Thirteenth USA/Europe Air Traffic Management Research and Development Seminar (ATM2019)

Optimizing Successive Airspace Configurations with a Sequential A^* Algorithm

David Gianazza
ENAC, Université de Toulouse, France

***Abstract*—In this paper, we introduce exact tree search algorithms which explore all the possible sequences of airspace partitions, taking into account some constraints on the transitions between two successive airspace configurations. The transitions should be simple enough to allow air traffic controllers to maintain their situation awareness during the airspace configuration changes. For the same reason, once a sector is opened it should remain so for a minimum duration.**

The proposed method is a sequential A^* algorithm with a rolling horizon. It finds a sequence of airspace configurations minimizing a cost related to the workload and the usage of manpower resources, while satisfying the transition constraints. This approach shows good results on 9 months of data from the french ATCC Aix (East), when compared with two baseline methods, one with a greedy approach and the other with no transition constraints.

***Keywords*—Air traffic management; workload; airspace configuration; hotspot prediction; tree search; branch & bound; $A*$ algorithm; A_star algorithm**

INTRODUCTION

A dynamic and flexible use of the airspace resources is a key element for improving the efficiency of Air Traffic Management (ATM), as expressed in many operational concepts such as [1]–[3]. Currently the airspace is divided in managerial units – Air Traffic Control Centers (ATCCs). The airspace of each ATCC is divided into *elementary airspace sectors*. These basic airspace modules can be combined together so as to form *control sectors* operated each by a small team of 2-3 controllers. Depending on the ATCC, the Air Traffic Controllers (ATCos) are qualified to operate any ATC sector of their center, or only those belonging to a specific *qualification zone*.

The partitionning of an ATCC's airspace – or a qualification zone – into ATC sectors is called an *airspace configuration*. It changes across the day, depending on the workload experienced by the ATCos. Sectors can be split[1] when the workload increases, or merged (or collapsed) when the workload decreases. More complex recombinations may sometimes be decided by the control room manager, when necessary.

Although it could be made even more flexible by multiplying the number of airspace modules and combining them in different ways [4], or by optimizing flexible sector boundaries so as to minimize the controller taskload [5], the current system is already very flexible and adaptive when facing traffic variations. It does lack predictability, though.

Several issues arise when trying to predict future airspace congestions (or hotspots). One of them is that the prediction must rely on a realistic workload model. Another one is to identify which control sectors might be open in the future, in order to detect which of them might become overloaded. The first issue has been adressed in a number of works, using linear models [6], neural networks [7], [8] or other machine learning techniques [9] to predict the controller workload from a set of relevant ATC (Air Traffic Control) complexity metrics. This is not the focus of this paper, however. The second issue – knowing which control sectors might be open at a given time in the future – is the problem being addressed here.

Our objective is to compute an optimal sequence of airspace configurations in a given time interval in the future, taking various constraints into account. The successive airspace configurations should minimize the overloads and also the number of working positions that are necessary to cope with the traffic. In addition, the workload should be as close as possible to a nominal value in each sector. The constraints include that the transitions between successive configurations should be simple enough to allow controllers to maintain their situation awareness. For the same reason, once an ATC sector is opened, it should remain so for a minimum duration.

Given an input traffic prediction and assuming we have a realistic workload model, one could use such an optimal sequence of predicted airspace configurations to anticipate future overloads that could not be solved simply by reconfiguring the sectors. This "hotspot" detection is crucial to the elaboration of traffic management measures such as rerouting or delaying flights well in advance.

The paper is organized as follows. Section I provides a very brief overview of previous research on airspace sectorization and configuration. The problem being addressed in this paper is described in section II. The *branch & bound* and A^* algorithms are presented in section III. Section IV shows how we apply these tree search algorithms to optimize sequences of airspace configurations. The datasets and experiment setup are described in section V, and the results are presented in section VI. Section VII concludes the paper and gives some perspectives of further research.

[1] Splitting a control sector requires that it is made of at least two elementary sectors.

I. Background

Many research has been done on airspace sectorisation and airspace configuration (see surveys in [10], [11]). Few works explicitely consider the transitions between configurations and try to optimize sequences of configurations, exploring all reachable configurations. In [12], dynamic programming methods are compared with greedy heuristics, considering a transition cost and exploring a limited subset of predefined configurations from a small ATCC with few sectors. Other works [13] use a distance between configurations to smooth sector configuration plans built using a combination of deterministic search and simulated annealing, considering a relatively small problem instance (Reims ATCC, France). In [14], a softmax regression provides a rough estimate of the number of controller working positions necessary to handle a given input traffic in Bordeaux ATCC (a bigger problem instance). A finer prediction is then obtained using a tree search method exploring a catalogue of operational configurations, considering higher-level clusters of control sectors that are used in operations (East, North, or South clusters).

This hierarchical partitionning – where one first decides to split the airspace into clusters, and then balances the workload by combining sectors within each cluster – highly reduces the number of combinations to consider.

Another aspect on which Flow Management Position (FMP) operators largely rely on is that typical traffic patterns are handled with typical airspace configurations. In european ATCCs, the FMP operators only consider a small number of these typical configurations, manually stored into databases, to build sector opening schemes.

Over the past decades, ATCCs have been multiplying the number of elementary airspace sectors and reducing their size, thus increasing dramatically the number of possible airspace configurations. In addition, there is a natural trend in ATM towards more flexibility in 4D-trajectory handling and airspace sectorisation and configuration, which might lead to the emergence of less predictable traffic patterns. Consequently, it seems interesting to explore all the possible partitions of airspace that can be built from controllable sectors, without limiting ourselves to a short list of typical configurations.

In our previous works [15]–[18] we introduced a *branch & bound* algorithm that explores all the possible partitions of the airspace into valid sector configurations. However, this approach did not take into account the transition constraints between successive airspace configurations. In the following, we propose another approach where an A^* algorithm is used to optimize sequences of airspace configurations satisfying transition constraints. The next section states the problem being addressed with more details.

II. Problem description

Ultimately, our aim is to find optimal sequences of airspace configurations satisfying a number of constraints including a maximum number of available working positions at any time of the day, a minimum duration length for any sector opening, and compliance to various constraints including transition rules which allow the controllers to maintain their situation awareness while reconfiguring the airspace.

A. Airspace Configurations

The airspace is divided into a number of elementary airspace sectors (modules). These airspace sectors are assigned to controllers' working positions (CWPs). The radar and planning controllers working on a CWP operate an *Air Traffic Control sector* (ATC sector) made of one or several elementary airspace sectors.

At any moment, all airspace sectors should be controlled (i.e. assigned to a CWP) and no airspace sector should be assigned to more than one CWP[2]. Consequently the assignement of airspace sectors to controllers' working positions form a partition of the airspace into ATC sectors, as illustrated on Figure 1 on a toy example involving 5 airspace sectors in a fictitious airspace.

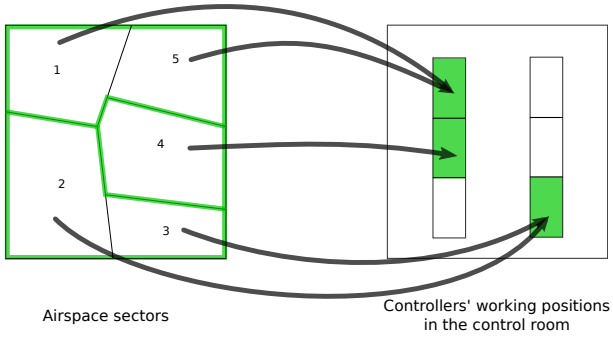

Figure 1: Airspace partitionning into ATC sectors

Not all airspace partitions are valid airspace configurations, though. Considering Figure 1, merging the airspace sectors 1 and 3 for example would not form a valid ATC sector, as these airspace sectors are not geographically connex. In practice, a list of valid ATC sectors is available from the ATC center databases.

Using this data, one can enumerate all the possible airspace configurations that can be obtained using controllable sectors – *i.e.* valid groups of sectors or elementary sectors that appear in the center's database. This enumeration is illustrated on Figure 2. In order to build valid airspace configurations, airspace sectors are considered sequentially, starting with sector 1 at the root of the tree. The ATC sectors compatible with each node are traced. When assigning airspace sector 1 to a CWP at the root of the tree, the compatible ATC sectors are $s = \{1\}$, $d = \{1, 5\}$ and $e = \{1, 2, 3, 4, 5\}$. Considering airspace sector 2, one can either assign it to the same CWP as 1 (left branch) , or assign it to a separate CWP (right branch). In the right branch, we see that group $e = \{1, 2, 3, 4, 5\}$ is no longer compatible with airspace sector 1 when 2 is assigned to a separate CWP. In the left branch, the only ATC sector compatible with 1 and 2 assigned to the same CWP is $e = \{1, 2, 3, 4, 5\}$. The process goes on with the other airspace sectors. The nodes having an empty list of compatible ATC

[2]Except maybe very briefly when transferring a sector from one CWP to another

sectors are no longer developped. The leaves of the tree give us all the valid airspace configurations.

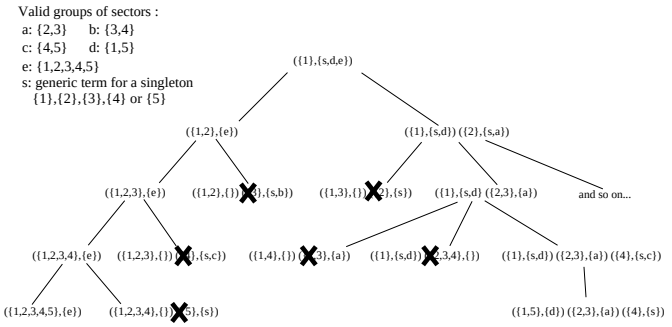

Figure 2: Building valid airspace configurations

Table I shows on the rightmost column the number of valid airspace configurations for each French ATC center (or qualification zone), in 2018. The computation time just to count these configurations ranges from a few milliseconds for the smallest centers to more than 2 hours and 20 minutes for Brest ATCC (using an Intel(R) Xeon(R) CPU E31270 3.40GHz octo-core desktop).

	Airspace sectors	Valid groups	Airspace config.
Aix (West)	20	45	47,377
Aix (East)	26	79	12,161,652
Bordeaux	36	69	99,522,406
Brest	32	92	233,281,435
Paris (West)	12	20	583
Paris (East)	15	23	833
Reims	22	51	224,691

TABLE I. Number of valid airspace configurations (rightmost column) in the French ATC centers, in January 2018

B. Sequences of airspace configurations

Let us denote C_{t_0} an initial configuration at time t_0, and let us consider a time interval $[t_0, t_0 + K\delta t]$, where δ_t is a chosen time step (typically one minute). Our objective is to predict an optimal sequence of airspace configurations $C_{t_0}, C_{t_0+\delta t}, \ldots, C_{t_0+K\delta t}$, minimizing the cost of the successive configurations. This cost should depend on the workload of each ATC sector, as well as the number of working positions. Various constraints can be taken into account, such as the maximum number of working positions that can be opened (depending on how many controllers are available), the minimum length of the time interval during which a sector should remain open, and the transition rules from one configuration to the next.

Several modeling choices can be envisionned when optimizing the sequences of successive airspace configurations. One can consider the successive configurations as independent one from the other, in the sense that any configuration can be the successor of a given configuration and the transition from the current configuration to the next is costless. In that case, the optimal sequence can be obtained by finding the optimal configuration at each time step, separately.

A more realistic model is to consider that a configuration in the sequence depends on its predecessor, *i.e.* not all transitions are allowed, or the transitions have a cost. In operations, transitions from one airspace configuration to another must allow controllers to maintain their situation awareness. Transitions such as the ones shown on Figure 3 are too complex and should be avoided, or at least considered only when there are very few flights in the sectors involved.

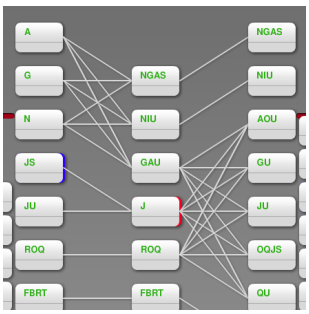

Figure 3: Complex airspace reconfigurations

In this article, we propose to introduce simple transition rules restricting the set of configurations that can be reached from the current configuration. When doing so, one cannot find the optimal sequence of configurations by optimizing each configuration independently anymore. We must consider all the possible sequences of configurations that can be reached, starting from the initial one, as illustrated on Figure 4.

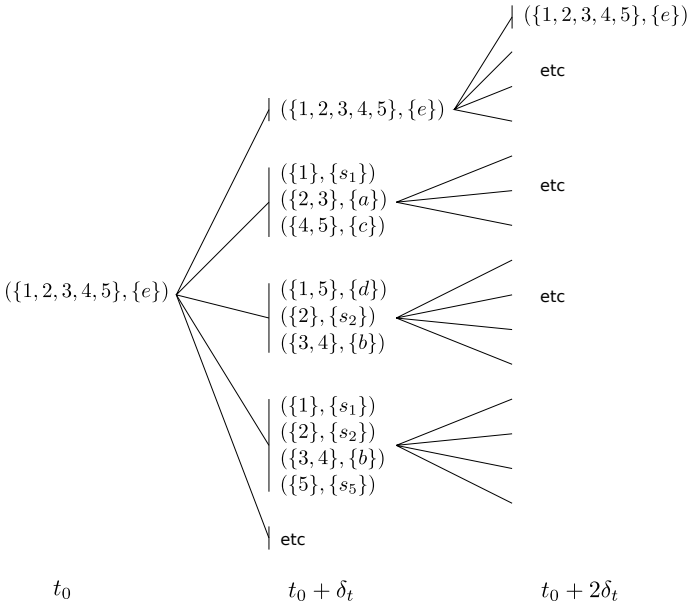

Figure 4: Possible sequences of successives airspace configurations

Denoting b the average branching factor of the tree of all possible sequences, the approximate number of sequences to explore in order to find the optimal sequence is b^K, when considering sequences of K steps. Taking for example a time horizon of 2 hours (*i.e.* 120 minutes) and a branching factor of 20, there are 20^{120} possible sequences of configurations, which is huge. Considering the number of possible configurations in table I, the actual branching factor might be much

higher than 20, depending on whether the chosen transition rules are restrictive of not.

C. Constraints

1) Transition rules: Any two successive airspace configurations C and C_+ in a sequence S should satisfy the following constraint :

$$C_+ \in \mathcal{A}_{\mathcal{T}}(C) \tag{1}$$

where $\mathcal{A}_{\mathcal{T}}(C)$ is the set of configurations that can be reached from C complying with a chosen set of transition rules $\mathcal{T}$

In this paper we shall consider two sets of transition rules, comprising only three kinds of actions: split, merge, or transfer sectors. A transition shall consist in one or several such actions. The two sets of transition rules used in this paper are described below:

$$\mathcal{T}_1 : \begin{array}{ll} \text{Split} & ab \rightarrow a, b \\ \text{Merge} & a, b \rightarrow ab \\ \text{Transfer} & ab, c \rightarrow a, bc \end{array} \tag{2}$$

$$\text{One action per transition} \tag{3}$$

$$\mathcal{T}_2 : \begin{array}{ll} \text{Split} & \text{one sector into up to } n_{split} \text{ sectors} \\ \text{Merge} & \text{up to } n_{merge} \text{ sectors into one} \\ \text{Transfer} & ab, c \rightarrow a, bc \end{array}$$

$$n_{act} \text{ actions per transition at most}$$

An action can be seen as an operation (split, merge, or transfer) taking a list of arguments (ATC sectors), and returning a list of new ATC sectors that replace the former ones in the new configuration. A 'split' operation applies to one ATC sector, resulting in up to n_{split} new ATC sectors. A 'merge' operation applies up to n_{merge} sectors, resulting in one ATC sector, and a 'transfer' operation applies to two ATC sectors, resulting in the same number of ATC sectors.

Considering $\mathcal{T}_2$ and taking for example $n_{act} = 2$ actions at most per transition, there are 9 possibilities in the choice of operations: [Split; Split]; [Split; Merge]; [Split; Transfer]; [Split]; [Merge; Merge]; [Merge; Transfer]; [Merge]; [Transfer; Transfer]; [Transfer]. For a partition containing 6 ATC sectors, there are 722 ways to choose up to $n_{act} = 2$ operations and to choose the arguments of these operations – *i.e* the ATC sectors being split, merged (with at most 3 sectors being merged, here), or implied in a transfer. With $n_{act} = 3$ this number goes up to 2062.

The actual number of airspace configurations that can be built applying these possible actions is even higher, as there can be many ways to split a given ATC sector into several smaller sectors, or to transfer some airspace sectors from one ATC sector to another.

2) Duration of sector openings: The time intervals during which any ATC sector is opened during a sequence $S = \{C_1, \ldots, C_n\}$ should have a minimum width of M minutes.

This constraint avoids multiple split, merge, or transfer operations involving a same sector in a short period of time, and allows controllers to maintain a better situation awareness.

3) Other constraints: Some other constraints can be defined, such as a maximum number of working positions that may vary across the day depending on the number of controllers on the duty roster. This constraint is easy to implement and has been taken into account in our previous works [15], [18]. However, we will not consider this constraint in the current paper, and focus on the difficulty to apply tree search methods to the problem of computing optimal sequences of airspace configurations.

Another constraint, already mentioned in section II-A, is the fact that one can use only some predefined groups of airspace sectors to form an ATC sector. This constraint is taken into account in the rest of the paper, where we use the list of valid ATC sectors provided by each ATC center to build valid airspace configurations.

Note that we do not restrict ourselves to a list of typical configurations, as currently done in operations. We consider all the possible configurations that can be obtained by partitionning the set of airspace sectors into valid ATC sectors.

D. Problem statement

To summarize our description of the problem, let us formalize it as a minimization problem. Denoting S a sequence of airspace configurations, $\mathcal{S}$ the set of all possible sequences, and given a cost function *cost* and a set of chosen transition rules $\mathcal{T}$, the problem can be formalized as follows :

$$\min_{S \in \mathcal{S}} cost(S) \text{ subject to} \tag{4}$$

$$C_+ \in \mathcal{A}_{\mathcal{T}}(C) \text{ for any successive configurations } C, C_+ \tag{5}$$

$$min\_open(atc\_sect) \geq M \text{ , } \forall atc\_sect \in \mathcal{O}(S) \tag{6}$$

$$\tag{7}$$

where $\mathcal{A}_{\mathcal{T}}(C)$ is the set of configurations that can be reached from C complying with the chosen transition rules $\mathcal{T}$, and $\mathcal{O}(S)$ is the set of ATC sectors opened during the sequence S.

The last constraint expresses the fact that an ATC sector, once opened, should remain open for at least M minutes. Note that this constraint does not apply to the duration of the configurations, which can be changed at any time, but to the ATC sector openings.

III. Tree search methods

A. Branch & bound

The branch & bound algorithm performs a *depth-first* search in a tree or a graph until it reaches a solution. It then backtracks in the tree to check for other solutions. At each node, a cost estimate is computed, that is a lower bound of the costs of all the solutions that can be reached in the subtree of the considered node. If this lower bound is worse than the best cost found so far, there is no need to continue exploring the subtree and the branch is cut. Otherwise the depth-first search continues, updating the best solution when necessary. The search stops when there are no more nodes to explore.

This algorithm can be implemented with a priority queue where the deepest nodes get the highest priority (*i.e.* a "last in first out" stack). The pseudocode is given in Algorithm 1.

Algorithm 1 *branch and bound* algorithm

Require: u_0 ▷ Initial state
$(best, cost_{best} \leftarrow (None, +\infty)$
$p_0 \leftarrow$ PRIORITY(u_0)
$Q \leftarrow$ INSERT($u_0, p_0, \emptyset$) ▷ Init priority queue (Last In First Out)
$D \leftarrow \emptyset$ ▷ Empty set of expanded nodes
loop
  if $Q = \emptyset$ and $best = None$ **then**
    return FAILURE ▷ No solution
  else if $Q = \emptyset$ and $best \neq None$ **then**
    return $(best, cost_{best})$ ▷ Return best solution
  else
    $(u, Q) \leftarrow$ EXTRACT(Q) ▷ Extract node of highest priority (*i.e.* the most recently inserted node)
    if ISSOLUTION(u) **then** ▷ u is a terminal state
      $cost_u \leftarrow$ COST(u) ▷ Store the cost of u
      if $cost_u < cost_{best}$ **then**
        $(best, cost_{best}) \leftarrow (u, cost_u)$ ▷ Update best solution
      end if
    else ▷ u is not a terminal state:
      $D \leftarrow D \cup \{u\}$ ▷ Add u to expanded nodes
      $S \leftarrow$EXPAND(u) ▷ Expand u (*i.e.* compute its successors)
      for all $v \in S$, $v \notin D$ **do**
        $eval_v \leftarrow$ COSTESTIMATE(v) ▷ Lower bound of reachable solutions
        if $eval_v < cost_{best}$ **then**
          $p_v \leftarrow$ PRIORITY(v)
          $Q \leftarrow$ INSERT(v, p_v, Q)
        end if ▷ Cut if lower bound above $cost_{best}$
      end for
    end if
  end if
end loop

B. The A^* algorithm

The A^* algorithm performs a *best-first* search in a tree or a graph. It extracts the node u of highest priority from the frontier F, that is the set of nodes that have been generated but not expanded yet, and expands this node by producing a list of successor nodes. If a successor node v has never been encountered before, or if the cost of going from the initial state u_0 to v is lower when passing through u than through any previous parent node of v, the successor v is inserted (or reinserted) in the frontier F with a priority $f(v)$. This process is repeated until a terminal state is reached.

The cost of node v is simply the cost of the parent node u plus the cost $k(u, v)$ of going from u to v: $cost_v = cost_u + k(u, v)$. The priority $f(v)$ is the estimated cost for going from the initial state to the final state while passing through v. The priority $f(v)$ is the sum of $cost_v$ and a heuristic $h(v)$ estimating the cost between v and the destination: $f(v) = cost_v + h(v)$.

A typical example is trying to find the shortest route when driving from one city to another. For this application, one can take $k(u, v)$ equal to the driving distance between u and v, two connected nodes in the network, and the heuristic function $h(v)$ equal to the distance as the crow flies from v to the destination.

The notations are summarized below, and the algorithm is fully described in Algorithm 2.

- u_0 : initial state
- T : set of terminal nodes (solutions)
- D : set of expanded nodes
- F : frontier, *i.e.* the set of nodes that have been generated but not expanded yet
- $h(u)$: heuristic function estimating the cost of the path from the current node u to a final state belonging to T
- $cost(u)$: the best cost stored for the path from the initial state u_0 to the current node u
- $f(u)$: estimate of the total cost for the path from u_0 to a terminal state, via the node u. $f(u) = cost(u) + h(u)$ aggregates the cost of the path already traveled from u_0 to u and the cost estimated for the path remaining to be travelled to a destination in T
- $k(u, v)$: cost of the transition between two successive states u and v
- $parent(v)$: the parent of v for which the path from u_0 to v is of lowest cost

In addition, the exploration of the state space requires to specify the following functions:

- EXPAND : function applying a set of production rules $\{p_1, p_2, ..., p_k\}$ to a state u, returning a set of successors $S = \{v_1, v_2, ..., v_k\}$. For example, when finding a path in a road network, it will simply return the cities (nodes in the network) connected to the current city u.
- BESTINFRONTIER : function returning the best node u in the frontier F, that is the one for which $f(u) = cost(u) + h(u)$ is the lowest.
- ADDORREPLACE : adds a node v with priority $f(v)$ to the frontier F, or replaces it if it was already in it.

Algorithm 2 The A^* algorithm.

Require: Initial state u_0
$F \leftarrow \{u_0\}$
$cost(u_0) \leftarrow 0$
while $F \neq \emptyset$ **do**
  $u \leftarrow$ BESTINFRONTIER(F) ▷ Find u with best priority $f(u)$ in frontier F
  $F \leftarrow F \setminus \{u\}$ ▷ Remove u from the frontier
  if $u \in T$ **then** ▷ u is a terminal state, return path
    return PATH(u_0, u)= $u :: parent(u) :: \ldots :: u_0$
  else
    $D \leftarrow D \cup \{u\}$ ▷ Add u in set of expanded nodes
    $S \leftarrow$ EXPAND(u) ▷ Expand u (*i.e.* compute its successors)
    for all $v \in S$ **do** ▷ For all successors
      if $v \notin D \cup F$ **or** $cost(v) > cost(u) + k(u, v)$ **then**
        $cost(v) \leftarrow cost(u) + k(u, v)$ ▷ Store cost of path $u_0, \ldots, u, v$
        $f(v) \leftarrow cost(v) + h(v)$ ▷ Compute $f(v)$ the priority of v
        $parent(v) \leftarrow u$ ▷ Memorize u as parent of v
        $F \leftarrow$ ADDORREPLACE($v, f(v), F$) ▷ Insert v in frontier F
      end if
    end for
  end if
end while
return FAILURE ▷ No solution when the frontier is empty

IV. Application to our problem

A. Air traffic controller workload

The cost we will try to minimize when computing optimal sequences of airspace configurations is related to the workload experienced by the air traffic controllers operating the ATC sectors of the successive configurations. There are many ways to model the air traffic controller workload. The notion of incoming flow – *i.e.* the number of incoming flights entering a sector in given time interval – has been widely used in Europe, together with the notion of capacity, defined as a threshold value for the incoming flow, above which the sector is considered as overloaded. These notions of flow and capacity are well suited to flow management and traffic regulation, but they are a poor estimate of the actual workload experienced by controllers.

A more realistic workload estimation is to count the number of aircraft within the sector boundaries at a given time, or within a given time period. These occupancy measures are usually employed together with peak and sustain monitoring alert parameters to determine potential overloads. Although better than the simple incoming flows and capacities, the number of aircraft and occupancy counts do not take into account the air traffic control complexity.

A lot of research has been done on understanding and expliciting the relationship between ATC complexity and controller workload [6], [19]–[23]. In previous works [8], [16], [24], [25], we proposed to predict the workload from a set of 6 indicators (sector volume, aircraft count, incoming flows with 15 or 60 mn time horizon, average absolute vertical speed, and trajectory crossing count), using a neural network, or other machine learning techniques [9], [26]. In these works, the model was trained on historical data made of measures of ATC complexity and records of past sector operations.

The focus of the current paper is not on the workload model, but on finding optimal sequences using tree search methods. In order to make the results easier to reproduce, a very simple workload model, based on the aircraft count, is chosen in this study.

For each control sector, we define a lower bound lb and an upper bound ub for the aircraft count. Sectors with an aircraft count below lb will be considered as underloaded, and those with an aircraft count above ub will be considered as overloaded.

In addition to the upper bound ub and lower bound lb, we also define for each ATC sector s a nominal workload value nw, with $\mathrm{lb} < \mathrm{nw} < \mathrm{ub}$.

With these three parameters, we quantify the workload in a given ATC sector s using the aircraft count $n_a(s)$ by measuring the excessive overload ol_s, the normal workload above or below nominal (nl_s^+ and nl_s^- respectively), and the excessive underload ul_s. These quantities are expressed as follows, where $\mathbb{1}_{[x;y]}(z)$ is equal to 1 when $x \leq z \leq y$, and 0 otherwise:

$$\mathrm{ol}_s = max(n_a(s) - \mathrm{ub}_s, 0) \tag{8}$$

$$\mathrm{nl}_s^+ = \mathbb{1}_{[\mathrm{nw}_s;\mathrm{ub}_s]}(n_a(s)) \times |n_a(s) - \mathrm{nw}_s| \tag{9}$$

$$\mathrm{nl}_s^- = \mathbb{1}_{[\mathrm{lb}_s;\mathrm{nw}_s]}(n_a(s)) \times |n_a(s) - \mathrm{nw}_s| \tag{10}$$

$$\mathrm{ul}_s = max(\mathrm{lb}_s - n_a(s), 0) \tag{11}$$

Given a set of values for $n_a(s)$, ub_s, nw_s and lb_s, at most one of the above cost measures ol_s, nl_s^+, nl_s^- and ul_s will have a non-zero value.

B. Cost of an airspace configuration

An airspace configuration C is made of several ATC sectors. The cost of a configuration C depends on the workload in each of its sectors and also of the number of working positions required to operate the sectors.

In the following, we have chosen to quantify the cost of an airspace configuration as a multidimensional vector whose components are the following costs, enumerated below in their order of importance:

$$\mathrm{ol}(C) = \sum_{s \in C} \mathrm{ol}_s \tag{12}$$

$$n_{\mathrm{cwp}}(C) = cardinal(C) \tag{13}$$

$$\mathrm{ul}(C) = \sum_{s \in C} \mathrm{ul}_s \tag{14}$$

$$\mathrm{nl}(C) = \sum_{s \in C} (\mathrm{nl}_s^+ + \mathrm{nl}_s^-) \tag{15}$$

When comparing two configurations, these quantities are compared in the order of priority of their enumeration.

C. Cost of a sequence of configurations

The cost of a sequence is an aggregation of the costs of the successive configurations and the costs of the transitions. For the current study, we have chosen a transition cost of zero, and the cost of the sequence is simply the sum of the costs of the configurations.

D. Baseline 1: optimizing without transitions

The first baseline method simply consists in optimizing the successive airspace configurations independently, by applying Algorithm 1 (*branch & bound*) at each time step. When removing the constraints on the transitions, we can compute an optimal airspace partition for each time step of the sequence, searching over all the possible partitions. A similar approach was used in our previous works [15]–[18], [27], with the difference that in this paper the workload model is a simple aircraft count.

Figure 5 shows how the *branch & bound* algorithm is used to compute an optimal airspace partition at a given time step. Taking the same example as in section II-A, the algorithm explores the same tree as in Figure 1, where the nodes are incomplete partitions, and the leaves are full partitions of the airspace (*i.e.* airspace configurations).

The function Cost of Algorithm 1 that evaluates the leaves (e.g. at step 6 on Figure 5) is the cost described in section IV-B. It is a vector of components $\mathrm{ol}(C)$, $n_{\mathrm{cwp}}(C)$,

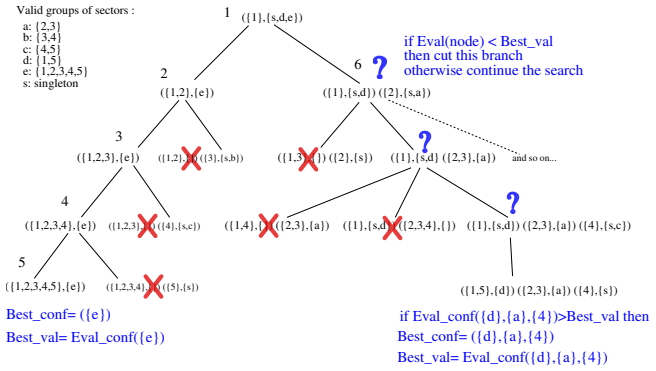

Figure 5: Computing an optimal partition with a *branch & bound* algorithm.

$\mathrm{ul}(C)$, $\mathrm{nl}(C)$ (assuming C is the airspace configuration at the evaluated leaf).

The function COSTESTIMATE in Algorithm 1 uses a similar cost structure when evaluating nodes. In order to obtain a lower bound of the costs of all leaves that can be reached from the evaluated node, we simply take – for each group of airspace sectors of the node – the minimum values of the workload costs over the compatible ATC sectors. A lower bound of the number of working position is given by the number of groups in the node. The order of priority of the different costs is taken into account when computing the minimum values over several possible ATC sectors.

Looking at the node visited at step 6 on Figure 5, we have two groups of sectors $(\{1\}, \{s_1, d\})$, and $(\{2\}, \{s_2, a\})$ so the lower bound for n_{cwp} is 2. For the first group, we will obtain a lower bound of the overload by taking the minimum over the overload values of the compatible ATC sectors $s_1 = \{1\}$ and $d = \{1, 5\}$. The same goes for the other workload quantities (normal workload above or below nominal, and underload). Doing the same with the other group $(\{2\}, \{s_2, a\})$, and aggregating these quantities we can compute a lower bound for the cost of the configurations that can be reached from the node.

E. Baseline 2: the greedy strategy

Taking into account the transition constraints, a greedy strategy is to compute the sequence of successive configurations by taking the best configuration that can be reached from the current configuration, at each time step.

In this approach, we explore the tree of all possible sequences of successive configurations (see Figure 4). However, this tree is explored without backtracking, following the best branch at each node until the sequence is complete.

This greedy method is a realistic baseline method with which we can compare our proposed method presented in the next section.

F. Sequential A^* with a rolling horizon

Ideally, we would like to find the best sequence among all the possible sequences. This is a difficult problem, though, considering the branching factor and the depth of the tree (see comments at the end of section II-B).

An intermediate strategy between the greedy strategy and the full search of the tree is to search the tree up to a limited depth, pick up the best next configuration and add it to the sequence, and reiterate the process from this configuration.

The Algorithm 2 (A^*) is used at each iteration to explore the tree up to a limited depth. This process is illustrated on Figure 6.

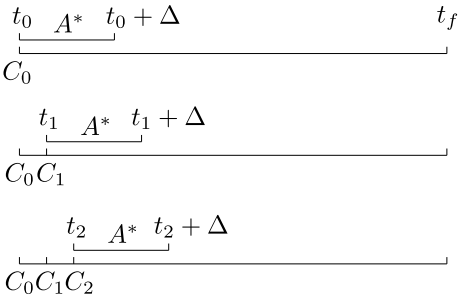

Figure 6: Sequential A^* with a rolling horizon.

For the A^* algorithm, the cost of the initial configuration C_0 is the same, whatever the explored sequence starting at C_0, and we can take 0 without changing the result.

The cost of any other configuration C is expressed as follows, where ol, n_{cwp}, ul, and nl are defined in section IV-B :

$$astar\_cost(C) = \alpha\,\mathrm{ol}^a + \beta\,n_{\text{cwp}} + \gamma\,\mathrm{ul}^b + \delta\,\mathrm{nl}^b \quad (16)$$

In the following, the parameter settings are $a = 1.3$, $b = 0.7$, $\alpha = 1e6$, $\beta = 1e2$, $\gamma = 1e-2$, and $\delta = 1e-6$.

If u is the current configuration and v one of its successors, the cost $k(u, v)$ for going from u to v (see algorithm 2) is simply $k(u, v) = astar\_cost(v)$. The heuristic function $h(v)$ is chosen as the sum of the costs of the best configurations with no transition constraints in the time interval between the current time and the rolling horizon. These costs are lower bounds of the costs obtained when enforcing the transition rules. They are computed using the baseline 1 method (*No-Trans*).

V. DATA AND EXPERIMENT SETUP

A. Datasets

In the current study, we compare our methods using real data from the Aix ATC Center, using the East qualification zone. We have used 279 days of recorded radar tracks from 2017 (January to December) to assess the workload thresholds (upper bounds) of each ATC sector.

The methods are then tried and compared on 254 days from 2018 (January to September). The radar tracks have been sampled in order to get 1 point per minute.

B. Workload thresholds

A rough estimate of the workload's upper bound – in terms of aircraft counts – can be obtained by boxplotting the number of aircraft in each sector for the 2017 traffic. Considering only sectors with at least 1200 data points, we have computed for each sector an upper bound for the number of aircraft using the following formula, where q is a quantile:

$$ub = q(75\%) + 1.5 \times (q(75\%) - (q(25\%)) \quad (17)$$

This formula is the one used in the R boxplot function to compute the upper limit of the whiskers.

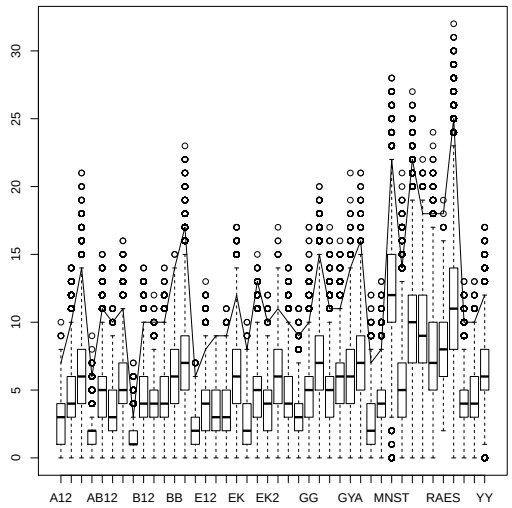

Figure 7: Boxplots of the aircraft counts, from Jan 2018 to Sep 2018, and workload upper bounds computed on 2017 data (broken line). The x-axis shows the sectors and the y-axis the number of aircraft.

In order to assess the relevance of these upper bounds estimated on 2017 traffic, we have compared them to the ones obtained using 2018 data. Figure 7 shows the boxplots of the number of aircraft in 2018, and the broken line representing the upper bounds obtained using equation 17 on the 2017 data. We see that the whiskers of the 2018 boxplots are globally close to the upper bound computed with data from 2017.

For sectors where we did not have enough data points to estimate an upper bound ub, we chose a default value of 14. For the lower bound lb, the chosen value is 6 for all sectors. The nominal workload nw is chosen as follows: $\mathrm{nw} = max(\mathrm{ub} - 3, \mathrm{lb})$.

VI. Results

In the following, we compare the two baseline methods – one without transition rules (*NoTrans*), and the other using a greedy strategy (*Greedy*) – with a *Sequential* A^* with several values for the time horizon Δ. The 254 days of results for the Aix East qualification zone in 2018 are presented as boxplots of various quantities (costs, branching factor, visited nodes, and processing time).

A. Results with transition rules $\mathcal{T}_1$

We first consider a set very simple transition rules $\mathcal{T}_1$ where one can split a sector into two sectors, collapse two sectors into one, or transfer one or several airspace sectors from one controller working position to another. Only one such operation per minute is allowed. No minimum duration for the sector openings is imposed here.

The left side of Figure 8 shows boxplots of the overload cost (our primary objective) for each of the tested methods. The *Greedy* baseline method on the left presents the highest cost. This was to be expected because it chooses the best

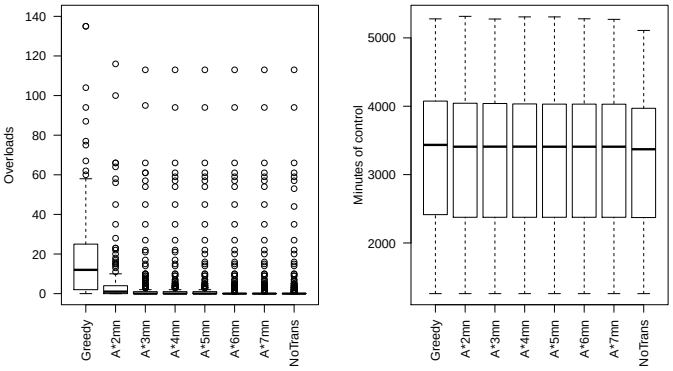

Figure 8: Overloads (left) and ATC sector openings (in minutes, right) for each method, with transition rules $\mathcal{T}_1$.

attainable configuration at each step without ever reconsidering this choice. The lowest overload cost is obtained with the *NoTrans* baseline method (on the right), i.e. when relaxing the transition constraints. The *NoTrans* reference is actually a lower bound of what can be obtained when optimizing sequences of airspace configurations : it computes an optimal configuration at each step without taking the transition constraints into account. The other boxplots show the decrease of the overload cost when using the *sequential* A^* with increasing values of the rolling horizon. We see that on this problem instance, there is no need to increase the search depth beyond 5 minutes or 6 minutes, as there is no more improvement of the overload costs.

The right part of Figure 8 shows the ATC sector openings (in minutes) for the different methods. This is the second cost (number of working positions at every minute of the day), in order of importance, that is being minimized by our algorithms. We observe that, in addition to being slightly more costly in terms of overloads than the *greedy* method, the *sequential* A^* solutions also require slighthly less resources in manpower.

The boxplots of the underload and normal load costs, not presented here, show similar trends.

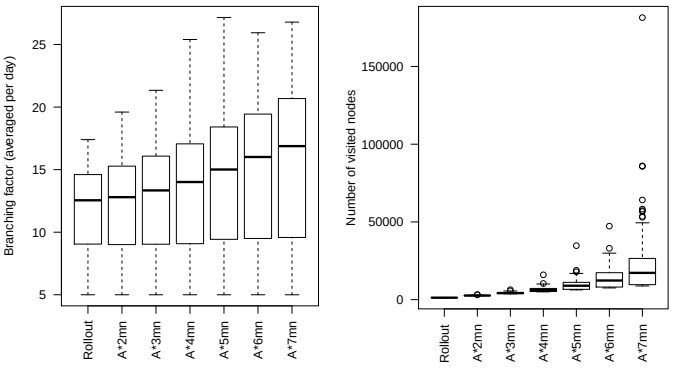

Figure 9: Branching factor (left) and number of visited nodes (right) for the sequential methods, with transition rules $\mathcal{T}_1$.

Increasing the search depth does increase the computational resources being used, though. Figure 9 shows the average

branching factor per day (left) and the number of nodes visited during the sequential search (right). We see that these two quantities increase rapidly with the search depth.

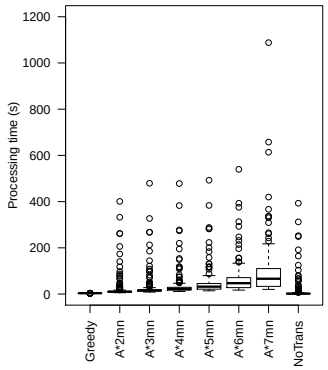

Figure 10: Processing time for each method, with transition rules $\mathcal{T}_1$.

Figure 10 shows the processing times on an AMD Ryzen Threadripper 1920X 12-Core Processor 3.3GHz with 32Gb of memory for all the methods[3]. As expected the processing time increases with the search depth of the sequential A^*. We also see that computation time of the *NoTrans* method is relatively important. We use this method in the computation of the heuristic of the A^* method, so a relatively large amount of the processing time of the *sequential* A^* can be imputed to the heuristic pre-computation. However, the processing time does increase very rapidly with the search depth, and we can observe a computation time above 1000 seconds for one day, when using a rolling horizon of 7 minutes.

B. Results with transition rules $\mathcal{T}_1$ and $M = 5$ minutes

Here, we consider the same set of transition rules $\mathcal{T}_1$, with the additional constraint that ATC sectors should be opened for a minimum duration of $M = 5$ minutes.

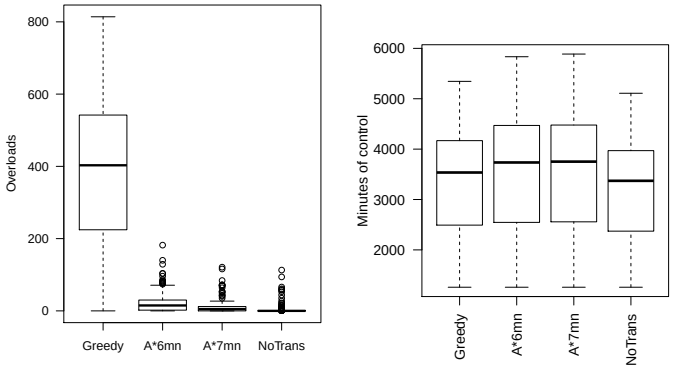

Figure 11: Overloads (left) and ATC sector openings (in minutes, right) for each method, with transition rules $\mathcal{T}_1$ and minimum sector opening time $M = 5$ minutes.

Figures 11, 12 and 13 show the same types of results as in the previous section. We observe on 11 (left) that the overloads are comparatively much higher for the *greedy* method when enforcing a minimum sector opening time of 5 minutes than with no minimum duration (Figure 8). This was to be expected because a bad choice of sector opening cannot be reconsidered by the *greedy* method for the next time steps when a minimum duration is imposed. To be effective, the time horizon of the *sequential* A^* must be chosen greater than the minimum duration M. With time horizons of 6 or 7 minutes, this method shows much less overloads than the *greedy* strategy. The performances are close the lower bound *NoTrans*.

Comparing Figures 12(left) and 9(left), we see that introducing the minimum sector opening duration constraint divides the average branching factor by 2 to 3. The number of visited nodes (right sides of the same figure) is higher with the additional constraint, certainly because the solutions satisfying the constraint are more difficult to find. The processing times (Fig. 13) are of the same order and increase rapidly with the time horizon.

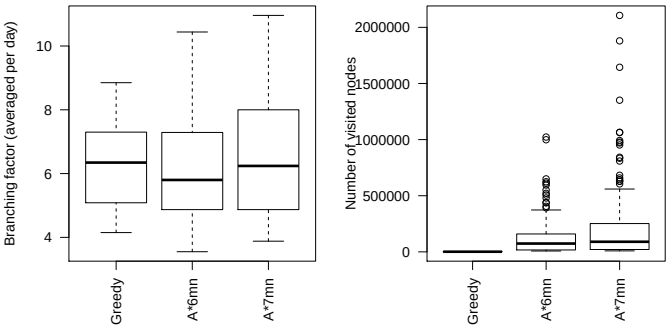

Figure 12: Branching factor (left) and number of visited nodes (right) for the sequential methods, with transition rules $\mathcal{T}_1$ and minimum sector opening time $M = 5$ minutes.

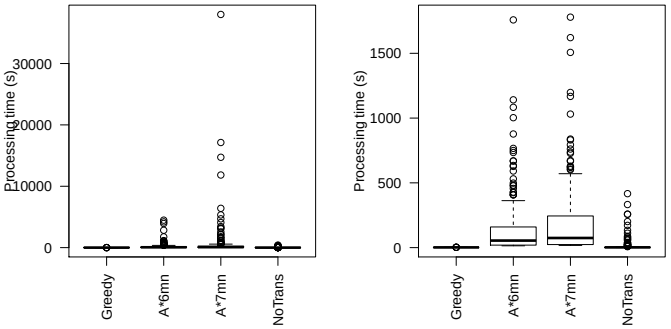

Figure 13: Processing time for each method, with transition rules $\mathcal{T}_1$ and minimum sector opening time $M = 5$ minutes. The left part shows all the points, and the right part zooms on the majority of the data points, discarding outliers with high computation times.

C. Results with transition rules $\mathcal{T}_2$

When using the transition rules $\mathcal{T}_2$ with up to $n_{act} = 2$ actions of sector reconfigurations (split, merge or transfer) at each time step, $n_{\text{split}} = 2$, $n_{\text{merge}} = 2$ and minimum sector

[3]Note that our code was not optimized for multi-core computation and used only one core.

opening time $M = 5$ minutes, we obtain results very similar to the ones of Figure 11, but with a computation time and a number of visited nodes more than 10 times higher (for a rolling horizon of 6 minutes).

With $n_{act} = 1$, $n_{\text{split}} = 3$, $n_{\text{merge}} = 2$ and $M = 5$, the results are very similar to the ones of section VI-B in all aspects, which would seem to indicate that situations where splitting a sector into 3 sectors are actually fairly rare.

VII. Conclusion

In conclusion, the greedy approach where a sequence of successive airspace configurations is obtained by picking up the best reachable configuration at each time step is clearly not optimal. The sequential A^* algorithm – which explores all possible sequences up to a chosen depth before making a decision at each time step – is much more efficient because it backtracks on the wrong choices in order to find better solutions. However, the computational cost of the A^* tree search is higher, and there is a trade-off between the quality of the solutions and the computation time. Allowing for more flexibility in the transition rules does not significantly improve the results, and induces higher computational costs when allowing more reconfiguration operations per time step.

The focus of this paper was to study how standard tree search methods perform on our problem of finding optimized sequences of airspace configurations. In order to make the problem challenging, we have voluntarily chosen a very flexible context in which the airspace configuration is reconsidered every minute. In current operations, the airspace is usually reconfigured only when an overload or a severe underload or imbalance is detected. Introducing this "lazy reconfiguration" rule – and other operational constraints and rules such as reconfiguring separately some sub-regions of the ATCC airspace – would certainly alleviate the difficulty of the problem and highly reduce the computational cost. This is left for further work.

The practical applicability of the proposed approach depends on its ability to provide responses in a short time. In the current operational context, standard determistic tree search algorithms such as the ones proposed in this paper might suffice to address the problem. However, they might not perform that well in a future context, with a greater number of airspace modules and a greater flexibility in the way to assemble them.

In future works, we might also try approximate or stochastic methods to address difficult problem instances with a limited budget of computation time.

References

[1] Global air traffic management operational concept. Technical report, International Civil Aviation Organization, 2005.

[2] H. Swenson, R. Barhydt, and M. Landis. Next Generation Air Transportation System (NGATS) Air Traffic Management (ATM)-Airspace Project. Technical report, National Aeronautics and Space Administration, 2006.

[3] SESAR Consortium. Milestone Deliverable D3: The ATM Target Concept. Technical report, 2007.

[4] A. Klein, P. Kopardekar, M. D. Rodgers, and H. Kaing. Airspace playbook: Dynamic airspace reallocation coordinated with the national severe weather playbook. In *Proceedings of the 7th AIAA Aviation Technology, Integration and Operations Conference*, 2007.

[5] Ingrid Gerdes, Annette Temme, and Michael Schultz. Dynamic airspace sectorisation for flight-centric operations. *Transportation Research Part C: Emerging Technologies*, 95:460 – 480, 2018.

[6] P. Kopardekar and S. Magyarits. Measurement and prediction of dynamic density. In *Proceedings of the 5th USA/Europe Air Traffic Management R & D Seminar*, 2003.

[7] Gano B Chatterji and Banavar Sridhar. Neural network based air traffic controller workload prediction. In *American Control Conference, 1999. Proceedings of the 1999*, volume 4, pages 2620–2624. IEEE, 1999.

[8] D. Gianazza and K. Guittet. Evaluation of air traffic complexity metrics using neural networks and sector status. In *Proceedings of the 2nd International Conference on Research in Air Transportation*. ICRAT, 2006.

[9] D. Gianazza. Learning air traffic controller workload from past sector operations. In *Proceedings of the 12th USA/Europe Air Traffic Management R & D Seminar*, 2017.

[10] Pierre Flener and Justin Pearson. Automatic airspace sectorisation: A survey. *arXiv preprint arXiv:1311.0653*, 2013.

[11] Shannon Zelinski and Chok Fung Lai. Comparing methods for dynamic airspace configuration. In *Digital Avionics Systems Conference (DASC), 2011 IEEE/AIAA 30th*, pages 3A1–1. IEEE, 2011.

[12] Michael Bloem and P Gupta. Configuring airspace sectors with approximate dynamic programming. In *International Congress of the Aeronautical Sciences 2010*, number ARC-E-DAA-TN1935, 2010.

[13] Judicaël Bedouet, Thomas Dubot, and Luis Basora. Towards an operational sectorisation based on deterministic and stochastic partitioning algorithms. In *The Sixth SESAR Innovation Days*, 2016.

[14] Judicaël Bedouet and Thomas Dubot. Tactical prediction of the number of control positions with softmax regression and tree search. In *The Eighth SESAR Innovation Days*, 2018.

[15] D. Gianazza and J. M. Alliot. Optimization of air traffic control sector configurations using tree search methods and genetic algorithms. In *Proceedings of the 21st Digital Avionics Systems Conference*, 2002.

[16] D. Gianazza. Airspace configuration using air traffic complexity metrics. In *Proceedings of the 7th USA/Europe Seminar on Air Traffic Management Research and Development*, 2007. best paper of "Dynamic Airspace Configuration" track.

[17] D. Gianazza, C. Allignol, and N. Saporito. An efficient airspace configuration forecast. In *Proceedings of the 8th USA/Europe Air Traffic Management R & D Seminar*, 2009.

[18] D. Gianazza. Forecasting workload and airspace configuration with neural networks and tree search methods. *Artificial Intelligence Journal, Elsevier*, 174(7-8):530–549, may 2010.

[19] ACT-540 NAS Advanced Concepts Branch. An evaluation of dynamic density metrics using RAMS. Technical report (draft) DOT/FAA/CT-TN, Federal Aviation Administration, April 2001.

[20] A. Yousefi, G. L. Donohue, and K. M. Qureshi. Investigation of en route metrics for model validation and airspace design using the total airport and airspace modeler (TAAM). In *Proceedings of the fifth USA/Europe Air Traffic Management R&D Seminar*, 2003.

[21] A. J. Masalonis, M. B. Callaham, and C. R. Wanke. Dynamic density and complexity metrics for realtime traffic flow management. In *Proceedings of the 5th USA/Europe Air Traffic Management R & D Seminar*, 2003.

[22] A. Majumdar, W. Y. Ochieng, G. McAuley, J.M. Lenzi, and C. Lepadetu. The factors affecting airspace capacity in europe: A framework methodology based on cross sectional time-series analysis using simulated controller workload data. In *Proceedings of the 6th USA/Europe Air Traffic Management R & D Seminar*, 2005.

[23] G.B. Chatterji and B. Sridhar. Measures for air traffic controller workload prediction. In *Proceedings of the First AIAA Aircraft Technology, Integration, and Operations Forum*, 2001.

[24] D. Gianazza and K. Guittet. Selection and evaluation of air traffic complexity metrics. In *Proceedings of the 25th Digital Avionics Systems Conference*. DASC, 2006.

[25] D. Gianazza. Smoothed traffic complexity metrics for airspace configuration schedules. In *Proceedings of the 3nd International Conference on Research in Air Transportation*. ICRAT, 2008.

[26] David Gianazza. Analysis of a workload model learned from past sector operations. In *SID 2017, 7th SESAR Innovation Days*, 2017.

[27] D. Gianazza, J. M. Alliot, and G. Granger. Optimal combinations of air traffic control sectors using classical and stochastic methods. In *Proceedings of the 2002 International Conference on Artificial Intelligence*, 2002.