Departure MANagement with a Reinforcement Learning Approach: *Respecting CFMU Slots*

Ivomar Brito Soares*†, Yann-Michaël De Hauwere*, Kris Januarius†, Tim Brys*, Thierry Salvant†, Ann Nowé*
Computational Modeling Lab
Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussels, Belgium
Email: *{ibritoso,ydehauwe,timbrys,ann.nowe}@vub.ac.be
Airtopsoft SA
Rue aux Fleurs 32, 1000 Brussels, Belgium
Email:†{ivomar.soares,kris.januarius,thierry.salvant}@airtopsoft.com

***Abstract*—This paper considers how existing *Reinforcement Learning* (RL) techniques can be used to model and learn solutions for large scale *Multi-Agent Systems* (MAS). The large scale MAS of interest is the context of the movement of departure flights in big airports, commonly known as the *Departure MANagement* (DMAN) problem. A particular DMAN subproblem is how to respect *Central Flow Management Unit* (CFMU) take-off time windows, which are time windows planned by flow management authorities to be respected for the take-off time of departure flights. A RL model to handle this problem is proposed including the *Markov Decision Process* (MDP) definition, the behavior of the learning agents and how the problem can be modeled using RL ranging from the simplest to the full RL problem. Several experiments are also shown that illustrate the performance of the machine learning algorithm, with a comparison on how these problems are commonly handled by airport controllers nowadays. The environment in which the agents learn is provided by the *Fast Time Simulator* (FTS) AirTOp and the airport case study is the *John F. Kennedy International Airport* (KJFK) in New York City, USA, one of the busiest airports in the world.**

I. Introduction

The number of large intelligent distributed systems, or large scale **Multi-Agent Systems** (**MAS**) [1], that we encounter in daily life is increasing. These systems are characterized by a highly dynamical nature with an increasing number of agents which are constantly joining or leaving the environment, resources that are disappearing or becoming available, increasing demands for a more efficient use of those resources, hidden constraints or modes of operation unknown even by the designers of the system, etc. A few examples are smart energy grids, intelligent traffic signs and **Air Traffic Control** (**ATC**) [2] operations. Because of these characteristics, it is often easier to build a generative model of the environment, that is, the representation of the system in the form of a simulator, rather than an explicit mathematical description of system dynamics.

A **Machine Learning** (**ML**) [3] paradigm called **Reinforcement Learning** (**RL**) [1] [4] appears as a very natural way to model these large scale MAS. The agent based modeling effort can be reduced significantly compared to other approaches since a full view of the system is not needed anymore. It can also allow the system to exceed human controller performance when multiple decisions need to be made in these highly dynamical environments. Furthermore, RL is by its nature adaptive and can change its decisions dynamically depending on the current environmental conditions. Finally, another interesting characteristic of RL is that the information needed to learn a solution can be obtained directly from a simulator. Other common ML approaches like **supervised learning** or **unsupervised learning** are unfeasible in such a setting. Supervised learning because of the lack of examples of the target function to be learned and unsupervised learning because of the absence of an error function to minimize [5].

An important large scale MAS problem today and for the future is the context of the movement of departure aircraft in big airports, commonly known as **Departure MANagement** (**DMAN**) [6] [7]. Several problems need to be handled in a DMAN context, such as: how to make aircraft take-off in a way that will increase runway usage, how to respect assigned CFMU windows assigned by flow management authorities, how to efficiently de-ice on winter climates, etc. All these tasks need to be performed in a way that reduces noise emissions, fuel consumption and general delays. These tasks are currently performed by the different human airport controllers with an increasing support of computer decision support systems.

The particular subproblem this work is interested in is on how to make departure aircraft respect assigned **Central Flow Management Unit** (**CFMU**) slots [8] by taking-off in an assigned time window. An RL model was built to handle such a problem. The environment used by the RL system to derive its solutions is provided by the **Fast Time Simulator** (**FTS**) AirTOp [9]. Finally, the airport being analyzed is the **John F. Kennedy International** (ICAO ID: **KJFK**) **Airport** in New York City, USA, one of the busiests airports in the world.

This paper is organized as follows. In **Section II**, an overview of the necessary theoretical background in RL needed to understand this work is presented. **Section III** presents an overall description of the ATC context. Next, in **Section IV**, the proposed RL model is described in detail, followed by **Section V** which presents experiments and collected results. Finally, in **Section VI**, conclusions and suggestions for future work are presented.

II. Reinforcement Learning Background

Reinforcement Learning (**RL**) [1] [5] is an approach to solve a **Markov Decision Process** (**MDP**). A finite MDP is

978-1-4673-6596-3/15 $31.00 © 2015 IEEE
DOI 10.1109/ITSC.2015.193

defined by the tuple $(S, A, \{T_{sa}\}, \gamma, R)$. $S = \{s_1, ..., s_N\}$ is a finite set of **states**. $A = \{a_1, ..., a_k\}$ are the **actions** available to the agent. Each combination of starting state s_i, action choice $a_l \in A$ and next state s_j has an associated **transition probability** $T(s_i, a_l, s_j)$ and immediate **reward** $R(s_i, a_l)$. $\gamma \in [0, 1)$ is the **discount factor**, which can be used to express that a reward in the near future is more important than a reward that is expected in the far future. The goal for the agent is to learn a **policy** π, which maps an action to each state so that the expected (E) future discounted reward J^π is maximized: $J^\pi \equiv E[\sum_{t=0}^{\infty} \gamma^t R(s(t), \pi(s(t)))]$.

This goal can also be expressed using **Q-values**, one for every action from each possible state, which explicitly store the expected discounted reward for every (s,a) pair: $Q(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q(s', a')$ (s' is the next state).

In order to find the optimal policy, the agent can learn these Q-values, and afterwards select in every state the action with the highest Q-value (**greedy action selection**). In RL, the agent typically does not have any knowledge about the underlying model, e.g., the transition probabilities and reward function are unknown. Watkins described an algorithm to iteratively approximate the optimal values Q^*. In this **Q-learning** algorithm [10], a table consisting of state-action pairs is stored. Each entry contains the estimated value for a specific state-action pair (s, a). The Q-values are updated according to the following update rule: $Q(s, a) \leftarrow Q(s, a) + \alpha_t [R(s, a) + \gamma \max_{a'} Q(s', a') - Q(s, a)]$, where α_t is the **learning rate** at time step t.

There are theoretical guarantees that if all states and actions are visited infinitely often and an appropriate learning rate is chosen, the estimates Q will converge to the optimal values Q^*.

A. Action Selection Mechanisms

When choosing actions to maximize an expected long-term numerical reward, an agent can maintain estimates over the actions and update these estimates over time. The choice the agent has to make is when to start to **exploit** its knowledge (i.e., to decide when to use the best estimated action) instead of continuing to **explore** the other options. Choosing this best action is called **greedy action selection**. If this is done too early, the value estimates of the agent might still be inaccurate and a suboptimal action might be selected. If this is done too late, the total collected reward can be low over the entire time period. The necessity of this balance between exploration and exploitation is commonly known as the **exploration-exploitation dilemma** of RL [1].

A common action selection mechanism in RL that tries to make this balance is the **ϵ-greedy action selection** strategy [1] [5]. This selection mechanism selects the best action most of the time, but with a small probability (ϵ) selects an action uniformly at random, independently of the action-value estimates. An important characteristic of this approach is that it keeps exploring, which is robust against problems in which dynamics or reward structure change over time.

In some settings, it is possible and often desirable to start with a value of ϵ of 1.0 in the beginning, to induce exploration and evaluate more actions, and to decay it with time to induce more exploitation towards the end. Considering ϵ_0 to be the initial value of ϵ and $\tau \in (0, 1)$ the decay, the value of ϵ at a given episode can be given by $\epsilon\,(episode) = \epsilon_0 * \tau^{episode}$.

B. From N-Armed Bandit to the Full Reinforcement Learning Problem

The simplest RL problem is known as the **n-armed bandit** [1], named by analogy to a slot machine with n levers. In this setting, after making a choice of a lever, a numerical reward is received. The goal is to find the lever that will give the maximum long term expected reward. Every action selection is called a **play**. This is considered to be a **Single-State** or stateless RL problem and each choice of a lever is an action.

The n-armed bandit problem is **non-associative**, in which actions do not need to be associated with different situations. It is very common, though, that in general RL problems, tasks are **associative**, meaning that actions need to be chosen in more than one situation, and the goal for the agent is to learn a policy: know which action is best in each case (**Multi-State**), as described previously. When tasks are associative and the effect of an action affects the next situation and future rewards, then the **full RL problem** is reached.

Pseudocode 1 shows how a RL agent chooses actions in an **episodic** task (the agent-environment breaks-down into a sequence of episodes), in a Multi-State setting with a finite MDP, and how it updates the estimates over these actions using Q-Learning. ϵ-Greedy with parameter decay is used as the action selection mechanism.

Algorithm 1 RL Update With Q-Learning and ϵ-Greedy with Decay

```
Initialize Q(s, a) (e.g., 0)
for each episode do
    Agent returns to initial state.
    Decay ε: ε ← ε_0 * τ^episode
    while Final/absorbing state not reached do
        Generate random number n ∈ [0, 1):
        if n ≤ ε then
            Explore: Choose a at random
        else
            Exploit: Choose among a with the highest Q
        Execute action a
        Observe reward r, next state s'
        Update Q of a: Q(s, a) ← Q(s, a) + α_t[R(s, a) +
    γ max_a' Q(s', a') − Q(s, a)]
```

Most of the traditional RL theory is based on the case in which there is only one learning agent, often called the **Single-Agent RL** problem [5]. When multiple agents are learning together in a shared environment, this becomes the **Multi-Agent RL** (**MARL**) [4] problem, which is often more complex then its Single-Agent counterpart and many of the assumptions that guarantee convergence of the former are violated. The DMAN problem we introduce below, is a MARL problem.

III. Departure MANagement

Departure MANagement (**DMAN**) [6] is the process of controlling a departure aircraft from its allocated departure

gate to a departure runway for take-off, respecting all the constraints present in the system and trying to do it in the most efficient way possible. Several problems must be handled in this context, such as: determine the moment an aircraft must leave the gate (**off-block time**) to respect a pre-defined take-off time window, choice of the runway and runway entry, the routing until the runway, the take-off order or sequence such that the usage of the runway and consequently the capacity of the airport is increased, etc. While performing these activities, delays, fuel consumption and noise emissions must be decreased as much as possible. The difficulty in finding satisfactory solutions to all these problems simultaneously increases when complex airport layouts are considered and in situations of intense traffic.

The rest of this Section is described as follows. **Section III-A** presents an introduction to take-off time windows and their importance in ATC, followed by, in **Section III-B**, of a description of how the different airport controllers try to respect those windows. **Section III-C** presents FTS tools for ATC studies. Finally, **Section III-D** shows a general description of KJFK airport.

A. Target Take-Off Time Window

It is common in many operational contexts that a departure flight has a pre-assigned time slot to fly in the airspace. These slots have been pre-calculated, e.g., by flow management authorities, such as the **Central Flow Management Unit** (**CFMU**) [8] in Europe, in order to avoid congestion and to allow for a better use of the whole ATC infrastructure.

In order for the aircraft to fly in its slot, its **Actual Take-Off Time** (**ATOT**) (wheels-off instant) must be within a certain time window. With every window, there is also a specific time, often centered in the window, which is called the **Target Take-Off Time** (**TTOT**). [1]

In airports worldwide, there is still room for improvement in respecting these windows. For example, [8] shows that for **Charles de Gaulle Airport** (**LFPG**) in Paris, France, roughly 80% of the flights succeed in taking off inside their slots. This works uses a genetic algorithm approach combined with a branch and bound graph exploration technique to find the best path and priority for the aircraft and respect the windows.

A **TTOT window** $TTOTW_i$ for a departure aircraft i (ac_i^{dep}) which has an assigned TTOT ($ac_i^{dep,TTOT}$) can be formally defined as follows: the **width** of the window is $TTOTW_i^w > 0$ and its **range** is $[TTOTW_i^{min}, TTOT_i^{max}]$, such that $TTOTW_i^{min} < TTOT_i < TTOTW_i^{max}$ and $TTOTW_i^{max} = TTOTW_i^{min} + TTOTW_i^w$. Finally, $ac_i^{dep,TTOT}$ respects its window if $ATOT_i \in [TTOTW_i^{min}, TTOTW_i^{max}]$.

B. Human Ground Controllers Approaches to Respecting TTOT Windows

A common approach that ground controllers nowadays use to respect TTOT windows is the following. Initially, the gate controllers make an estimation of the time duration between the off-block and take-off times (T_i^{ot}) for $a_i^{dep,TTOT}$ some time before $TTOT_i$. To make this estimation, the controller evaluates the traffic situation in the airport, aircraft type, total taxi length, etc. S/he will then clear the aircraft to off-block at $TTOT_i - T_i^{ot}$. In this work, T_i^{ot} was calculated using the following two approaches:

Average ($T_i^{ot,a}$) = $\sum$ average push back duration (00:02:35), average taxi time duration (total taxi length / 14kt), average runway line up duration (00:00:28), runway acceleration duration. [2]

Exact ($T_i^{ot,e}$): For every aircraft type (A320, B777 etc) and departure gate pair used in the scenario, this is measured with the aircraft taxiing alone in the airport.

Later on, the runway controller, by receiving $a_i^{dep,TTOT}$ in its area of responsibility can make it wait before lining up, if it estimates that it will miss its window by taking-off too early: $ATOT_i < TTOTW_i^{min}$.

C. Fast Time Simulation

While performing studies to improve air traffic, e.g., when developing new **ATC** procedures or improving existing capacity, researchers often need a set of computational tools to allow them to realistically model and simulate those new procedures. They can provide an immediate feedback in terms of the impact on controller workload, distance or fuel burned by the aircraft, general delays, etc.

One type of these simulation tools is the **Fast-Time Simulator** (**FTS**) of which several commercial off-the-shelf versions currently exist on the market. Important characteristics of these tools are that they offer a low cost solution and provide fast and reliable feedback. Because of this, they are used in the early stages of the development of ATC projects. The term **fast time** is derived from the fact that the simulation clock runs faster than a regular clock. [3]

Different FTS have different capabilities, but as a whole they are capable of modeling the following aspects of ATC and **Airport Ground Operations** (**AGO**) domains: *EnRoute* flight phase, **Terminal Maneuvering Area** (**TMA**), aircraft and airport handlers, vehicles ground movements, etc.

D. John F. Kennedy International Airport

In 2011, **John F. Kennedy International airport** (**KJFK**) in New York City, USA, was ranked the 6th busiest airport in the US and 17th in the world, with a total of 48 million passengers and 400.000 movements in that year [6]. It covers a surface of 21 km^2, with eight terminals and a total of 128 gates. The terminals are surrounded by a dual ring of taxiways and the total taxiway length is 40km. There are two pairs of parallel runways around the terminal area and outside the dual ring of taxiways: 13R-31L, 4R-22L, 4L-22R, 13L-31R. See **Figure 1** for an overview of the KJFK model used.

[1] Depending on the ATC system generating these windows, TTOT can have other names such as Calculated Take Off Time (CTOT) and be used in an operational context different from the CFMU slots.

[2] These values, including the taxi-speed used, are averages measured over all departure flights of KJFK. Runway acceleration time is estimated for each aircraft individually.

[3] The simulation speed is only limited by the performance of the computer on which it is running.

Fig. 1: John F. Kennedy International Airport, New York City, USA (Yellow nodes are the gates, orange are the runway entry/exit points. Terminal buildings in the center area of the airport are highlighted in gray).

IV. Departure MANagement Reinforcement Learning Model

This Section describes the proposed RL model used to make departure flights respect their TTOT windows. Initially, in **Section IV-A**, a description of the environment at which the agents learn is presented. Followed by, in **Section IV-B**, a description of the MDP. Thirdly, in **Section IV-C**, it is shown how this probleman be mapped from the N-armed bandit to the full RL problem. Finally, in **Section IV-D**, an example is given for a Single-Agent in a Multi-State context to have a better intuition on how the model works.

A. Environment

The whole ATC environment, either on ground or in the air, is provided by the FTS **AirTOp** [9] (**Air Traffic Optimization**) simulator. In this environment, a **Flight Plan** (**FP**) is assigned to every flight in the simulation. The FP defines, among other things, the origin and destination airports, gates to use, the routing in the airspace to be used by the flight, etc.

For **departure flights** (ac_i^{dep}), the aircraft is expected to off-block at the nominal time defined by the planning authorities in its FP. Once it off-blocks, it is assigned the shortest path to the runway entry and will taxi at the maximum speed allowed for its aircraft type and the taxiway it is on. After entering the runway by the most appropriate entry, it will accelerate and take off. It will fly its pre-assigned **Standard Instrument Departure** (**SID**) route followed by its ***EnRoute*** one [6].

Arrival flights (ac_i^{arr}), after their *EnRoute* phase, are assigned a **Standard Terminal Arrival Route** (**STAR**), with a clearly defined approach phase, prior to their landing in the airport. After landing, they decelerate and leave the runway at the most appropriate runway exit and taxi to their assigned arrival gate.

While taxiing on ground, several **safety** distance requirements must be met. These distance requirements are often dependent on safety and operation constraints of the particular airport and airspace in its vicinity. One common one is based on **wake vortex separations** [6], which define the minimum distance between the movement of two consecutive aircraft, due to trailing air turbulence generated by the movement of the leading aircraft.

Finally, the taxipath in the airport is represented as a network of nodes and links as in a graph, similar to the grid world environments [5] typically considered in RL. Each of the elements that make up the ground routing of an aircraft, e.g., gates, runway entries etc, is represented by one of these nodes. See **Figure 1** for a schema of the layout of KJFK airport. This scenario is modeled and solved with a RL approach detailed at next.

B. Markov Decision Process

In this section an MDP process is introduced by defining its agents, states, actions, and the reward function.

Agent: A FP controller agent a_i. One for each $ac_i^{dep,TTOT}$. $ac_i^{dep,TTOT}$ receives instructions from a_i that it needs to respect.

States (S): Three different types of state are defined depending on the phase of $ac_i^{dep,TTOT}$ from the moment it appears at the gate until its $ATOT_i$:

1) **Parked** (initial state) (s_i^p): a_i enters this state when $ac_i^{dep,TTOT}$ parks at its departure gate and until its **Actual Off-Block Time** (**AOBT**). It is defined by:

- **Departure Gate** (g_d): Assigned departure gate.

- **Entry Time** (e_p): Instant $ac_i^{dep,TTOT}$ appears at g_d.

2) **Taxiing** (intermediate states) ($S_{i,j}^t = s_{i,1}^t, ..., s_{i,N}^t$): a_i initially enters this state at $AOBT_i$. [4] It switches to the next taxiing state after it has crossed the exit node of the current state. It is defined by:

- **Entry Node** (n_e): Entry node for the current portion of the ground path of the aircraft.

- **Exit Node** (n_x): Exit node for the current portion of the ground path of the aircraft.

- **Entry Time** (e_t): Instant $ac_i^{dep,TTOT}$ crosses n_e.

3) **Taken-Off** (goal/absorbing states): a_i enters this state at $ATOT_i$. Two possible absorbing states of this category are possible:

- **Taken-Off Inside Window** ($s_i^{TTOTW,in}$): $ATOT_i \in TTOTW_i$.

- **Taken-Off Outside Window** ($s_i^{TTOTW,out}$): $ATOT_i \notin TTOW_i$.

Actions (A): Different actions are available for the agent:

- **Delay Off-Block** ($A^o = a_1^o, ..., a_L^o$): Choice of the time duration to delay the nominal off-block time. Available at s_i^p.

[4] Strictly speaking from an ATC point of view, the aircraft undergoes different phases, such as pushing back, taxiing, queuing to enter runway etc. To simplify the model, it was decided to group all these phases in one state called taxiing state.

- **Delay During Taxiing** ($A^t = a^t_1, ..., a^t_M$): Choice of the amount of delay to apply to $ac_i^{dep,TTOT}$ at n_x. The aircraft comes to a full stop. Available at S^t_i.

- **Take-Off** (a^e) : Action taken when the controlled aircraft reaches its lift-off speed at the runway. It is always executed. Available at S^t_i.

Reward Function (R): a_i receives a reward at $ATOT_i$. If its final state is $s_i^{TTOTW,in}$, a reward $r_i^{TTOTW,in}$ is observed. This reward consists of a max value (r^{max}) from which is deducted a penalty ($p_i^{taxiing}$) equal to the total time duration $ac_i^{dep,TTOT}$ was stopped during taxiing ($d_i^{taxiing}$) times a penalty factor ($f^{taxiing} > 0$): $r_i^{TTOTW,in} = r^{max} - p_i^{taxiing} = r^{max} - f^{taxiing} * d_i^{taxiing}$. [5] If the final state is $s_i^{TTOTW,out}$, a reward $r^{TTOTW,out}$ is granted. For the agents to have an incentive to transition to $s_i^{TTOTW,in}$, the following relation must be respected: $r_i^{TTOTW,in} > r^{TTOTW,out}, \forall d_i^{taxiing}$.

C. From Single-Agent Single-State to Multi-Agent Multi-State

Different RL models can be used depending on the configuration of the learning setting: the number of learning agents, Single-Agent or Multi-Agent, and whether a Single-State or Multi-State approach is being used. The goal is to identify the simplest model that can be used to learn an appropriate solution for each case. Transitions in complexity follow roughly the order shown below. The motivation for the increase in complexity, from the ATC point of view, is included in the description.

- **Single-Agent Single-State**: There is only one learning agent and the only state considered is the Parked State with the Delay Off-Block actions. It tries to respect the window by absorbing all the delay at the gate. These are often desired solutions in an ATC context, since all the delay to respect the window is absorbed at the gate and the aircraft engines are turned off, thus minimizing fuel consumption.

- **Multi-Agent Single-State**: The scenario is similar to the Single-Agent Single-State case described above, but in this case multiple agents are learning in a shared environment.

- **Single-Agent Multi-State**: The transition from a Single-State to a Multi-State case can occur since to respect the window, it is not always possible to absorb all the delay at the gate. E.g., the aircraft needs to leave the gate because an arriving flight is requesting it, or to avoid the traffic in the vicinity of its gate close to its AOBT. Since the aircraft is leaving the gate earlier than desired, it might be necessary to further delay the aircraft during its path to the runway.

- **Multi-Agent Multi-State**: The scenario is similar to the Single-Agent Multi-State case, but with multiple learning agents in a shared environment.

D. Single-Agent Multi-State Example

In order to have a better understanding of the RL model, namely the state transitions and the final reward received, this Section presents as an example a Single-Agent in a Multi-State setting for the airport layout shown at **Figure 2**. In this airport, a departure aircraft is parked at gate a and will taxi to runway 27, crossing nodes b and c and taking-off at d.

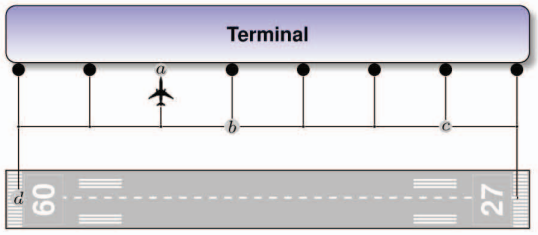

Fig. 2: Airport layout.

To keep the representation short and easier to visualize, Taxiing state transitions and Delay actions are only defined at nodes a, b and c. The nominal off-block time in the FP is 08:00:00, $T^{ot,e}_{a-b} = T^{ot,e}_{b-c} = T^{ot,e}_{c-d} = 5min$, $T^{ot,e} = 15min$. [6] $TTOTW$ definition: $TTOTW^{min}$ = 08:16:00, $TTOTW^{max}$ = 08:18:00. If the AOBT is 08:00:00 and the aircraft taxi to runway 27 with no delay applied, its ATOT will be 08:15:00, missing its window ($ATOT_i < TTOTW_i^{min}$).

The following set of Delay Off-Block actions $A^o = \{0s, 30sec, 60sec\}$ can be applied at the gate a. The same set can be used as A^t at nodes b and c. The reward function parameters are: $r^{max} = 100$, $r^{TTOTW,out} = 0$, $f^{taxiing} = 0.5$.

The state transition diagram is shown on **Figure 3**. [7] Inside each Parked and Taxiing states, e_p and e_t are shown. Each column groups the states associated with a given n_e/n_x pair, shown at the bottom. The arrows connecting the s^p_i and S^t_i indicate the delays available in each state (the uppermost arrow is always $0sec$). The final ones leading to the Taken-Off states represent the Take-Off actions.

Multiple solutions exist that will make the aircraft respect its window. The one with a maximum of delay applied at the gate and a minimum during taxiing is a=$60sec$, b=$0sec$, c=$0sec$, ATOT=08:16:00 $\in TTOTW_i$, $r_i^{TTOTW,in} = 100$. This solution minimizes fuel consumption. If the solution a=$0sec$, b=$30sec$, c=$60sec$, ATOT=08:16:30 $\in TTOTW_i$ is adopted, $r_i^{TTOTW,in} = 100 - 0.5 * 90 = 45$.

V. Experiments

Several learning scenarios were derived from a baseline model of KJFK consisting of 1409 FPs, being 698 departures and 711 arrival FPs to KJFK. The total simulation time spans roughly two days of operation of this airport. All departure flights of KJFK used in this study use runway 31L (which is also used by arrivals). In **Section V-A**, it is described how the different learning scenarios were derived from this baseline model. At next, in **Section V-B**, a description of the RL set up used is given. Finally, in **Section V-C**, simulation results that compare the performance of the learning algorithm and the human ground controllers are shown.

[5] Delay on ground and fuel consumption are two correlated objectives. If delay on ground increases, fuel consumption also increases. In this sense, fuel consumption can also be penalized instead of taxiing delay.

[6] Deceleration and acceleration times on a full stop are considered to be zero, and no speed variations are considered to simplify the example.

[7] Transition probabilities are all 1 in this representation and are omitted.

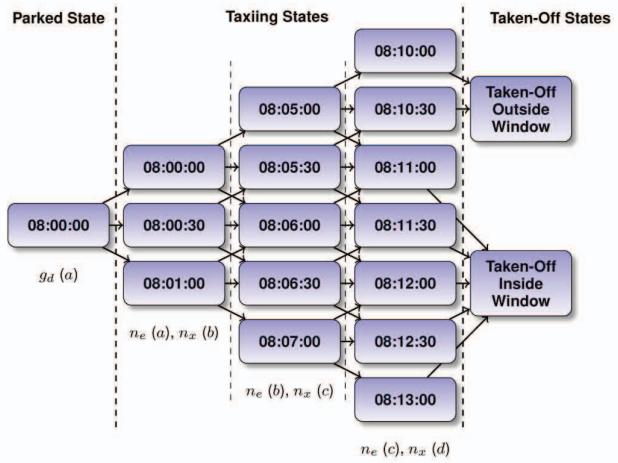

Fig. 3: State transition diagram: Single-Agent Multi-State.

A. Setting Up Learning Scenarios

Initially, a TTOTW was created for every departure flight of KJFK (there were 698). First, each $TTOTW_i$ was generated by measuring $ATOT_i$ for every $ac_i^{dep,TTOT}$, when it taxis alone in the airport. All measured $ATOT_i$ were then sorted in ascending order and whenever $ATOT_{i+1} - ATOT_i < 3min \rightarrow ATOT_{i+1} = ATOT_i + 3min$ to achieve this min separation. [8] These sorted/shifted ATOTs were converted into TTOTs. Finally, a window was created centered around each $TTOT_i$ with $TTOT_i^w = 2min$. In **Table I**, a few examples of TTOTW generated for some $ac_i^{dep,TTOT}$ of KJFK are shown.

FP Callsign	TTOTW		
	TTOTW^{min}	TTOT	TTOTW^{max}
AAL0005D	07:19:12	07:20:12	07:21:12
DAL0067D	07:22:12	07:23:12	07:24:12
JBU0065D	07:25:12	07:26:12	07:27:12
AAL0007D	07:28:12	07:29:12	07:30:12
DAL0009D	07:31:12	07:32:12	07:33:12

TABLE I: Some TTOT windows generated for learning scenario 6 (AAL = American Airlines, JBU = Jet Blue, DAL = Delta Airlines).

Afterwards, different learning scenarios were defined from this baseline model. The learning scenario with index 0 is generated by grouping all departures of KJFK that have their TTOT within a 1h interval starting from the first TTOT. This window is then shifted 1h and the subsequent departures are grouped in another learning scenario and so on. The arrivals considered for each case are the ones that are touching down in KJFK during this time frame. This is a first approximation for a rolling horizon window commonly used in these types of planning systems [6]. In total, 42 learning scenarios were generated comprising all departure flights from KJFK. **Table II** shows the number of departure flights and the learning problem type, in terms of the number of learning agents, per learning scenario index.

[8] This value is enough to account for runway separation requirements of KJFK.

Scenario	6-38	5	0	39	1,3,4	2,41	40
# of Dep	20	13	8	6	3	1	0
# of AT	MA	MA	MA	MA	MA	SA	-

TABLE II: Number of departure flights per learning scenario index (Dep: Departure Flights, AT: Agents Type, MA: Multi-Agent, SA: Single-Agent).

B. RL Set Up

Agents are all independent learners (they do not take each other into consideration) and use Q-learning with $\alpha = 0.2$, $\gamma = 0.8$. [9] Action selection mechanism is ϵ-greedy with parameter decay: $\epsilon_0 = 1.0$, $\tau = 0.995$. A learning trial ends when $\epsilon = 0.001$ for all agents (1378 episodes). Every agent is trying to maximize its own expected reward (competitive setting [5]) with $r^{max} = 10,000$, $f^{taxiing} = -1.0$. [10]

Simulations are performed on both **deterministic** and **stochastic** settings. In the stochastic case, the amount of time the pilot spends in making her/his final check after receiving the clearance of the control tower to off-block is modeled as a Gaussian distribution ($\mu = 20sec$, $\sigma = 5sec$). Uniform ground speed reductions up to $5kt$, every $3min$ of taxiing time, also happen for every flight (arrivals and departures). In a multi-agent setting, the environment as experienced by the agents becomes non-deterministic because of the interactions with the other learning agents, so the terms deterministic and stochastic are used to differentiate when those probability distributions are used. [11]

Finally, at both Single-State and Multi-State settings, A^o were defined with a range of $[-10min, 10min]$ centered around $TTOT - T^{ot,e}$ and a step of $10sec$ for every agent (121 actions per s_i^p). In the Multi-State case, A^t were defined with a range of $[0, 1min]$ and a step of $10sec$ (7 actions per S_i^t) close to each apron exit. [12]

C. Results

All 42 learning scenarios were executed initially with a Single-State approach on both deterministic and stochastic settings. The results are averages over 10 learning trials. For the learning case, the values shown in **Figures 10 - 11** are the ones measured at the last episode, when all agents are behaving greedy. Whenever the maximum possible number of take-offs inside window was not reached, a Multi-State approach was tried for it. The same scenarios were also evaluated with the human ground controllers trying to respect all $TTOTW_i$ on the same two settings. One final experiment was performed only modeling the behavior of the human gate controllers.

Initially, a more detailed analysis of learning scenario 6 in the deterministic setting is shown through **Figures 4 - 9**. This scenario is chosen because the Single-State approach could not

[9] In most practical applications, a constant low α (e.g., 0.1 or 0.2) is used. γ is chosen close to one.

[10] These were chosen such that the reward function constraint holds true for KJFK.

[11] These are approximate values used in an operational context.

[12] In an operational context, it is common for a few airports in the U.S.A., for the aircraft to stop for some time in the apron exit in order to wait for a radio frequency switch from the gate controllers to the taxiing controllers.

find a solution ($ATOT_i \in TTOTW_i$) for all agents, but the Multi-State approach did.

Figure 4 shows the number of take-offs inside window as learning progresses, the Single-State case stabilizes on 19 on average while the Multi-State case in 20 (the maximum possible). Fuel consumption, shown on **Figure 5**, for the Multi-State case is higher because of the additional stop the departure aircraft face at the apron exit. In a post-analysis evaluation of this scenario, the Single-State case did not succeed in finding a solution because arrival flights taxiing in the vicinity of the gate prevent the departure flight from continuing. The Multi-State approach succeeds in this task, because the agent learns to make the aircraft off-block earlier and also stop for some time. If the aircraft would only leave the gate earlier, with no additional stop, it would be too early in the runway to take-off ($ATOT_i < TTOTW_i^{min}$) and miss its window.

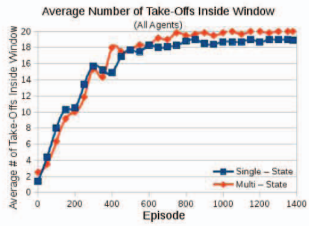

Fig. 4: Average # of $TTOTW^{in}$.

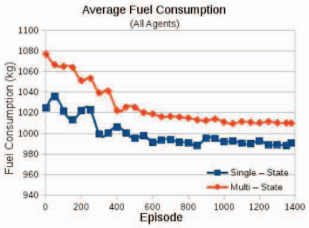

Fig. 5: Average fuel.

Figure 6 shows the average reward for all agents in the learning scenario. In the Multi-State case, it converges to a value close to the maximum possible (r^{max} * number of a_i). In **Figure 7** the average reward for a few selected agents of this scenario is shown (the middle ones shown on **Table I**). It can be seen that JBU0065D, in the Single-State case, has an average reward of 0 during the whole learning process. This is an indication that $ATTOT_{JBU0065D} \notin TTOTW_{JBU0065D}$ in this setting. In the Multi-State case, this problem is solved.

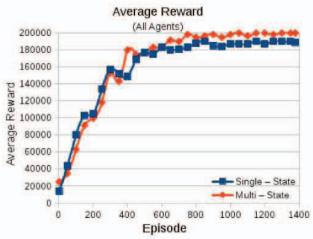

Fig. 6: Average reward (all).

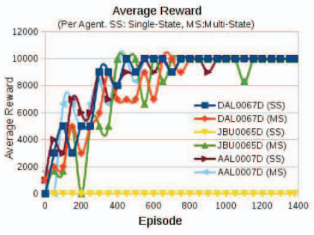

Fig. 7: Average reward (per).

Figures 8 - 9 show a profile of the general delays on ground for the departure aircraft as learning progresses. **Figure 8** presents the average gate delay. The Single-State case has higher gate delay because the only solution available to meet the window is to delay the off-block of the aircraft. In **Figure 9** the average taxiing delay is higher in the Multi-State case because of the additional stop at the apron exit, as described previously.

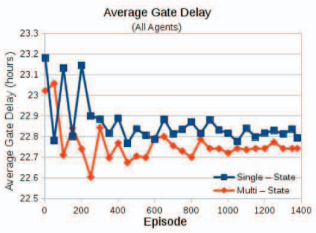

Fig. 8: Average gate delay.

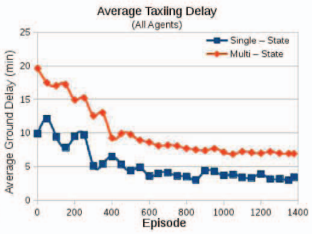

Fig. 9: Average taxiing delay.

Finally, let's consider the state-space size. In the last learning trial for the Multi-State approach, there were 2111 taxiing states visited for the deterministic setting and 2168 visited taxiing states for the stochastic setting.

Figures 10 - 11 show a comparison of the performance of machine learning and the human ground controllers in terms of the percentage of windows respected and fuel consumption for each scenario. The results shown are a combination of the Single-State and Multi-State approaches. The Single-State approach is the one shown by default because it had equal or higher percentage of $TTOTW_i$ respected in most cases. A Multi-State is displayed (highlighted in the x axis index) if its percentage was higher.

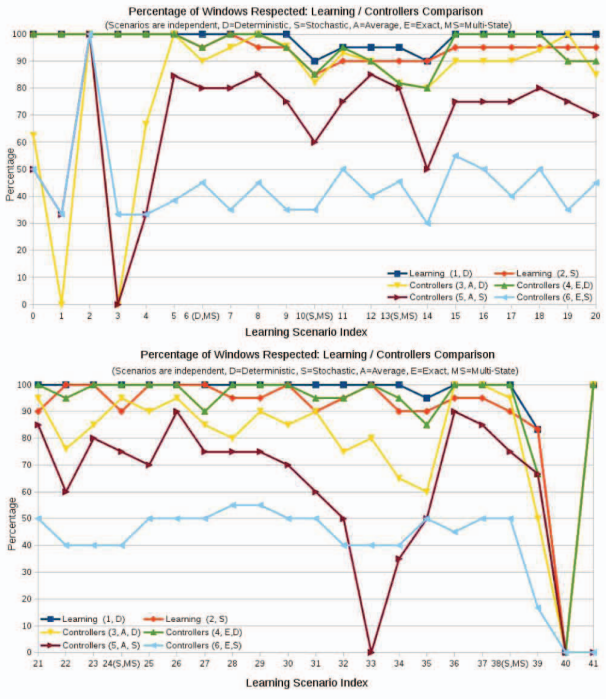

Fig. 10: Percentage of KJFK departure flights that take off inside window. (MS besides the learning index indicate that a Multi-State approach was used.)

For the percentage of windows respected, shown in **Figure 10**. Although the human controllers perform reasonably well in the deterministic setting, machine learning still outperforms

they because it succeeds in finding solutions that overcome situations where the departure aircraft faces traffic disturbance on its path to the runway (departures or arrivals). In the stochastic setting, the performance of the human controllers decreases considerably, but machine learning still succeeds in maintaining a rate of more than 90% of windows respected in most of the scenarios evaluated. **Table III** shows a percentage of windows respected for all scenarios together.

Percentage of Windows Respected			
Case		**Environment**	
		Deterministic	**Stochastic**
Machine Learning		99	96
Gate Controllers	$T^{ot,a}$	85	71
	$T^{ot,e}$	97	44
Gate + Runway Controllers	$T^{ot,a}$	87	70
	$T^{ot,e}$	96	44

TABLE III: Percentage of windows respected for all scenarios.

For fuel consumption, shown in **Figure 11**, learning in most of the scenarios, provides lower fuel consumption for the departure aircraft. Fuel consumption is higher in the stochastic setting compared to deterministic one because of the speed reductions provided by the uniform taxi speed variation which causes the aircraft to taxi for a longer time. **Table IV** shows the fuel consumption comparison for all scenarios together.

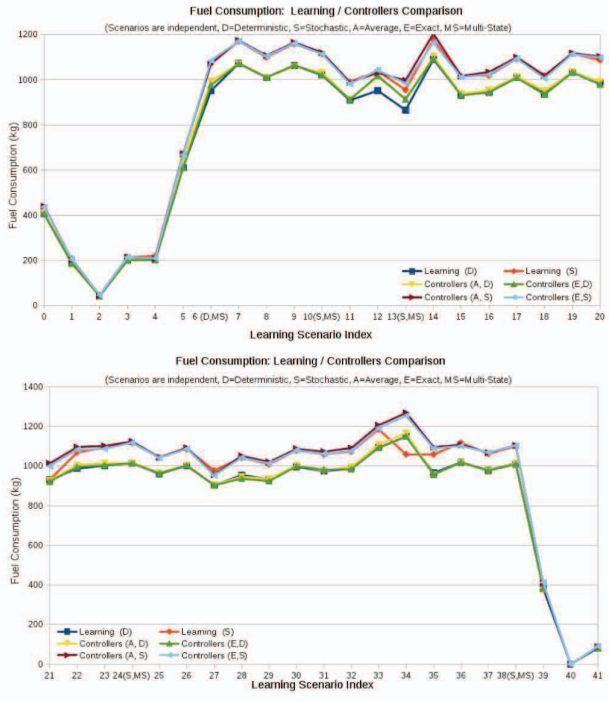

Fig. 11: Average fuel consumption for departure flights.(An M besides the learning index indicates that a Multi-State approach was used.).

Fuel Consumption (kg)			
Case		**Environment**	
		Deterministic	**Stochastic**
Machine Learning		34,806	37,989
Gate Controllers	$T^{ot,a}$	35,057	38,106
	$T^{ot,e}$	34,839	37,865
Gate + Runway Controllers	$T^{ot,a}$	35,412	36,613
	$T^{ot,e}$	34,847	37,872

TABLE IV: Fuel consumption for departure aircraft for all scenarios.

VI. Conclusions and Future Work

RL has showen to have good potential for modeling and finding solutions for respecting assigned take-off windows for departure aircraft. In a complex and busy airport such as KJFK, in a deterministic setting, it managed to find solutions for almost all cases. In a stochastic setting its performance rate dropped slightly but remained above 90%, while the performance of the human airport controller modeled on the same task starts to decrease more drastically. The Single-State setting shows the advantages of reduced fuel consumption and a reduced learning problem since there are no visited taxiing states. It has the disadvantage of increased gate delay and not being able to find a solution for all cases, e.g., when it needs to avoid traffic taxiing in the vicinity of the gate, which were successfully solved by the Multi-State approach in some situations. Future steps for this research, from a learning point of view, is to evaluate cases in which the agents take each other into consideration (joint action learning) and on how to further generalize from previous experience. From an operational point of view, interesting follow up steps are: try the same approach on different airport layouts with different operational constraints, different window configurations, such as generated by a real flow management system.

References

[1] A. G. Barto, *Reinforcement Learning: An Introduction*. MIT press, 1998.

[2] K. Tumer and A. Agogino, "Improving air traffic management with a learning multiagent system," *IEEE Intelligent Systems*, vol. 24, no. 1, pp. 18–21, 2009.

[3] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, *Machine learning: An Artificial Intelligence Approach*. Springer Science & Business Media, 2013.

[4] A. Nowé, P. Vrancx, and Y.-M. De Hauwere, *Game Theory and Multi-agent Reinforcement Learning*. Springer, 2012, ch. Reinforcement Learning: State of the Art, pp. 441–470.

[5] Y.-M. De Hauwere, "Sparse interactions in multi-agent reinforcement learning," 2011.

[6] R. De Neufville and A. Odoni, *Airport Systems: Planning, Design and Management*, 2013.

[7] S. Stroiney, B. Levy, and C. Knickerbocker, "Departure management: Savings in taxi time, fuel burn, and emissions," in *Integrated Communications Navigation and Surveillance Conference (ICNS)*. IEEE, 2010, pp. J2–1–7.

[8] J.-B. Gotteland, N. Durand, and J.-M. Alliot, "Handling cfmu slots in busy airports," in *5th USA/Europe Air Traffic Management Research and Development Seminar*, 2003.

[9] Airtopsoft, "Airtop fast time simulator," http://www.airtopsoft.com/, 2005, [Online; accessed 23-February-2015].

[10] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.