Multi-Agent Reinforcement Learning for Complex and Dynamic Graph-Based Planning Problems

Jonathan Cawalla
Fraunhofer FKIE
Wachtberg, Germany
jonathan.cawalla@fkie.fraunhofer.de

Marvin Gülhan
Fraunhofer FKIE
Wachtberg, Germany
marvin.guelhan@fkie.fraunhofer.de

***Abstract*—Command and Control (C2) planning problems are often difficult to address with conventional optimization methods due to their complexity and inherent dynamicity. As a result, practitioners often rely on manually crafted heuristics, which are time-consuming to develop and require costly re-engineering when requirements change. As an alternative, we investigate Graph Reinforcement Learning (GRL) as a data-driven approach for learning policies in a dynamic, multi-agent, graph-structured planning problem. As a case study, we explore the Airlift Planning Problem (APP), in which multiple aircraft must transport dynamically requested cargo under strict time windows, resource constraints, and frequent disruptions in the flight network. We model system states as per-aircraft sub-graphs, process them with a Graph Neural Network, and train decentralized aircraft policies using Multi-Agent Proximal Policy Optimization (MAPPO) with shared parameters. To ensure stable training, we combine reward shaping, dynamic action masking, and rule-based handling of simple sub-tasks. Experiments on benchmark APP scenarios show that the learned GRL policies are competitive with, and often outperform, manually crafted heuristics.**

This paper was originally presented at the International Conference on Military Communication and Information Systems (ICMCIS), organized by the Information Systems Technology (IST) Scientific and Technical Committee, IST-224-RSY – the ICMCIS, held in Bath, United Kingdom, 12-13 May 2026

***Index Terms*—Reinforcement Learning, Multi-Agent, Graph Neural Networks, Neural Combinatorial Optimization, Decision Support, Airlift Planning, Heuristics**

I. Introduction

Command and Control (C2) environments are characterized by high complexity, operational constraints, and significant uncertainty. Problems arising in such settings can be addressed with methods of the broader field of Operations Research, such as optimization models, heuristics, and metaheuristics. While these established approaches can be effective, applying them to C2 problems is often challenging in practice. Exact models may be impractical due to modeling effort and runtime constraints, and hand-crafted heuristics require substantial domain expertise and repeated tuning. When requirements change, which is common in dynamic military environments, these methods typically require significant re-engineering.

In the combinatorial optimization literature, Neural Combinatorial Optimization (NCO) [1], [2], also called Graph Reinforcement Learning (GRL) [3], [4] has emerged as a data-driven alternative that learns heuristics from a simulation via Reinforcement Learning (RL). Existing RL work from the NCO literature has focused primarily on classic optimization problems and on routing problems such as Vehicle Routing Problems (VRP) [5]. In contrast, NCO/GRL approaches to military C2 planning problems have received comparatively little attention so far [6], [7].

In this paper, we study the application of GRL to a complex C2 problem, namely the Airlift Planning Problem (APP), which exemplifies this research gap through its dynamic events, time-critical requests, and resource constraints. The APP requires dynamically managing the transport of numerous cargo items to their destinations under tight time constraints using various planes. Rapid decision-making is critical, especially given frequent route disruptions and dynamic delivery orders. The problem environment originates from the Airlift Challenge developed by the Air Force Research Laboratory (AFRL) [8].

We model the APP state as a graph and process it with a Graph Neural Network (GNN) to obtain agent-centric embeddings. These embeddings are used to define a stochastic policy for each aircraft, which is treated as an individual agent. The shared network parameters are optimized with Multi-Agent Proximal Policy Optimization (MAPPO) [9] using the trajectories collected from all agents.

To make RL training reliable in this setting, we combine several mechanisms that shape and constrain agent behavior. Reward design is one important component. In addition, we use action masking to remove actions that are infeasible or clearly counter-productive, thereby reducing exploration noise. We also employ a limited form of hybrid control, where simple, unambiguous sub-tasks, such as unloading cargo upon arrival at its destination, are handled by fixed rules instead of being learned. Together, these mechanisms provide predictable bounds on agent behavior: the reward guides the training process, the mask constrains the action space, and fixed rules cover trivial decisions that need not be learned.

Our contributions are as follows:

- Firstly, we demonstrate how to effectively address a complex C2 planning problem, such as the APP, using Graph Reinforcement Learning. This represents, to the best of our knowledge, the first exploration of this technique in the airlift planning domain, which has previously

been addressed primarily through heuristic and classical optimization approaches [8], [10].
- Secondly, we provide a detailed study of reward design and action masking, demonstrating how these choices influence performance and training stability.
- Finally, we evaluate whether GRL-learned policies can outperform manually designed heuristics for the APP.

The remainder of the paper is structured as follows: Section II introduces the APP in more detail. Section III summarizes related literature and describes heuristics for the APP. Section IV details our methodology. Section V presents the experimental results and compares our approach to several heuristic baselines. Finally, Section VI concludes with a summary and an outlook on future research.

II. Airlift Planning Problem

The airlift planning problem (APP) involves efficiently transporting cargo between airports under constraints such as fuel limits, airport capacity, and route availability. The goal is to deliver all cargo on time at minimal cost in a dynamic environment with delays, bottlenecks, and disruptions, requiring rapid route recalculation [10], [11].

To address this problem, the AFRL designed an APP simulator and created the Airlift Challenge[1], a publicly hosted challenge which was run in 2023 [10] and 2024 [8]. The authors also made the simulator code publicly available[2], which we use to demonstrate our RL approach in this work.

The essence of a scenario in the APP can be illustrated as shown in Figure 1. There is a pickup zone, containing airports where cargo may appear, and a drop-off zone, where cargo must be delivered to designated airports. Airports within each zone are typically located close to one another.

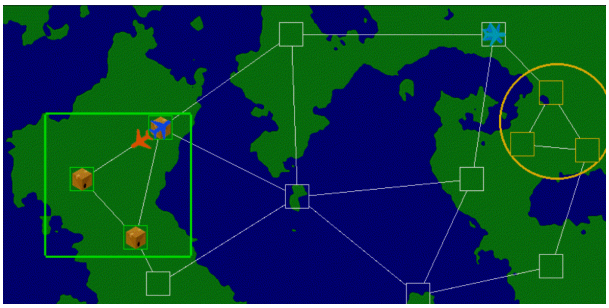

Fig. 1: A simple example of an airlift scenario[1]: the green square indicates the pickup zone where cargo appears, while the yellow circle marks the drop-off zone.

Every cargo item is defined by an origin airport, a destination airport, a soft deadline, a hard deadline, and a weight. The soft deadline marks the point after which a small cost is incurred for each unit of delay, while the hard deadline marks the time after which the cargo is treated as missed and no longer contributes to the objective. Each aircraft type has a defined maximum range, speed, and maximum cargo weight it can carry. Airports themselves are subject to constraints such as a capacity limit (the number of aircraft they can serve simultaneously) and a processing time per aircraft. Airports are categorized into three types: departure, transit, and destination airports.

[1]https://airliftchallenge.com/chapters/main.html
[2]https://github.com/airlift-challenge/airlift

III. Manual Heuristic Design

The APP, as introduced in Section II, poses a set of challenges that fundamentally distinguish it from classical routing and logistics optimization problems. Cargo requests arrive dynamically over time and are subject to soft and hard deadlines, while aircraft operate under heterogeneous constraints such as limited range, speed, and cargo capacity. Airports introduce additional coupling between agents through capacity limits and processing times, creating congestion effects when multiple aircraft arrive simultaneously. Furthermore, flight routes may become temporarily unavailable due to malfunctions, and future cargo demands are unknown at decision time. These characteristics make it difficult to directly apply standard formulations such as the Vehicle Routing Problem (VRP) [12] or Pickup and Delivery Problems (PDP) [13]. Classical VRP and PDP models typically assume a static setting with full knowledge of all requests in advance and rely on centralized planning over a fixed problem instance. In contrast, the APP requires continuous online decision-making under uncertainty, with multiple agents acting concurrently and influencing one another through shared infrastructure. While there are dynamic variants to these static problems, such as the Dynamic VRP with Time Windows (DVRPTW) [14] or Dynamic PDP with Time Windows (DPDPTW) [15], due to these many constraints none of them are directly applicable to the APP without substantial extensions.

As a result, successful approaches to the APP rely on heuristic online policies that trade optimality for scalability and robustness to dynamic change. This perspective is reflected in prior work on the AFRL Airlift Challenge. In particular, Kolen [16] shows that robust, lightweight heuristics are well suited to dynamic airlift planning and serve as strong baselines for learning-based methods. The author proposes two approaches: one uses a layered heuristic design that combines shortest-path-based goal assignment, congestion-aware flight scheduling, and greedy cargo loading; the other adopts a simplified “build-fast” strategy that minimizes replanning, limits the number of active aircraft to reduce airport congestion, and often waits for disrupted routes to recover instead of recomputing alternatives. [16]

Although this approach proved highly effective in practice, the lack of publicly available implementation details prevents direct comparison. Therefore, we focus on three Heuristics:

The Shortest Path (SP) heuristic represents a baseline routing strategy given by the AFRL Team [8]. At each decision point,

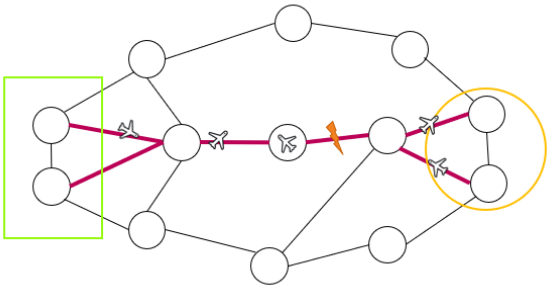

Fig. 2: Illustrative shortest path routing in a simple airlift network. All aircraft select the shortest path (red) between pickup and delivery zone. The disruption (lightning) on this path forces agents to replan, potentially leading to congestion and cascading delays.

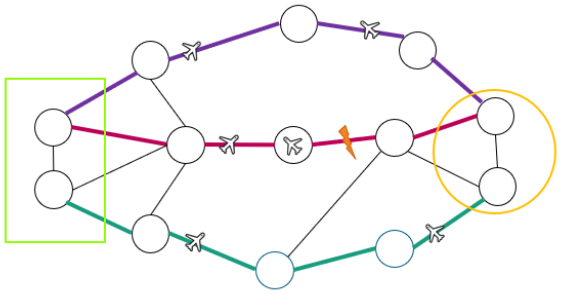

Fig. 3: Illustrative corridor based routing in a simple airlift scenario. The corridors (marked in purple, red, green) connect the pickup and delivery zone. The same disruption as in Fig. 2 no longer triggers complete replanning and potential cascading effects for all agents. Only the red corridor is affected.

an aircraft selects its next destination based on the shortest path to the current target, typically the pickup or drop-off location associated with its assigned cargo. Decisions are made independently by each agent without any anticipation of congestion effects. If a route is unavailable the agent will reroute to the next shortest path. An example of how disruptions affect this method is shown in Fig. 2.

The **Corridor-Based Routing Heuristic (CH)** was our submission to the second iteration of the Airlift Challenge and achieved 2nd place [8]. It coordinates aircraft by restricting their movements to a set of pre-computed corridors through the airlift network which are essentially k-shortest paths with no overlapping edges outside the pickup and drop-off zones. A *corridor* can then be understood as a path connecting specific pickup nodes in the pickup zone to the drop-off zone and serves as a stable routing structure during execution. The agents are mapped to these corridors. By constraining agents to these fixed corridors, CH reduces path overlap, alleviates airport congestion, and improves robustness to disruptions, while maintaining a simple decentralized execution policy. Agents are rerouted only if rerouting is faster than waiting. Cargo loading is performed greedily based on urgency, measured as the remaining time until the hard deadline. After delivery, agents return to their assigned corridor origin. An example of the superiority to the SP approach in case of disruption is shown in Fig. 3.

The **Decentralized Corridor Heuristic Adapted (CHA)** is a variant of the CH method. The key difference to CH is that CHA removes the centralized corridor planning phase and operates fully decentralized, with each agent selecting actions solely based on its local observation. Unlike SP, congestion awareness is explicit since agents consider node capacity and local agent density when choosing routes, resulting in behavior that retains similarities to the characteristics of corridor-based planning such as CH. Loading of cargo is done in the same manner as explained in CH.

Although scenarios with multiple aircraft-specific subgraphs are considered in the Airlift Challenge [8], our experiments focus on settings without explicit subgraph separation.

IV. Graph Reinforcement Learning Methodology

Despite the original intent of the authors and the design of the APP simulator to motivate RL-based solutions, no submitted solution has successfully employed RL [8], [10]. This failure is not specific to airlift planning, but reflects broader challenges that arise when applying RL to complex planning problems.

The APP simulator exposes three related issues that we address with our methodology. Firstly, the simulator defines complex, composite state and action spaces that are incompatible with most standard RL methods. We resolve this by transforming the environment into a tractable Markov Decision Process (MDP). Secondly, the simulator's reward function is designed to measure airlift performance rather than to facilitate RL credit assignment, which, as we will see later, leads to significant training issues. We address this by redesigning the reward. Thirdly, the action space allows unconstrained and operationally meaningless actions, which we mitigate

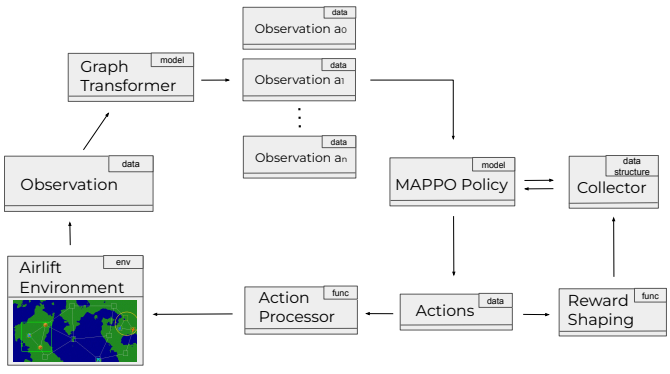

Fig. 4: Overview of the multi-agent reinforcement learning pipeline for the airlift environment.

through action masking and hybrid control that encodes basic operational knowledge to guide learning.

Finally, we introduce a Graph Reinforcement Learning model architecture that operates on the transformed environment. The overall process of how the RL agents interact with the environment is shown in Figure 4. Observations generated by the Airlift environment are first encoded using a Graph Transformer to produce per-agent representations, which are then consumed by a MAPPO policy to generate actions. These actions are post-processed to enforce operational constraints before being executed in the environment, while a collector aggregates trajectories, and shaped rewards for policy optimization.

While some components and parameters of our approach are specific to airlift planning, the overall design choices can serve as a guideline for applying RL to other complex C2 planning problems that face similar challenges.

A. Markov Decision Process Transformation

RL methods typically assume fixed-size state and action spaces. This is not the case in the original APP simulator, which makes the underlying MDP difficult to address using standard RL techniques. Each action specifies a destination, an optional list of cargo items to load or unload, and a priority. The state includes a graph structure along with multiple dynamic lists, such as the number of cargo items at each node or inside each aircraft. These dynamic lists are the main obstacle. Furthermore, not all state information is relevant for every agent.

To address this, we construct a fixed-size observation and discrete action space suitable for decentralized, partially observable Markov decision processes (DEC-POMDP), similar to the formulation in [9], based on the underlying Airlift MDP [8]. The resulting DEC-POMDP is defined as $\langle \mathcal{S}, \mathcal{A}, O, R, P, n \rangle$.

The global state space $\mathcal{S}$ encodes the full Airlift environment. Each agent $i \in 1, \ldots, n$ receives a local observation $o^{(i)} = O(s, i)$ derived from $s \in \mathcal{S}$. This observation is represented as an agent subgraph $G^{(i)} = (V^{(i)}, E^{(i)})$ with associated node and edge features.

The key design choice in our observation model is to avoid dynamic, variable-length lists (e.g., "list of agents present" or "list of cargo items at a node"). Instead, every piece of information is encoded with a fixed-size, bounded representation: quantities such as the fraction of agents at a node, the fraction of cargo waiting, or the proportion of airport workload are expressed as normalized scalars in the range $[0, 1]$, while the earliest five cargo-pickup deadlines at an airport are stored in the pre-allocated slots d_1–d_5. This yields a compact observation space that can be processed efficiently by graph neural networks while preserving all information essential for high-level planning.

The full list of nodes and edge features follows:

1) Node features (per node in each agent's sub-graph):

- **agent_position**: boolean value that indicates whether the agent is currently located at the node.
- **cargo_deadline**: normalized value that reflects the urgency of the cargo already loaded on the agent (values close to 1 denote high urgency).
- **amount_agents**: normalized count of agents present at the node.
- **amount_cargo**: normalized count of cargo items waiting at the node.
- **reachable**: boolean value that indicates whether the node lies within the agent's current flight range.
- **drop_off**: boolean value that indicates that the node is the destination of the cargo currently loaded on the agent.
- **source**: boolean value that indicates that the node belongs to the pickup zone.
- **destination**: boolean value that indicates that the node belongs to the drop-off zone.
- **d1–d5**: normalized values that store the earliest five cargo-pickup deadlines at this airport.
- **cargo_onboard**: boolean value that indicates whether the agent is carrying any cargo.
- **airport_workload**: normalized measure of the current usage of the airport's processing capacity.
- **agent_state**: encoding of the agent's current state (PROCESSING, READY, WAITING, MOVING).
- **num_cargo_onboard**: normalized count of cargo items already loaded onto the agent.
- **cargo_urgency**: normalized scalar that grows as deadlines become closer; it is zero when the agent has no cargo.
- **time_elapsed**: normalized representation of the current timestep relative to the maximum possible deadline (identical for all nodes).

2) Edge features (per edge in each agent's sub-graph):

- **cost**: normalized value that represents the travel cost of traversing the edge.
- **time**: normalized value that indicates the travel time required to fly the edge, taking distance and aircraft speed into account.
- **malfunction**: boolean value that indicates whether the edge is currently experiencing a malfunction.
- **incident**: boolean value that indicates whether the edge is incident to the node where the agent is currently positioned.

We define a discrete action space $\mathcal{A}$ by restricting agents to choosing their next node from their current neighbors. The joint action space is $\mathcal{A} = \mathcal{A}^{(1)} \times \cdots \times \mathcal{A}^{(n)}$, where the individual action space for agent i is:

$$\mathcal{A}^{(i)} = \{v \in V^{(i)} \mid (c_i, v) \in E^{(i)}\}$$

with c_i denoting agent i's current node. Based on this action space, agents do not explicitly choose cargo but instead we use a fixed heuristic: if an agent is at a pickup node, it

automatically selects the cargo with the earliest hard deadline. If an agent moves to drop-off node assigned by a cargo, it will be automatically delivered.

The transition function $P(s_{t+1}|s_t, A)$ describes the probability of doing a joint action $A = (a^{(1)}, \ldots, a^{(n)})$ in state s_t and transitioning to state s_{t+1}. In this transition function, the state is updated including the plane positions, dynamic route disruptions and cargo information.

B. Reward function design

The design of the reward function R is one of the most challenging aspects when defining a RL environment for a complex problem. We begin by describing the original reward function used in the Airlift Challenge. It combines four components: a movement penalty proportional to the fuel cost of the flown edge, a step-wise late-delivery penalty applied after the soft deadline, a one-off penalty incurred when the hard deadline is exceeded, and a small negative reward for issuing a "stay" action. Overall, this formulation aims to minimise total flight cost while avoiding late or missed deliveries.

In practice, this reward leads to two problematic behaviours in RL training:

1) Firstly, because the late-delivery penalty is only applied after the soft deadline, the agent receives a strong negative signal late in the episode, which makes it difficult to attribute the failure to earlier decisions.
2) Secondly, the per-step movement cost incentivises agents to hover on short back-and-forth routes with negligible cost, even though no cargo is transported.

These issues motivated the design of the three alternative reward configurations described next.

These reward configurations share the following main idea: Let R denote the global reward function. At time step t the environment is in state s_t and the joint action of all agents is $A_t = \{a_t^{(1)}, \ldots, a_t^{(n)}\}$. The total reward is the sum of the individual-agent rewards:

$$\begin{aligned} R(s_t, A_t) &= \sum_{i=1}^{n} R^{(i)}(s_t, a_t^{(i)}) \\ &= \sum_{i=1}^{n} \Big(r_t^{\text{move}(i)} + r_t^{\text{delivery}(i)} + r_t^{\text{penalty}(i)} \Big). \end{aligned}$$

Thus each agent receives a per-step scalar that is the sum of three components:

- $r_t^{\text{move}(i)}$: a directional movement penalty based on the change in average distance to target or destination nodes.
- $r_t^{\text{delivery}(i)}$: a positive incentive awarded when the agent successfully drops a cargo at its destination
- $r_t^{\text{penalty}(i)}$: a small penalty for bottlenecks, waiting, or simply for taking a time step.

Note, that only the terms that are enabled by a particular reward configuration contribute to the final value. We use the term reward throughout, with the understanding that it may take negative values and thus also represent a cost.

Variant 1: Drop-off-only

Variant 1 is the simplest possible reward function that addresses the problematic avoidance of longer paths by not rewarding movement at all. It also reverses the reward structure: instead of assigning a penalty when a cargo deadline is missed, we provide a positive reward when a delivery is successfully completed. This makes credit attribution significantly easier. Despite its simplicity, the reward sends a clear signal to the learning agent about the primary objective, namely delivering cargo.

We define the following reward terms:

$$\begin{aligned} r_t^{\text{move}} &= 0, \\ r_t^{\text{delivery}} &= \begin{cases} 1, & \text{if a cargo is delivered successfully,} \\ 0, & \text{otherwise,} \end{cases} \\ r_t^{\text{penalty}} &= 0. \end{aligned} \tag{1}$$

All shaping terms are disabled; the agent receives a single positive unit reward *only* when a cargo reaches its final airport. No step cost, wait penalty, or movement penalty is applied.

Variant 2: Drop-Off + Constant Penalty Reward

In this reward configuration, the only positive signal is the delivery reward, while every action (including "wait") incurs a constant cost. This follows a similar idea to the first reward configuration, but signals the agent which trajectories are more efficient based on the step cost.

We define the following reward terms:

$$\begin{aligned} r_t^{\text{move}} &= 0, \\ r_t^{\text{delivery}} &= \begin{cases} 1, & \text{cargo is delivered,} \\ 0, & \text{otherwise,} \end{cases} \\ r_t^{\text{penalty}} &= -0.05 \qquad \text{(small cost incurred every step).} \end{aligned} \tag{2}$$

Variant 3: Drop-Off + Directional Movement Reward

This reward configuration defines a directional signal that discourages exploiting low-cost oscillations that reduce penalties, as was possible in the original reward scheme. By making movement costs negative, regardless of direction, the agent is incentivized to minimize travel distance while still pursuing deliveries, since the delivery reward is comparatively large.

We define the following reward terms:

$$
\begin{aligned}
r_t^{\text{move}} &= \begin{cases} -\dfrac{1}{\kappa}\,\dfrac{d_{\text{old}}}{d_{\text{new}}}, & d_{\text{new}} < d_{\text{old}}, \\ -\dfrac{\lambda}{\kappa}\left(2 - \dfrac{d_{\text{old}}}{d_{\text{new}}}\right), & d_{\text{new}} \geq d_{\text{old}}, \end{cases} \\
r_t^{\text{delivery}} &= \begin{cases} 3, & \text{cargo is delivered}, \\ 0, & \text{otherwise}, \end{cases} \\
r_t^{\text{penalty}} &= \begin{cases} -0.10, & \text{agent waits}, \\ 0, & \text{otherwise}. \end{cases}
\end{aligned} \tag{3}
$$

Let d_{old} denote the average Euclidean distance from the current location to the set of target nodes, defined as pick-up locations when empty, and drop-off locations when carrying cargo. Let d_{new} denote the corresponding average distance after the move. The ratio $\frac{d_{\text{old}}}{d_{\text{new}}}$ therefore measures the directional effect of the move: values greater than 1 indicate movement towards the target nodes, while values below 1 indicate movement away from them.

The movement term is constructed to be always negative, so that it acts as a penalty, but with a substantially larger magnitude when the move increases the distance to the relevant set. When $\frac{d_{\text{old}}}{d_{\text{new}}} \geq 1$, the penalty is given by $-1/\frac{d_{\text{old}}}{d_{\text{new}}}$, which lies in $[-1, 0)$ and becomes milder as the move brings the aircraft closer to the target. When $\frac{d_{\text{old}}}{d_{\text{new}}} < 1$, the penalty is instead defined as $\frac{d_{\text{old}}}{d_{\text{new}}} - 2$, which lies in $(-2, -1)$. The constant offset -2 ensures a strict separation between the two cases: any move away from the target incurs a penalty more severe than any move towards it, while preserving continuity at the boundary $\frac{d_{\text{old}}}{d_{\text{new}}} = 1$.

The parameter κ controls the overall magnitude of the movement penalty, with $\kappa = 10$ chosen to discourage wasteful travel without dominating the learning signal. The parameter λ further amplifies the penalty for moves that increase the distance to the relevant nodes, namely pick-up locations when empty and drop-off locations when loaded. Setting $\lambda = 3$ makes such moves roughly three times more costly than moves in the correct direction, providing a clear directional cue while still allowing some exploration.

C. Action Mask

The action space described in Section IV-A permits any neighbour of the current node to be selected. In early experiments we observed agents repeatedly traversing a low-cost edge simply to avoid the high penalties associated with more expensive edges, even when those costly moves would lead to a delivery in the future. Reward shaping can mitigate such behaviours partly, but a more direct approach is to prune the set of actions at each step. We therefore introduce a configurable action-masking layer that removes actions before they are presented to the policy network. We implemented the following options:

- **Directional Action Mask:** removes actions that do not bring the agent strictly closer (using either the static or dynamic distance matrix) to the current goal node set. The goal is pickup when the aircraft is empty, and drop-off when cargo is onboard.
- **Dynamic Distance Matrix:** when enabled, the directional action mask uses the current, time-varying distance matrix that reflects temporary edge failures or congestion; otherwise, a static precomputed matrix is used.
- **Prevent Backtrack:** disallows the move that would return the aircraft to the node it occupied in the previous timestep, except when the agent chooses the explicit wait action. This eliminates trivial oscillations.
- **Wait Enabled**: enables the explicit wait action, subject to the Max Wait Steps limit and only when the agent is at a pickup node or when all paths to any target are blocked.
- **Max Wait Steps:** caps the number of consecutive wait actions an agent may execute.

However, we have to note that masking reduces the algorithm's exploration space and might lead to overly restricting the actions spaces. This can lead to agents not finding optimal routes in certain situations. Thus, we experiment with different variants based on the above masking options.

D. Graph Reinforcement Learning Model Architecture

After defining the Markov Decision Process, we now describe how the problem is addressed within a reinforcement learning framework.

Reinforcement Learning (RL) studies how an agent can learn a decision-making strategy through interaction with an environment [17]. At each time step, the agent selects an action, observes the resulting transition, and receives a reward signal that guides learning. The agent's behavior is defined by a policy, which specifies a probability distribution over actions conditioned on the agent's observations.

Since our environment consists of multiple decision-making entities, the problem falls within the scope of Multi-Agent Reinforcement Learning (MARL) [18]. We model each plane in the environment as an individual agent that interacts with the shared environment and with other agents. In MARL, the presence of multiple agents introduces additional challenges, most notably non-stationarity from the perspective of each agent, as well as the need to coordinate with others.

Several training and execution paradigms have been studied in MARL. In centralized training and execution (CTE), both learning and action selection rely on globally shared information and a central decision-making mechanism. In decentralized training and execution (DTE), each agent learns and acts independently based only on its local observations. A commonly used intermediate paradigm is centralized training with decentralized execution (CTDE), where learning can exploit additional information during training, while policies are executed independently using only local observations.

In this work, we adopt a training paradigm that lies between decentralized training and execution and centralized training with decentralized execution (i.e., between DTE and CTDE). We employ a decentralized actor-critic architecture with parameter sharing based on Proximal Policy Optimization (PPO), following the Multi-Agent PPO (MAPPO) formulation of Yu et al. [9]. All agents share the same actor and critic networks, and training is performed jointly using experiences collected from all agents. Each agent is trained using individual reward signals based on its own behavior. The critic is conditioned only on agent-centric observations and actions and does not take joint observations or joint actions as input. However, these agent-centric observations include summarized information about other agents in the environment, for example relative positions in the underlying graph structure. As a result, training leverages shared experience and structured information about other agents without relying on a centralized global state or a centralized critic, while execution remains fully decentralized.

To represent the graph-structured state for both actor and critic, we use a Graph Neural Network (GNN) to compute node and edge embeddings [19]. The GNN updates node features over multiple message-passing layers by aggregating information from neighboring nodes and incident edges, thereby creating node embeddings. The last layer of the GNN maps the previous node embeddings to a scalar logit. Applying a Softmax over these logits yields a stochastic policy over nodes, from which the agent samples an action that is executed in the environment.

At the core of our learning approach is PPO [20]. Starting from random initialization, we update the shared policy parameters using the per-agent rewards defined in Section IV-A. PPO performs clipped policy-gradient updates that increase the likelihood of actions with positive advantage estimates while constraining the change in the policy between successive updates [20]. In our multi-agent setting, the PPO objective is optimized using trajectories collected from all agents, and gradients are aggregated to update the shared actor and critic parameters.

The concrete PPO implementation is based on our previous work [7]. We also reuse the same overall GNN design, which has proven robust across related graph decision problems. In particular, we stack multiple Transformer Convolution layers with residual connections for message passing [21]. Compared to our earlier implementation, we additionally include an explicit Kullback–Leibler (KL) divergence penalty between the updated policy and the previous policy used to generate the trajectories. Empirically, this stabilizes training in our multi-agent setting.

V. Experiments

A. Scenarios

Based on the APP simulator [8] (see Section II), we create the following scenarios for our experiments:

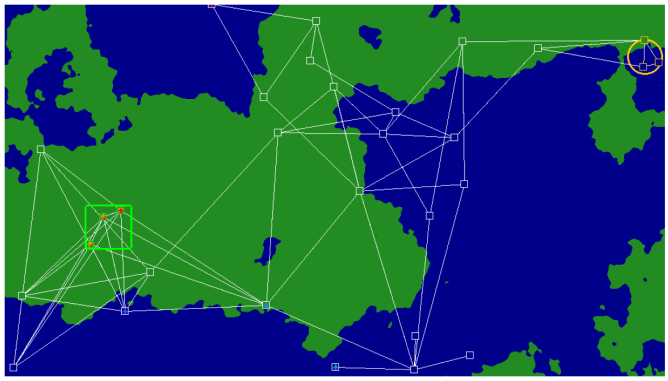

Fig. 5: Graphs sampled from the simulator under the smaller scenario AP30-A6-DM.

1) **AP30-A6-S (Static).** This scenario represents a fully static planning problem with 30 airports and 6 aircraft operating on a fixed route network without malfunctions. All cargo tasks are available from the beginning of the episode and no new tasks appear over time. Airport processing capacity is tight, with only one aircraft handled at a time, which creates local congestion. Deadlines are relatively strict and fixed, making this scenario primarily a test of static routing under capacity constraints.
2) **AP30-A6-DM (Dynamic with malfunctions).** This scenario uses the same number of airports and aircraft as the static case but introduces substantial dynamics and uncertainty. A large initial backlog of cargo is present, with additional cargo appearing stochastically during the episode and becoming available at staggered times. Routes can fail randomly and remain unavailable for extended periods, requiring solutions to continuously replan as the network and cargo demands change.
3) **AP50-A9-DM (Dynamic with malfunctions).** This scenario scales the environment to 50 airports and 9 aircraft. Route malfunctions are present as in the previous dynamic scenario, but the larger fleet and network allow for more routing alternatives.

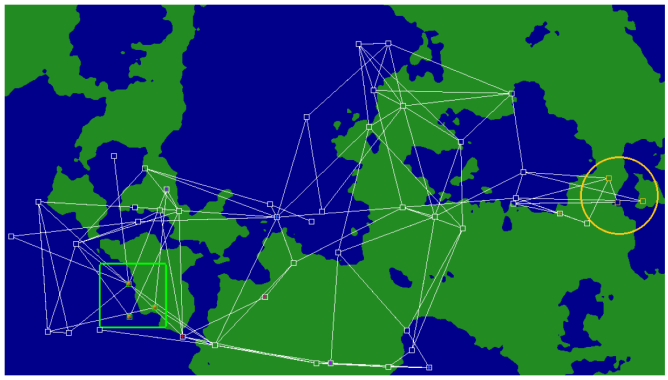

Fig. 6: Graphs sampled from the simulator under the larger scenario Scenario AP50-A9-DM.

Figures 5 and 6 show two example graphs from two different scenarios. Note, that the simulator is able to generate an unlimited amount of different graphs based on a scenario configuration. This allows us to train our RL model on a large number of graphs, while we set specific hold-out seeds for validation and testing.

B. Baselines

We use the baselines described in Section III and repeat a short description here as a reminder:

1) **Random**: Each agent randomly chooses an available action.
2) **Shortest Path (SP)**: Each agent independently routes cargo along the shortest path to its current target, without considering congestion, agent interactions, or cargo urgency.
3) **Corridor-Based Routing Heuristic (CH)**: Agents are centrally assigned to corridors and greedily transport the most urgent cargo along these fixed routes, reducing congestion through structured coordination.
4) **Decentralized Corridor Heuristic Adapted (CHA)**: CHA removes centralized corridor planning and operates fully decentralized, with agents selecting routes and cargo based on local observations, urgency, and capacity information, leading to implicit coordination.

C. Experiment Definitions

Experiments 1 (Reward Design) and Experiment 2 (Action Mask Design) aim to identify the best design choices for reward and action masking. These experiments are run only on scenario AP30-A6-DM. The results are then incorporated into the final Experiment 3 (GRL vs. Heuristic Solutions), where we compare the best RL approach against the manually designed heuristic solutions from Section III across all scenarios. All experiments use the MAPPO + GNN approach described in Section IV-D. Across all experiments, we use the same GRL setup: identical actor and critic networks, default PPO hyperparameters [20], trajectories of 10,000 steps collected in parallel from 16 collectors and processed in 6 minibatches, and a 3-layer GNN with 128 hidden units per layer. Each variant was trained with three random seeds for 5 million steps and evaluated on 50 episodes (50 different graphs sampled from the scenario's graph distribution). Each training run took roughly 5–6 hours.

We record the following metrics:

- **Average score:** Average episode score, which consists of penalties for missed deliveries, lateness, and flight cost [10] (Note: lower is better).
- **Average missed deliveries:** The number of deliveries missed on average. We include this because it is easier to compare and interpret than the average score (Note: lower is better).

It is important to note that we do not have access to the optimal solution for a generated problem instance. As a result, any number of missed deliveries obtained by either a heuristic baseline or our GRL approach represents the best known solution available to us. In many scenarios, delivering all cargo appears to be impossible, most likely due to strict deadlines and capacity constraints. Consequently, a solution can still be considered good even when it involves missed deliveries.

D. Results

TABLE I: Performance comparison of different reward configurations.

Reward Variant	Avg missed deliveries	Avg score
Original	48.25 ± 1.45	506.34 ± 16.53
Drop-off Only	11.59 ± 4.23	137.45 ± 42.78
Drop-off + Constant	9.03 ± 1.79	112.13 ± 19.15
Drop-off + Movement	8.60 ± 1.76	106.14 ± 18.41

1) Experiment 1 - Reward Design: In the first experiment (see Table I), we tested the reward variants described in Section IV-B. We used the dynamic action mask for these experiments, as we expected this choice to yield the best results. We observe that the original reward design is not well suited for training. Even our simplest variant, which rewards drop-offs only, significantly outperforms the original reward design, which was not optimized for RL. Surprisingly, we observed little difference between the Drop-off + Constant and Drop-off + Movement variants. Apparently, computing a directional movement penalty does not add much benefit compared with the simpler constant penalty, which also gives a clear signal to the agents.

2) Experiment 2 - Action Mask Design: In Experiment 2, we chose the best reward variant from experiment 1 and then test the following Action Mask variants:

1) Dynamic Mask Full (backtrack prevention enabled, wait enabled and max wait steps of 5)
2) Dynamic Mask NW (backtrack prevention enabled, wait disabled)
3) Dynamic Mask NB (backtrack prevention disabled, wait enabled)
4) Static Mask Full (backtrack prevention enabled, wait enabled and max wait steps of 5)
5) No Mask

From the results shown in Table II, it becomes clear that action masking can significantly improve performance. The

TABLE II: Performance of different masking configurations.

Mask Variant	Avg missed deliveries	Avg score
Dyn. Mask Full	8.60 ± 1.76	106.14 ± 18.41
Dyn. Mask + NW	10.77 ± 2.55	130.53 ± 27.99
Dyn. Mask + NB	11.87 ± 2.53	138.82 ± 25.62
Static Mask Full	9.46 ± 0.52	115.79 ± 5.91
No Mask	22.67 ± 5.41	248.58 ± 57.69

TABLE III: Performance comparison of baseline heuristics and GRL algorithms across three scenarios with 50 test graphs each. RL algorithms are run for three seeds and we report the mean and confidence interval. Since the optimal solution is unknown, the reported results reflect the best solutions found by the evaluated methods, and zero missed deliveries may be impossible for some graphs.

Algorithm	**AP30-A6-S**		**AP30-A6-DM**		**AP50-A9-DM**	
	Avg score	**Avg missed deliveries**	**Avg score**	**Avg missed deliveries**	**Avg score**	**Avg missed deliveries**
Random	500.17	50.00	1646.65	164.52	1255.63	125.22
SP	314.71	30.36	1181.30	116.16	949.96	93.18
CH	251.09	24.76	134.09	11.28	**248.78**	**21.64**
CHA	**211.08**	**20.16**	175.90	14.20	430.82	40.06
MAPPO (Original Rew., No AM)	500.22 ± 0.09	50.00 ± 0.00	1645.90 ± 1.51	164.45 ± 0.15	1255.84 ± 0.03	125.24 ± 0.00
MAPPO (Dir. M. Rew., Dyn. AM)	238.57 ± 16.08	23.13 ± 1.75	$\mathbf{106.14 \pm 18.41}$	$\mathbf{8.60 \pm 1.76}$	$\mathbf{240.13 \pm 17.09}$	$\mathbf{21.76 \pm 1.78}$

Full Dynamic Action Mask achieves an average score of 106.14, whereas the absence of masking results in 248.58. Backtrack prevention and waiting also have a noticeable positive effect on performance. Somewhat unexpectedly, the static mask performs well when combined with backtrack prevention and waiting, possibly because these additional constraints already eliminate many suboptimal actions, reducing the benefit of adapting the mask to time-varying distances.

3) Experiment 3 - GRL vs. Heuristic Solutions: The final experiment compares all solutions introduced in this paper. We evaluate the four heuristic baselines from Section III, i.e., Random, Shortest-Path (SP), Corridor-Based Heuristic (CH), and Decentralised Corridor Heuristic Adapted (CHA) against two MAPPO configurations:

- **MAPPO (Original Reward, No AM)**: a vanilla implementation of MAPPO that uses the reward function defined in the original APP benchmark and does not apply any action masking. This configuration is included solely as a check to demonstrate how much the reward shaping and dynamic action-masking contribute to performance.
- **MAPPO (Directional Movement Reward, Dynamic AM)**: the configuration that combines the Directional Movement reward (Section IV-B) with the Dynamic Action Mask (Section IV-C). This is the best RL variant identified in Experiments 1 and 2.

All results are reported in Table III. As expected, the Random baseline rarely delivers any cargo. Shortest Path heuristic improves on Random by routing each aircraft along the static shortest path to its current target, yet it still suffers heavily from congestion and capacity constraints, resulting in considerably higher travel costs and a larger number of missed deliveries than the more sophisticated heuristics.

Between the two corridor-based approaches we observe a clear trade-off. In the static scenario (AP30-A6-S) the CHA heuristic outperforms CH (211.08 vs. 251.09 average score; 20.16 vs. 24.76 missed deliveries). CHA routes each aircraft along the globally shortest path as long as airport processing capacities are not saturated, which yields more direct flights and lower overall travel cost when the network is stable. By contrast, CH deliberately spreads aircraft across multiple pre-defined corridors to hedge against possible route failures. In a static environment where disruptions are absent, this early distribution introduces unnecessary detours. However, in the dynamic scenarios (AP30-A6-DM and AP50-A9-DM) the advantage of corridor-based routing becomes apparent. CH outperforms CHA in the larger dynamic scenario (AP50-A9-DM) by roughly a factor of two in missed deliveries (21.64 for CH versus 40.06 for CHA).

The vanilla MAPPO configuration (Original Reward, No AM) fails to learn any useful behaviour: its performance mirrors the Random baseline across all three scenarios (e.g. 500.22 score and 50.00 missed deliveries in the static scenario). This collapse can be attributed to the original reward's poor credit-assignment properties and the unrestricted action space (see Experiment 1 and 2).

When equipped with the Directional Movement reward and the Dynamic Action Mask, MAPPO attains significantly better results. In the static scenario (AP30-A6-S) the learned policy reaches an average score of 238.57 ±16.08 with 23.13 ±1.75 missed deliveries, which is comparable to CHA (211.08 score, 20.16 missed) and substantially better than CH.

In the first dynamic scenario (AP30-A6-DM) MAPPO (Dir. Move Reward, Dyn. AM) outperforms every heuristic baseline, achieving the lowest average score (106.14 ±18.41) and the fewest missed deliveries (8.60 ±1.76). This demonstrates that the GRL agent can successfully adapt to stochastic cargo arrivals and route malfunctions, learning a coordinated replanning strategy that surpasses the manually engineered heuristics.

For the larger dynamic scenario (AP50-A9-DM) the MAPPO configuration remains competitive: its average score (240.13 ±17.09) and missed deliveries (21.76 ±1.78) are on par with the best heuristic (CH, which attains 248.78 score and 21.64 missed deliveries). Importantly, the confidence intervals reveal non-negligible variance across random seeds: in some runs MAPPO performs slightly worse than the heuristics, while

in others it exceeds them. This seed-sensitivity suggests that additional hyper-parameter tuning or longer training horizons could improve training stability and further improve the MAPPO approach.

VI. Conclusion and Outlook

This work investigated Graph Reinforcement Learning (GRL) as a data-driven alternative to hand-crafted heuristics for dynamic Command and Control planning, using the Airlift Planning Problem (APP) as a case study. By modeling the environment as per-aircraft sub-graphs, processing them with a Graph Neural Network, and training decentralized policies with Multi-Agent Proximal Policy Optimization and shared parameters, we showed that GRL can handle stochastic cargo arrivals, tight time windows, and network disruptions in a multi-agent setting.

Beyond the specific APP domain, our results highlight design principles for applying GRL in complex C2 environments. We showed that carefully structured reward designs, combined with dynamic action masking and rule-based handling of simple sub-tasks, can improve training stability. This hybrid control scheme constrains exploration to operationally meaningful behaviors. Empirically, the learned GRL policies are competitive with, and often outperform, manually designed heuristics. These findings indicate that GRL can reduce reliance on manually engineered heuristics and provide adaptable decision-support in dynamic, graph-structured planning problems.

Two directions are particularly relevant for future work. Firstly, the current formulation relies on parameter sharing without explicit inter-agent communication. Incorporating learnable communication mechanisms could enable more structured coordination under congestion and disruptions. Secondly, further evaluation on larger and more diverse scenarios is required to assess the scalability of the approach.

References

[1] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, L. Song, Learning combinatorial optimization algorithms over graphs, Advances in neural information processing systems 30.

[2] Y. Bengio, A. Lodi, A. Prouvost, Machine learning for combinatorial optimization: a methodological tour d'horizon, European Journal of Operational Research 290 (2) (2021) 405–421.

[3] Q. Cappart, D. Chételat, E. B. Khalil, A. Lodi, C. Morris, P. Velickovic, Combinatorial optimization and reasoning with graph neural networks., Journal of Machine Learning Research 24.

[4] V.-A. Darvariu, S. Hailes, M. Musolesi, Graph Reinforcement Learning for Combinatorial Optimization: A Survey and Unifying Perspective. arXiv:2404.06492.

[5] B. Li, G. Wu, Y. He, M. Fan, W. Pedrycz, An overview and experimental study of learning-based optimization algorithms for the vehicle routing problem 9 (7) 1115–1138.

[6] J. Cawalla, R. Pardhi, Graph Reinforcement Learning for Courses of Action Analysis, in: 2024 International Conference on Military Communication and Information Systems (ICMCIS), pp. 1–9.

[7] J. Cawalla, Learning heuristics for course of action analysis with reinforcement learning, in: 2025 International Conference on Military Communication and Information Systems (ICMCIS), IEEE, 2025, pp. 1–9.

[8] A. Delanovic, C. Chiu, J. F. Kolen, M. Gülhan, J. Cawalla, A. Beckus, Airlift challenge: A competition for optimizing cargo delivery (2025). arXiv:2404.17716.

[9] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, Y. Wu, The surprising effectiveness of ppo in cooperative multi-agent games, Advances in neural information processing systems 35 (2022) 24611–24624.

[10] A. Delanovic, C. Chiu, J. Kolen, A. Gnanasekaran, A. Surana, K. Srivastava, H. A. Zhu, Y. Lin, N. Bukingolts, D. Willis, Results of the airlift challenge: A multi-agent AI planning competition, in: Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications V, Vol. 12538, SPIE, pp. 374–393.

[11] D. Bertsimas, A. Chang, V. V. Mišić, N. Mundru, The Airlift Planning Problem 53 (3) 773–795.

[12] P. Toth, D. Vigo, Vehicle Routing: Problems, Methods, and Applications, 2nd Edition, SIAM, 2014.

[13] M. W. P. Savelsbergh, M. Sol, The general pickup and delivery problem, Transportation Science 29 (1) (1995) 17–29.

[14] H. N. Psaraftis, M. Wen, C. A. Kontovas, Dynamic vehicle routing problems: Three decades and counting, Networks 67 (2016) 3–31.

[15] J. Cai, Q. Zhu, Q. Lin, L. Ma, J. Li, Z. Ming, A survey of dynamic pickup and delivery problems, Neurocomputing 554 (2023) 126631.

[16] J. Kolen, Optimizing dynamic airlift operations: Winning strategies in the afrl airlift challenge, The International FLAIRS Conference Proceedings 37 (2024).

[17] R. S. Sutton, A. G. Barto, Reinforcement Learning: An Introduction, MIT press.

[18] S. V. Albrecht, F. Christianos, L. Schäfer, Multi-Agent Reinforcement Learning: Foundations and Modern Approaches, MIT Press, 2024. URL https://www.marl-book.com

[19] W. L. Hamilton, Graph representation learning, Synthesis Lectures on Artifical Intelligence and Machine Learning 14 (3).

[20] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, arXiv preprint arXiv:1707.06347.

[21] Y. Shi, Z. Huang, S. Feng, H. Zhong, W. Wang, Y. Sun, Masked label prediction: Unified message passing model for semi-supervised classification. arXiv:2009.03509.