder of this paper is organized as follows: Section 2 details the workflow of the proposed TMCTS framework, including its algorithmic architecture and model components. Section 3 describes the experimental setup, covering data sources, parameter configurations, and training procedures. Section 4 presents the computational results and provides a comparative analysis with benchmark methods. Finally, Section 5 concludes the paper by summarizing the findings and suggesting directions for future research.

2. Methodology

2.1. Algorithm Framework

Figure 2 illustrates a deep reinforcement learning-based research framework for flight scheduling, which comprises three core components:

1.Transformer-based Scheduling Architecture: Flight data are initially encoded into feature vectors. These vectors are then processed through multiple encoder layers, each consisting of multi-head attention mechanisms and feedforward neural networks, to capture complex dependencies within flight sequences. After multi-layer transformations, two output heads generate prior knowledge and state-value estimates, respectively.

2.MCTS Guided Strategy Optimization: Leveraging the prior probabilities provided by the Transformer, MCTS iteratively refines flight scheduling strategies through a cyclic process of selection, expansion, simulation, and backpropagation. At each decision node, the improved policy and corresponding value derived from MCTS serve as supervisory signals to train the Transformer network.

3.Actor-Critic Reinforcement Learning with Transformer Core: An Actor-Critic architecture, integrated with policy gradient methods, employs the Transformer as its central model. Decision-making is guided by MCTS during exploration, while collected trajectory data are used to update the Transformer parameters, thereby facilitating continuous policy optimization for the scheduling agent.

This integrated framework enables synergistic interaction between representation learning, strategic search, and policy improvement, enhancing both the efficiency and robustness of flight scheduling under complex operational constraints.

 doi:10.20944/preprints202603.0874.v1

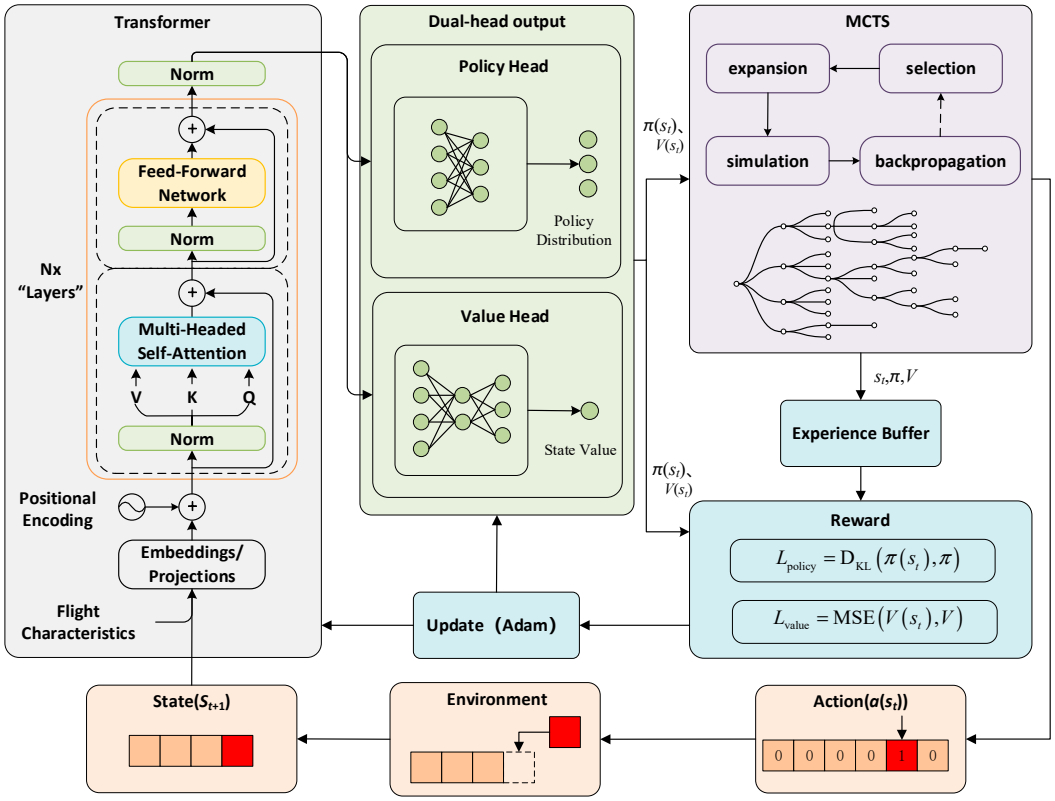

Figure 2. Research Framework.

2.2. Aircraft Landing Issues and Reinforcement Learning Models

2.2.1. Mathematical Model for ALP

The objective of the time-window constrained multi-flight optimization model is to determine an optimal scheduling sequence π^* for N flights and compute the corresponding optimal landing time $t_{\pi^*}^i$ for each flight under sequence π^*, such that the total cost $TC_{\pi^*}^N$ for all N flights is minimized. This objective can be formulated as follows:

$$TC_{\pi^*}^N = argmin\, t\, c_\pi \quad (\pi \in \Pi) \tag{1}$$

where π denotes a candidate landing sequence, Π represents the set of all feasible sequences of N flights, and tc_π (total cost) is the minimum total landing cost under sequence π, which can be expressed as:

$$tc_\pi = min \sum_{i=1}^{N} f(t_\pi^i) \tag{2}$$

Here, $f(t_\pi^i)$ represents the cost incurred when the i-th flight in sequence π lands at time t_π^i, expressed as:

$$f(t_\pi^i) = g_\pi^i * max(0, T_\pi^i - t_\pi^i) + \lambda_\pi^i * max(0, t_\pi^i - T_\pi^i) \tag{3}$$

where T_π^idenotes the optimal landing time for flight i under sequence π; and g_π^i and λ_π^i are the unit cost coefficients for earliness and tardiness, respectively, relative to T_π^i.

The model is subject to the following operational constraints:

1.Wake separation interval constraint:

$$t_\pi^j - t_\pi^i \geq S_{\pi(i,j)}, \forall i,j \in \{1,\cdots,N\}, i \neq j \tag{4}$$

 doi:10.20944/preprints202603.0874.v1

where $S_{\pi(i,j)}$ denote the minimum wake turbulence separation interval between flights i and j under sequence π.

2.Earliest landing time constraint:

$$t_\pi^i \geq E_\pi^i, \quad \forall i \leq N \tag{5}$$

where E_π^i is the earliest permissible landing time for flight i .

3. Latest landing time constraint:

$$t_\pi^i \leq L_\pi^i, \quad \forall i \leq N \tag{6}$$

where L_π^i is the latest permissible landing time for flight i.

These constraints ensure that the scheduling sequence respects safety, operational, and time-window requirements while minimizing the total landing cost.

2.2.2. Markov Decision Process (MDP) Modeling

The ALP is formulated as a Markov Decision Process (MDP), defined by the quadruple (S,A,R,P) :

1.State space

The state space integrates all essential aircraft information and scheduling progress. Each aircraft is represented by a 7-dimensional state vector, defined as:

$$s_i(t) = \left[E_\pi^i(t), T_\pi^i(t), L_\pi^i(t), g_\pi^i(t), h_\pi^i(t), m_\pi^i(t), d_\pi^i(t)\right] \tag{7}$$

whereE_π^i, T_π^i, and L_π^i denote the time window information; g_π^i and h_π^i are penalty coefficients; m_π^i is a binary indicator of whether flight i has been scheduled; and d_π^i represents the required wake separation between unscheduled flight i and the last scheduled flight.

2.Action Space

The action space comprises the set of aircraft that can be selected for landing in the current state. A valid action must satisfy two conditions: the aircraft has not yet been assigned a landing slot, and it complies with safety separation constraints relative to the preceding aircraft. The action space is formally expressed as:

$$A(t) = \left\{ a_i(t) | a_i(t) \in \{0,1\}, \sum_n a_i(t) = 1 \right\} \tag{8}$$

Here $a_i(t)$ denotes the action for the i-th flight. $a_i(t)$=0 indicates that the i-th flight is not selected, while $a_i(t)$=1 indicates selection.

3.Reward function

The reward function is designed to guide the agent toward minimizing cost and completing the scheduling task. It consists of three components:

- Cost penalty: A negative reward proportional to the cumulative cost incurred at the current scheduling step, with dynamically adjusted weighting throughout the process.
- Completion reward: A positive reward granted upon successful scheduling of all flights. The global reward is computed using the method described in the previous section for fixed-order ALP, yielding the total cost under the constructed sequence.
- Deviation Bonus: An additional reward provided when the final scheduling cost outperforms the historical best recorded value.

4.State transition.

Given the current state $S(t)$ and the next state $S(t+1)$, the state transition probability after executing action $A(t)$ is defined as $P(S(t+1) | S(t),A(t))$. In this paper, the state transition is deterministic, i.e., $P(S(t+1) | S(t),A(t))=1$.

The overall reinforcement learning-based decision-making process for flight scheduling is illustrated in Figure 3.

 doi:10.20944/preprints202603.0874.v1

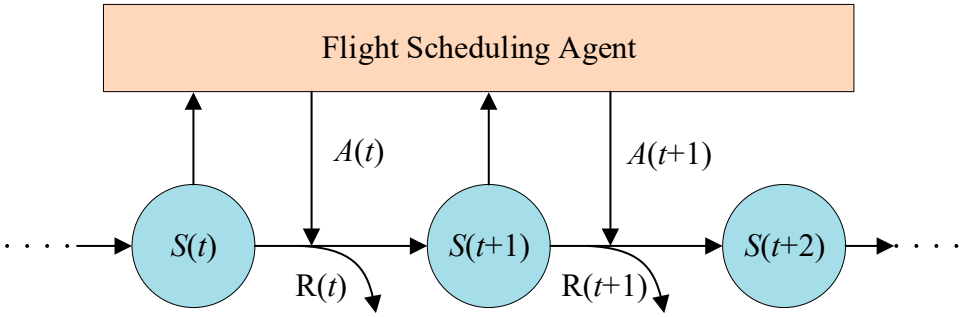

Figure 3. Reinforcement Learning Decision Process for Flight Scheduling.

2.3. Solving for Aircraft Landing Times Under Fixed Sequence

The ALP can be decomposed into two subproblems: the first subproblem entails determining landing times given a fixed sequence, while the second subproblem involves identifying the optimal landing sequence. In the proposed algorithm, each time the Monte Carlo Tree Search (MCTS) selects an aircraft, it computes the cost of the corresponding sequence, assesses its quality, and treats this as the ground-truth cost. This value is subsequently employed in gradient descent alongside the value estimate generated by the value head of the Transformer network.

To address the problem of determining aircraft landing times under a fixed sequence, this paper employs the Dynamic Programming Approach to Limit Optimization (DPALO) [29]. First, the landing scheduling problem with a fixed flight sequence is formulated as an integer programming problem for determining the landing time of each flight. This problem is then solved using the Dynamic Programming Algorithm (DPA).

Let $F(t_\pi^i)$ denote the minimum cumulative total cost generated by the first i flights when the i-th flight in sequence π lands at time t_π^i. Given that the first flight in the sequence has cost $F(t_\pi^1)$=f(t_π^1), the i-th flight has cost $F(t_\pi^i) = f(t_\pi^i) + min(F(t_\pi^{i-1}))$. Establish a time-based recursive function for $F(t_\pi^i)$ as follows:

$$\mathrm{F}\left(t_\pi^i\right)=\begin{cases}\mathrm{f}\left(t_\pi^i\right) & i=1\\ \mathrm{f}\left(t_\pi^i\right)+\min\limits_{E_\pi^{i-1}\le t_\pi^{i-1}\le \min\left(t_\pi^i-S_{\pi(i-1,i)},\ L_\pi^{i-1}\right)}\left(\mathrm{F}\left(t_\pi^{i-1}\right)\right) & 1<i\le n\end{cases} \tag{10}$$

Equation (2) can be further simplified to Equation (11).

$$\mathrm{tc}_\pi = min(F(t_\pi^i)) \quad (\pi \in \Pi) \tag{11}$$

Based on Equations (5) and (6), the range of values for dependent variable t_π^i of function $f(t_\pi^i)$ is determined to be $[E_\pi^i, L_\pi^i]$. Considering Equation (4) and the optimal value of objective function tc_π, the range of values for dependent variable t_π^i can be further narrowed. This section will further analyze the range of values for dependent variable t_π^i. Figure 4 shows the flowchart of the dynamic programming algorithm for solving the ALP. In Figure 4, $f\pi[i][t]$ equals $f(t_\pi^i)$, where $f\pi[i][t]$ represents the cost of landing a flight at that time. For example, $f(E_\pi^i)$ denotes the cost of landing the i-th flight in sequence π at its earliest possible landing time.$F\pi[i][t]$ equals $F(t_\pi^i)$, where $F\pi[i][t]$ represents the landing time of a flight at that moment and the minimum cumulative total cost generated by all preceding flights. For example, $F(E_\pi^i)$ denotes the minimum cumulative total cost generated by the preceding i-1 flights when flight i lands at its earliest possible landing time. By calculation, the optimal landing time and total cost for flights under a fixed sequence can be determined.

 doi:10.20944/preprints202603.0874.v1

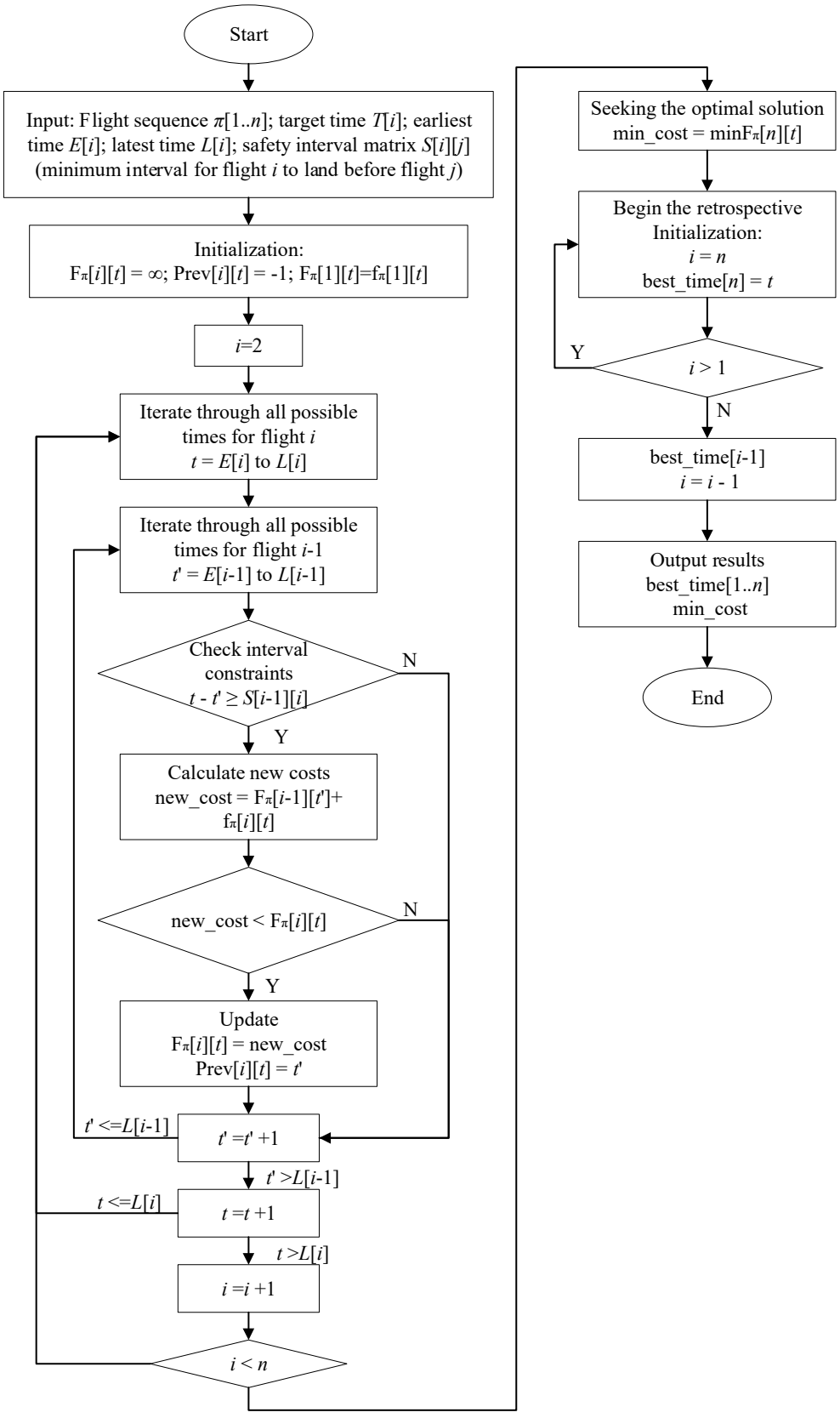

Figure 4. Flowchart of Dynamic Programming Algorithm.

Figure 5 illustrates the search range for flight landing costs under the discretized π sequence, using three flights as an example. Flight landing times must simultaneously satisfy constraints (4), (5), and (6). When $E_\pi^2 > E_\pi^1 + S_{\pi(1,2)}$ occurs, the second flight cannot land at time E_π^2, $E_\pi^1 + S_{\pi(1,2)}$), as shown in dashed box 1; when $E_\pi^3 < E_\pi^2 + S_{\pi(2,3)}$ occurs, the third flight cannot land at time E_π^3,

 doi:10.20944/preprints202603.0874.v1

$E_\pi^2 + S_{\pi(2,3)}\big)$, as shown in dashed box 2; When $L_\pi^2 > L_\pi^1 + S_{\pi(1,2)}$ occurs, since deviation costs increase linearly with time, $f(L_\pi^2) > f\big(L_\pi^1 + S_{\pi(1,2)}\big)$ holds. Therefore, when calculating the total landing cost for the first two flights, $L_\pi^1 + S_{\pi(1,2)}$, L_π^2 within the second flight's time window can be omitted from calculation, as shown in dashed box 3. When $L_\pi^1 + S_{\pi(1,2)}$, L_π^2 occurs, the third flight cannot land at time L_π^3, $L_\pi^2 + S_{\pi(2,3)}$, as indicated by dashed box 4.

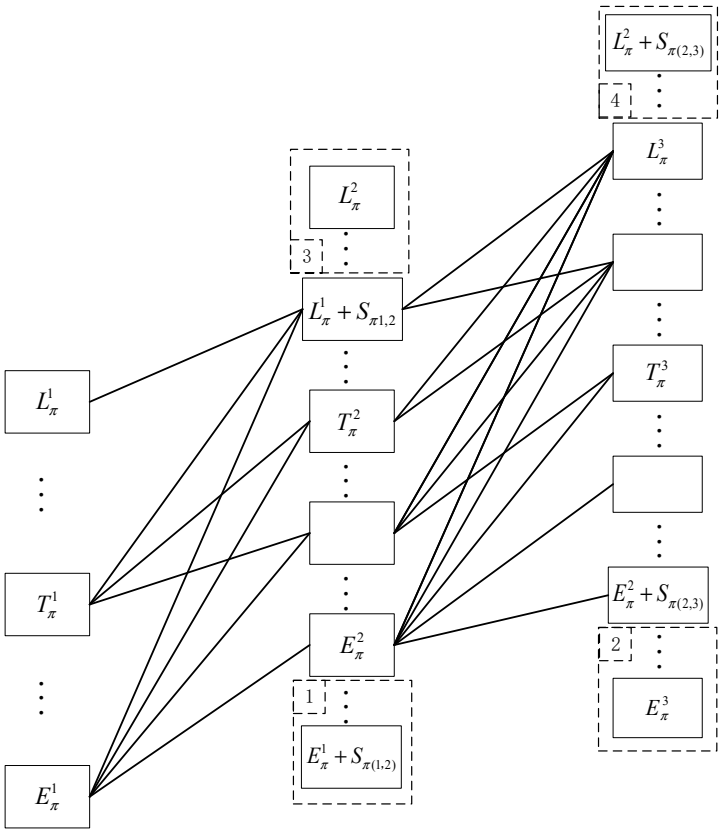

Figure 5. Initial search range.

Proposition 1: Let π denote a fixed landing sequence for flights. Under constraints (3), (4), and (5), sequence $\boldsymbol{t} = \{t_\pi^{i\,*} | i = 1,2,3,\ldots,n)\}$ represents the landing times $tc_\pi = \sum_{i=1}^n f(t_\pi^{i\,*}) = min(F(t_\pi^n)) = F(t_\pi^{n*})$ for each flight that minimize tc_π. Then: $t_\pi^{1\,*} \le T_\pi^1$. Prove by contradiction as follows: First assume that under sequence π, there exists sequence $\boldsymbol{t}' = \{t_\pi^{1\,'} | i = 1,2,3,\ldots,n\}$ such that the minimum value tc_π is obtained, $tc_\pi = \sum_{i=1}^n f(t_\pi^{i\,'})$, and $t_\pi^{1\,'} > T_\pi^1$.Then, the landing time of the first flight is advanced from $t_\pi^{1\,'}$ to T_π^1. Since this adjustment does not affect the safety intervals or landing times of subsequent flights, their arrival times remain unchanged. Thus, the π sequence lands in the order determined by $\boldsymbol{t}'' = \{T_\pi^1, t_\pi^{1\,'}, \ldots, t_\pi^{n\,'}\}$.At this point, $tc_\pi = f(T_\pi^1) + \sum_{i=2}^n f(t_\pi^{i\,'}) = 0 + \sum_{i=2}^n f(t_\pi^{i\,'}) < \hbar_\pi^1 \times \left(t_\pi^{1\,'} - T_\pi^1\right) + \sum_{i=2}^n f(t_\pi^{i\,'})$, sequence $\boldsymbol{t}''$ can achieve a lower cost. Assuming this contradicts Proposition 1, $t_\pi^{1\,'} \le T_\pi^1$ is proven. The time window for the first flight can be optimized to $[E_\pi^1,\ T_\pi^1]$.

Inference 1: Similarly, by contradiction, the optimal landing time for the last flight n cannot be earlier than its earliest landing time. Therefore, the time window for the nth flight can also be constrained to optimize to $[T_\pi^n,\ L_\pi^n]$.

Inference 2: Based on Proposition 1, Corollary 1, Constraint (3), and Constraint (4), Equations (12) and (13) can be derived.

$$e_\pi^i = \begin{cases} T_\pi^i & i = n \\ max\big(E_\pi^i, min\big(T_\pi^i, e_\pi^{i+1} - S_{\pi(i+1,i)}\big)\big) & i < n \end{cases} \tag{12}$$

 doi:10.20944/preprints202603.0874.v1

$$l_{\pi}^{i} = \begin{cases} T_{\pi}^{i} & i = 1 \\ min\left(L_{\pi}^{i}, max\left(T_{\pi}^{i}, l_{\pi}^{i-1} + S_{\pi(i-1,i)}\right)\right) & i > 1 \end{cases} \quad (13)$$

Inference 2 provides the landing time window $[e_{\pi}^{i}, l_{\pi}^{i}]$ after boundary optimization. Figure 6 shows the search range after boundary optimization, using three flights as an example. For Flight 1, the search range is narrowed to $[E_{\pi}^{1}, T_{\pi}^{1}]$; for Flight 3, it is narrowed to $[T_{\pi}^{3}, L_{\pi}^{3}]$. For Flight 2, if $T_{\pi}^{2} > e_{\pi}^{3} - S_{\pi(2,3)}$, $E_{\pi}^{2} > e_{\pi}^{3} - S_{\pi(2,3)}$ and $T_{\pi}^{2} > l_{\pi}^{1} + S_{\pi(1,2)}$ occurs, its search range is narrowed to $[E_{\pi}^{2}, T_{\pi}^{2}]$. The dashed box indicates the search range eliminated compared to DPA optimization, while the dashed line denotes omitted search steps. The search strategy for the remaining portions remains unchanged. The search range required to find the minimum landing cost for the first i+1 flights also correspondingly decreases, $tc_{\pi}^{i+1} = min(F(e_{\pi}^{i+1}), \ldots, F(l_{\pi}^{i+1}))$. Compared to DPA, this approach significantly reduces computational steps and simplifies algorithmic complexity.

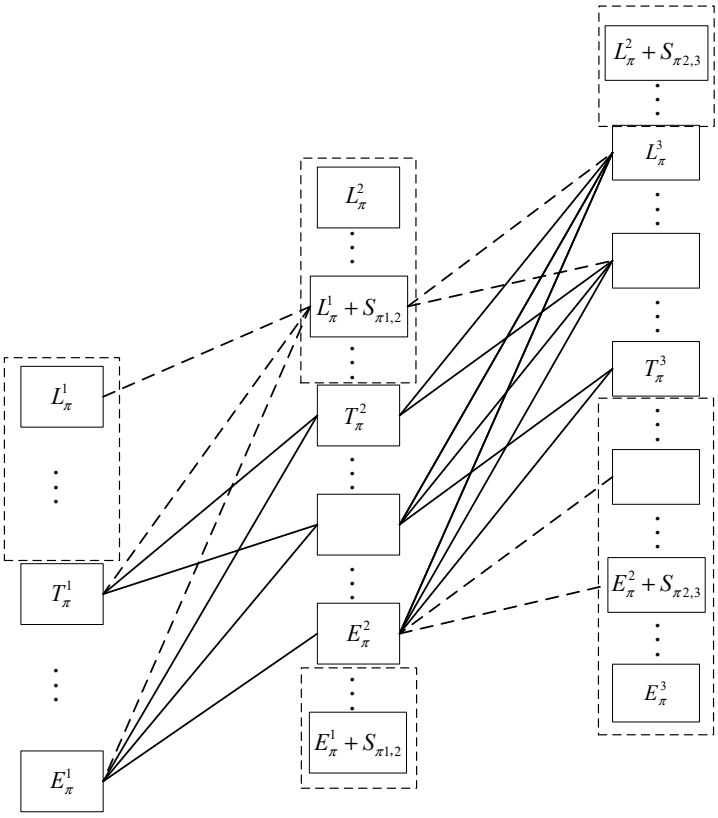

Figure 6. Search range after limit optimization.

Figure 7 shows the initial time windows $[E_{\pi}^{i}, L_{\pi}^{i}]$ and the boundary-optimized time windows $[e_{\pi}^{i}, l_{\pi}^{i}]$ for 10 flights in the Airland1 dataset from OR-Library. The dashed lines indicate the search range for DPA, while the solid lines represent the search range for DPALO. Boundary optimization significantly reduces the computational scope of the algorithms.

 doi:10.20944/preprints202603.0874.v1

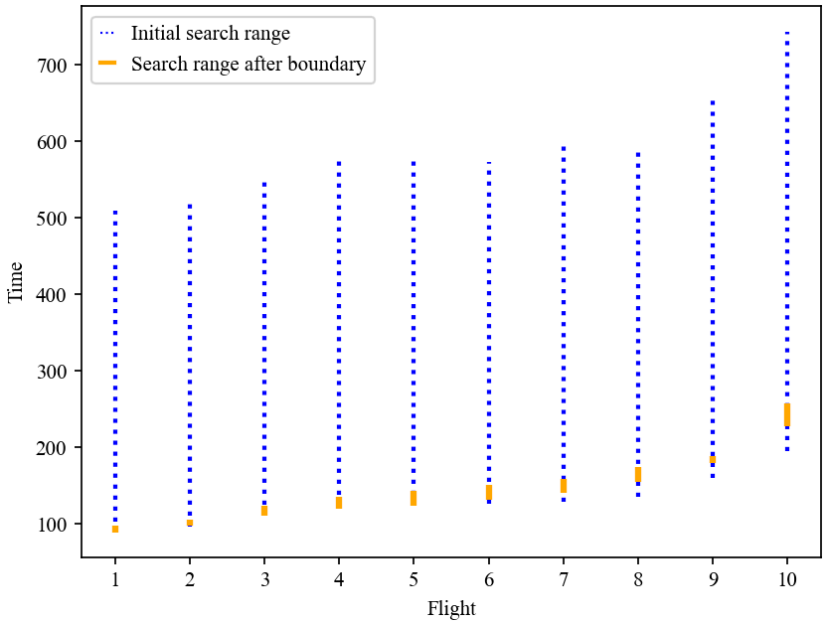

Figure 7. Algorithm search range.

2.4. Solving for the Optimal Landing Sequence

2.4.1. Monte Carlo Tree Search

To solve for the optimal landing sequence, the TMCTS approach is employed. The landing time and landing cost, computed using the method described in Section 2.3, serve as the optimization objective for the TMCTS algorithm.

MCTS is a heuristic search algorithm based on random simulation and tree structures, particularly suitable for scenarios with large state spaces where exhaustive search methods are computationally infeasible. Figure 8 illustrates the workflow of the MCTS process. The specific steps are as follows:

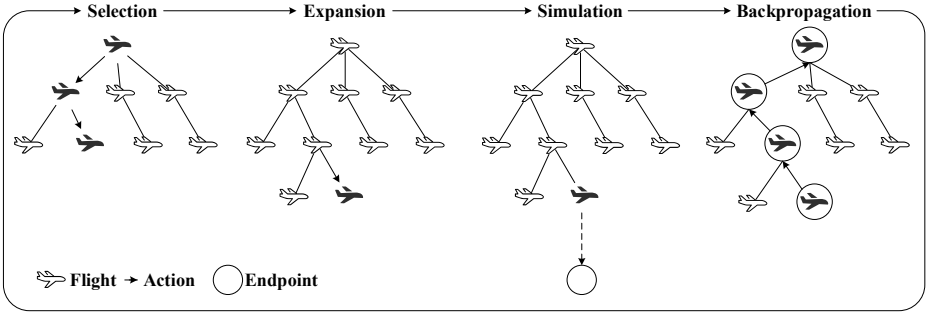

Figure 8. Schematic Diagram of MCTS.

(1) Selection. Starting from the root node, the algorithm traverses downward through the tree structure to identify a node worthy of further exploration. The core action selection principle is based on the Upper Confidence Bound (UCB), which balances exploration and exploitation. The UCB formula is defined as follows:

$$UCB = \frac{Q(n)}{N(n)} + C \times \sqrt{\frac{\ln N_p(n)}{N(n)}} \quad (14)$$

 doi:10.20944/preprints202603.0874.v1

Among these, $Q(n)$ represents the cumulative value of node n; $N(n)$ denotes the number of visits to node n; $\frac{Q(n)}{N(n)}$ is the utilization term; C is the exploration coefficient; $N_P(n)$ indicates the number of visits to the parent node; $\sqrt{\frac{ln\, N_p(n)}{N(n)}}$ is the exploration term.

Traditional MCTS relies on fully random simulations to evaluate node values, resulting in numerous futile searches that reduce computational efficiency. This paper employs a Transformer to generate action policies, state value estimates, and specialized decisions, providing MCTS with prior probabilities. This optimizes search logic during expansion and simulation phases, reducing ineffective exploration. The optimized UCB formula is:

$$UCB = \frac{Q(n)}{N(n)} + C \times p(a) \times \sqrt{\frac{ln\, N_p\,(n)}{N(n)}} \tag{15}$$

Among these, $p(a)$ is the prior probability of selecting action a at node n.

(2) Expansion. When node n is not in a terminal state and unexplored actions exist, perform tree extension on node n. Randomly select an action a from the unexplored action set of node n, generate the corresponding child node n', and initialize the utility and exploration values of the new node. After simulation, proceed with scoring. The scoring calculation process for each node is as follows:

$$Q = Val_{nn} \times e^{-\frac{c-c_g}{c_g}} \tag{16}$$

Among these, val_{nn} represents the value output of the neural network, indicating the probability of finding a solution by selecting this node. c denotes the cost incurred when choosing this node, while c_g represents the cost of the TTS algorithm at this step.

(3) Simulation. The potential value of a newly expanded node n' is rapidly evaluated through simulation. Starting from node n', actions are continuously executed according to a predefined policy until a terminal state is reached, ultimately yielding the reward value R for that trajectory. By repeatedly performing simulations, MCTS progressively explores different ordering strategies.

(4) Backpropagation. During the backpropagation phase, the reward R obtained from the simulation is propagated backward through all ancestral nodes along the path (from n' back to the root node). This process updates the node ratings and visit counts of each node, providing evaluation criteria for subsequent selection phases.

By iteratively executing these four steps, a search tree is progressively constructed. Through a finite number of iterations, the tree structure gradually converges toward high-value paths, ultimately yielding an optimal sequencing result.

2.4.2. Transformer Architecture

The ALP involves multiple constraints, strong interdependencies, and sequential dependencies. This paper proposes a Transformer-based flight sequencing algorithm. The Transformer architecture is primarily employed to address aircraft scheduling decisions by capturing the features and interdependencies of each aircraft within the sequence. It outputs action policies and state values, providing prior probabilities for MCTS.

The Transformer architecture, as illustrated in Figure 9, primarily consists of two components:

 doi:10.20944/preprints202603.0874.v1

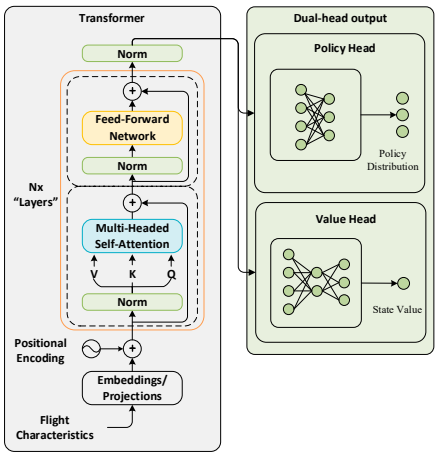

Figure 9. Transformer Architecture Diagram.

(1) Transformer Encoder

The model architecture consists of stacked encoder layers, each comprising a multi-head attention module, a feed forward neural network, residual connections, and layer normalization. Through multi-layer stacking, feature encoding progressively becomes more abstract and global in scope.

Embeddings Projections Layer maps the multidimensional features of each flight into a fixed-dimensional space via linear transformation, generating flight feature vectors. This enables the model to process features as numerical representations. Sine-cosine positional encoding is applied to incorporate positional information of flights within the sequence, endowing the model with awareness of temporal order. Positional encodings are combined with state embeddings before being fed into the model, as defined by the following equations:

$$PE_{pos,2i} = sin(\frac{pos}{10000^{2i/d_{\text{model}}}}) \tag{17}$$

$$PE_{pos,2i+1} = cos(\frac{pos}{10000^{2i/d_{model}}}) \tag{18}$$

where *pos* denotes the position of a flight within the sequence, and *i* represents the feature dimension index.

The Multi-Headed Self-Attention layer enables each flight feature to attend to all other flight features, capturing global dependencies. It allows the model to identify relationships between aircraft while simultaneously handling interactions between individual and multiple flights. Flight features, after positional encoding and embedding, are used as Query, Key, and Value inputs. Attention weights are computed between each flight and every other flight in the sequence.

The feed-forward network layer applies a nonlinear transformation to the output vectors from the self-attention mechanism, enhancing the model's expressive capacity to learn more complex feature patterns. Residual connections and layer normalization are incorporated to mitigate gradient vanishing during training, ensuring stable feature extraction for flight states.

(2) Multi-Task Output Layer

Based on the learned decision features, the model outputs two types of results. The Policy Head outputs a probability distribution over aircraft to be selected as the next action, which guides the MCTS in choosing subsequent moves. The Value Head estimates the value of the current state, outputting the probability that a feasible solution exists for that state. This value is used to compute node scores within MCTS.

 doi:10.20944/preprints202603.0874.v1

3. Model Training

3.1. Training Framework

During training, an Actor-Critic framework guided by MCTS is employed. As illustrated in Figure 10, the Transformer simultaneously outputs a policy distribution and a state value estimate. By interacting with the environment to collect data optimized through MCTS, the network is updated via the joint optimization of policy loss and value loss, enabling the agent to improve its policy within the target environment.

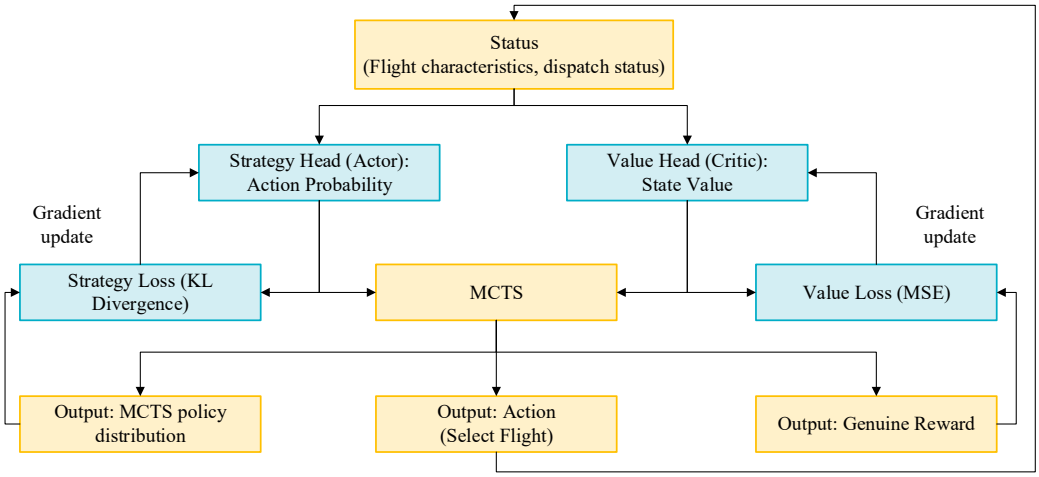

Figure 10. Actor-Critic Training Framework.

First, flight information and state representations are encoded into feature vectors by the Transformer encoder, which outputs both a policy distribution and a value estimate for the current state. The policy output provides prior action probabilities for MCTS, while the value estimate guides MCTS in action selection. MCTS constructs a search tree and ultimately outputs an MCTS policy distribution, a ground-truth reward (representing the probability of finding a feasible solution in the current state), and a selected action.

Each flight in the dataset is characterized by seven attributes. The final attribute specifies the required wake separation time interval between the current flight and its predecessor. This information is structured as a matrix where n denotes the number of flights, and n_{ij} represents the wake separation interval required for flight j to follow flight i. The value of n_{ij} depends on the aircraft types of both flights.

3.2. Parameter Settings

In the flight scheduling analysis based on deep MCTS, a Transformer model combined with an MCTS strategy is employed, with the corresponding parameters listed in Table 2.

Table 2. Parameter Settings.

Model	Parameter	Value
MCTS	Number of simulations	512
	Exploration Constant	1.414
	Batch Search Quantity	32
Transformer	Basic Feature Dimensions	256
	Number of attention heads	8
	Number of encoder stacking layers	6
	Dropout	0.1

 doi:10.20944/preprints202603.0874.v1

Initial learning rate	0.0003
Experience Replay Buffer Capacity	10000
Number of training batches	128

The parameter design of the model balances the complexity of the ALP with computational efficiency. For the search strategy, the MCTS module uses a simulation search depth of 512 and a batch size of 32, enabling thorough exploration and exploitation of the decision space within limited computational resources. The exploration constant is set to 1.414, achieving a balanced trade-off between exploration and exploitation based on the classic Upper Confidence Bound (UCB) formula, thereby preventing premature convergence to local optima.

Regarding the model architecture, a Transformer-based encoder structure is adopted with an embedding dimension of 256. It consists of six encoding layers, each equipped with eight self-attention heads. Each attention head focuses on feature representations within a 32-dimensional subspace, enabling the parallel capture of multidimensional scheduling features such as time window constraints, penalty cost weights, and separation requirements. The hidden layer of the feedforward network has a dimension of 512. During training, dropout regularization is applied with a rate of 0.1 to mitigate overfitting. Positional encoding supports sequences of up to 500 elements, accommodating large-scale scheduling scenarios.

The experimental setup employs the Adam optimizer with a learning rate of 3×10^{-4}, an experience replay buffer capacity of 10,000, and a batch size of 128.

3.3. Training Methods

The training process adopts a hybrid approach combining Behavior Cloning and Reinforcement Learning. It is divided into two main stages: pre-training (distillation phase) and reinforcement learning training.

Given the limited volume of data in existing datasets and the fact that Transformer models require substantial data to learn effective strategies, a pre-training phase is conducted before the main training. A large amount of flight-like data was generated based on the original dataset. For ALP of different scales, genetic algorithms were used to produce extensive trajectory data. These trajectories include sequences of decisions from initial to terminal states, along with corresponding expert actions for each decision. The Transformer network then underwent behavior-cloning training on this trajectory data, learning to imitate the decision-making of the genetic algorithm. The goal of this phase is to enable the network to initially capture basic scheduling decision patterns, providing a solid foundation for subsequent reinforcement learning. In this work, 200,000 simulated flight data points were generated, and the dataset was trained for 300 epochs.

Following pre-training, MCTS is employed as the policy improvement algorithm. Data is generated through the “select-expand-simulate-backtrack” process, and this data is used to further train the network. At each training step, the current network guided the MCTS search. The network was then updated based on the policy distribution obtained from the MCTS search and the final outcome. An experience replay buffer was used to store training data, with samples periodically drawn from the buffer for training. Curriculum learning was employed, starting with a small dataset and gradually increasing its size as model performance improved.

After pre-training, MCTS is employed as the policy improvement algorithm. Data are generated through the “selection–expansion–simulation–backpropagation” cycle and used to further train the network. At each training step, the current network guides the MCTS search. The network is then updated based on the policy distribution obtained from MCTS and the final outcome. An experience replay buffer stores training data, with samples periodically drawn from the buffer for training. Curriculum learning is applied, starting with a small dataset and gradually increasing its size as model performance improves.

 doi:10.20944/preprints202603.0874.v1

4. Results and Discussion

The experimental setup includes an Intel Core i5-12400F processor, an NVIDIA GeForce RTX 4060 Ti graphics card, and 16 GB of RAM. All implementations were developed in Python.

Figure 11 illustrates the loss curves of the TMCTS model over 300 training epochs. The policy network loss converged to 0.0275, the value network loss converged to 0.0046, and the total loss converged to 0.0229. The value network loss decreased rapidly, converging and stabilizing within the first 10 epochs. The policy network loss gradually converged to 0.0275 as training progressed, stabilizing after approximately 250 epochs.

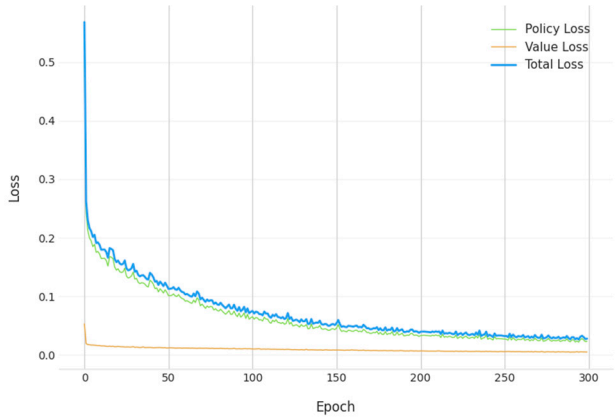

Figure 11. Loss of the TMCTS Model.

For the ALP, MCTS should prioritize depth-first over breadth-first search logic. The decision tree structure generated by MCTS is illustrated in Figure 12, which depicts the action selection process. A key characteristic of ALP lies in the sequential dependency of decisions: selecting a specific aircraft for landing not only affects the scheduling window of adjacent aircraft but also alters the feasible time intervals for distant aircraft through the propagation of separation constraints. Therefore, the search algorithm must thoroughly explore the complete outcomes of decision sequences. While a breadth-first strategy can evaluate more actions at shallow levels, fully exploring the scheduling sequences of each branch leads to an exponential increase in computational complexity.

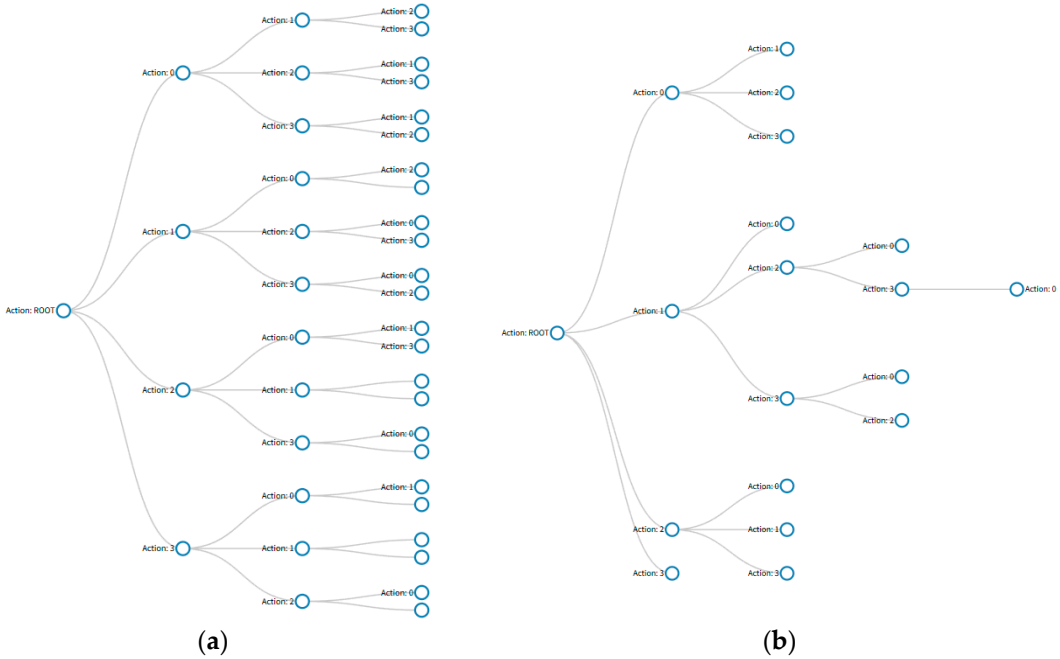

Figure 12. Decision Tree Structure (**a**) Breadth-First Search; (**b**) Depth-First Search.

 doi:10.20944/preprints202603.0874.v1

The number of MCTS simulations influences both model accuracy and computational time. In this study, MCTS simulations were performed with 32, 64, 128, and 256 iterations. The results obtained for each simulation count, compared against the best-known values (BKV) [30], are presented in Table 3.

Table 3. Computational Results for Different Simulation Runs.

Group	BKV	TMCTS							
		Num=32		Num=64		Num=128		Num=256	
	TC	TC	t(s)	TC	t(s)	TC	t(s)	TC	t(s)
1	700	700	0.75	700	0.977	700	1.573	700	2.517
2	1480	1480	0.351	1480	0.683	1480	1.511	1480	2.804
3	820	820	0.457	820	0.962	820	1.919	820	3.677
4	2520	2520	0.536	2520	0.884	2520	2.032	2520	3.675
5	3100	6260	0.451	3100	0.889	3100	2.111	3100	3.656
6	24442	24442	3.114	24442	6.301	24442	12.844	24440	25.559
7	1550	1550	1.569	1550	3.214	1550	6.686	1550	13.819
8	1950	2285	1.221	2025	2.499	1860	5.196	1860	9.972
9	5611	5907.85	5.821	5861.51	11.423	5748.73	24.811	5706.44	58.409
10	12329	14142.58	11.902	13696.76	22.52	12703.58	53.382	12793.52	119.577
11	12418	12698.92	20.691	12675.54	39.035	12503.92	85.725	12503.92	172.669
12	16209	17192.69	31.114	17318.36	66.428	16725.44	121.36	16803.22	256.906
13	44832	38134.56	145.918	38549.12	302.504	38629.45	644.307	38444.89	1221.15
Average difference	——	13.277	——	-247.9	——	-398.298	——	-403	——
Average time	——	——	17.22	——	35.255	——	74.112	——	145.722
Average time for the first 8 groups	——	——	1.056	——	2.051	——	4.234	——	8.21
Average time for the last 5 groups	——	——	43.089	——	88.382	——	185.917	——	365.742

As shown in Table 3, when the number of simulations is 128, the average difference reaches -390.483, and the computation time is relatively fast.

On small-sample datasets (first 8 datasets), achieving BKV is possible with 128 simulations, and outperforming BKV occurs in the 8th dataset. When simulations exceed 128, computational results remain unchanged while average computation time increases from 4.234s to 8.21s—a 93.9% rise significantly reducing computational efficiency. When the simulation count is below 128, computational efficiency improves only marginally while compromising accuracy. For the eighth dataset, a simulation count of 64 yields a cost of 2025, exceeding the 1860 cost achieved at 128 simulations.

For large-scale datasets (the last five datasets), the number of simulations substantially affects computation time, which approximately doubles with each doubling of the simulation count. Comparisons indicate that 128 simulations provide both shorter computation times and higher computational accuracy. Therefore, the results obtained with 128 simulations are selected for subsequent comparative analysis.

This paper compares five algorithms—FCFS, DPALO+GA [29], DPALO+PSO [29], CPLEX [5], and CAO [10]—against the proposed TMCTS approach. The mean interpolation values and average

 doi:10.20944/preprints202603.0874.v1

computation times of these algorithms, along with BKV, were calculated on the same dataset. The comparison results are presented in Table 4.

Table 4. Comparison of results from various algorithms.

Group	BKV	FCFS	TMCTS		DPALO+GA		DPALO+PSO		CPLEX		CAO	
			TC	t(s)	TC	t(s)	TC	t(s)	TC	t(s)	TC	t(s)
1	700	1280	700	1.573	700	0.004	700	0.006	700	0.8	700	0.0025
2	1480	1790	1480	1.511	1480	0.028	1482	0.046	1480	0.8	1480	0.0034
3	820	1790	820	1.919	820	0.025	820	0.058	820	1.3	820	0.0042
4	2520	4890	2520	2.032	2520	0.004	2520	0.006	2520	2.3	2520	0.0043
5	3100	6470	3100	2.111	3100	0.043	3244	0.159	3100	3.7	3100	0.1945
6	24442	24442	24442	12.844	24442	0.004	24442	0.006	24442	0.7	24442	0.0521
7	1550	1550	1550	6.686	1550	0.005	1550	0.007	1550	1.2	1550	0.0857
8	1950	18870	1860	5.196	1863.75	0.056	1863.25	0.185	1950	2.2	1995	1.711
9	5611	18937.16	5748.73	24.811	5694.52	2.541	5675.01	3.223	5611.7	932.2	5611	6.272
10	12329	27660	12703.58	53.382	13826.42	5.727	14799.94	4.615	12292.2	3600	12457	29.314
11	12418	35532.91	12503.92	85.725	12671.36	4.706	12922.42	5.506	12418.3	3600	12439	31.717
12	16209	46471.72	16725.44	121.36	17249.11	6.72	17626.09	6.849	16122.2	3600	16262	47.328
13	44832	99776.58	38629.45	644.307	38724.21	21.91	41323.23	12.052	37848.9	3600	38573	267.125
Average difference	——	12423.03	-398.298	——	-255.36	——	77.457	——	-546.6	——	-462.5	——
Average time	——	——	——	74.112	——	3.214	——	2.517	——	1180.4	——	29.5

Table 4 demonstrates that the TMCTS approach achieves a significant improvement in average deviation compared to FCFS. Adhering to FCFS under large sample sizes would incur substantial landing costs—specifically, a landing cost of 99,776.58 when handling 500 flights, more than double that of the adjusted methods. Compared to heuristic algorithms DPALO+GA and DPALO+PSO, improvements of 142.938 and 475.755 were achieved respectively. In computational time, TMCTS achieved a 93.7% improvement over the MIP solver CPLEX., though it still lags behind other algorithms. This is primarily because relying solely on the Transformer's direct output leads to significant computational inaccuracies. On certain datasets, it may fail to complete the sequencing task, necessitating MCTS to search for the optimal ordering sequence. During MCTS simulation, the complexity of the action space and state space grows exponentially. To achieve higher computational accuracy, substantial time is required for MCTS to explore better paths.

When the sample size is less than 50 (first 8 datasets), optimal results can be directly provided. All algorithms achieved the BKV value in the first 7 datasets. For the eighth dataset, the TMCTS computation yielded 1860, outperforming both BKV's 1950 and previous algorithms while simultaneously ensuring computational accuracy and speed. When the sample size exceeded 100 (the last five datasets), the TMCTS results showed minimal deviation from existing algorithms, maintaining computational accuracy. However, computational efficiency was limited due to algorithmic constraints.

5. Conclusion

For the ALP, this paper proposes a deep reinforcement learning method based on MCTS and Transformers. Output the landing sequence and landing times for flights to minimize landing costs. The approach formulates flight scheduling as a MDP and designs a dual-head Transformer network comprising policy and value heads. Leveraging its global attention mechanism, the network captures dependencies among flights and provides prior policy guidance for MCTS. Training is conducted within an Actor–Critic framework, combining behavior cloning pretraining with reinforcement learning optimization.

The experiment utilized the OR-Library dataset. Results indicate that the TMCTS approach achieved a significant improvement in average deviation compared to FCFS, with relative

 doi:10.20944/preprints202603.0874.v1

improvements of 142.938 and 475.755 over DPALO+GA and DPALO+PSO, respectively. In terms of computation time, TMCTS achieved a 93.7% improvement over the MIP solver CPLEX., though it remains slower than other benchmark algorithms. The proposed method is able to approximate or outperform known optimal solutions in most cases. However, the number of MCTS simulations constrains the algorithm's search efficiency. Future improvements could involve expanding training data and extending training time to enhance the quality of the initial policy, thereby reducing the reliance on MCTS simulations and improving overall algorithmic efficiency. The proposed method can assist the AMAN system. First, by statistically analyzing historical flight trajectory information and integrating aircraft performance data with the terminal area route network, it calculates the landing time window (i.e., the earliest and latest possible landing times) for each flight. Second, by receiving real-time flight trajectory data, it computes the landing sequence and timing with the minimum cost for each flight. Compared to traditional rule-based optimization methods, the proposed approach is data-driven and capable of identifying optimization patterns from historical data, resulting in a flight sequence with a lower total cost. This provides new technical support for practical air traffic arrival management and holds significant reference value.

Author Contributions: Conceptualization, J.H. and S.Z.; methodology, S.Z, X.F.; software, J.H. and S.Z.; validation, S.Z., X.F. and X.W.; formal analysis, S.Z.; investigation, S.Z.; resources, J.H., X.F. and X.W.; data curation, S.Z.; writing—original draft preparation, S.Z.; writing—review and editing, J.H., X.F. and X.W.; supervision, X.F. and X.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by [The State Key Laboratory of Air Traffic Management System]grant number[SKLATM202402]; [The National Natural Science Foundation of China] grant number[62173332]]; [Jiangsu Province's Major Science and Technology Project] grant number[BG2025007].And The APC was funded by [The State Key Laboratory of Air Traffic Management System] grant number[SKLATM202402].

Data Availability Statement: Publicly available datasets were analyzed in this study. This data can be found here: http://people.brunel.ac.uk/~mastjjb/jeb/orlib/airlandinfo.html (accessed on 11 October 2025).

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

References

1. 2023 Statistical Bulletin on the Development of the Civil Aviation Industry.Available online:http://www.caac.gov.cn/XXGK/XXGK/TJSJ/202505/t20250515_227513.html(2026/1/1)
2. Li, Y.Y. Analysis and Prediction of Airport Delay Based on Big Data Mining. Master's thesis, Nanjing University of Aeronautics and Astronautics, Nanjing, China, **2019**.
3. Balakrishnan, H.; Chandran, B.G. Algorithms for Scheduling Runway Operations Under Constrained Position Shifting. *Operations Research* **2010**, *58*, 1650–1665, doi:10.1287/opre.1100.0869.
4. Xu, B. An Efficient Ant Colony Algorithm Based on Wake-Vortex Modeling Method for Aircraft Scheduling Problem. *Journal of Computational and Applied Mathematics* **2017**, *317*, 157–170, doi:10.1016/j.cam.2016.11.043.
5. B.S., G. An Efficient Hybrid Particle Swarm Optimization Algorithm in a Rolling Horizon Framework for the Aircraft Landing Problem. *Applied Soft Computing* **2016**, *44*, 200–221, doi:10.1016/j.asoc.2016.04.011.
6. Feng, X.R.; Gao, Z.D.; Wang, J.; Wang, X.L.; Hui, K.H. Research on Flight Landing Scheduling Problem Based on Compact Subsequences. J. Beijing Univ. Aeronaut. Astronaut.**2024**, *50*, 2421–2431. https://doi.org/10.13700/j.bh.1001-5965.2022.0656.
7. Beasley, J.E.; Krishnamoorthy, M.; Sharaiha, Y.M.; Abramson, D. Scheduling Aircraft Landings—The Static Case. *Transportation Science* **2000**, *34*, 180–197, doi:10.1287/trsc.34.2.180.12302.
8. Beasley, J.E.; Krishnamoorthy, M.; Sharaiha, Y.M.; Abramson, D. Displacement Problem and Dynamically Scheduling Aircraft Landings. *Journal of the Operational Research Society* **2004**, *55*, 54–64, doi:10.1057/palgrave.jors.2601650.
9. Pinol, H.; Beasley, J.E. Scatter Search and Bionomic Algorithms for the Aircraft Landing Problem. *European Journal of Operational Research* **2006**, *171*, 439–462, doi:10.1016/j.ejor.2004.09.040.

 doi:10.20944/preprints202603.0874.v1

10. Yu, S.-P.; Cao, X.-B.; Zhang, J. A Real-Time Schedule Method for Aircraft Landing Scheduling Problem Based on Cellular Automation. *Applied Soft Computing* **2011**, *11*, 3485–3493, doi:10.1016/j.asoc.2011.01.022.
11. Gui, D.; Le, M.; Luo, X.; Huang, Z. A Metaheuristic Algorithm for Efficient Aircraft Sequencing and Scheduling in Terminal Maneuvering Areas. *Optim Lett* **2025**, *19*, 579–604, doi:10.1007/s11590-024-02151-8.
12. Pamplona, D.A.; Alves, C.J.P. A Fast Heuristic for Aircraft Landing Scheduling with Time Windows: Application to Guarulhos Airport. *Aerospace* **2025**, *12*, 1008, doi:10.3390/aerospace12111008.
13. Chen, X.; Yu, H.; Cao, K.; Zhou, J.; Wei, T.; Hu, S. Uncertainty-Aware Flight Scheduling for Airport Throughput and Flight Delay Optimization. *IEEE Trans. Aerosp. Electron. Syst.* **2020**, *56*, 853–862, doi:10.1109/TAES.2019.2921193.
14. Junfeng Z.; Lubao Y.O.U.; Ming Z.; Chunwei Y.; Bo K. Multi-objective arrival sequencing and scheduling based on point merge system. *bjhkhtdxxb* **2021**, *49*, 66–73, doi:10.13700/j.bh.1001-5965.2021.0199.
15. Wang, J.; Ding, X.; Wang, S. Collaborative Sequencing of Arrival and Departure Aircraft Considering Potential Conflicts. *Journal of Transportation Systems Engineering and Information Technology* **2023**, *23*, 312, doi:10.16097/j.cnki.1009-6744.2023.05.032.
16. Chen, K.; Situ, T.; Fang, Y. An Improved Multi-Objective Restart Variable Neighborhood Search Algorithm for Aircraft Sequencing Problem with Complex Interdependent Runways. *Journal of Air Transport Management* **2025**, *127*, 102807, doi:10.1016/j.jairtraman.2025.102807.
17. Jiang, H.; Liu, J.X.; Zhou, W.S. Bi-level Programming Model for Joint Scheduling of Arrival and Departure Flights Based on Traffic Scenario. Trans. Nanjing Univ. Aeronaut. Astronaut.**2021**, *38* (4), 671–684. doi:10.16356/j.1005-1120.2021.04.013
18. Zhou, D.K. Terminal Area Arrival and Departure Flight Sequencing Based on Reinforcement Learning. Master's thesis, Civil Aviation Flight University of China, Guanghan, China, **2024**
19. Kang, R.; Yang, M.; Lin, Z.Y.; Yang, Z.Y. Optimization of Arrival Flight Sequencing Based on EoR Operations. Sci. Technol. Eng.**2025**, *25*, 8289–8296.DOI:10.12404/j.issn.1671-1815.2405453
20. Pamplona, D.A.; Alves, C.J.P. A Fast Heuristic for Aircraft Landing Scheduling with Time Windows: Application to Guarulhos Airport. *Aerospace* **2025**, *12*, 1008, doi:10.3390/aerospace12111008.
21. Dönmez, K.; Bakır, M.; Cecen, R.K. A Comprehensive Data-Driven MCDM Approach to Determine the Best Single Objective Function for the Aircraft Sequencing and Scheduling Problem. *Expert Systems with Applications* **2026**, *296*, 129172, doi:10.1016/j.eswa.2025.129172.
22. Coulom, R. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In *Computers and Games*; Van Den Herik, H.J., Ciancarini, P., Donkers, H.H.L.M., Eds.; Lecture Notes in Computer Science; Springer Berlin Heidelberg: Berlin, Heidelberg, 2007; Vol. 4630, pp. 72–83 ISBN 978-3-540-75537-1.
23. Silver, D.; Huang, A.; Maddison, C.J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature* **2016**, *529*, 484–489, doi:10.1038/nature16961.
24. Song, W.S.; Ren, B.Y.; Guan, T. Research on Arch Dam Placement Sequencing Based on Deep Monte Carlo Tree Search. J. Hydroelectr. Eng.**2024**, *43*, 120–130.doi:10.11660/slfdxb.20240311
25. Wang, G.; Pu, H.; Song, T.; Li, W.; Zhang, H.; Hu, G.; Wang, G.; Pu, H.; Song, T.; Li, W.; et al. Integrated Method Based on Reinforcement Learning and Monte Carlo Tree Search for Railway Alignment Optimization. *Tiedao Xuebao/Journal of the China Railway Society* **2025**, *47*, 103–110.
26. Peng, J.; Zhu, G.L.; Wu, Q.S.; Li, Y.F.; He, S.; Jin, Y.Y.; Xu, M.L. Scheduling Method for Carrier Aircraft Support Operations Based on Monte Carlo Tree Search. Acta Aeronaut. Astronaut. Sin.**2025**, *0*, 1–17.doi: 10.7527/S1000-6893.2025.32444.
27. Ze Y.U.; Mingying H.U.O.; Yeguang W.; Shipeng W.; Zheng L.I.; Yang Z.; Bo Q.I.; Naiming Q.I. A UAV Mission Planning Method Based on Improved Monte Carlo Tree Search. *yhxb* **2025**, *46*, 874–883, doi:10.3873/j.issn.1000-1328.2025.05.005.
28. Pang, Y.; Zhao, P.; Hu, J.; Liu, Y. Machine Learning-Enhanced Aircraft Landing Scheduling under Uncertainties. Transportation Research Part C: Emerging Technologies 2024, 158, 104444, doi:10.1016/j.trc.2023.104444.

 doi:10.20944/preprints202603.0874.v1

29. Feng, X.R.; Zhang, S.; Qiu, D.L.; Wang, X.L. A Bounding Optimization Method for Solving Flight Landing Scheduling Problems. J. Nanjing Univ. Aeronaut. Astronaut.**2024**, *56*, 1024–1035. https://doi.org/10.16356/j.1005-2615.2024.06.005.
30. Sabar, N.R.; Kendall, G. An Iterated Local Search with Multiple Perturbation Operators and Time Varying Perturbation Strength for the Aircraft Landing Problem. Omega 2015, 56, 88–98, doi:10.1016/j.omega.2015.03.007.