A Deep Ensemble Method for Multi-Agent Reinforcement Learning: A Case Study on Air Traffic Control

Supriyo Ghosh,[1] Sean Laguna,[1] Shiau Hong Lim,[1] Laura Wynter,[1] Hasan Poonawala [2]
[1] IBM Research AI, Singapore 018983
[2] Amazon Web Services (AWS), United Kingdom
supriyog@ibm.com, {slaguna,shonglim,lwynter}@sg.ibm.com, hasanp1987@gmail.com

Abstract

Reinforcement learning (RL), a promising framework for data-driven decision making in an uncertain environment, has successfully been applied in many real-world operation and control problems. However, the application of RL in a large-scale decentralized multi-agent environment remains a challenging problem due to the partial observability and limited communications between agents. In this paper, we develop a model-based kernel RL approach and a model-free deep RL approach for learning a decentralized, shared policy among homogeneous agents. By leveraging the strengths of both these methods, we further propose a novel deep ensemble multi-agent reinforcement learning (MARL) method that efficiently learns to arbitrate between the decisions of the local kernel-based RL model and the wider-reaching deep RL model. We validate the proposed deep ensemble method on a highly challenging real-world air traffic control problem, where the goal is to provide effective guidance to aircraft to avoid air traffic congestion, conflicting situations, and to improve arrival timeliness, by dynamically recommending adjustments of aircraft speeds in real-time. Extensive empirical results from an open-source air traffic management simulation model, developed by Eurocontrol and built on a real-world data set including thousands of aircrafts, demonstrate that our proposed deep ensemble MARL method significantly outperforms three state-of-the-art benchmark approaches.

Introduction

Reinforcement learning (RL), formulated under the framework of Markov decision process (MDP), experienced a surge in popularity when deep models demonstrated superior performance on game play, particularly on Atari suites and on the game of Go (Silver et al. 2016). In many practical sequential decision making problem, multiple agents share a common environment with the goal of maximizing their individual long-term rewards which can effectively be modeled using cooperative multi-agent reinforcement learning (MARL) model (Zhang, Yang, and Başar 2019). Cooperative MARL has been used successfully in solving real-world applications ranging from taxi fleet optimization (Lin et al. 2018), to distributed traffic light management (Chu et al. 2019) to autonomous driving (Shalev-Shwartz, Shammah, and Shashua 2016). While cooperative MARL enjoys a rich research history (Hernandez-Leal, Kartal, and Taylor 2019), learning in a large-scale decentralized multi-agent environment remains a challenging problem due to partial observability and limited communications between agents. Moreover, solving a cooperative MARL problem as a centralized single-RL problem is intractable since the complexity of the problem increases exponentially with the number of agents due to the joint action or state space. While a straightforward workaround is to consider the agents independently, such as through independent Q-learning (Tan 1993), that leads generally to unstable learning and non-stationarity problem (Matignon, Laurent, and Le Fort-Piat 2012). A popular approach to tackle this non-stationarity issue is to perform centralized learning with decentralized execution (Lowe et al. 2017). In a similar vein, we consider the problem of learning a decentralized, shared policy among homogeneous agents, as done in Gupta, Egorov, and Kochenderfer (2017).

In this paper, we develop two complementary RL methodologies to learn a shared policy by leveraging experience samples collected from all the agents: (i) a model-based kernel RL approach employing an agent-centric state representation that captures local information in the neighborhood; and (ii) a model-free deep MARL model that takes into consideration additional wider-reaching state information. The Kernel-based method is expected to perform well during execution if an encountered state has a dense neighborhood of training experience samples, but it extrapolates poorly. In contrast, a deep MARL method is more flexible and able to generalize well, but is prone to various pathologies and can be brittle, even in regions of dense training data. Therefore, we propose a novel general purpose deep ensemble MARL method wherein we train a separate deep neural network to leverage the pre-trained kernel and deep MARL policies to obtain our final ensemble policy. As a result of that, our ensemble learner can effectively capture the multi-agent interactions during training and efficiently learns to arbitrate between the decisions of these two pre-trained policies during execution; thus leveraging strengths of both these model-based and model-free methods.

We validate the efficacy of proposed deep ensemble MARL method on a highly challenging real-world application of augmenting the capabilities of human air traffic controllers. The passenger demand of airlines is growing rapidly with global passenger demand is expected to be doubling in the next two decades (IATA 2019). In 2018, the world's airlines served 4.3 billion passengers over roughly 22,000 routes, an increase of 1,300 routes and more than 200 million journeys from the previous year alone. With the increase in passenger demand for air travel as well as new forms of urban aircraft, from air taxis to drones, the densification of air traffic will require more sophisticated technologies, including intelligent automated real-time decision support tools for air traffic controllers. In air traffic control, multiple aircrafts need to coordinate in a highly dynamic and stochastic environment, and the decisions taken by one aircraft impact the outcome of that and other aircraft in the future. We model this inherently sequential decision making problem involving multiple actors influencing each other into the MARL framework with the goal of minimizing the combined cumulative long-term effect of conflicts, congestion, delays and fuel costs, by dynamically recommending adjustments of aircraft speeds in real-time. To that end, our key contributions are as follows:

- We model the air traffic control problem using a MARL framework which, different from existing approaches, seeks to minimize the combined cumulative long-term effect of conflicts, congestion, delays and fuel costs.
- We design a kernel based approach employing agent-centric local observation information and a deep MARL model with extended richer state information.
- We develop a novel ensemble learner that leverages the pre-trained policies from the kernel and deep MARL techniques to efficiently arbitrate the complex boundary between the effectiveness of these two methods.
- We conduct extensive experiments by leveraging an open-source air traffic simulator, developed by Eurocontrol and built on a real-world data set consisting of thousands of aircrafts, and empirically demonstrate that our proposed ensemble learner significantly outperforms three state-of-the-art benchmarks: (a) a baseline approach; (b) a local search approach; and (c) a MARL model from Brittain and Wei (2019).

Related Work

Multi-agent reinforcement learning: There is a substantial literature on cooperative MARL, where multiple agents jointly learn a policy to optimize a cumulative future expected reward (Bu et al. 2008). We refer to the recent survey papers by Hernandez-Leal, Kartal, and Taylor (2019) and Zhang, Yang, and Başar (2019) for the details of existing research on cooperative MARL. As the joint action and state space in cooperative MARL grow exponentially with the number of agents, a straightforward approach to tackle this scalability issue is through independent Q-learning (Tan 1993), which leads to unstable learning and non-stationarity issues (Matignon, Laurent, and Le Fort-Piat 2012). Several approaches have been proposed to address this problem, including training with other agents' policy parameters (Tesauro 2004), using explicit communication (Foerster et al. 2016), opponent modeling (Billings et al. 1998), and through centralized learning with decentralized execution (Lowe et al. 2017). In the similar direction of Gupta, Egorov, and Kochenderfer (2017), we consider learning a decentralized, shared policy among homogeneous agents, where the policy parameters are updated using experience transition samples from all the agents.

Ensemble reinforcement learning: While several works have proposed ensemble methods for (single-agent) RL, we are unaware of such in the MARL domain. Existing methods for single-agent ensemble RL rely on either (a) averaging the Q-values from multiple target networks so as to improve stability during the learning process (Anschel, Baram, and Shimkin 2017; Chen et al. 2018) or (b) performing policy-level aggregation during testing through mechanisms such as majority voting or Boltzmann multiplication (Wiering and Van Hasselt 2008; Hans and Udluft 2010). The former cannot, however, take advantage of kernel-based methods, and the latter is not suitable when there are multi-agent interactions. In a similar spirit to our approach, but in a single-agent RL setting, Peng et al. (2018) designed an ensemble method for personalized medical recommendations that learns probabilistic weights for a deep RL and a kernel policy. In that work, the discounted return for an ensemble policy can be computed using off-policy evaluation with a fixed set of patient treatment trajectory data. In our multi-agent setting, however, switching policies across agents according to a probabilistic weight would drastically deteriorate performance. In contrast, our ensemble method involves training a separate master deep neural network to handle the effects of multi-agent interactions during training by taking advantage of a real-world simulator.

Automation of air traffic control: The automation of air traffic control was addressed more than a decade ago by Wollkind, Valasek, and Ioerger (2004), who provide an incremental bargaining process between two aircrafts so as to choose a solution that is Pareto optimal. Farley and Erzberger (2007) propose an auto-resolver that iteratively computes candidate air traffic trajectories until a suitable trajectory is found that satisfies all of the conflicts. Tumer and Agogino (2007) design a comprehensive reward function, but do so independently by considering a fixed geographical location as an agent due to computational limitations. In addition, their learning process did not include state transition dynamics. Recently, Brittain and Wei (2019) addressed the problem of ATC automation using MARL framework. However, they limited their model to handling only potential aircraft conflict resolution, which reduces significantly the complexity of their model. However, as discussed in Tumer and Agogino (2007), air traffic congestion and arrival timeliness are important factors, and conflict resolution in the absence of those considerations will typically lead to unrealistic solutions such as excessive arrival delays. Therefore, we consider a compact reward function combining the penalties for conflict, congestion, arrival delays and fuel cost, and show that our proposed MARL solutions scale gracefully to handle wide-area flight zones with thousands of aircraft.

Motivation: Air Traffic Control

We model the decision making problem of the air traffic control (ATC) in the framework of decentralized multi-agent reinforcement learning (MARL), where each aircraft is represented as an agent. In this section, we begin with a formal definition of the decentralized MARL framework, and then provide the details of different components of the MARL framework for the ATC problem.

A standard reinforcement learning (RL) problem is typically defined with respect to a Markov Decision Process (MDP), given by a tuple $\langle S, A, P, R, \gamma \rangle$, where S denotes the set of all possible states in the environment, A represents the set of permissible actions for the agent, $P : S \times A \rightarrow \mathcal{P}(S)$ denotes the transition function providing the next-state distribution after executing action $a \in A$ in state $s \in S$, $R : S \times A \rightarrow \mathbb{R}$ denotes a reward function providing the expected immediate reward for executing a in s, and $\gamma \in [0, 1]$ is a discount factor. A (potentially randomized) policy $\pi : S \rightarrow \mathcal{P}(A)$ specifies the action to take in each state $s \in S$. The Q-value of a state-action pair (s, a) with respect to a policy π is given by the expected total reward after executing a in s and following π,

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s, a_0 = a\right].$$

The goal of the agent is to learn a policy π^* to maximize $Q(s, a)$ for all (s, a). In finite-state MDP, algorithms such as tabular Q-learning (Watkins and Dayan 1992) can be shown to asymptotically learn the optimal Q function. For large or continuous state spaces, the design of state features plays an important role and some form of function approximation is typically needed.

Analogous to Gupta, Egorov, and Kochenderfer (2017), we propose learning a decentralized, shared policy among multiple homogeneous agents to solve a cooperative multi-agent decision problem. This approach can be seen as solving a multi-agent cooperative Markov game, where all agents share and update the same policy (i.e., $\boldsymbol{\pi}_{\theta}$) simultaneously using experience samples from all the active agents. The game is defined by N agents with their corresponding actions $\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_N$. Let $\mathcal{S}$ denote the global state space and $\mathcal{O}_1, \mathcal{O}_2, ..., \mathcal{O}_N$ represent local observations of the agents. In each time step, the agents execute their actions based on the current policy and local observations producing the next state depending on the transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A}_1 \times \mathcal{A}_2 \times ... \times \mathcal{A}_N \rightarrow \mathcal{S}$. After the transition, each agent i receives a local observation $o_i : \mathcal{S} \rightarrow \mathcal{O}_i$ and a reward $r_i : \mathcal{O}_i \times \mathcal{A}_i \rightarrow \mathbb{R}$.

We now provide the details of the MARL components (i.e., state space, action space and reward function) for each aircraft agent of the ATC problem.

STATE SPACE: To have a rich yet compact representation, we include the following information in the local observation of an aircraft agent:

- *Local information:* the aircraft's geographical location (i.e., latitude and longitude), velocity, direction, distance to the destination and timeliness of the journey with respect to the scheduled arrival time; and
- *Neighbors' information:* For the N nearest aircrafts within a given radius, the relative distance and relative velocity of the aircraft.

Figure 1 illustrates the state representation. Defined this way, the state space remains of constant size as the number of agents increases. We further extend the state space for the Deep MARL so as to enable a richer representation of the domain. For the deep MARL, the state space includes additional two sets of 3×3 grid image information, a coarse grid that covers a large area and a fine grid that covers a smaller area around the aircraft. Each cell is identified by a Geo-hash encoding and the cell size, governed by the precision of the Geo-hash, contains the information about the number of aircraft present in that cell.

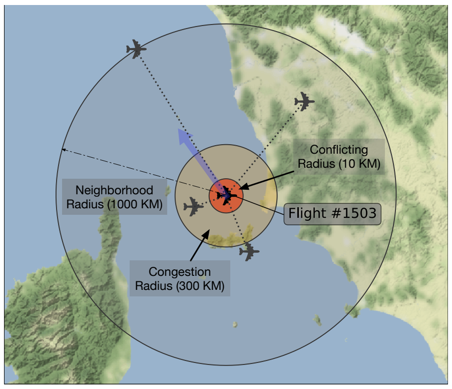

Conflicting Radius (10 KM)
Neighborhood Radius (1000 KM)
Flight #1503
Congestion Radius (300 KM)

Figure 1: State representation used to model en-route ATC.

ACTION SPACE: Our model uses an analogous action space to that of Brittain and Wei (2019), which allows for aircraft speed increase, decrease or no change. Specifically, at every time step, each agent decides whether to maintain, increase, or decrease the aircraft speed by δ meters/second[1], bounded by a minimum and maximum speed. That is,

$$A_t = \{\max(v_{min}, (v_{t-1} - \delta)), v_{t-1}, \min(v_{max}, (v_{t-1} + \delta))\}.$$

REWARD FUNCTION: The reward includes the following four components:

- *Conflict cost:* To ensure a separation distance, if the relative distance between two aircraft is less than a threshold, a high penalty cost is imposed. It should be noted that aircrafts are equipped with automated collision avoidance systems (FAA 2017); thus, a conflict does not lead to collision. However, as conflict resolution is a costly operation in practice, a high penalty cost is set for reducing the number of conflicts;
- *Congestion cost:* To avoid congestion, if the number of aircraft within a given congestion radius exceeds a threshold, a penalty is imposed;
- *Lateness cost:* To minimize the delays, a penalty is imposed if the aircraft is behind schedule, where the penalty is proportional to the delay; and
- *Fuel cost:* The fuel cost penalty is defined as a quadratic function of the deviation of speed from the optimal speed for that type of aircraft, as shown in Figure 2.

[1]The value of δ is set to 10 metres/second in the experiments.

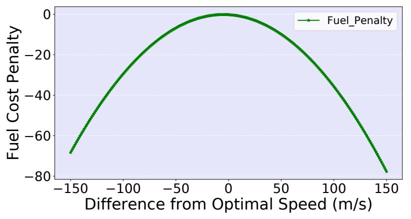


Figure 2: Fuel cost penalty function

Formally, the reward for an agent i at time t is:

$$r_t^i = \alpha \cdot I(o_t^i, R^s) + \beta \cdot I(o_t^i, R^c, N^c) + \gamma \cdot \max(0, \tilde{d}_t^i - d_t^i) + \delta \cdot F(v_t^i - v_0^i)$$

The value of $I(o_t^i, R^s)$ is set to 1 if the relative distance from the nearest aircraft is less than the separation radius R^s according to the current observation o_t^i, and otherwise it is set to 0. The value of $I(o_t^i, R^c, N^c)$ is set to 1 if the number of aircraft within the congestion radius R^c exceeds the threshold value N^c, and 0 otherwise. Let $\tilde{d}_t^i$ denote the expected distance to travel by aircraft i at time t according to the given schedule and d_t^i denote the actual distance traveled; then $\max(0, \tilde{d}_t^i - d_t^i)$ characterizes the amount of delay at time t. $F(v_t^i - v_0^i)$ represents the quadratic fuel cost function depending on the deviation of current speed from the optimal speed v_0^i, which is shown in Figure 2. Finally, α, β, γ and δ denote the weights for conflict, congestion, delay and fuel cost penalties, respectively.

Solution Methodology

In this section, we provide three methodologies for solving a decentralized MARL problem: (a) a kernel based approach employing agent-centric local observation information; (b) a deep MARL model with richer state information; and (c) a deep ensemble approach to leverage the pre-trained policies from the kernel and deep MARL approaches.

Kernel Based RL

Kernel Based RL [KBRL] (Ormoneit and Sen 2002) is a Q-value approximation scheme which has the benefit of having strong theoretical properties. In particular, KBRL algorithms can be proven to converge to a unique fixed point and the approximation error can be bounded asymptotically. KBRL is a model-based approach relying on a set of sample transitions $\mathbb{S} \equiv \{s_k^a, r_k^a, \hat{s}_k^a | k = 1, 2, ..., n_a\}$ associated with each action $a \in A$. These experience transition samples are collected from all the agents by simulating the environment with a particular behavioral (e.g., random or greedy) policy. The complexity of KBRL increases with the number of sample transitions and the size of the state space, thereby limiting its applicability in practice. KBRL was thus extended to achieve better scalability through a modification known as Kernel Based Stochastic Factorization [KBSF] (Barreto, Precup, and Pineau 2016). In contrast to KBRL wherein each sample transition gives rise to a state, KBSF generates a set of representative states and uses stochastic factorization to reduce the size of the transition matrix.

Algorithm 1: SOLVEKBSF()

```
1  Inputs: $S^a = \{s_k^a, r_k^a, \hat{s}_k^a | k = 1, 2, ..., n_a\} \forall a \in \mathcal{A}$;
2  $\bar{S} = \{\bar{s}_1, ..., \bar{s}_m\}$ ▷ $m$ representative states
3  Outputs: $Q^*$ matrix of size $|\bar{S}| \times \mathcal{A}$;
4  Define Gaussian kernel $\bar{\kappa}_{\bar{\tau}}$ and $\kappa_{\tau}^a$ using Equation 1;
5  for each $a \in \mathcal{A}$ do
6      Compute matrix $D^a : d_{ij}^a = \bar{\kappa}_{\bar{\tau}}(\hat{s}_i^a, \bar{s}_j)$ ;
7      Compute matrix $K^a : k_{ij}^a = \kappa_{\tau}^a(\bar{s}_i, s_j^a)$ ;
8      Compute reward $r^a : r_i^a = \sum_j k_{ij}^a r_j^a$;
9      Compute transition probabilities: $P^a = K^a D^a$ ;
10 Solve MDP $\{\bar{S}, \mathcal{A}, P^a, r^a, \gamma = .99\}$ and obtain $Q^*$;
11 return $Q^*$
```

The KBSF algorithm is outlined in Algorithm 1. For the KBSF method, we first collect n transitions (s_t, a_t, r_t, s_{t+1}) by simulating a random policy. We then compute m representative states, $\bar{S} = \{\bar{s}_1, ..., \bar{s}_m\}$ from n transitions using *k-means* clustering. Next, we define a Gaussian kernel, $\bar{\kappa}$ to build the components of the transition matrix of the MDP:

$$\bar{k}_{\bar{\tau}}(s, s') = \bar{\phi}\left(\frac{||s - s'||}{\bar{\tau}}\right); \bar{\kappa}_{\bar{\tau}}(s, \bar{s}_i) = \frac{\bar{k}_{\bar{\tau}}(s, \bar{s}_i)}{\sum_{j=1}^{m} \bar{k}_{\bar{\tau}}(s, \bar{s}_j)} \quad (1)$$

where $||.||$ measures the distance under a Euclidean norm, $\bar{\phi}(x) = \exp(-x^2)$, and $\bar{\tau}$ denotes the kernel width. Using the Gaussian kernel from Equation (1), we define: (a) D^a a matrix of size $n \times m$ with kernel width $\bar{\tau}$; $D^a : d_{ij}^a = \bar{\kappa}_{\bar{\tau}}(\hat{s}_i^a, \bar{s}_j)$ and (b) K^a a matrix of size $m \times n$ with kernel width τ; $K^a : k_{ij}^a = \kappa_{\tau}^a(\bar{s}_i, s_j^a)$. The reward function is computed as $r^a : r_i^a = \sum_j k_{ij}^a r_j^a$ and the transition probability matrix is defined as $P^a = K^a D^a$. The MDP is solved by *Policy Iteration* (Howard 1960) to obtain the optimal Q^* values for the representative state-action pairs. Finally, during the validation phase, we compute the policy $\bar{\pi}(s)$ for any given state s using Equation (2).

$$\bar{Q}(s,a) = \sum_{i=1}^{m} \bar{\kappa}_{\bar{\tau}}(s, \bar{s}_i) Q^*(\bar{s}_i, a); \bar{\pi}(s) = \underset{a}{\operatorname{argmax}} \bar{Q}(s,a). \quad (2)$$

Deep MARL

To solve the Q-learning problem with a much larger state-space, Mnih et al. (2015) proposed using a neural network approximator which they termed deep Q-learning, or DQN. DQN uses a parameter θ learnt using gradient descent on the loss function: $\mathcal{L}_\theta = \mathbb{E}\big[(y^{DQN} - Q(s, a; \theta))^2\big]$, where $y^{DQN} = r + \gamma \max_{a'} Q(s', a'; \theta^-)$ is the target value. Numerous methods have since then been proposed for RL with neural network approximators, including those based on policy gradient (Mnih et al. 2016). The Proximal Policy Optimization (PPO) (Schulman et al. 2017) in particular was introduced to overcome the shortcomings of approaches based on policy gradient. PPO has been shown to offer better performance than both DQN and advantage actor-critic (A2C) (Mnih et al. 2016). PPO as its name implies, seeks to find a proximal policy, and as such to avoid large policy updates.

Let θ denote the parameters of the policy network at time t and $r_t(\theta)$ represent the ratio $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$. The PPO loss function is given by equation (3), where ϵ is a hyperparameter that determines the bound for $r_t(\theta)$ and $A_t := R_t - V(s_t)$ is an estimator of the advantage function at time t.

$$\mathcal{L}^{\text{CLIP}}(\theta) = \mathbb{E}_t\big[\min(r_t(\theta)A_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)\big] \quad (3)$$

We employ the PPO method for learning a shared decentralized MARL policy by leveraging experience samples from all the active agents. We begin the training process by initializing the policy parameters θ to θ_0. In each episode k, the environment is reset to an initial state s_0. Let N_t denote the number of active agents at time t. In each time step t, all active agents sample an action using the current policy parameters θ_k. The joint action $\boldsymbol{a}_t = (a_t^1, ..., a_t^{N_t})$ is then executed and the immediate rewards are observed along with the subsequent state. The N_t sample transitions are also stored in a replay buffer D. After accumulating the samples for an episode, M rounds of update are performed with a minibatch of transitions sampled from buffer D using stochastic gradient descent (SGD) on the loss function of equation (3).

Deep Ensemble MARL

The KBSF has the advantage of its theoretical convergence and asymptotic bounds for the model learnt from experience training samples. We expect KBSF to perform very well during execution if the encountered state lies in a region with dense training examples. However, this is only feasible if the dimension of the state-space remains small. We can therefore afford a relatively compact state representation that includes only a local view of the global state space. With a deep MARL policy we can afford to employ a more ambitious state representation. However, a trained deep RL policy may have pathologies due to non-linear function approximation and can be brittle in certain state regions. We therefore propose an ensemble learner that is able to combine these two approaches across the highly complex boundary between them. Unfortunately, a hybrid (simultaneous) training of the Deep MARL neural network with KBSF, and with the ensemble network, would not be readily feasible since we do not have a gradient of the KBSF method to leverage in the gradient descent step of the neural network training. Therefore, we develop a novel ensemble learning approach wherein we train a separate deep neural network to leverage the pre-trained KBSF and Deep MARL policies to obtain our final ensemble policy. The action space consists of two actions: choose the KBSF policy or choose that of the deep MARL, and we employ the same set of features used in the KBSF method to represent the state space. The architecture of our deep ensemble method is shown in Figure 3.

The key steps of our proposed deep ensemble MARL are provided in Algorithm 2. We begin by loading the trained kernel model $\tilde{K}$ and the trained deep MARL policy $\tilde{\pi}$ parameterized by $\tilde{\theta}$. Let θ denote the parameters of the policy network of our ensemble MARL initialized to θ_0. In each episode k, we reset the environment to an initial state s_0 with a new agent scenario. At each time step t, all active agents,

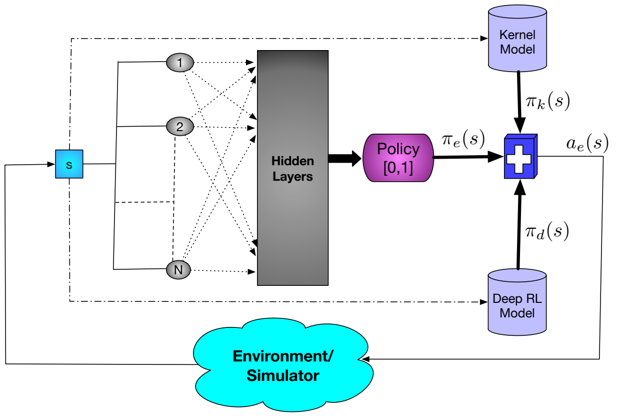


Figure 3: Architecture of the deep ensemble MARL method.

N_t sample actions using the current ensemble policy parameters θ_k. If the sampled action a_t^i for agent i is 0, then we choose an action $\tilde{a}_t^i$ recommended by the KBSF policy and otherwise, the action $\tilde{a}_t^i$ is sampled using the deep MARL policy $\tilde{\pi}(\tilde{\theta})$. We execute the joint action $\tilde{\boldsymbol{a}}_t = (\tilde{a}_t^1, ..., \tilde{a}_t^{N_t})$ in the environment and the agents obtain the immediate rewards along with the data from the next state. The N_t transitions $(s_t^i, a_t^i, r_t^i, s_{t+1}^i)$ are stored in a replay buffer D. Once all the samples are generated for an episode, we perform M rounds of update to the current policy parameters θ_k with a minibatch of transitions sampled from buffer D to optimize the loss function (3).

Experimental Design

In this section, we describe implementation issues concerning the simulator as well as our experiment setup.

Air Traffic Simulator

We employ an open source air traffic simulator developed by Eurocontrol for training and evaluation (Gurtner et al. 2017). The simulator was developed as part of the Enhanced Large Scale Aeronautical Telecommunication Network deployment (ELSA) consortium funded by the Single European Sky Air Traffic Management Research (SESAR) program[2]. The training and testing scenarios make use of aircraft schedule data from a region of airspace in southern Europe. The ELSA Agent-Based Model (ELSA-ABM) simulator allows for simulating any number of aircraft within a set of adjacent sectors. The path of an aircraft f is defined by a sequence of navigation points $\mathcal{P}^f = \{p_1^f, p_2^f, ..., p_n^f\}$ where each point $p_i^f \in \mathcal{P}^f$ is represented by a tuple $\langle l_i, t_i, s_i \rangle$. The location l_i is specified by a latitude, longitude, and altitude value, t_i represents the scheduled time to reach the navigation point and s_i represents the expected speed between the point p_i^f and p_{i+1}^f. At each time-step, the simulator updates positions of an active aircraft using the *Haversine* distance according to current location, speed and other parameters (e.g., wind speed) that may affect the aircraft's movement.

[2]The open-source ELSA ABM simulator codes are accessed from https://github.com/ELSA-Project/ELSA-ABM

Algorithm 2: SolveEnsembleMARL()

```
1  Initialize replay buffer $D$ to capacity $C$;
2  Initialize policy parameters to $\theta_0$;
3  Load trained kernel model $\tilde{K} = (\tilde{S}, \tilde{Q}, \tilde{\kappa_\tau}, \tilde{\mu}, \tilde{\sigma})$;
4  Load trained deep MARL model and policy $\tilde{\pi}(\tilde{\theta})$;
5  for $k = 1$ to max_iterations do
6      Reset environment to state $s_0$ with a schedule $P_k$;
7      for $t = 1$ to $T$ do
8          for $i = 1$ to $N_t$ do
9              Sample action $a_t^i$ according to current observation $s_t^i$ and policy $\pi(\theta_k)$;
10             if $a_t^i == 0$ then
11                 Compute action $\tilde{a}_t^i$ using kernel parameters $\tilde{K}$ in equation (2);
12             else
13                 Sample action $\tilde{a}_t^i$ using trained deep RL policy $\tilde{\pi}(\tilde{\theta})$ for observation $s_t^i$;
14         Execute the joint action $\tilde{\boldsymbol{a}}_{\boldsymbol{t}} = (\tilde{a}_t^1, ..., \tilde{a}_t^{N_t})$ in simulator and get the next state $\boldsymbol{s}_{t+1}$;
15         for $i = 1$ to $N_t$ do
16             Store transition $(s_t^i, a_t^i, r_t^i, s_{t+1}^i)$ in $D$;
17     for $m = 1$ to $M$ do
18         Sample a minibatch of transitions from $D$ ;
19         Update policy parameters $\theta_k$ to maximize the PPO objective function (3);
```

Message Passing Architecture

Due to heavy computational requirements, the ELSA-ABM simulator is written in 'C' language and computes in-memory updates of all aircraft data. However, it does not provide an easy interface for an RL agent to interact with it as an environment. Therefore, we designed and built an adapter for the simulator to allow an external RL agent to extract the aircraft's state and to submit actions via a pre-defined set of definitions. To do so, we designed a two-way message passing system between the simulator and an RL agent using the Nanomsg Next Generation (NNG) library. At each time step, the simulator collects all necessary state and reward information for all active aircrafts and sends it as serialized data to the agent who in turn sends the set of actions to execute to all active aircrafts (refer to Figure 4(a)).

Experimental Settings

The aircraft schedule data covers one 24-hour day and includes, for 1668 aircraft, the source, destination and departure time, a set of navigation points with their corresponding geographical locations and the expected arrival time and speed of each aircraft at each navigation point. From this set, we generate 1000 training and 30 test scenarios by introducing random delay from -30 to +30 minutes to the aircraft's departure time. The 24-hour period is subdivided into 360 time steps, each lasting 4 minutes. We consider 9 state features for both KBSF and ensemble MARL approaches, whereas deep MARL states consists of 44 features. The conflict, congestion and neighborhood radii are defined as 10, 300 and 1000 kilometers, respectively. Penalties are set to -1000 for each conflict, -100 for congestion. The delay penalty for an aircraft is set to -1 for every kilometer it is behind of its expected location. The fuel cost penalty follows the quadratic function illustrated in Figure 2, which is centered at the average expected speed for each aircraft. Three fuel cost scenarios are considered, high, medium and low with a penalty weight factor 5, 2.5 and 1, respectively. Figure 2 illustrates the medium fuel cost scenario. The low, medium and high fuel cost settings are referred as "LowFC", "MediumFC" and "HighFC", respectively. We compare our ensemble method with the following three benchmark approaches:

Baseline: The schedule provided by our data set gives the expected speed between each pair of subsequent navigation points. A baseline is thus derived by simulating the aircraft using those speeds. As our cost function is centered around the baseline schedule speed (thus, the fuel cost plus delay penalty is always lowest for the baseline approach), we compare our approaches against this baseline schedule.

Local search: We employ a myopic best improvement based local search approach. In each time step, each active aircraft chooses its best action (the action with the highest global reward) assuming that other agents will maintain the same speed from the previous time step.

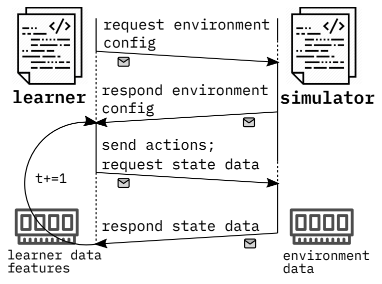
DDMARL: We employ the deep distributed MARL method from Brittain and Wei (2019) as a benchmark approach. For a fair comparison, we adopt their state space information and consider our reward function and action definition to implement this benchmark approach.

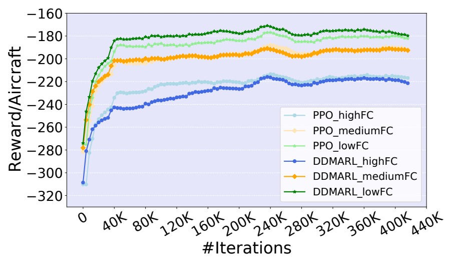
Empirical Results

In this section, we provide the performance comparison results[3] of our proposed ensemble method against other approaches during both the training and evaluation phases.

Hyperparameter Settings: For the training process, 600,000 experience transition samples were generated using a random policy for the KBSF method. These samples gave rise to 1000 representative states using *K-means* clustering. We perform a grid search on a validation set from 0.001 to 1000 to identify the optimal kernel width. For both deep and ensemble MARL PPO method, we use neural network with 2 hidden layers, each consisting of 256 hidden units with *tanh* nonlinearity at the first hidden layer. We set the default parameter values as follows: the discounting factor $\gamma = 0.99$, minibatch size $b = 128$, learning rate $lr = 0.0005$ and clip parameter $\epsilon = 0.3$. In addition, we use a mean standard deviation filter to normalize the state features. We train the deep MARL and DDMARL for 1 million

[3]The experiments were performed on an Ubuntu 16.04 virtual machine with 8-core CPU, 64 GB of RAM, and a single Nvidia Tesla P100 GPU. The distributed Ray framework and RLlib (Liang et al. 2017) were used for the PPO method.


request environment config
respond environment config
send actions; request state data
respond state data
learner
simulator
t+=1
learner data features
environment data
(a)


(b)

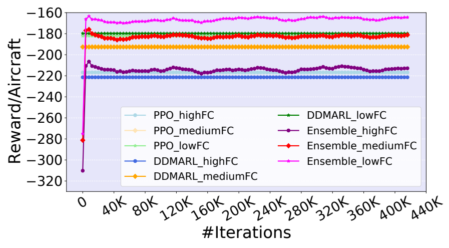

(c)

Figure 4: (a) Message passing architecture built to interface between the ATC simulator and an external RL agent; (b) Training performance of DDMARL vs. PPO Deep MARL; and (c) Training performance of Deep Ensemble MARL vs. DDMARL and PPO.

iterations which on an average takes 36 hours on the GPU. The ensemble approach is trained for 0.5 million iterations.

Training Performance Comparison

Figure 4(b) illustrates the training performance of deep MARL PPO and the benchmark approach DDMARL for the three fuel cost settings. The Y-axis represents the expected reward per aircraft-agent and the X-axis denotes the iteration number. While the PPO performs slightly better for low fuel cost setting, the DDMARL achieves a better training performance in case of hight fuel cost. Figure 4(c) illustrates the training performance of our proposed deep Ensemble method against the best training performance achieved by the PPO and DDMARL for the three fuel cost settings. In all three settings, our deep Ensemble method outperforms the best training reward attained by both the DDMARL and the PPO approach. We note that the Ensemble method does have the advantage that it starts with two pre-trained policies as its "actions" (although learning to properly exploit both policies is still non-trivial), and therefore, Figure 4(c) does not provide a comparison of training computation times across the methods. The computation time needed to train the ensemble method (including 1 million iterations for deep MARL and 0.5 million iterations for ensemble method) is around 1.5 times higher than training the DDMARL method. For all the settings, the Ensemble learner parameters initially overfit to the initial training scenarios. This leads to the somewhat surprising early peak in the training performance within first few iterations. Then, as model training continues, the learner is exposed to other, different training scenarios which help to stabilize the learning process. This initial overfitting effect can be easily confirmed experimentally by running the policies obtained at the early reward peak in the training phase to the test data and observing that those policies perform less well than the ones obtained after more extensive training, in spite of the slightly lower training phase reward of the latter. The key findings from Figure 4(c) are two-fold: (i) The slower convergence of the Ensemble method after the initial peak leads to a more robust solution than that of the initial peak and it offers better testing performance; and (ii) The average reward after convergence of the ensemble method outperforms that of DDMARL.

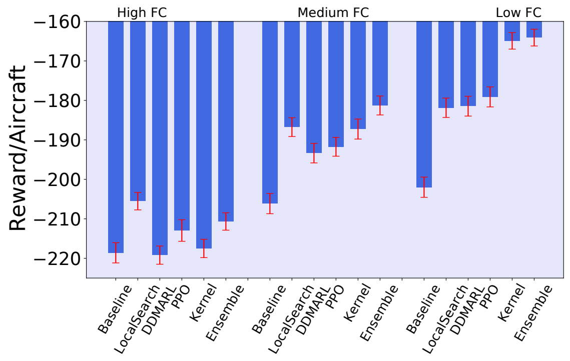


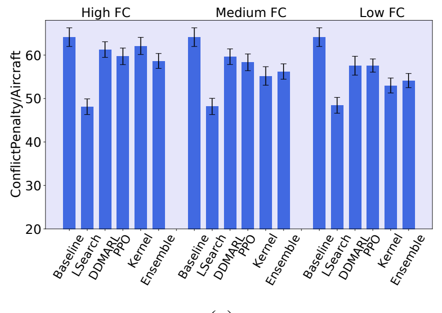
Figure 5: Performance comparison on 30 testing scenarios.

Performance Comparison on Test Data Sets

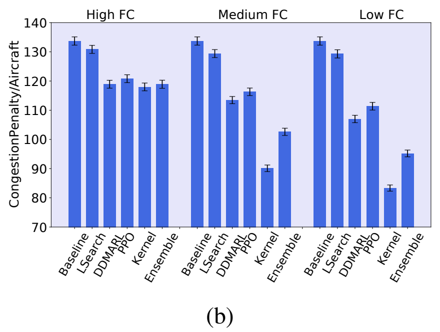
We evaluate the performance of our deep Ensemble method against the three benchmark approaches on 30 test scenarios. Figure 5 depicts the average reward per aircraft agent for our proposed method, the component PPO and Kernel methods, as well as for baseline approaches for the 3 fuel cost settings. The red "I"-shaped indicators provide the standard error of the expected reward. In the case of a high fuel cost, the baseline schedule provides a near optimal solution as deviating from the scheduled speed incurs a significant fuel cost-related penalty. The reward associated with the baseline schedule is quite close to the solution of DDMARL. By myopically improving the baseline schedule, the local search method managed to achieve a 5.5% gain over the baseline. Both PPO and our proposed Ensemble method provide around 3.6% expected reward gain over the baseline.

The medium fuel cost scenario can be considered the most realistic of the three fuel cost settings as it is generated using a polynomial function that replicates the deviation from optimal speed vs. fuel consumption curve of Airbus (shown in Figure 2). In this setting, the proposed Ensemble method significantly outperforms the other approaches. On average, DDMARL, local search, PPO and Kernel provide 6.2%, 8.9%, 7% and 9.2% gain in the expected reward over the baseline, respectively. The Ensemble method provides a 12.1% improvement beyond the baseline.

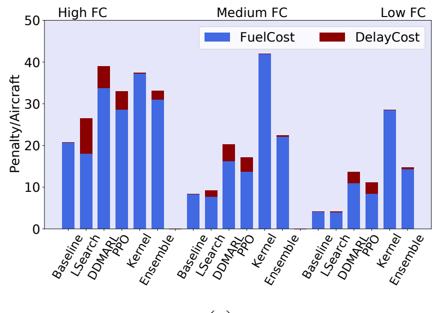
For low fuel costs, the KBSF approach is able to signif-



(a)



(b)



(c)

Figure 6: Performance on test data for reward components: (a) conflicting penalty; (b) congestion penalty; and (c) fuel cost and delay penalty.

icantly outperform PPO and all other approaches. In this case, the deep Ensemble method is able to match the performance of the KBSF approach. The average performance gain of DDMARL, local search, PPO, and Kernel over baseline are 10.3%, 10.1%, 11.3%, 18.3%, respectively, while our proposed Ensemble method beats the baseline by 18.8%.

Figure 6 illustrates the performance of each method on each component of the reward function: the conflict, congestion, fuel cost and delay penalty. These plots demonstrate that unlike other approaches (e.g., kernel, ppo or local search), the ensemble approach does not go to extreme end to optimize an individual component of the reward function and thus, able to maintain the best trade-off over reward components so as to maximize the overall combined reward value (as shown in Figure 5).

Figure 7 illustrates the distribution of the actions taken for the three fuel cost settings. As expected, when the fuel cost is high, the action distribution of all approaches is skewed towards the baseline, since there is little appetite for speed deviation in this case. In the low fuel cost scenario, the distribution is skewed towards speedup, which has the benefit of reducing delays and hence increasing reward. Notice, however, that the local search method is always skewed towards the baseline schedule and thus performs poorly especially in the low-cost case. Our deep Ensemble method succeeds in leveraging the best of both worlds from the fine-grained localized KBSF policies and the more global PPO policies; indeed the Ensemble method gains more from the speedup action than does PPO, whilst not going to the extreme in proposing only speedup actions as is the case with the KBSF. We observe that the ensemble classifier output is skewed towards deep MARL for high fuel cost, whereas the output is mostly skewed towards KBRL actions for low fuel cost setting. In case of medium fuel cost, almost 70% of ensemble actions arise from deep MARL.

In a nutshell, the empirical results demonstrate that our ensemble method can efficiently learn to arbitrate between the Kernel and deep MARL policies. The performance of the Ensemble method was further validated using the reward function of Brittain and Wei (2019), which takes into account only a conflict cost. Notably, the Ensemble method (by leveraging power of KBSF method) outperforms DDMARL on their own reduced reward function by 17% .

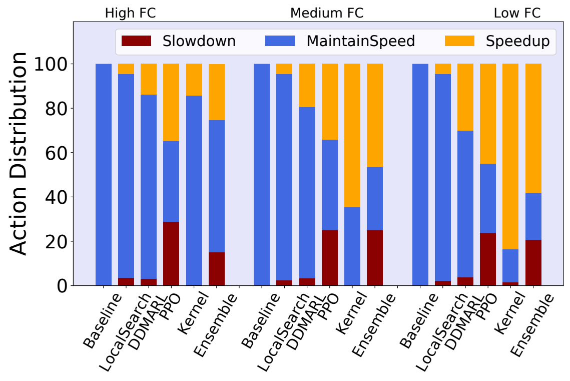


Figure 7: Action distribution on 30 testing scenarios.

Conclusions

In this paper, we propose a novel deep ensemble method that can efficient learn the multi-agent interactions during training so as to efficiently arbitrate between the decisions of a pre-trained local model-based kernel policy and a wider-reaching model-free deep MARL policy. We further demonstrate that the complexity of the air traffic control problem can be concisely and accurately captured by learning an efficient deep ensemble policy within a MARL framework. Experimental results on a real-world simulation model demonstrate the feasibility of using MARL for partially automating the problem of real-time en-route air traffic control as well as the benefits of our proposed deep ensemble method. Thus, this work marks a solid step into using reinforcement learning, and MARL in particular, for large-scale operational control. In future, we see the potential for utilizing our proposed deep ensemble MARL method to combine power of multiple complementary MARL approaches. From an application perspective, we acknowledge that the proposed MARL-based automation is more suitable for en-route aircraft management problem, than the take off and landing problems. Moreover, it will be important to extend the action space with additional controls, including heading and altitude changes, for operational deployment. In both of these extensions, we see that constrained RL has a role to play, to enable taking into account explicitly the complex business rules that govern those expanded control actions.

References

Anschel, O.; Baram, N.; and Shimkin, N. 2017. Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 176–185. JMLR.

Barreto, A.; Precup, D.; and Pineau, J. 2016. Practical kernel-based reinforcement learning. *The Journal of Machine Learning Research* 17(1): 2372–2441.

Billings, D.; Papp, D.; Schaeffer, J.; and Szafron, D. 1998. Opponent modeling in poker. *AAAI/IAAI* 493: 499.

Brittain, M.; and Wei, P. 2019. Autonomous Air Traffic Controller: A Deep Multi-Agent Reinforcement Learning Approach. *ICML Workshop Reinforcement Learning for Real Life, arXiv preprint arXiv:1905.01303* .

Bu, L.; Babu, R.; De Schutter, B.; et al. 2008. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38(2): 156–172.

Chen, X.-l.; Cao, L.; Li, C.-x.; Xu, Z.-x.; and Lai, J. 2018. Ensemble network architecture for deep reinforcement learning. *Mathematical Problems in Engineering* 2018.

Chu, T.; Wang, J.; Codecà, L.; and Li, Z. 2019. Multi-agent deep reinforcement learning for large-scale traffic signal control. *IEEE Transactions on Intelligent Transportation Systems* 21(3): 1086–1095.

FAA, U. 2017. Introduction to TCAS II, Version 7.1.

Farley, T.; and Erzberger, H. 2007. Fast-time simulation evaluation of a conflict resolution algorithm under high air traffic demand. In *7th USA/Europe ATM 2007 R&D Seminar*.

Foerster, J. N.; Assael, Y. M.; de Freitas, N.; and Whiteson, S. 2016. Learning to communicate to solve riddles with deep distributed recurrent q-networks. *arXiv preprint arXiv:1602.02672* .

Gupta, J. K.; Egorov, M.; and Kochenderfer, M. 2017. Cooperative multi-agent control using deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*, 66–83. Springer.

Gurtner, G.; Bongiorno, C.; Ducci, M.; and Miccichè, S. 2017. An empirically grounded agent based simulator for the air traffic management in the SESAR scenario. *Journal of Air Transport Management* 59: 26–43.

Hans, A.; and Udluft, S. 2010. Ensembles of neural networks for robust reinforcement learning. In *2010 Ninth International Conference on Machine Learning and Applications*, 401–406. IEEE.

Hernandez-Leal, P.; Kartal, B.; and Taylor, M. E. 2019. A survey and critique of multiagent deep reinforcement learning. *Autonomous Agents and Multi-Agent Systems* 33(6): 750–797.

Howard, R. A. 1960. *Dynamic programming and markov processes.* John Wiley.

IATA. 2019. Healthy Passenger Demand Continues in 2018 with Another Record Load Factor. *IATA press release* .

Liang, E.; Liaw, R.; Nishihara, R.; Moritz, P.; Fox, R.; Gonzalez, J.; Goldberg, K.; and Stoica, I. 2017. Ray rllib: A composable and scalable reinforcement learning library. *arXiv preprint arXiv:1712.09381* .

Lin, K.; Zhao, R.; Xu, Z.; and Zhou, J. 2018. Efficient large-scale fleet management via multi-agent deep reinforcement learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1774–1783.

Lowe, R.; WU, Y.; Tamar, A.; Harb, J.; Pieter Abbeel, O.; and Mordatch, I. 2017. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. In *Advances in Neural Information Processing Systems 30*, 6379–6390.

Matignon, L.; Laurent, G. J.; and Le Fort-Piat, N. 2012. Independent reinforcement learners in cooperative markov games: a survey regarding coordination problems. *The Knowledge Engineering Review* 27(1): 1–31.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, 1928–1937.

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.

Ormoneit, D.; and Sen, Ś. 2002. Kernel-based reinforcement learning. *Machine learning* 49(2-3): 161–178.

Peng, X.; Ding, Y.; Wihl, D.; Gottesman, O.; Komorowski, M.; Lehman, L.-w. H.; Ross, A.; Faisal, A.; and Doshi-Velez, F. 2018. Improving sepsis treatment strategies by combining deep and kernel-based reinforcement learning. In *AMIA Annual Symposium Proceedings*, volume 2018, 887. American Medical Informatics Association.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* .

Shalev-Shwartz, S.; Shammah, S.; and Shashua, A. 2016. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295* .

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.

Tan, M. 1993. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, 330–337.

Tesauro, G. 2004. Extending Q-learning to general adaptive multi-agent systems. In *Advances in neural information processing systems*, 871–878.

Tumer, K.; and Agogino, A. 2007. Distributed agent-based air traffic flow management. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, 255. ACM.

Watkins, C. J.; and Dayan, P. 1992. Q-learning. *Machine learning* 8(3-4): 279–292.

Wiering, M. A.; and Van Hasselt, H. 2008. Ensemble algorithms in reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 38(4): 930–936.

Wollkind, S.; Valasek, J.; and Ioerger, T. 2004. Automated conflict resolution for air traffic management using cooperative multiagent negotiation. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 4992.

Zhang, K.; Yang, Z.; and Başar, T. 2019. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *arXiv preprint arXiv:1911.10635* .