HAL open science

CONFLICT RESOLUTION WITH TIME CONSTRAINTS IN THE TERMINAL MANEUVERING AREA USING A DISTRIBUTED Q-LEARNING ALGORITHM

Antoine Henry, Daniel Delahaye, Alfonso Valenzuela

To cite this version:

Antoine Henry, Daniel Delahaye, Alfonso Valenzuela. CONFLICT RESOLUTION WITH TIME CONSTRAINTS IN THE TERMINAL MANEUVERING AREA USING A DISTRIBUTED Q-LEARNING ALGORITHM. International Conference on Research in Air Transportation (ICRAT 2022), Jun 2022, Tampa, United States. hal-03701660

HAL Id: hal-03701660

https://enac.hal.science/hal-03701660v1

Submitted on 22 Jun 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

CONFLICT RESOLUTION WITH TIME CONSTRAINTS IN THE TERMINAL MANEUVERING AREA USING A DISTRIBUTED Q-LEARNING ALGORITHM

Antoine HENRY
ENAC
7 avenue Edouard Belin
31400 Toulouse, France
Email : antoine.henry@alumni.enac.fr

Daniel Delahaye
ENAC
7 avenue Edouard Belin
31400 Toulouse, France
Email : delahaye@recherche.enac.fr

Alfonso Valenzuela
ETSI, Universidad de Sevilla
Camino de los Descubrimientos s/n
41092 Sevilla, Spain
Email : avalenzuela@us.es

***Abstract*—With the growing number of flights, more and more conflicts have to be solved in Terminal Manoeuvring Areas (TMAs). In order to keep a fluid flow of aircraft arriving on an airport, air traffic controllers use softwares to help them to solve conflicts and sequence aircraft on runways. This paper faces the sequencing and merging problem using a reinforcement learning algorithm (Q-Learning) in order to measure its performance. This algorithm has been run on a scenario representing a regular day at Paris Charles de Gaulle airport (CDG), and gives satisfying results. Then, it has been benchmarked on heavily-loaded scenarios, with more aircraft than the previous ones in order to see the limits of reinforcement learning efficiency. The Q-Learning algorithm can not only solve conflicts on this heavily-loaded scenario but it also has a reasonable computational time. By using a Q-learning algorithm in a distributed way, we aim to find an optimized solution on heavily-loaded scenarios without compromising the computational time.**

I. Introduction

Air traffic keeps increasing each year since the mid - twentieth century. This increasing number of flights forces air traffic controllers to be more efficient. The capacity has to be improved in the TMA without impacting the safety. This means that controllers will have to resolve more and more potential conflicts between aircraft. In order to decrease the workload, most of international airports are equipped with softwares to assist them to manage traffic. These softwares, called arrival managers (AMANs), not only sequence aircraft on runways but also ensure a minimum separation between aircraft in the TMA. In summary, such tools organize the traffic in order to maximize the runway throughput by computing decisions for each aircraft which are delivered by the controllers.

The aircraft separation problem is complex since there are many decision variables inducing a large space state. By considering only speed and flight level as decision variables, a pre-sequence can be computed in the En-route airspace [1]. The generated flight instructions could be directly used by the controller, shaping a pre-sequence without any conflict when aircraft enter the TMA. Using this pre-sequence composed by aircraft entering at a certain time in the TMA, the algorithm should be able to solve conflicts in a heavy-loaded scenario. In the TMA, aircraft are usually controlled by speed regulation. When such decision are not enough, as when there are too much uncertainties, tactical vectorings are also assigned to some aircraft in order to solve the remaining conflicts. Such vectorings induce extra workload for the controller.

In order to reach the same objective, the Point Merge System (PMS) has been developed in the 90s. Such PMS consists in a circular leg in the TMA where aircraft are assigned to follow for a given amount of distance (2,3,4,..NM) in order to create efficient separations between aircraft. It is an efficient way to solve conflict problem when several Standard Arrival Routes (STARs) converge towards the same runway. This technique allows aircraft to add delay between each other by flying different paths. Some previous related works [2], [3] have used optimization algorithms for computing efficiently the amount of time aircraft have to stay in the PMS leg for building an optimal sequence at the runway.

The aim of this paper is to figure the best performance of a machine learning algorithm on sequencing and merging problem. Since this problem could be modelized as a Markovian Decision Process (MDP), the Q-Learning algorithm (QL) has been chosen for its simplicity and performance on these problems. It is a model-free reinforcement learning algorithm which finds the optimal policy corresponding to the maximum reward that an agent can get. This problem can be addressed in a centralized way by using global optimization approach [1] but the associated computation time may be too high. In order to reduce this computation time, a selective simulated annealing algorithm has been proposed in [4] for which the computation can be divided by 20 compared with [1]. This approach still need some centralized communication between aircraft in order to be efficient.

In the current work, we propose a fully decentralized sequencing and merging algorithm based on a distributed Q-learning methodology, meaning every aircraft is considered as a learning agent. A difficult point is that every agent learns

without being aware of the state or actions of others. This is why the QL used in this paper is not only distributed but also selected: all agents are not learning at the same time. Agents with the worst reward are considered critical, they learn before others. Furthermore, this problem is simplified by using a time decomposition approach [5] [6]. Instead of solving the problem for a whole day, the algorithm is applied to a set of time windows sliding along the day. This approach presents two main advantages. First, it considers aircraft in the near future for which uncertainty is reduced, thus increasing the validity of the decisions taken by the algorithm (as a matter of fact, it is irrelevant to take decisions in the morning for aircraft that will land in the evening). Second, taking into account aircraft in the next time window will reduce the size of the problems the algorithm will have to address. Therefore, the QL is run only on flights in a certain time window, which slides during the whole day.

This algorithm has been tested over 24-hour scenarios. These scenarios represent either a normal day at Paris Charles de Gaulle airport or heavily-loaded scenarios - with up to 50% more traffic. The results of the algorithm will be considered as satisfying when every conflict in a scenario will be solved, with a minimized average delay on aircraft.

The structure of the paper is summarised as follows. The next section presents mathematical modeling of this problem. The third section describes the Q-learning algorithm used to solve this problem. The fourth section presents benchmarks on Paris Charles de Gaulle airport.

II. Mathematical model

This section describes the mathematical model of aircraft, network and decision variables.

A. Network

Based on the model developed in [7], STARs are represented with a graph $\mathcal{G}(\mathcal{N}, \mathcal{L})$ where $\mathcal{N}$ is the set of nodes and $\mathcal{L}$ the set of links. Among the set of nodes one can identify two subsets: $\mathcal{N}_e \subset \mathcal{N}$ is the subset of entry points of the TMA and $\mathcal{N}_r \subset \mathcal{N}$ is the subset of runways. The final links connecting the runways are also grouped in a link subset: $\mathcal{L}_r \subset \mathcal{L}$. The CDG current network is represented in figure 1. We propose in this study to feed both runways, 27R and 26L, by a double point merge system which will merge on initial fix of each runway (IF_27R - RWY_27R and IF_26L - RWY_26L, respectively).

For each aircraft, the procedure is flown at constant speed. This speed is the landing speed which, in this work, we assume, for simplicity, that depends on its wake vortex category.

The PMS structure is represented on figure 2. For each aircraft, the length of PMS arc will be considered as a decision variable for the algorithm.

B. Aircraft data

A flight f is characterized with the following information:

- $V_{0,f}$: the initial true air speed of the aircraft
- $t_{0,f}^{TMA}$: the initial entry time in the TMA

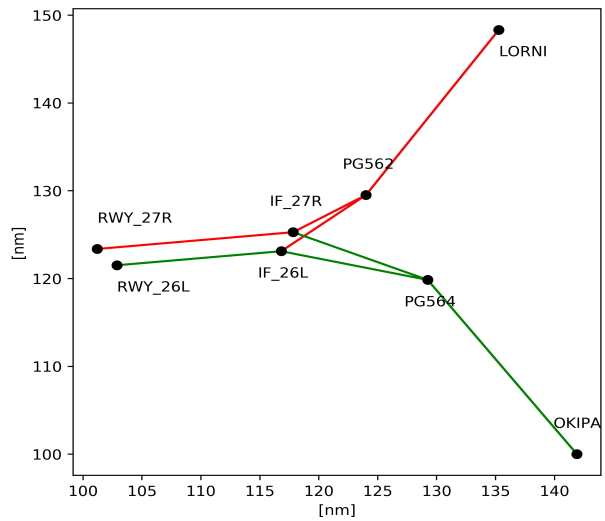

Fig. 1. Simplified STAR model at CDG: aircraft enter at LORNI or OKIPA. Merge points are located on both IF_27R and IF_26L

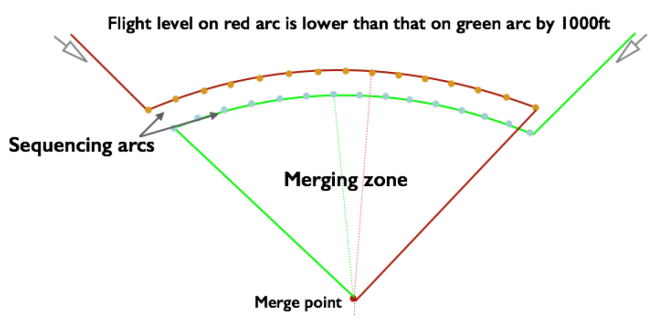

Fig. 2. Merge Point Topology : for each aircraft, the length of the flown sequencing arc is a decision variable

- $r_{0,f} \in \mathcal{N}_r$ the runway on which the aircraft is planned to land
- t_f^{RTA} the time at which the aircraft is required to land
- C_f : the wake vortex category

C. Decision Variables

For each flight f, the following decision variables are considered :

- the speed of the aircraft, V_f
- the entry time in the TMA, t_f^{TMA}
- the runway assigned for landing, r_f
- the length of the merge point arc, l_f^{MP}

Such decision variable have to meet some constraints which are now introduced.

Speed: The speed of the aircraft has to stay in a given range of the initial speed:

$$V_f \in V_{0,f} + p\Delta V$$

with p the number of increments and ΔV the speed increment:

$$p \in \mathbb{Z} \text{ and } p\Delta V \in \left[\Delta V^{min}, \Delta V^{max}\right]$$

and:

- ΔV^{max} the maximum speed increase from $V_{0,f}$ that can be assigned to an aircraft;
- ΔV^{min} the minimum speed decrease $V_{0,f}$ that can be assigned to an aircraft which depends on the wake vortex category.

Entry time in TMA: The entry time decision corresponds to a delay which can be absorbed in the En-Route airspace before the aircraft enters the TMA. In this airspace, the aircraft can be slowed down or accelerated in a given range. As a result, the entry time in TMA of the aircraft could also change in a given range:

$$t_f^{TMA} \in t_{0,f}^{TMA} + p\Delta T$$

with p the number of increments and ΔT the time increment:

$$p \in \mathbb{Z} \text{ and } p\Delta T \in \left[\Delta T^{min}, \Delta T^{max}\right]$$

and:

- ΔT^{max}, ΔT^{min} the maximum and minimum time increments from $t_{0,f}^{TMA}$ that can be assigned to an aircraft.

Runway: To keep a balanced flow between runways, it could be sometimes more appropriate to change the landing runway of an aircraft ($r_f \in \mathscr{N}_r$).

Length of merge point arc: As the network contains merge points, one of the decision variable, l_f^{MP}, is the length of the arc that an aircraft will fly in one of the merge point.

$$l_f^{MP} \in p\Delta L$$

with p the number of increments and ΔL the length increment:

$$p \in \mathbb{N} \text{ and } p\Delta L \leq L_{max}^{MP}$$

and L_{MP}^{max} the maximum arc length that a merge point could have.

III. Optimization algorithm

A. Markovian Decision Process

In this problem, each flight is a Markovian Decision process MDP$\{S, A, P_a, R_a\}$. All decision variables represent the state space S. It means that, for every aircraft, a state is defined by {speed, entry time in the TMA, PMS arc length, runway assignment}. In each state, the following actions are considered: $A = \{$increasing/decreasing the speed, increasing/decreasing the entry time in the TMA, increasing/decreasing the PMS arc length, changing the landing runway, no action$\}$. For states that are not direct neighbors of the current state, the transition function value is 0. For neighbor states, the transition function is an equiprobalistic one:

$$\mathrm{P_a(s, s')} = \begin{cases} 0, \text{ if s' is not a neighbor of s} \\ \frac{1}{Card(A)}, \text{ otherwise} \end{cases}$$

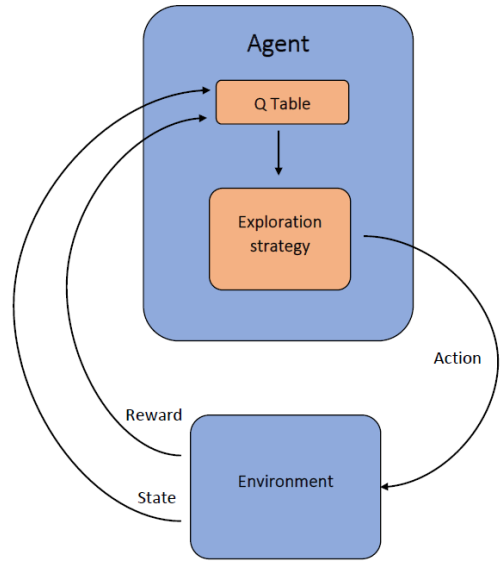

Fig. 3. The agent - environment interaction in a Q-Learning : the agent does not know his environment but only the state and reward following an action

With $Card(A)$ the number of elements in A, eight in this case.

B. Q-learning

The Q-Learning (QL) is a model-free reinforcement learning algorithm [8]. This means that the algorithm does not need a model of the environment, it only interacts with the environment without knowing it, see figure 3. It is also an off-policy algorithm: the policy only determines which state-action pairs are explored. For a correct performance, policies need to ensure that every state-action pairs are continuously explored.

Every aircraft is considered as an agent, making it a multi-agent Markovian Decision Process (MDP) problem. The QL has been well studied on MDP [9]. The QL is used to learn the optimal policy of a certain MDP. This is done by computing the Q-function for each aircraft - representing the expected reward that an agent can receive if he takes a given action in a given state. The QL used is a distributed one, meaning that the reward of each agent is treated individually at each iteration. Therefore, they are considered as independent learners.

For each agent, the expected reward $Q(s, a)$ in a given state s, for a given action a, is updated as follows:

$$Q(s, a) += \alpha(R + \gamma\, max_{a'} Q(s', a') - Q(s, a)) \quad (1)$$

with:

- s' the new state when the action a is taken in the state s;
- R the reward the agent will receive by making the action a in s;
- α the learning rate;
- γ the discount factor.

The expected reward $Q(s, a)$ in a given state s, for a given action a is updated at each iteration considering an estimation

of the optimal future value $max_{a'}Q(s', a')$. This is done independently of the policy being followed. Precisely, this is a one-step QL since the estimation is only done by looking one iteration ahead.

The pseudo code of the Q-Learning is given in Section III-B.

Algorithm 1 Q-Learning

Initialisation for each agent $Q(\cdot,\cdot) = Q_0$
Require: $T_0, \beta, T_f, iterations$
  while $T \geq T_f$ **do**
    for $i \geq iterations$ **do**
      for agent $\in$ critical flight set **do**
        Choose action a from s using the Boltzmann strategy;
        Take the action a, observe R and s';
        Update $Q(s, a)$ for this agent according to (1);
        $s \leftarrow s'$
      end for
      $i \leftarrow i + 1$
    end for
    $T \leftarrow \beta \times T$
  end while

In order to balance the ratio between exploration and exploitation, a softmax exploration (or Boltzmann exploration) has been used. For a state $s \in S$, an action $a \in A$ and the temperature T, the probability $\pi(s, a)$ to choose a in s is given by:

$$\pi(s, a) = \frac{e^{Q(s,a)/T}}{\sum\limits_{a' \in A} e^{Q(s,a')/T}}$$

The temperature at the iteration number k, is given by a geometric law of parameter β, i.e. $T_k = T_0\beta^k$, with T_0 the initial temperature. This temperature set the trade-off between exploration and exploitation : a relatively high temperature will promote the exploration of the Q table, contrary to a low temperature which is in favor of the exploitation of the Q table.

In this distributed Q-Learning, every aircraft is considered as a learning agent and consequently has a Q-table. All the Q-tables are initialized at a Q_0 value chosen relatively low to enforce the state exploration. It is done on purpose since the reward - and the Q-table - of an aircraft depends on agents close to it (meaning which could be in conflict). Every agent is seen as an independent learner. Then, for a given agent, the choice of an action does not consider the chosen action of other agents but only their actual states. Therefore, between two decision makings of an agent, its environment could have been changed. In order to reduce this possibility, an agent could choose the specific action of doing nothing, since then, its state will not change.

C. Reward function

For each aircraft, a reward function is computed to be used by the reinforcement learning algorithm. The reward given at each state and action depends on other aircraft's state and is computed as the weighted sum of rewards described below. All rewards presented here are negative, therefore the function reward has to be maximized.

$$\begin{aligned} R =& w_{rta}(R_{rta} + 5R_{runway}) + \\ & w_{conflict}(\sum R_{link} + \sum R_{node}) \end{aligned}$$

If an aircraft f does not land on $r_{0,f}$, its preferred runway, the reward added is 5 times the R_{runway} weighed by w_{rta}. This is equivalent to a 5 minutes penalty. w_{rta} and $w_{conflict}$ will be established in section IV.

Next, the different terms of the reward function are described.

Required Time of Arrival: All airlines have a schedule for each aircraft and on-time aircraft should have a better reward. Then, a reward corresponding to the absolute difference between the RTA and the real arrival time is added to every aircraft.

$$R_{rta} = -|t_f^{RTA} - t_{arrival}|$$

Runway Number: This model considers airport with two runways used for landing. Such as before, airlines have dedicate terminals, therefore their aircraft prefer to land on a runway close to the gate in order to reduce taxi time.

$$R_{runway} = \begin{cases} 0, \text{ landing on the required runway, } r_{0,f} \\ -1, \text{ otherwise} \end{cases}$$

Conflicts: When an aircraft is flying on a network, loss of separation may only appears on nodes or on links. Then, our model considers two kinds of conflicts: link conflict when two aircraft do not respect the wake vortex category separation and node conflict when aircraft do not respect the 3 NM horizontal separation at merge points [7].

Links Conflicts: For each link, at the entrance and the exit, the minimum separation between two aircraft f and g has to correspond to the table I.

Category		**Leading Aircraft, f**		
		Heavy	***Medium***	***Light***
Trailing Aircraft, g	***Heavy***	4	3	3
	Medium	5	3	3
	Light	6	5	3

TABLE I
SEPARATION MINIMA FOR LINK CONFLICT IN NM

Assuming that $s_{f,g}$ is the separation minimum and $d_{f,g}$ is the actual distance between the leading aircraft f and the trailing g, see figure 4, the criticality of a potential conflict, C_{link}, is proportional to the distance between aircraft. Overtakings are also computed, if this occurs, then $d_{f,g} < 0$ and the criticality of the conflict is set to -1.

$$C_{link} = \begin{cases} -1, \text{ if } d_{f,g} < 0 \\ -\frac{|s_{f,g} - d_{f,g}|}{s_{f,g}}, \text{ if } d_{f,g} < s_{f,g} \\ 0, \text{ otherwise} \end{cases}$$

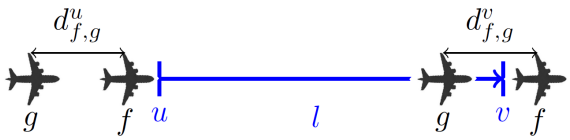

Fig. 4. Link conflict detection based on the comparison of distance between aircraft at the beginning or the end of a link with the separation minima

C_{link} is a piecewise linear and continuous function which is necessary for the learning algorithm: between two decision makings, the algorithm needs to know if the conflict is getting better or worst. Since C_{link} could be close to 0 such as R_{rta}, the learning algorithm could improve the R_{rta} instead of resolving the conflict. In order to prioritize the conflict solving objective, the reward function for the link conflict is artificially set between -0.3 and -1 while preserving the linear variation using the formula:

$$R_{link} = -0.3 + (C_{link} * (1 - 0.3))$$

Node Conflicts: Even if there is no link conflict between two aircraft f and g, there still could be conflicts on nodes. In TMA, every aircraft has to be separated by 3 NM from others in order to respect the separation distance. But as [7] shows, in many airports, thanks to the network geometry, the detection area could be reduced to a 2.2 NM radius circle, see figure 5. As for the links, the criticality of a conflict on a node is given by the formula:

$$C_{node} = \begin{cases} -\frac{2.2 - d_{f,g}}{2.2}, \text{ if } d_{f,g} < 2.2 \\ 0, \text{ otherwise} \end{cases}$$

As for the links, the reward function used for a node conflict is artificially set between -0.3 and -1 while preserving the linear variation by the formula:

$$R_{node} = -0.3 + (C_{node} * (1 - 0.3))$$

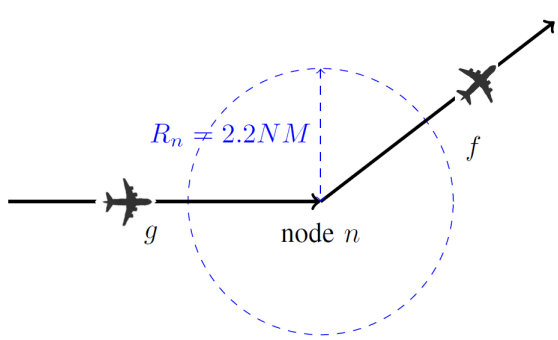

Fig. 5. Node conflict based on a circle detection area

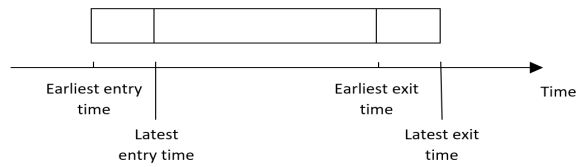

Fig. 6. Representation of an aircraft in a Sliding Window

Sliding window length	21 min
Time shift for the sliding window	7 min

TABLE II
SLIDING WINDOW PARAMETERS

D. Sliding Window

In this problem, if an aircraft enters in the TMA many hours before another one, their decisions can be considered as independent. Therefore, it is not relevant to optimize the whole problem in one attempt. This is why we introduce a sliding window and run the optimization algorithm only on it. It reduces the number of irrelevant decision variables and the computational time [5] [6].

Aircraft are classified into four groups, see figure 6:

- *Completed*, the latest landing time is before the starting time of the window;
- *On going*, the earliest entry time is before the starting time of the window. For these, decisions have already been made
- *Active*, the earliest entry time and the latest entry time is in the time window
- *Planned*, the latest entry time is after the end of the sliding window

At each iteration of the sliding window, the optimization algorithm is run on active flights.

The difference between the earliest entry time and the latest entry time for a given aircraft is about 7 minutes. Based on this, the sliding window wide has been fixed at 21 minutes (see table II). Also the time shift between two consecutive sliding windows has also been fixed to 7 minutes.

E. Critical Flight Set

Running the algorithm on every active flight in the sliding window is not efficient enough, some them have already a good reward and others have multiple conflicts. In order to speed up the optimization process, decisions are changed with a higher priority on aircraft with the worse reward. Those aircraft are indicated as critical flights.

Those critical flights are computed by using a threshold, which is compared to the reward. Aircraft in this critical flight set are these above 70% of the worst aircraft reward. Since these aircraft are learning, their rewards decrease and more and more aircraft are considered as "critical" as the average reward lowers.

Q_0	-150
α	0.1
γ	0.95
w_{rta}	1
$w_{conflict}$	100

TABLE III
THE QL PARAMETERS CHOSEN

IV. APPLICATION

A. Scenario

The benchmarks have been done on heavily-loaded scenarios but only considering LORNI and OKIPA entries. The initial fight set representing a regular day at CDG on those two entry points contains a total of 458 aircraft. However, in order to simulate extra demand, some of them have been duplicated and added to the initial set. In order to be coherent with the reality, aircraft have been added in regards with the density of aircraft in the initial set. The probability for an aircraft to be added at a high density time is higher than to be added at a low density time. The algorithm has been run on sets up to 50% bigger than the initial demand, i.e. 687 aircraft.

The maximum speed of aircraft in the TMA has been fixed at 250kts since they are below flight level 100. The minimum approach speed they can have depends on the wake vortex category: 220kt for heavy aircraft and 190kt for medium aircraft. This approach speed is a variable of the QL. This scenario does not contain light aircraft. After the merge point, in the final leg, the speed is fixed at 185kt for heavy aircraft and 155kt for medium aircraft.

For each different decision variable, different steps as been chosen : 10kt for the speed step, 20 seconds for the time step and 0.5 nm for the PMS arc length step. With that, the number of state possible is 5040 for a Medium aircraft and 2520 for a Heavy aircraft.

B. QL Parameters Optimized by Genetic Algorithm

As [10] shows, the Boltzmann exploration strategy required temperature parameters to be well chosen in order to perform well. To find the best combination of parameters, a genetic algorithm [11] has been used. The QL parameters optimized by the genetic algorithm are the following: T_0, T_f, β, iterations, critical flight set threshold.

The other QL parameters is given by the table III. w_{rta} has been set to 1 and $w_{conflict}$ to 100 to ensure that the algorithm is first focused on conflict solving then on minimizing delays.

Evaluation: In this algorithm, an individual is a combination of Q-Learning parameters. To evaluate an individual, the QL is run 5 times with the corresponding parameters. The fitness associated to this individual is the average of final rewards of the five runs minus the standard deviation of these rewards - reminder, rewards are all negatives. This is done to not only optimize the best result that the QL can have but also the stability of the QL with these parameters. The pseudo-code of such evaluation is summarized in algorithm 2.

Algorithm 2 Evaluate an individual

```
procedure EVALUATE(individual)
    for i ← 1, j do                ▷ j = 5 is enough to be precise
        Run the QL with the parameters given by the individual
        Observe the reward R_i
    end for
    return The average of all R_i minus the standard deviation of all R_i
end procedure
```

Parameter	Value
T_0	100
T_f	1
β	0.99
Iterations	50
Critical flight set Threshold	0.7

TABLE IV
THE BEST QL PARAMETER COMBINATION FOUND BY THE GENETIC ALGORITHM

Selection: Individuals used to generate a new generation are those with the best fitness.

Crossover: A uniform crossover has been chosen, for every child parameter, the value is randomly chosen between the two parents.

Mutation: For every parameter of every individual, the probability of a mutation is 0.2. The new value is chosen randomly in the parameter set.

The overall genetic algorithm is given in algorithm 3.

Algorithm 3 Genetic Algorithm

```
Initialisation Choose randomly individuals in the initial parameters set
Require: n                          ▷ The number of generations
for i ← 1, n do
    for individual in population do
        EVALUATE(individual)
    end for
    Selection
    Crossover
    Mutation
end for
The final parameter set is composed by the parameters contained in the final population
```

At the end, the best parameter combination is represented in table IV. With these parameters, the QL results are presented in table V.

In order to avoid the genetic algorithm to overfit the QL parameters, it has been run multiple times on different heavily-loaded scenarios. On some scenarios, every conflict has been solved and the algorithm couldn't go further. This is showed in figure 7, the critical flight set threshold parameter gives the same result in the range [0.7, 0.9].

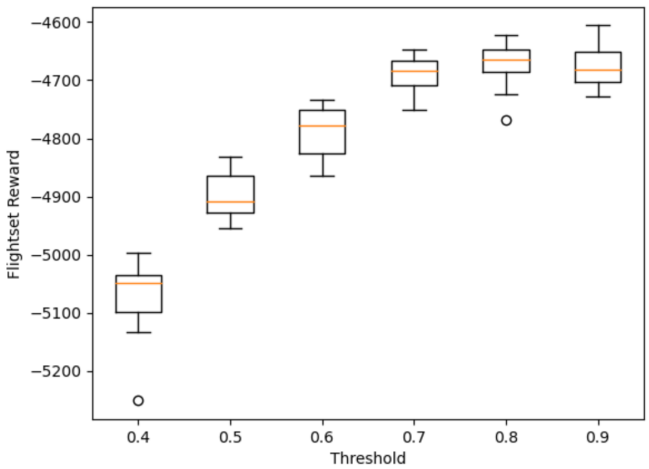

Fig. 7. Genetic algorithm results on critical flight set threshold parameter, 150 individuals evaluated over 10 generations

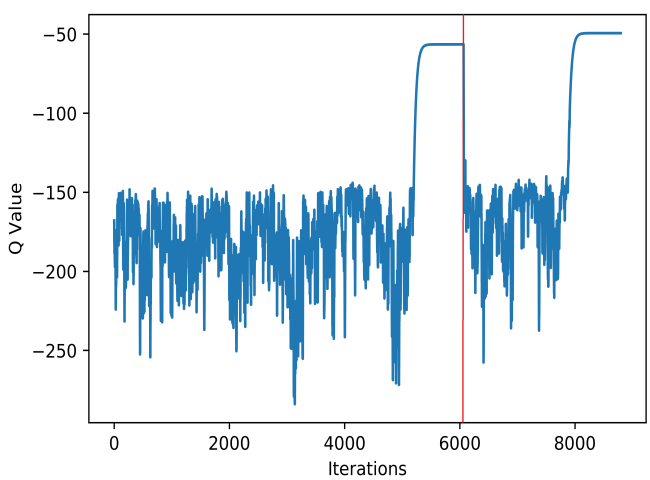

Fig. 8. Q value for a given aircraft over two sliding windows (separated with the red line).

C. Results

Figure 8 shows the value of the Q-function for an agent over iterations. Two exponential curves can be seen, these represent the exploitation part of the Boltzmann exploration strategy. This occurs at every sliding window when the temperature drops if, and only if, the aircraft has a so low reward that it is selected. Here, the red line separate the two different sliding windows. In this case, the Q-table was initialized to -150. This is why many values on this graph equal -150, it is the default value until this state and its neighbors have been more deeply explored.

Since the optimization is run only on worst aircraft first, the threshold drops all along the iterations. This means that more and more aircraft are selected in the critical flight set.

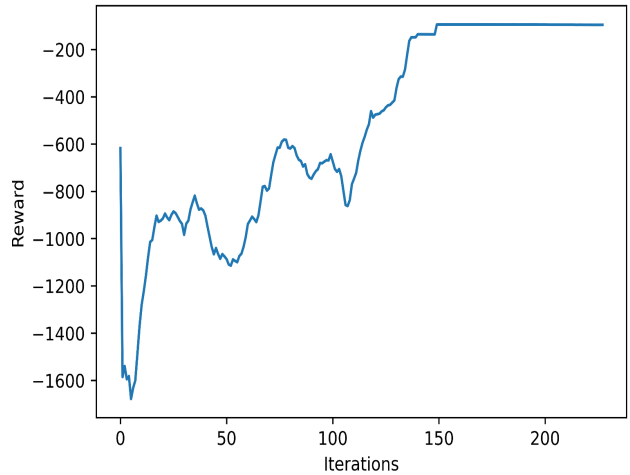

Fig. 9. Evolution of the total reward (sum of all aircraft's reward) of a given Sliding Window, this could be seen as the learning curve of the algorithm over a certain sliding window

Parameter	Average Value
Delay	75 sec
Number of conflict	0
Runway change	5.3
Worst reward	-8.7
Computational time	32 sec

TABLE V
AVERAGE RESULT (COMPUTED ON 10 DAY OPTIMIZATION) OF THE QL WITH THE PARAMETERS FOUND IN THE PREVIOUS SUBSECTION (FOR 687 AIRCRAFT PER DAY).

Depending on the temperature at which aircraft enter in the critical set, the Boltzmann strategy allows the exploration to go more or less further. At the end of iterations, the temperature is very low and the best states discovered in the exploration part are now more likely to be reached. This bring the total reward of the sliding window to strongly improve (see figure 9).

The figure 10 represents aircraft rewards at LORNI for the first and the final state of a scenario. Each bar represents an aircraft, and the length of bars are proportional to the rewards - be aware, scales are different between the two graphs. If the aircraft is in conflict at some point, the bar is in red, in blue otherwise. The QL algorithm is very efficient on hot spots, since every conflict has been solved, the delay is now shared between all aircraft, maximizing the reward for each one.

Table V gives the average results over 10 iterations of the QL on a heavily-loaded scenario. The process was run on a 3.1GHz core i5 CPU with Java code.

Figure 11 represents aircraft delays histogram. The time step used is the same as the time step used in the model: 20 seconds. The average delay of aircraft in this scenario is 73 sec.

Finally, figure 12 gives the histogram of the arc length

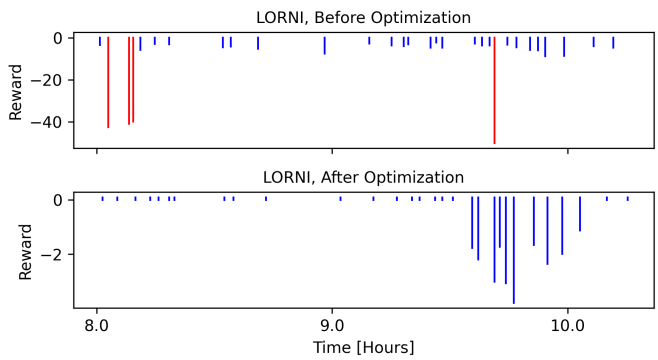

Fig. 10. Reward for aircraft in the flight set at LORNI; aircraft in red are in conflict, in blue otherwise. Results are shown before optimization on the top and after on the bottom.

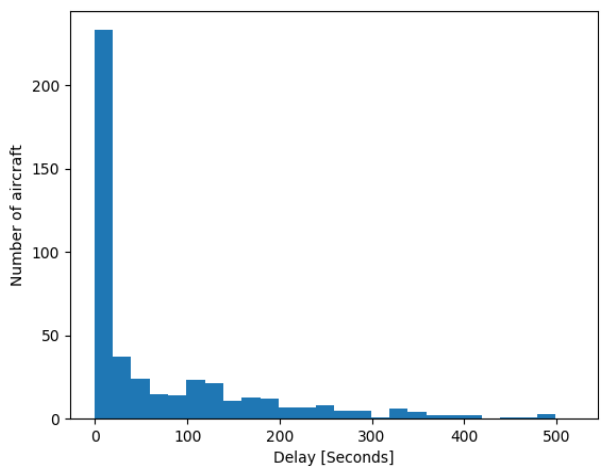

Fig. 11. Histogram representing the delay of aircraft in the scenario (687 aircraft)

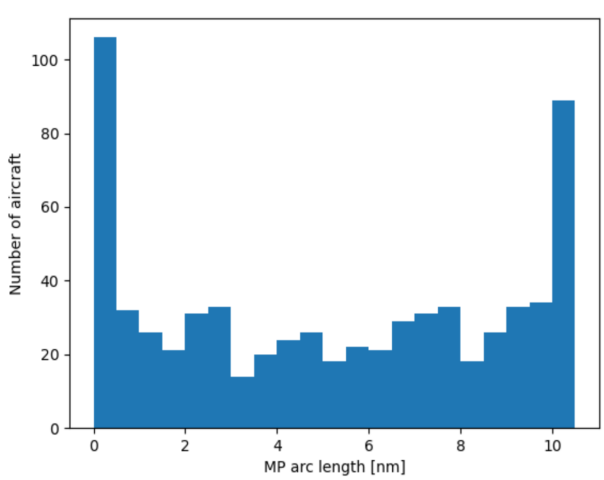

Fig. 12. Histogram representing the arc length that aircraft have flown in the PMS

that aircraft have flown. The length step used is 0.5 nm. The average arc length for an aircraft in this scenario is 4.9 nm.

V. Conclusion

Centralized AMAN has been extensively studied in the literature but few initiatives have been proposed for a decentralized version of such algorithm. This paper has presented an efficient decentralized sequencing and merging algorithm. A mathematical model, based on a network model, has been presented for which aircraft decision variables and constraints have been introduced. A distributed algorithm based on Q-learning has been proposed, for which parameters have been optimally tuned thanks to a genetic algorithm. This algorithm has been implemented by using a sliding window mechanism. It has been successfully applied to CDG airport for which demand has been artificially increased with a total of 687 aircraft landings. Conflict free solution for a full day of traffic has been computed in less than 30s which is really adapted for real time planning.

This approach is relevant for aircraft landing but is even more adapted for UTM, for which no centralized system is expected to manage the landing of many UAV at the same location.

In future work, such algorithm will be improved by taking into account uncertainties (wind, etc..) but also by using a more realistic aircraft model based, for instance, on BADA model.

References

[1] S. Abba-Rapaya, P. Notry, and D. Delahaye, "Coordinated Sequencing of Traffic on Multiple En-route Constraint Points," in *6th ENRI International Workshop on ATM/CNS, ENRI*, vol. 731. Springer, Oct. 2019.

[2] M. Liang, D. Delahaye, M. Sbihi, and J. Ma, "Multi-layer Point Merge System for Dynamically Controlling Arrivals on Parallel Runways," Sep. 2016.

[3] EUROCONTROL, "Point Merge implementation A quick guide," May 2021.

[4] A. A. Deshmukh, Y. Huo, D. Delahaye, P. Notry, and M. Sbihi, "Algorithmic Efficiency Comparison of Centralised and Distributed Arrival Management (AMAN) Problem In Terminal Airspace," in *10th Sesar Innovations Days*, Dec. 2020.

[5] J. Ma, D. Delahaye, M. Sbihi, and M. Mongeau, "Merging Flows in Terminal Moneuvering Area using Time Decomposition Approach," Jun. 2016.

[6] M. Xiangwei, Z. Ping, and L. Chunjin, "Sliding window algorithm for aircraft landing problem," in *2011 Chinese Control and Decision Conference (CCDC)*, 2011, pp. 874–879.

[7] J. Ma, D. Delahaye, M. Sbihi, and M. Mongeau, "Integrated Optimization of Terminal Manoeuvring Area and Airport," in *6th SESAR Innovation Days*, Nov. 2016.

[8] C. Watkins, "Learning from Delayed Rewards," Ph.D. thesis, King's college, London, May 1989.

[9] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*, second edition ed., ser. Adaptive computation and machine learning. Cambridge, Mass: MIT Press, 2014, 2015.

[10] A. D. Tijsma, M. M. Drugan, and M. A. Wiering, "Comparing exploration strategies for Q-learning in random stochastic mazes," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*. Athens, Greece: IEEE, Dec. 2016, pp. 1–8.

[11] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, ser. Adaptive computation and machine learning. Springer-Verlag, 1996.