Autonomous Aircraft Sequencing and Separation with Hierarchical Deep Reinforcement Learning

自主飞机序列与分层深度强化学习分离

Marc Brittain
Marc Brittain
Aerospace Engineering Department
航空航天工程系
Iowa State University
爱荷华州立大学
Ames, IA, USA
Ames, IA, USA
mwb@iastate.edu
Peng Wei
彭伟
Aerospace Engineering Department
航空航天工程系
Iowa State University
爱荷华州立大学
Ames, IA, USA
Ames, IA, USA
pwei@iastate.edu

Abstract-With the increasing air traffic density and complexity in traditional controlled airspace, and the envisioned large volume vertical takeoff and landing (VTOL) operations in low-altitude airspace for personal air mobility or on-demand air taxi, an autonomous air traffic control system (a fully automated airspace) is needed as the ultimate solution to handle dense, complex and dynamic air traffic in the future. In this work, we design and build an artificial intelligence (AI) agent to perform air traffic control sequencing and separation. The approach is to formulate this problem as a reinforcement learning model and solve it using the hierarchical deep reinforcement learning algorithms. For demonstration, the NASA Sector 33 app has been used as our simulator and learning environment for the agent. Results show that this AI agent can guide aircraft safely and efficiently through "Sector 33" and achieve required separation at the metering fix.

摘要-随着传统受控空域内空中交通密度的增加和复杂性的提高，以及预计的大规模低空垂直起降 (VTOL) 操作，用于个人空中出行或按需空中出租，一个自主空中交通控制系统 (完全自动化的空域) 是未来处理密集、复杂和动态空中交通的最终解决方案。在这项工作中，我们设计并构建了一个人工智能 (AI) 代理来执行空中交通控制序列和分离。该方法是将此问题构建为一个强化学习模型，并使用分层深度强化学习算法来解决。为了演示，我们使用了 NASA 的 Sector 33 应用程序作为代理的模拟器和学习环境。结果显示，这个 AI 代理能够安全、高效地引导飞机通过 "Sector 33" 并实现所需的计量点分离。

Keywords: Artificial Intelligence, Autonomous Air Traffic Control, Hierarchical Deep Reinforcement Learning

关键词: 人工智能，自主空中交通控制，分层深度强化学习

I. INTRODUCTION

I. 引言

A. Motivation

A. 研究动机

The original proposal of an autonomous air traffic control system was from Heinz Erzberger and his NASA colleagues, where they believe that a fully automated system, referred as the Advanced Airspace Concept (AAC), is the ultimate solution to accommodate dense, complex and dynamic air traffic in the controlled airspace in the future. The core element of the AAC is called the Autoresolver. It was designed to detect and resolve conflicts in en route and terminal airspace. In their papers [1, 2 , 3], such an autonomous air

traffic control system for automated sequencing and separation is expected to augment human air traffic controllers to increase airspace capacity and enhance operation safety.

最初提出自主空中交通控制系统的建议来自 Heinz Erzberger 及其 NASA 同事，他们认为完全自动化的系统，即所谓的先进空域概念 (Advanced Airspace Concept, AAC)，是未来容纳受控空域中密集、复杂和动态空中交通的最终解决方案。AAC 的核心元素称为自动解析器 (Autoresolver)。它被设计用于检测并解决航路和终端空域中的冲突。在他们的论文 [1, 2 和 [3] 中，这种自动化的空中交通控制系统预计将增强人工空中交通管制员的作业，以提高空域容量和提升运行安全性。

In the recent proposals for low-altitude airspace operations, including UAS Traffic Management (UTM) [4, 5, 6] for remote-piloted or unmanned autonomous drone operations and Urban Air Mobility (UAM) [7, 8, 9, 10] for vertical takeoff and landing (VTOL) personal air travel or urban air taxi operations, an autonomous air traffic control system is needed to communicate with future intelligent aircraft, facilitate on-board autonomy or human operator decisions, and cope with envisioned high-density and on-demand air traffic by providing automated sequencing and separation advisories.

在近期关于低空空域运行的提案中，包括针对遥控或无人自主无人机运行的无人机交通管理 (UTM)[4, 5, 6]，以及针对垂直起降 (VTOL) 个人空中旅行或城市空中出租车运行的 Urban Air Mobility(UAM)[7, 8, 9, 10]，需要一种自主空中交通控制系统来与未来智能飞机通信，便利机上自主或人工操作员的决策，并通过提供自动化的排序和分离建议来应对预期的高密度和按需空中交通。

The inspiration of this paper is twofold. First, the authors were amazed by the fact that an artificial intelligence agent called AlphaGo built by DeepMind defeated the world champion Ke Jie in three matches of Go in May 2017 [11]. This notable advance in AI field demonstrated the theoretical foundation and computational capability to potentially augment and facilitate human tasks with intelligent agents and AI technologies. Therefore, the authors want to apply the most recent AI frameworks and algorithms to revisit the autonomous air traffic control idea. Second, the authors were granted the software called NASA Sector 33 for education and outreach purposes [12]. It is an air traffic control game designed to interest students in air transportation and aviation related careers. It contains 35 problems featuring two to five aircraft. The player needs to apply speed and route controls over these aircraft to satisfy sequencing and separation requirements, as well as, minimize delay. The authors decide to use this software as the game environment and simulator for performance evaluation of our proposed framework and algorithms.

本文的灵感来源于两个方面。首先，作者们对 DeepMind 公司开发的人工智能程序 AlphaGo 在 2017 年 5 月击败世界冠军柯洁的三场围棋比赛感到震惊 [11]。这一在人工智能领域的显著进步展示了理论基础的建立和计算能力，这些能力有可能通过智能代理和人工智能技术增强并促进人类任务的完成。因此，作者希望应用最新的人工智能框架和算法来重新审视自主空中交通控制的想法。其次，作者们为了教育和推广目的获得了名为 NASA Sector 33 的软件 [12]。这是一个空中交通控制游戏，旨在吸引学生对航空运输和航空相关职业的兴趣。游戏包含 35 个问题，涉及两到五架飞机。玩家需要对这些飞机应用速度和航线控制以满足排序和分离要求，同时最小化延迟。作者决定使用这个软件作为游戏环境和模拟器，以评估我们提出的框架和算法的性能。

In this paper, a hierarchical deep reinforcement learning framework is proposed to enable autonomous air traffic sequencing and separation, where the input of this agent is the air traffic controller's screen. Through the computer vision techniques, the agent will "see" the screen, comprehend the air traffic situation, and perform online sequential decision making to select actions including speed and route option for each aircraft in real time with ground-based computation. Unlike Erzberger's approach, in our paper the reinforcement learning agent integrates conflict detection and conflict resolution together via a deep reinforcement learning approach. Like Erzberger, we also assume that the computed speed and route advisories will be sent from ground to aircraft via data link. The series of actions will guide the aircraft quickly through "Sector 33" and maintain required safe separation. Our proposed framework and algorithms provide another promising potential solution to enable autonomous air traffic control system.

在本文中，提出了一个分层深度强化学习框架，用于实现自主空中交通排序和分离，其中该代理的输入是空中交通管制员的屏幕。通过计算机视觉技术，代理将“看到”屏幕，理解空中交通情况，并进行在线顺序决策，实时为每架飞机选择包括速度和航线选项在内的行动，并进行地面计算。与 Erzberger 的方法不同，在我们的论文中，强化学习代理通过深度强化学习方法将冲突检测和冲突解决集成在一起。与 Erzberger 一样，我们也假设计算出的速度和航线建议将通过数据链从地面发送到飞机。这一系列行动将引导飞机快速通过“Sector 33”并保持所需的安全间隔。我们提出的框架和算法为实现在自主空中交通控制系统中提供了另一种有前景的潜在解决方案。

B. Related Work

B. 相关工作

There have been many important contributions to the topic of autonomous air traffic control. One of the most promising and well-known lines of work is the Autoresolver designed and developed by Heinz Erzberger and his NASA colleagues [1, 2, 3]. It employs an iterative approach, sequentially computing and evaluating candidate trajectories, until a trajectory is found that satisfies all of the resolution conditions. The candidate trajectory is then output by the algorithm as the conflict resolution trajectory. The Autoresolver is a physics based approach that involves separate components of conflict detection and conflict resolution. It has been tested in various large-scale simulation scenarios. In addition, the Autoresolver is being verified and validated by NASA researchers using formal methods.

在自主空中交通控制领域有许多重要的贡献。其中最具有前景且广为人知的工作之一是由 Heinz Erzberger 及其 NASA 同事设计并开发的 Autoresolver [1, 2, 3]。它采用迭代方法，依次计算和评估候选轨迹，直到找到满足所有解决条件的轨迹。然后算法输出候选轨迹作为冲突解决轨迹。Autoresolver 是一种基于物理的方法，涉及冲突检测和冲突解决分开的组件。它已经在各种大规模仿真场景中进行了测试。此外，NASA 研究人员正在使用形式化方法对 Autoresolver 进行验证和确认。

Reinforcement learning and deep Q-networks have been demonstrated to play games such as Go, Atari and Warcraft [13, 14, 15] . The results from these papers show that a well-designed, sophisticated AI agent is capable of performing high-level tasks, as well as, learning complex strategies. Therefore, we are encouraged to apply the reinforcement learning framework to solve an air traffic control game and set up an environment for the AI agent to learn the fundamental air traffic control tasks, i.e., aircraft sequencing and separation.

强化学习和深度 Q 网络已经在玩 Go、Atari 和 Warcraft [13, 14, 15] 等游戏方面得到演示。这些论文的结果表明，设计精良、复杂的人工智能代理能够执行高级任务，同时还能学习复杂的策略。因此，我们受到鼓舞，将强化学习框架应用于解决空中交通控制游戏，并建立一个环境，让 AI 代理学习基本的空中交通控制任务，即飞机排序和分离。

In this paper, the reinforcement learning framework and a novel hierarchical deep agent algorithm are developed to solve the sequencing and separation problem with delay minimization for autonomous air traffic control. The results show that the algorithm has very promising performance.

在本文中，开发了一种强化学习框架和一种新颖的分层深度代理算法，用于解决自主空中交通控制的排序和分离问题，并最小化延迟。结果显示，该算法具有非常令人期待的性能。

The structure of this paper is as follows: in Section II, the background of reinforcement learning and deep Q-network will be introduced. In Section III, the description of the problem and its mathematical formulation of reinforcement learning are presented. Section IV presents our designed hierarchical deep agent algorithm to solve this problem. The numerical experiment and results are shown in Section V. And Section VI concludes this paper.

本文的结构如下: 在第二部分，将介绍强化学习和深度 Q 网络的相关背景。在第三部分，将呈现问题的描述以及强化学习的数学公式化。第四部分介绍我们设计的分层深度代理算法来解决此问题。第五部分展示了数值实验和结果。第六部分对本论文进行总结。

II. BACKGROUND

II. 背景

A. Reinforcement Learning

A. 强化学习

Reinforcement learning is one type of sequential decision making where the goal is to learn how to act optimally in a given environment with unknown dynamics. A reinforcement learning problem involves an environment, an agent, and different actions the agent can take in this environment. The agent is unique to the environment and we assume the agent is only interacting with one environment. Let t represent the current time, then the components that make up a reinforcement learning problem are as follows:

强化学习是一种顺序决策的类型，其目标是学习如何在给定的未知动态环境中如何最优地行动。一个强化学习问题包括环境、代理以及代理在这个环境中可以采取的不同动作。代理对该环境是唯一的，我们假设代理只与一个环境交互。令 t 表示当前时间，那么构成强化学习问题的组成部分如下:

- S - The state space S is a set of all possible states in the environment
- S - 状态空间 S 是环境中所有可能状态的集合
- A - The action space A is a set of all actions the agent can take in the environment
- A - 动作空间 A 是代理在环境中可以采取的所有动作的集合
- $\mathrm{r}\left(s_t, a_t, s_{t+1}\right)$ - The reward function determines how much reward the agent is able to acquire for a given $\left(s_t\right.$, $\left.a_t, s_{t+1}\right)$ transition
- $\mathrm{r}\left(s_t, a_t, s_{t+1}\right)$ - 奖励函数决定了代理在给定的 $\left(s_t\right.$, $\left.a_t, s_{t+1}\right)$ 转变中能够获得多少奖励
- $\gamma \in [0,1]$ - A discount factor determines how far in the future to look for rewards. As $\gamma \rightarrow 0$, only immediate rewards are considered, whereas, when $\gamma \rightarrow 1$, future rewards are getting prioritized.
- $\gamma \in [0,1]$ - 折扣因子决定了要考虑未来的奖励有多远。当 $\gamma \rightarrow 0$ 时，只考虑即时奖励，而当 $\gamma \rightarrow 1$ 时，未来的奖励被优先考虑。

S contains all information about the environment and each element s_{t+1} can be considered a snapshot of the environment at time t . The agent accepts s_t and with this, the agent then decides an action, a_t . By taking action a_t , the state is now updated to s_{t+1} and there is an associated reward from making the transition from $s_t \rightarrow s_{t+1}$. How the state evolves from $s_t \rightarrow s_{t+1}$ given action a_t is dependent upon the dynamics of the system, which is unknown. The reward function is user defined, but needs to be carefully designed to reflect the goal of the agent. Fig. 1 shows the progression of a reinforcement learning problem.

S 包含了关于环境的所有信息，每个元素 s_{t+1} 可以被视为在时间 t 的环境快照。代理接受 s_t 并在此基础上决定一个动作 a_t 。通过采取动作 a_t ，状态现在更新为 s_{t+1} ，并且有一个与从 $s_t \rightarrow s_{t+1}$ 过渡相关的奖励。状态如何从 $s_t \rightarrow s_{t+1}$ 给定动作 a_t 发展取决于系统的动态，这是未知的。奖励函数由用户定义，但需要仔细设计以反映代理的目标。图 1 显示了强化学习问题的进展。

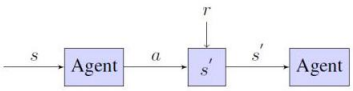

Figure 1. Progression of a reinforcement learning problem within an environment.

图 1. 在环境中强化学习问题的进展。

From this framework, the agent is able to learn the optimal decisions in each state of the environment by maximizing a cumulative reward function. We call the sequential actions the agent makes in the environment a policy. Let π represent some policy and T represent the total time for a given environment, then the optimal policy can be defined as:

从这个框架出发，智能体能够通过最大化累积奖励函数来学习环境中每个状态的最优决策。我们将智能体在环境中采取的顺序动作称为策略。设 π 表示某种策略，T 表示给定环境的总时间，那么最优策略可以定义为:

$$\pi^* = \arg\max_{\pi} \mathrm{E}\left[\sum_{t=0}^{T}\left(r\left(s_t, a_t, s_{t+1}\right) \mid \pi\right)\right]. \tag{1}$$

If we define the reward for actions we deem "optimal" very high, then by maximizing the total reward, we have found the optimal solution to the problem.

如果我们将我们认为的“最优”动作的奖励设置得很高，那么通过最大化总奖励，我们就找到了问题的最优解。

B. Q-Learning

B. Q-学习

One of the most fundamental reinforcement learning algorithms is known as Q-learning. This popular learning algorithm was introduced by Watkins and the goal is to maximize a cumulative reward by

selecting an appropriate action in each state [16]. The idea of Q-learning is to estimate a value Q for each state and action pair(s, a)in an environment that directly reflects the future reward associated with taking such an action in this state. By doing this, we can extract the policy that reflects the optimal actions for an agent to take. The policy can be thought of as a mapping or a look-up table, where at each state, the policy tells the agent which action is the best one to take. During each learning iteration, the Q-values are updated as follows:

最基本的强化学习算法之一被称为 Q-学习。这个流行的学习算法由 Watkins 提出，目标是通过对每个状态选择一个适当的行为来最大化累积奖励 [16]。Q-学习的思想是为环境中的每个状态和动作对 (s, a) 估计一个值 Q ，这个值直接反映了在这个状态下采取这样的动作将带来的未来奖励。通过这种方式，我们可以提取出反映智能体应采取的最优行为的策略。策略可以被视为一种映射或查找表，在每种状态下，策略告诉智能体哪个动作是最好的。在每次学习迭代中，Q 值将按以下方式更新:

$$Q\left(s_t, a_t\right) \leftarrow Q\left(s_t, a_t\right) +$$

$$\alpha\left(r+\gamma \max_{a_{t+1}} Q\left(s_{t+1}, a_{t+1}\right)-Q\left(s_t, a_t\right)\right). \tag{2}$$

In (2), α represents the learning rate, r represents the reward for a given state and action, and γ represents the discount factor. One can see that in the max $Q\left(s_{\mathrm{t}+1}, a_{\mathrm{t}+1}\right)$ term, the idea is to determine the best possible future reward by taking this action.

在公式 (2) 中，α 表示学习率，r 表示给定状态和动作的奖励，γ 表示折扣因子。可以注意到，在 max $Q\left(s_{\mathrm{t}+1}, a_{\mathrm{t}+1}\right)$ 项中，想法是通过采取这个动作来确定可能获得的最佳未来奖励。

C. Deep Q-Network (DQN)

C. 深度 Q 网络 (DQN)

While Q-learning performs well in environments where the state-space is small, as the state-space begins to increase, Q-learning becomes intractable. It is because there is now a need for more experience (more game episodes to be played) in the environment to allow convergence of the Q-values. To obtain Q-value estimates in environments where the state-space is large, the agent must now generalize from limited experience to states that may have not been visited [17]. One of the most widely used function approximation techniques for Q-learning is deep Q-networks (DQN), which involves using a neural network to approximate the Q-values for all the states. With standard Q-learning, the Q-value was a function of $Q\left(s, a\right)$, but with DQN the Q-value is now a function of $Q\left(s, a, \theta\right)$, where θ is the parameters of the neural network. Given an n - dimensional state-space with an m -dimensional action space, the neural network creates a map from $\Re^n \rightarrow \Re^m$. As mentioned by Van Hasselt et al., incorporating a target network and experience replay are the two main ingredients for DQN [18]. The target network with parameters θ^- , is equivalent to the online network, but the weights (θ^-) are updated every τ time steps. The target used by DQN can then be written as:

当状态空间较小时，Q 学习在环境中表现良好，但随着状态空间的增大，Q 学习变得不可处理。这是因为现在需要更多的经验 (在环境中进行更多的游戏回合) 以允许 Q 值的收敛。为了在状态空间较大的环境中获得 Q 值估计，代理必须从有限的经验中泛化到可能未访问过的状态 [17]。Q 学习中最广泛使用的函数逼近技术之一是深度 Q 网络 (DQN)，它涉及使用神经网络来逼近所有状态的 Q 值。在标准 Q 学习中，Q 值是 $Q\left(s, a\right)$ 的函数，但在 DQN 中，Q 值现在变成了 $Q\left(s, a, \theta\right)$ 的函数，其中 θ 是神经网络的参数。给定一个 n 维状态空间和 m 维动作空间，神经网络创建了一个从 $\Re^n \rightarrow \Re^m$ 的映射。正如 Van Hasselt 等人所提到的，引入目标网络和经验回放是 DQN 的两大主要组成部分 [18]。目标网络带有参数 θ^- ，与在线网络等价，但其权重 (θ^-) 每隔 τ 时间步更新一次。DQN 使用的目标可以写成:

$$Y_t^{DQN} = r_{t+1} + \gamma \max_{a_{t+1}} Q\left(s_{t+1}, a_{t+1}; \theta_t^-\right). \tag{3}$$

The idea of experience replay is that for a certain amount of time, observed transitions are stored and then sampled uniformly to update the network. By incorporating the target network, as well as, experience replay, this can drastically improve the performance of the algorithm [18].

经验回放的想法是，在一段时间内，观察到的转换被存储起来，然后均匀抽样以更新网络。通过引入目标网络以及经验回放，这可以显著提高算法的性能 [18]。

D. Double Deep Q-Network (DDQN)

D. 双重深度 Q 网络 (DDQN)

In Q-learning and DQN there is the use of a max operator to select which action results in the largest potential future reward. Van Hasselt et al. showed that due to this max operation, the network is more likely to overestimate the values, resulting in overoptimistic Q-value estimations [19]. The idea introduced by Hasselt was to decouple the max operation to prevent this overestimation to create what is called double deep Q-network (DDQN) [20]. To decouple the max operator, a second value function must be introduced, including a second network with weights θ' . During each training iteration, one set of weights determines the greedy policy and the other then determine the Q-value associated. Formulating (3) as a DDQN problem:

在 Q 学习和 DQN 中，使用了最大值操作符来选择导致最大潜在未来奖励的动作。Van Hasselt 等人表明，由于这个最大值操作，网络更有可能高估价值，导致过于乐观的 Q 值估计 [19]。Hasselt 提出的想法是将最大值操作分离，以防止这种高估，从而创建了所谓的双深度 Q 网络 (DDQN)[20]。为了分离最大值操作符，必须引入第二个价值函数，包括一个带有权重 θ' 的第二个网络。在每次训练迭代中，一组权重确定贪婪策略，另一组则确定关联的 Q 值。将 (3) 公式化为 DDQN 问题:

$$Y_t^{DDQN} = r_{t+1} + \gamma Q\left(s_{t+1}, \arg\max_{a_{t+1}} Q\left(s_{t+1}, a_{t+1}; \theta_t\right); \theta_t'\right). \tag{4}$$

In (4), it can be seen that the max operator has been removed and we are now including an argmax function to determine the best action due to the online weights. We then use that action, along with the second set of weights to determine the estimated Q-value.

在 (4) 中，可以看到最大值操作符已被移除，我们现在包含一个 argmax 函数来确定由于在线权重而产生的最佳动作。然后我们使用这个动作以及第二组权重来确定估计的 Q 值。

III. PROBLEM FORMULATION

III. 问题公式化

A. Problem Statement

A. 问题陈述

In real world practice, the air traffic controllers in en route and terminal sectors are responsible for sequencing and separating aircraft. In our research, we used the NASA Sector 33 web-based application (a video game) as our air traffic control simulator. The application has a set of problems whose solutions are difficult to find. To evaluate the performance of our hierarchical deep agent algorithm, we selected three different game scenarios with various difficulty levels and constraints.

在实际操作中，航路和终端区的空中交通管制员负责飞机的排序和分离。在我们的研究中，我们使用了 NASA Sector 33 基于网络的应用程序 (一种视频游戏) 作为我们的空中交通控制模拟器。该应用程序有一组难以找到解决方案的问题。为了评估我们分层深度代理算法的性能，我们选择了三种不同难易程度和约束的游戏场景。

1) Objective: The objective in the NASA Sector 33 game environment is to maintain a safe separation between aircraft, resolve conflict, and minimize delay by making appropriate route changes and providing speed advisories. Ultimately, we want to guide each aircraft quickly through the metering fix. Unlike the real-world air traffic control scenario, in this game environment there is also a unique "final metered position" right after the metering fix for each aircraft. It is not too difficult for a good human player to obtain a feasible solution in the game, where the aircraft are close to their final metered position, safe separation is maintained, and conflicts are resolved. In order to obtain the optimal solution in this environment, the aircraft have to maintain safe separation, resolve conflict, and arrive at their final metered position with no delay. However, obtaining the optimal solution in this environment is much more difficult than obtaining the feasible solution, due to the fact that each aircraft has to follow the optimal speed and route at every time-step. If one speed change is made incorrectly, then the optimal solution will not be achieved.

1) 目标: 在 NASA Sector 33 游戏环境中，目标是保持飞机之间的安全距离，解决冲突，并通过适当的航线变更和提供速度建议来最小化延误。最终，我们希望引导每架飞机快速通过计量修正点。与现实世界的空中交通管制场景不同，在这个游戏环境中，每架飞机在计量修正点之后还有一个独特的“最终计量位置”。对于优秀的玩家来说，在游戏中获得一个可行解并不困难，在这种情况下，飞机接近它们的最终计量位置，保持安全距离，并解决冲突。为了在这个环境中获得最优解，飞机必须保持安全距离，解决冲突，并且不延误地到达它们的最终计量位置。然而，由于每架飞机在每一个时间步都必须遵循最优速度和航线，因此在游戏中获得最优解比获得可行解要困难得多。如果速度变更有一次错误，那么就无法实现最优解。

2) Constraints: There are many constraints in the NASA Sector 33 game environment to help resemble a real-world air traffic environment. For each problem, there is a fixed number of aircraft. For some problems, there is only two aircraft, some there is three aircraft, and this number increased to a maximum of five aircraft. The next constraint imposed a limit on the number of route changes for a given aircraft. For example, one problem might allow for a single aircraft to change routes, while another problem would allow both aircraft to change routes, thus increasing the complexity. Weather also imposed an additional limit on the number of aircraft route changes. In some of the problems, there is a storm blocking one of the routes, so now there is no option to select the corresponding route anymore. One of the strictest constraints in the game is time. In each problem, there is a timer for how long the episode lasts and there is only enough time for the optimal solution to be obtained. This meant that to obtain the optimal solution, the aircraft has to be at their final metered position when the timer is at 0:00, otherwise it is not an optimal solution. The last constraint in the game is the individual speed of each aircraft. Each aircraft has the ability to fly at six different speeds ranging from 300 knots to 600 knots.

2) 约束条件: 在 NASA Sector 33 游戏环境中，有许多约束条件以模拟现实世界的空中交通环境。每个问题中都有固定数量的飞机。某些问题中只有两架飞机，某些问题中有三架飞机，这个数字最多增加到五架飞机。接下来的约束条件是对给定飞机的航线变更次数的限制。例如，一个问题可能允许一架飞机改变航线，而另一个问题则允许两架飞机都改变航线，从而增加复杂性。天气也对飞机航线变更次数提出了额外的限制。在某些问题中，一场风暴阻碍了其中一条航线，因此现在无法选择相应的航线。游戏中最为严格的约束之一是时间。每个问题都有一个计时器，用于计算情节的持续时间，且只有足够的时间获得最优解。这意味着为了获得最优解，飞机必须在计时器显示 0:00 时处于其最终的计量位置，否则就不算是最优解。游戏中的最后一个约束是每架飞机的个体速度。每架飞机都能以六种不同的速度飞行，范围从 300 节到 600 节。

B. Reinforcement Learning Formulation

B. 强化学习公式化

Here we formulate the NASA Sector 33 environment as a reinforcement learning problem and define the state-space, action-space, and reward functions for the parent agent, as well as, the child agent.

在这里，我们将 NASA Sector 33 环境公式化为一个强化学习问题，并定义了父代理和子代理的状态空间、动作空间和奖励函数。

1) State Space: A state contains all the information the AI agent needs to make decisions. The state information was composed of different information for the parent agent and child agent. For the parent agent, the information included in the state was a screen-shot of the game screen. For the child agent, if we let i represent a given aircraft, then the information included in the state was: aircraft positions (x_i, y_i) , aircraft speeds v_i , and route information. Route information included the combination of routes for both aircraft, defined as C_j , where j represents a given route combination. From this, we can see that the state-space for the parent agent is constant, since it only depends on the number of pixels in the screen-shot. Suppose there are $m \times m$ pixels in the screen-shot and n number of aircraft, then the state-space can be represented with $m \times m$ numbers for the parent agent and $2 \times n + n + 1$ for the child agent. Fig. 2 shows an example of a state in the NASA Sector 33 game environment.

1) 状态空间: 状态包含 AI 代理做出决策所需的所有信息。状态信息由不同信息组成，对于父代理和子代理而言。对于父代理，状态中包含的信息是游戏屏幕的截图。对于子代理，如果我们让 i 表示一个给定的飞机，那么状态中包含的信息是: 飞机位置 (x_i, y_i) ，飞机速度 v_i 和航线信息。航线信息包括两个飞机的路线组合，定义为 C_j ，其中 j 表示一个给定的路线组合。由此我们可以看出，父代理的状态空间是固定的，因为它只依赖于截图中的像素数量。假设截图中有 $m \times m$ 个像素和 n 架飞机，那么状态空间可以用父代理的 $m \times m$ 个数字和子代理的 $2 \times n + n + 1$ 个数字来表示。图 2 显示了 NASA Sector 33 游戏环境中的一个状态示例。

If we consider Fig. 2 as an example, we can acquire all of the state information we need from the

game-screen. For the parent agent, the state will be represented as follows:

如果我们以图 2 为例，我们可以从游戏屏幕获取所需的所有状态信息。对于父代理，状态将如下表示:

$$S_P = (p_1, p_2, p_3, \ldots, p_{m\times m}),$$

where p_k represents the intensity tuple of pixel k . For each pixel in the game screen, there is an associated pixel intensity tuple (red, green, blue) that contains the intensity for each color ranging from $[0, 255]$. For example, the color green is represented as(0,255,0), the color blue is represented as $(0, 0$, 255) and the color red is represented as(255,0,0). For the child agent, the state is defined as:

其中 p_k 表示像素 k 的强度元组。游戏屏幕中的每个像素都有一个相关的像素强度元组 (红色、绿色、蓝色)，其中包含每种颜色的强度，范围从 $[0, 255]$ 。例如，绿色表示为 (0,255,0)，蓝色表示为 $(0, 0$, 255) ，红色表示为 (255,0,0)。对于子代理，状态定义为:

$$S_C = (x_1, y_1, x_2, y_2, v_1, v_2, C_j),$$

where the subscript represents a specific aircraft and j is the combination of route that the aircraft will take.

其中下标表示特定的飞机，j 是飞机将要采取的路线组合。

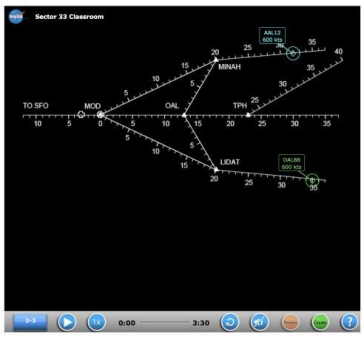

Figure 2. Example of a state in the NASA Sector 33 game environment.

图 2。NASA Sector 33 游戏环境中的一个状态示例。

2) Action Space: At each time-step, the parent agent and child agent can make a decision to change the route of the aircraft and change the speeds of the aircraft, respectively. The only difference is the decision time-step for the parent agent and child agent. The parent agent takes one action every episode, where an episode is defined as an entire play through the game. The child agent takes one action every four seconds within the episode to provide more control over the aircraft once the route combination is determined. The action-space for the parent agent can be defined as follows:

2) 动作空间: 在每一个时间步，父代理和子代理可以做出决策，分别改变飞机的航线和飞机的速度。唯一的区别是父代理和子代理的决策时间步。父代理在每一幕中采取一个动作，其中一幕被定义为游戏的完整进行。子代理在幕内每四秒采取一个动作，以在确定航线组合后提供对飞机的更多控制。父代理的动作空间可以定义如下:

$$A_P = (C_1, \ldots, C_j)\, \forall j,$$

where j is the number of route combinations for the aircraft. If we consider the example in Fig. 2, the action-space for the parent agent will be:

其中 j 是飞机的航线组合数量。如果我们考虑图 2 中的示例，父代理的动作空间将是:

$$A_C = (C_1, C_2, C_3, C_4).$$

This is because each aircraft can take two unique routes, which equates to four unique route combinations.

这是因为每架飞机可以采取两条独特的航线，这等于四种独特的航线组合。

For the child agent, the action-space is defined as:

对于子代理，动作空间定义为:

$$A_C = (U_1, \ldots, U_k)\, \forall k,$$

where we define U as all of the possible combinations of speeds for the aircraft and k as a unique speed combination.

其中我们定义 U 为飞机的所有可能速度组合，k 为一个独特的速度组合。

3) Terminal State: Termination in the episode could be achieved in three different ways:

3) 终止状态: 在幕中的终止可以通过三种不同的方式实现:

- Goal reached (optimal) - All aircraft made it to their final metered position, maintained safe separation, and avoided collision. This meant that:
- 目标达成 (最佳)- 所有飞机都到达了它们最终的计量位置，保持了安全间隔，并避免了碰撞。这意味着:

$$|g_{x_i} - x_i| = 0, \forall i$$

- Out of time (feasible) - The aircraft did not arrive at their final metered position, but might have given more time.
- 时间耗尽 (可行)- 飞机没有到达它们最终的计量位置，但如果给予更多时间可能会到达。

$$|g_{x_i} - x_i| > 0, \forall i$$

- Collision - The aircraft collided with one another.
- 碰撞- 飞机之间发生了碰撞。

$$\sqrt{(y_j - y_i)^2 + (x_j - x_i)^2} < \delta, \forall i \neq j$$

where δ is in terms of the pixel distance, and g_x is defined as the set of goal positions for each aircraft.

其中 δ 是以像素距离来计算的，g_x 被定义为每个飞机的目标位置集合。

By observing the current state, we were able to see if any of the terminal states were obtained. For example, let the current state be defined as $(x_1, y_1, x_2, y_2, v_1, v_2, \mathrm{C}_j)$, then if

通过观察当前状态,我们能够判断是否获得了任何终端状态。例如,假设当前状态定义为 $(x_1, y_1, x_2, y_2, v_1, v_2, \mathrm{C}_j)$ ，那么如果

$$|g_{x_1} - x_1| + |g_{x_2} - x_2| = 0,$$

we have obtained the optimal solution to the problem.

我们已经获得了问题的最优解。

4) Reward Function: The reward function for the parent agent and child agent needed to be designed to reflect the goal of this paper: safe separation, minimizing delay of arriving at final metered position, and choosing the optimal route combination. We were able to capture our goals in the following reward functions for the parent agent and child agent:

4) 奖励函数: 为了反映本文的目标，父代理和子代理的奖励函数需要被设计为: 安全分离，最小化到达最终计量位置的时间延迟，以及选择最优的路线组合。我们能够在以下父代理和子代理的奖励函数中捕捉到我们的目标:

Parent agent

父代理

$$r_P = \frac{1}{\sum_{i=1}^{N} |g_{x_i} - x_i|}$$

Child agent

子代理

$$r_C = 0.001 \sum_{i=1}^{N} v_i - 0.6,$$

where N is the number of aircraft. In the reward for the child agent, we included two constants (0.001 and -0.6). The reason for adding the factor of 0.001 is to scale the rewards between [- 1, 1] . The addition of 0.6 is to penalize slower aircraft speeds and to reward faster aircraft speeds. These rewards were obtained at each time-step for the parent and child agent. If a terminal state was reached, then there was an additional reward that was added for each scenario: -10 for collision, -3 for out of time, and +10 for optimal solution. With these reward functions, the AI agent will prioritize choosing the route combination that allows the aircraft to arrive as quickly as possible, as well as, choosing the fastest speed for the aircraft without creating any conflict.

其中 N 是飞机的数量。在子代理的奖励中，我们包含了两个常数 (0.001 和 -0.6)。添加因子 0.001 的原因是将奖励缩放到 [- 1, 1] 之间。添加 0.6 是为了惩罚较慢的飞机速度，并奖励较快的飞机速度。这些奖励是父代理和子代理在每个时间步获得的。如果达到终端状态，则每个场景都会增加额外的奖励: 碰撞为 -10，超时为 -3，最优解为 +10。有了这些奖励函数，AI 代理将优先选择允许飞机尽快到达的路线组合，同时选择飞机的最快速度而不产生任何冲突。

IV. SOLUTION APPROACH

IV. 解决方案方法

To solve the NASA Sector 33 environment, we designed and developed a novel reinforcement learning algorithm called the hierarchical deep agent. In this section, we introduce and describe the algorithm, then we explain why this algorithm is needed to solve this game.

为了解决 NASA Sector 33 环境，我们设计并开发了一种新的强化学习算法，称为分层深度代理。在本节中，我们介绍并描述该算法，然后解释为什么需要这种算法来解决这个游戏。

A. Hierarchical Deep Agent

A. 分层深度代理

To formulate this environment as a reinforcement learning problem, we found that we were unable to formulate this environment as a typical single agent environment due to the non-Markovian property that this problem involves. The route change for an aircraft can only happen during a small window of time, therefore, if formulated as a single agent problem, the action of changing routes would not be chosen nearly as often leading to a very slow convergence time.

为了将这个环境公式化为强化学习问题，我们发现由于这个问题涉及的非马尔可夫性质，我们无法将其公式化为典型的单一智能体环境。飞机的航线变更只能在很小的时间窗口内发生，因此，如果将其公式化为单一智能体问题，改变航线的行动几乎不会被选择，导致收敛时间非常慢。

To solve this problem, we used a parent agent who has a second agent nested within. The parent agent will take an action (changing route) and then the child agent will control the actions of changing speeds. One of the important differences between the parent and child agents is the state-space. The state-space for the parent agent can be represented as screen pixels or in terms of the aircraft positions and speeds. The child agent has the same state as the parent agent if we use the aircraft positions and speeds, but we also add another dimension to the child agent state. This new dimension we add is the action of the parent agent.

为了解决这个问题，我们使用了一个包含第二个子智能体的父智能体。父智能体将采取一个行动 (改变航线)，然后子智能体将控制改变速度的行动。父智能体与子智能体之间一个重要的区别是状态空间。父智能体的状态空间可以表示为屏幕像素或飞机位置和速度。如果使用飞机位置和速度，子智能体的状态与父智能体相同，但我们还在子智能体状态中增加了一个维度。这个新增加的维度是父智能体的行动。

We do this to decouple the action sets of changing route and changing speed. Then we can have the parent agent take one action in the beginning of the episode, followed by the child agent adding this action to its state and proceeding with the actions of changing speed. Fig. 3 provides a diagram of the progression of information from the parent agent to the child agent. We can see the initial state s_P is input into the parent agent, then the parent agent takes action a . From there, action a is now included into the child agent state, s_C . From there, the child agent is able to make all successive decisions with this information. Algorithm 1 provides a pseudocode for formulating a hierarchical deep reinforcement learning problem.

我们这样做是为了解耦改变航线和改变速度的行动集合。然后我们可以让父智能体在剧集开始时采取一个行动，接着子智能体将这个行动加入其状态并继续进行改变速度的行动。图 3 提供了一个从父智能

体到子智能体信息进展的图表。我们可以看到初始状态 s_P 输入到父智能体，然后父智能体采取行动 a 。从这里，行动 a 现在包含在子智能体状态中，s_C 。从这里，子智能体能够利用这些信息做出所有后续决策。算法 1 提供了构建分层深度强化学习问题的伪代码。

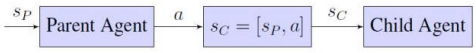

Figure 3. Progression from Parent agent to Child agent.
图 3. 从父智能体到子智能体的进展。

Algorithm 1 Hierarchical Deep Agent
算法 1 分层深度智能体
Initialize: Parent Agent
初始化: 父智能体
Initialize: Child Agent
初始化: 子智能体
Initialize: s_P
初始化: s_P
reward $= 0$
奖励 $= 0$
number of episodes $= n$
回合数 $= n$
for $i = 1$ to n do
对于 $i = 1$ 到 n 执行
$a_P =$ ChooseAction(ParentAgent)
$a_P =$ 选择动作 (ParentAgent)
$s_C = [s_P, a_C]$
repeat
重复
$a_c =$ ChooseAction(ChildAgent)
$a_c =$ 选择动作 (ChildAgent)
$s', r_C =$ SimulateEnvironment
$s', r_C =$ 模拟环境
receive parent agent reward $= r_P(s')$
接收父代理奖励 $= r_P(s')$
reward $=$ reward $+r$
奖励 $=$ 奖励 $+r$
update(ChildAgent)
更新 (ChildAgent)
until Terminal
直到终端
update(ParentAgent)
更新 (父代理)
end for
结束循环

B. Overall Approach

B. 总体方法

In our formulation of the hierarchical deep agent, we decided to use the game screen (raw pixels) as input for the parent agent. By using the game screen, the parent agent can ”see” and ”comprehend” the air traffic situation by abstracting the important features through the hidden layers of double deep Q-network (DDQN), such as relative aircraft positions without explicitly providing the information. Then the parent agent is able to make route selections for all aircraft at the output layer. In this way, the parent agent DDQN integrates the conflict detection task (”seeing” and ”comprehension”) and first part of conflict resolution task of route selection (”decision making”). Fig. 4 shows an illustrative example of the parent agent architecture. In Fig. 4, the ”hidden layer” is a simplified illustration to represent

the entire DDQN. In this specific game scenario shown in Fig. 4, we have two aircraft to control. The upper one has two route options, and the lower one has only one route option. Therefore, the total route combinations for these two aircraft is two.

在我们对分层深度代理的表述中，我们决定使用游戏屏幕 (原始像素) 作为父代理的输入。通过使用游戏屏幕，父代理可以“看到”并“理解”空中交通情况，通过双层深度 Q 网络 (DDQN) 的隐藏层抽象出重要特征，例如相对飞机位置，而不需要明确提供这些信息。然后父代理能够在输出层为所有飞机进行航线选择。这样，父代理 DDQN 集成了冲突检测任务 (“看到”和“理解”) 以及冲突解决任务中的第一部分——航线选择 (“决策制定”)。图 4 展示了父代理架构的示例。在图 4 中，“隐藏层”是简化示意图，用以代表整个 DDQN。在图 4 所示的具体游戏场景中，我们控制两架飞机。上面的一架有两个航线选项，下面的一架只有一个航线选项。因此，这两架飞机的总航线组合为两种。

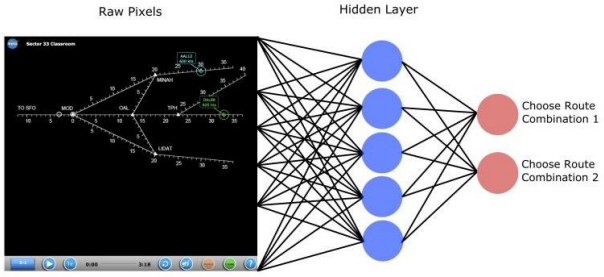

Figure 4. Illustrative example of the decision-making process for the parent agent.

图 4. 父代理决策过程的示例

With the route combination selected by the parent agent, the child agent will include this information to its current state and proceed with learning how to compute the speed adjustment advisories. Fig. 5 illustrates a specific example for the child agent. Here we define the hidden layer as the entire DDQN representation introduced, and the output as the speed controls the child agent can implement. In our case study, the output layer is the speed combinations of the aircraft.

在父代理选择的航线组合下，子代理将这一信息纳入当前状态，并继续学习如何计算速度调整建议。图 5 为子代理的一个具体示例。在这里，我们将隐藏层定义为引入的整个 DDQN 表示，输出为子代理可以实施的速度控制。在我们的案例研究中，输出层是飞机的速度组合。

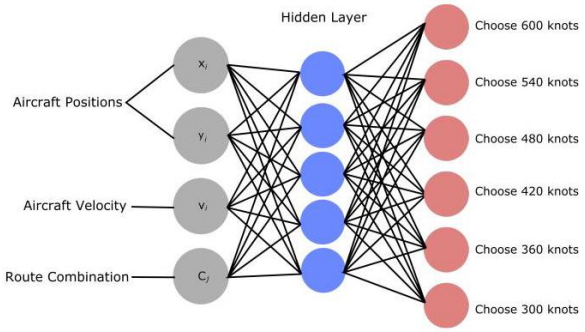

Figure 5. Illustrative example of the decision-making process for the child agent.

图 5. 子代理决策过程的示例

V. NUMERICAL EXPERIMENTS

V. 数值实验

A. Interface

A. 界面

To test the performance of our proposed algorithm, an interface was built between the web-based NASA Sector 33 application and our code so that the AI agent can operate all functions in the game. The state-

space information for the agent was obtained using computer vision techniques. By capturing screen-shots of the game screen every four seconds, we could then extract the aircraft position in terms of the (x_i , y_i) pixel location. By game design, when restarting a particular problem, all objectives were the same: final aircraft metered positions, aircraft initial position, and available speed changes did not change. We also included an additional feature that is not available by default in the web-based version of NASA Sector 33. This feature is to terminate the game if two aircraft are within in a 30-pixel radius (aircraft collision).

为了测试我们提出算法的性能，我们在基于网络的 NASA Sector 33 应用程序和我们的代码之间建立了一个接口，以便 AI 代理能够操作游戏中的所有功能。使用计算机视觉技术获取了代理的状态空间信息。通过每四秒捕获一次游戏屏幕截图，我们可以提取飞机的位置，以 (x_i , y_i) 像素位置表示。根据游戏设计，在重新启动特定问题时，所有目标都是相同的: 最终飞机的计量位置、飞机的初始位置以及可用的速度变化都没有改变。我们还包括了一个在 NASA Sector 33 的网络版中默认不可用的附加功能。这个功能是在两架飞机在 30 像素半径内时终止游戏 (飞机碰撞)。

B. Environment Setting

B. 环境设置

For each problem in NASA Sector 33, we discretized the environment into episodes, where each run through the game counted as one episode. We also introduced a time-step, Δt , so that after the child agent selected an action, the environment would evolve for Δt seconds until a new decision was made. This was to allow for a noticeable change in state from $s_t \rightarrow s_{t+1}$, as well as, allow the game to be played at different speeds. There were many different parameters that had to be tuned and selected to achieve the optimal solution in this game. We implemented the DDQN concept that was mentioned earlier, with a hidden layer of 64 nodes. We used an ε -greedy search strategy for both the parent and child agents. For the child agent, ε started at 1.0 and exponentially decayed until 0.01 . For the parent agent, ε also started at 1.0 and exponentially decayed to 0.15 . The reason for the parent agent' s ε decaying until 0.15 was to allow for the slightly larger probability of exploring the other routes even when many episodes had already been run. In harder game environments, we could obtain a near optimal feasible solution on the non-optimal route. By forcing the lower bound on ε , this ensured that we were able to converge to the optimal route. We used a tanh activation function for the child agent and the ReLU activation function for the parent agent. This was because for the parent agent, instead of using a multi-layer perceptron network, we used a convolutional neural network whose input was the current game screen. Our memory length for both the parent and the child agents was set to 500,000 to allow for a large number of the previous transitions to be made available for training.

对于 NASA Sector 33 中的每个问题，我们将环境离散化为多个阶段，每次运行游戏算作一个阶段。我们还引入了时间步长 Δt ，以便在子代理选择一个动作后，环境会在 Δt 秒内发展，直到做出新的决策。这样做的目的是为了允许从 $s_t \rightarrow s_{t+1}$ 观察到状态的变化，同时也允许游戏以不同的速度进行。为了在这个游戏中实现最佳解决方案，需要调整和选择许多不同的参数。我们实现了之前提到的 DDQN 概念，并使用了一个包含 64 个节点的隐藏层。我们为父代理和子代理都采用了 ε -贪婪搜索策略。对于子代理，ε 从 1.0 开始指数衰减至 0.01。对于父代理，ε 也从 1.0 开始指数衰减至 0.15。父代理的 ε 衰减至 0.15 的原因是为了在已经运行了许多阶段后，仍然允许有稍微大一点的概率去探索其他路线。在更困难的游戏环境中，我们可以在非最优路线上获得接近最优的可行解。通过为 ε 设置下限，这确保了我们能够收敛到最优路线。我们为子代理使用了 tanh 激活函数，为父代理使用了 ReLU 激活函数。这是因为对于父代理，我们没有使用多层感知器网络，而是使用了一个卷积神经网络，其输入是当前游戏屏幕。我们为父代理和子代理设置的内存长度为 500,000，以允许大量的先前转换用于训练。

C. Case Study: two aircraft with four routes

C. 案例研究: 两架飞机与四条路线

In this problem, we considered two aircraft where both aircraft have the ability to change routes. Fig. 6 shows the game screen for this problem. With both aircraft able to change routes, this leads to four different route combinations and greatly increases the complexity of the problem. What also makes this problem interesting is that even if the aircraft take the incorrect route, a near-optimal feasible solution can be obtained which makes choosing the route more difficult.

在这个问题中，我们考虑了两架飞机，两架飞机都具有改变航线的能力。图 6 展示了这个问题的游戏屏幕。由于两架飞机都能够改变航线，这导致了四种不同的航线组合，极大地增加了问题的复杂性。使这个问题有趣的是，即使飞机选择了错误的航线，也能获得接近最优的可行解，这使得选择航线变得更加困难。

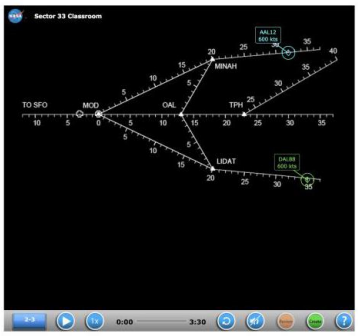

Figure 6. In the case study, both aircraft in this problem have the ability to change routes.
图 6。在案例研究中，这个问题中的两架飞机都具有改变航线的能力。

By training the AI agent on around 7000 episodes and choosing a time-step of four seconds, we were able to obtain the optimal solution for this problem. The number of episodes required to achieve the optimal solution was due to the complexity of the problem and the fact that different routes could achieve very similar results. It is also important to note that in this problem, if we chose a different route, the AI agent would achieve the optimal solution specific for that route. Therefore, in a real-world scenario, if there was a last-minute decision to change from the optimal route to a feasible route, the AI agent would be able to maintain safe separation and minimize delay on this new route. Fig. 7 shows the game screen after obtaining the optimal solution to this problem.

通过在约 7000 个回合中训练 AI 代理，并选择四秒的时间步长，我们能够获得这个问题的最优解。达到最优解所需的回合数是由于问题的复杂性和不同航线可能获得非常相似结果的事实。值得注意的是，在这个问题中，如果我们选择了不同的航线，AI 代理将会达到那个特定航线的最优解。因此，在现实世界的场景中，如果最后时刻决定从最优航线改为可行航线，AI 代理将能够在这条新航线上保持安全间隔并最小化延迟。图 7 展示了在这个问题获得最优解后的游戏屏幕。

D. Algorithm Performance

D. 算法性能

In this section, we analyze the algorithm's performance on the two-aircraft problem. Fig. 8 shows the score per episode during training for the case study.

在这一节中，我们分析了算法在两架飞机问题上的性能。图 8 展示了案例研究中训练期间每个回合的得分。

We can see that throughout training the score tends to oscillate frequently, but is on an increasing trend. This is due to the value of ε being close to 1 and as this value decreases through the episodes, the score increases significantly. We define the score in Fig. 8 to be the cumulative reward at the end of each episode. The score involves the reward of the child agent at each time step, as well as, the termination rewards: -10 for collision, -3 for out of time, and +10 for optimal solution.

我们可以看到，在整个训练过程中，得分 tends to oscillate frequently(频繁波动)，但总体呈上升趋势。这是由于 ε 的值接近 1，随着这个值在各个回合中减小，得分显著增加。我们定义图 8 中的得分为每个回合结束时累积的奖励。得分包括在每个时间步骤中子代理的奖励，以及以下终止奖励: 碰撞为 -10，超时为 -3，最佳解决方案为 +10。

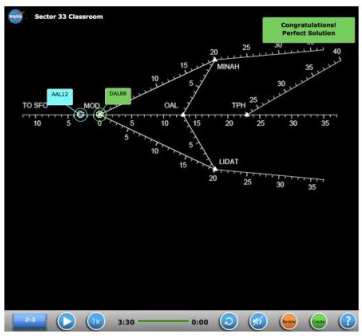

Figure 7. Problem 2-3 of the NASA Sector 33 game environment after obtaining the optimal solution. We can see that both aircraft are on their correct terminal when there is 0:00 left on the timer.

图 7. 在 NASA Sector 33 游戏环境中获得最佳解决方案后的问题 2-3。我们可以看到，当计时器上剩余时间为 0:00 时，两架飞机都处于它们正确的终端位置。

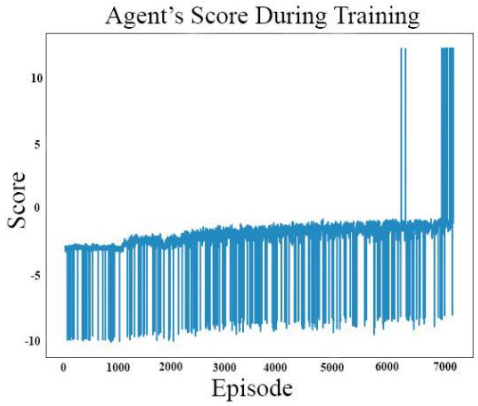

Figure 8. Agent's score throughout the training process for the case study. We can see that as the episode number increases, the score increases as well.

图 8. 研究案例中代理在训练过程中的得分情况。我们可以看到，随着回合数的增加，得分也在增加。

VI. CONCLUSION

VI. 结论

A novel hierarchical deep reinforcement learning algorithm is proposed in this paper to sequence and separate aircraft as a core component in an autonomous air traffic control system. The problem is formulated in a reinforcement learning framework with the actions of changing aircraft route and adjusting aircraft speed. The problem is then solved by using the hierarchical deep agent algorithm. Numerical experiments show that this proposed algorithm has promising performance to help aircraft maintain safe separation, resolve conflict, and arrive at their final metered positions in a case study. The proposed algorithm provides a potential solution framework to enable autonomous sequencing and separation for a fully automated air traffic control system in a structured airspace.

本文提出了一种新颖的分层深度强化学习算法，用于序列化和分离飞机，作为自主空中交通控制系统的核心组成部分。问题被表述在强化学习框架中，其动作包括改变飞机路线和调整飞机速度。然后使用分层深度代理算法解决这个问题。数值实验表明，这个提出的算法在帮助飞机保持安全间隔、解决冲突，并在一个案例研究中到达它们最终的计量位置方面表现出了有希望的成效。所提出的算法为实现在结构化空域中完全自动化的空中交通控制系统的自主序列化和分离提供了一个潜在的解决方案框架。

According to our knowledge, the major contribution of this research is that we are the first research group to investigate the feasibility and performance of autonomous aircraft sequencing and separation with deep reinforcement learning framework to enable an automated, safe and efficient airspace. In addition, we propose the novel hierarchical deep reinforcement learning architecture, which is demonstrated capable of solving complex online sequential decision-making problems. The promising results

from our numerical experiments encourage us to conduct future work on more advanced air traffic control simulators that can model operational uncertainties.

根据我们的知识，这项研究的主要贡献在于我们是第一个研究小组，研究自主飞机排序和分离的可行性和性能，采用深度强化学习框架以实现自动化、安全和高效的空域管理。此外，我们提出了新颖的分层深度强化学习架构，已证明其能够解决复杂的在线顺序决策问题。我们数值实验的积极结果激励我们未来在能够模拟操作不确定性的更先进空中交通控制模拟器上进行研究工作。

ACKNOWLEDGMENT

致谢

The authors would like to thank Heinz Erzberger for the discussions on autonomous air traffic control and Michelle Eshow for pointing us to the NASA Sector 33 simulation environment.

作者们想要感谢 Heinz Erzberger 关于自主空中交通控制的讨论，以及 Michelle Eshow 引导我们使用 NASA Sector 33 模拟环境。

REFERENCES

参考文献

[1] Erzberger, H., 2005. "Automated conflict resolution for air traffic control."

[2] Erzberger, H., 2007. "Fast-time simulation evaluation of a conflict resolution algorithm under high air traffic demand."

[3] Erzberger, H., Heere, K., 2010. "Algorithm and operational concept for resolving short-range conflicts." Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering 224 (2), 225-243.

[4] Amazon, A. P., 2015. "Revising the airspace model for the safe integration of small unmanned aircraft systems." Amazon Prime Air.

[5] Google, 2015. "Google uas airspace system overview." URL https://utm.arc.nasa.gov/docs/GoogleUASAirspaceSyst
ger[1].pdf

[6] Kopardekar, P. H., 2015. "Safely enabling civilian unmanned aerial system (uas) operations in low-altitude airspace by unmanned aerial system traffic management (utm)."

[7] Airbus, 2016. "Future of urban mobility." Ul http://www.airbus.com/newsroom/news/en/2016/12/My-Kind-Of-Flyover.html

[8] Gipson, L., 2017. "Nasa embraces urban air mobility, calls for market study." URL https://www.nasa.gov/aero/nasa-
embraces-urban-air-mobility

[9] Holden, J., Goel, N., 2016. "Fast-forwarding to a future of on-demand urban air transportation." San Francisco, CA.

[10] Uber, 2018. "Uber elevate - the future of urban air transport." URL https://www.uber.com/info/elevate

[11] OpenAI, 2017. Alphago at the future of go summit, 23-27 may 2017. URL https://deepmind.com/research/alphago
china/

[12] NASA, 2013. "NASA sector 33 application." https://www.atcsim.nasa.gov/simulator/sim2/sector33.html

[13] Amato, C., Shani, G., 2010. "High-level reinforcement learning in strategy games." In: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1. International Foundation for Autonomous Agents and Multiagent Systems, pp. 75-82.

[14] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M., 2013. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602.

[15] 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., 2016. "Mastering the game of go with deep neural networks and tree search." nature 529 (7587), 484-489.

[16] Watkins, C. J. C. H., 1989. "Learning from delayed rewards. Ph.D. thesis, King's College, Cambridge."

[17] Kochenderfer, M. J., Amato, C., Chowdhary, G., How, J. P., Reynolds, H. J. D., Thornton, J. R., Torres-Carrasquillo, P. A., U" re, N. K., Vian, J., 2015. "Decision Making Under Uncertainty: Theory and Application, 1st Edition." The MIT Press.

[18] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al., 2015. "Human-level control through deep reinforcement learning." Nature 518 (7540), 529.

[19] Van Hasselt, H., Guez, A., Silver, D., 2016. "Deep reinforcement learning with double q-learning." In: AAAI. Vol. 16. pp. 2094-2100.

[20] Hasselt, H. V., 2010. "Double q-learning." In: Advances in Neural Information Processing Systems. pp. 2613-2621.