Fourteenth USA/Europe Air Traffic Management Research and Development Seminar (ATM2021)

Predicting Air Traffic Congested Areas with Long Short-Term Memory Networks

Loïc Shi-Garrier, Daniel Delahaye
Ecole Nationale de l'Aviation Civile
Université de Toulouse
Toulouse, France

Nidhal C. Bouaynaya
Dept. of Electrical and Computer Engineering
Rowan University
New Jersey, USA

***Abstract*—The Extended ATC Planning (EAP) function aims at bridging the gap between Air Traffic Flow & Capacity Management (ATFCM) and Air Traffic Control (ATC) by predicting air traffic congested areas tens of minutes before their formation, and by suggesting real-time and fine-tuning measures to alleviate airspace "complexity". Current Air Traffic Flow Prediction methods focus on crude aircraft count in a sector, and hence are unable to distinguish between low and high complexity situations for a similar aircraft count. Complexity indicators, on the other hand, aggregate air traffic measurements and workload to describe the perceived complexity. However, the evaluation of workload is a long-debated issue and an inherently ill-posed problem. In this work, we present an intrinsic complexity metric, independent of any traffic control system, and address the prediction task of the EAP function using a novel Encoder-Decoder Long Short-Term Memory (LSTM) neural network. The complexity is measured by the eigenvalues of a linear dynamical system fitted to the aircraft's speed vectors. The Encoder-Decoder LSTM network uses a sequence of (discrete) aircraft states to predict the complexity of the air traffic in all areas of an airspace, in a time horizon of 40 minutes. Simulated traffic corresponding to one day of traffic over the French airspace is used to train and validate the model. Our experiments show that the proposed model achieves a Mean Absolute Error of 0.08 in predicting the normalized complexity value 40 minutes in the future.**

***Keywords*—Air traffic flow prediction; Complexity metrics; Long short-term memory (LSTM) networks.**

I. Introduction

A. Background and Motivation

With the continuous rise in air transportation demand (a 4.3% annual growth of the worldwide passenger demand is expected until 2035 [1]), the capacity of the Air Traffic Management (ATM) system is reaching its limits, leading to increased flight delays (expected to achieve 8.5 minutes per flight in 2035 with the current system [2]). The Covid-19 crisis has severely impacted the aviation industry, with a 90% decline of RPK (Revenue Passenger Kilometer) in July 2020 with respect to the same period in 2019. However, the traffic is expected to regain its growth rate once the crisis is over, with the worst-case scenario predicting a return to normality before 2025.

Congestion is defined as a situation where a set of trajectories strongly interact in a given area and time interval, leading to potential conflicts and hence requiring high monitoring from the controllers. Air traffic complexity is a measure of the difficulty that a particular traffic situation will present to air traffic control [3], and was shown to negatively affect the controller's decision-making ability and increase the error rate [4]. To address this complexity and improve the service provision to airspace users through reduced delays, better punctuality, less ATFCM regulations, and enhanced safety, the Extended ATC Planning (EAP) function was introduced by SESAR JU as the Solution #118 [5]. The EAP function relies on automation tools that enable early measures to be taken by ATC before traffic enters overloaded sectors. The implemented tools aim at bridging the gap between Air Traffic Flow and Capacity Management (ATFCM) and Air Traffic Control (ATC) by facilitating the communication between the local flow management position (FMP) and the controllers' working positions, and providing information to help taking actions at tactical time (less than one hour) such as rerouting or sector configurations. The gap between ATFCM and ATC can be described as follows: although ATFCM measures have managed to balance the capacity with the demand, high complexity areas may appear at the control level. These "hot spots" or "traffic bursts" require an intense surveillance from air traffic controllers in order to decide if some trajectories should be modified to avoid any conflict. Two problems must be addressed to reduce this increased workload: 1) the prediction of hot spots tens of minutes before their formation, and 2) the mitigation of the hot spots by appropriate early actions. This paper focuses on the first problem of the prediction of congested areas tens of minutes ahead of formation.

B. Related Work

We can group the literature to tackle the general problem of congestion prediction into three main approaches. The first approach considers conflict detection tools in a time horizon lower than thirty minutes [6], [7]. The second approach focuses on predicting the air traffic flow [8], [9], [10], [11], [12], [13]. The third approach computes various air traffic complexity metrics [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [3], [24], [25].

In the first approach of short-term conflict detection methods [6], the Medium-Term Conflict Detection (MTCD) flight data processing is perhaps the most popular system. MTCD is

designed to warn the controller of potential conflicts between flights in a time horizon extending up to thirty minutes ahead. The system integrates predictive tools that performs trajectory prediction, conflict detection, trajectory update, and trajectory edition using "what-if" scenarios and tools [6]. The MTCD applications, currently implemented in operational context, e.g., the Eurocontrol MTCD project [7], are mainly based on pairwise conflicts rather than on a global approach. Moreover, for longer time horizons, the uncertainties on the trajectories make it difficult to predict the exact trajectories that will be involved in conflicts.

Air traffic flow prediction (ATFP) focuses on aggregate models [8] rather than simulating the trajectories of individual aircraft. Trajectory-based models result in a large number of states, which are susceptible to error and difficult to design and implement for air traffic flow management [10]. The ATFP approach develops models of the behavior of air traffic that can be used for the analysis of traffic flow management. Several methods were considered, ranging from probabilistic algorithms modeling the flow over a network [8], [9], [10] to more recent machine learning algorithms [11], [12], [13]. In [8], a linear time variant traffic flow model was developed based on historical data. The dimension of this model depends only on the number of considered control volumes and not on the number of individual aircraft. This model is able to forecast the aircraft count at an Area Control Center level with a time interval of ten minutes. Another flow-based model was later developed in [9], called Link Transmission Model. In this model, a flight path is defined as a sequence of directed links passing through sectors. An aircraft is assumed to cross each of the links within an estimated crossing time such that the state of each link can be easily tracked. Aggregation of links in each sector yields a traffic forecast for that sector. This approach is extended in [10] where the authors introduce a dynamic network of air traffic flow characterizing both the static airspace topology and the dynamics of the traffic flow. Historical data is used to estimate the travel time corresponding to the weight of each edge of the network. Then, a probabilistic prediction method is implemented for the short-term prediction (fifteen minutes) of the air traffic flow.

Recently, several machine learning models were proposed for ATFP. In [11], the authors define a 3D grid over a given airspace containing the number of flights in each cell. They rely on neural network models that combine convolutional operations to extract spatial features and recurrent operations to extract time-dependent features. The 3D grids of the last ten minutes are inputted into the model to predict the 3D grid of the next timestep. A similar task is addressed in [12] using several 3D convolutional neural networks to extract spatial features for one timestep then recurrent layers to extract temporal features. In [13], the authors used support vector machines (SVM) and recurrent neural networks to predict the hourly air flow in predefined routes from time information (such as the time of the day, or the season and holiday indexes). The main drawback of all the above-mentioned ATFP methods is that they do not define any concept of congestion since the predicted variable is the aircraft count whether in a sector, a route, or crossing a predefined waypoint. Hence, these models are unable to discriminate low complexity situations from high complexity situations for a similar aircraft count.

To address this limitation, the third approach redefines the forecast objective of the ATFP problem by considering complexity metrics. Research focusing on defining and quantifying air traffic complexity, and on analyzing its impact on air traffic controller workload, was extensively conducted during the last two decades [14], [15], [16], [17], [18], [19], [20], [21]. Extensive reviews on this topic can be found in [3] and [24]. The traditional measure of air traffic complexity is the traffic density, defined as the number of aircraft crossing a given sector in a given period [16]. The traffic density is compared with the operational capacity, which is the acceptable number of aircraft allowed to cross the sector at the same time period. This crude metric does not take into account the traffic structure and the geometry of the airspace. Hence, a controller may continue to accept traffic beyond the operational capacity, or refuse aircraft even though the operational capacity has not been reached. The Dynamic Density introduced in [14] aims at producing an aggregate measure of complexity by combining complexity factors, including static air traffic characteristics (such as airway crossings) and dynamic air traffic characteristics (such as the number of aircraft and the closing rates). A list of these complexity factors can be found in [16]. The complexity factors can be combined either linearly as in [15] or through a neural network [17], [18]. Several versions of Dynamic Density were advanced [19], [20]. For example, the interval complexity introduced in [21] is a variant of Dynamic Density where the chosen complexity factors are averaged over a time window.

As elaborated in [3], these complexity indicators aggregate air traffic measurements and workload to describe the perceived complexity. However, the evaluation of workload is a long-debated issue and an inherently ill-posed problem. The difficulty of obtaining reliable and objective workload measures is the main motivation for investigating complexity metrics that are independent of the ATC workload. To address this issue, another approach has been developed in the literature: the *intrinsic complexity metrics* approach [22], [23]. In [24], a difference is made between the control workload and the traffic complexity. The control workload measures the difficulty for the traffic control system (human or not) to remedy a situation, whereas the traffic complexity is an intrinsic measurement of the complexity of the air traffic, independently of any traffic control system. In other words, intrinsic complexity metrics aim at modelling the level of disorder and the organization structure of the air traffic distribution, irrespective of its effect on the ATC workload. In [22], the authors interpolate a velocity vector field satisfying certain constraints, e.g., the field shall be equal to the velocity of an aircraft if an aircraft is present at this point, the field shall be flyable by an aircraft. The problem is modeled as a mixed integer linear program. The complexity is measured as the number of constraints that must be relaxed to obtain a feasible problem. In our previous work [23], we defined

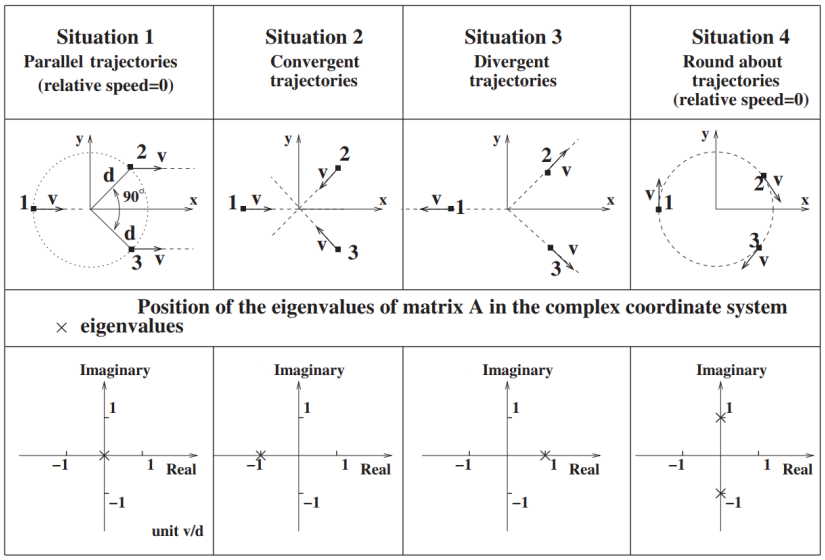


Figure 1: Eigenvalues loci of the dynamical system matrix A for 4 typical situations: parallel, convergent, divergent, and round about trajectories. Observe that the real part of the eigenvalues are negative if and only if the trajectories are convergent.

;

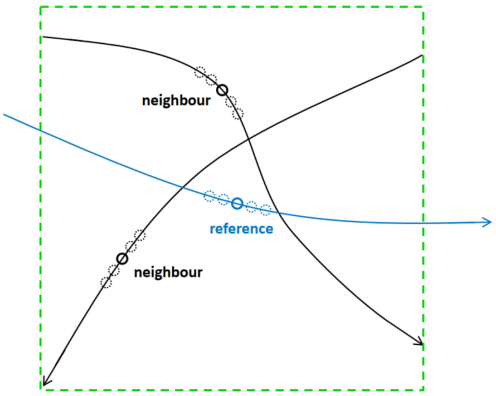


Figure 2: The neighborhood of a reference aircraft to compute the complexity metric. This neighborhood is defined as a 24.8 NM × 24.8 NM box and ±30 FL altitude, centered on the reference aircraft. Trajectories can be augmented to account for the uncertainties along the curvilinear abscissa (dotted points).

an intrinsic complexity metric using linear and nonlinear dynamical system models. The air traffic situation is modeled by an evolution equation (the aircraft trajectories being integral lines of the dynamical system). The complexity is measured by the Lyapunov exponents of the dynamical system. The Lyapunov exponents capture the sensitivity of the dynamical system to initial conditions: if a small variation in the current air traffic situation leads to a very different dynamical system, the complexity of the situation is considered to be high.

This paper uses state-of-the-art machine learning techniques to integrate ATFP with the intrinsic complexity metric developed in [23]. The main objective of this work is to provide congestion predictions in an extended time horizon, between forty to sixty minutes. Due to the uncertainties related to such time scales, we will directly predict congested areas rather than address the classical problem of trajectory prediction or conflict detection and resolution. In contrast to classic ATFP approaches that predict the aircraft count, we introduce a set of aircraft states (e.g., latitude, longitude, altitude, ground speed, heading, and climb rate) and use a Long Short-Term Memory (LSTM) network to predict congestion. We use an Encoder-Decoder framework where the input is the state of the aircraft and the output is the spatio-temporal complexity map measuring the spatial complexity at every time instant.

The remainder of the paper is organized as follows. Section II surveys the mathematically-grounded intrinsic complexity metric considered in this paper. Section III introduces the machine learning model, namely the encoder-decoder LSTM architecture. In Section IV, the data and the pre-processing techniques that were performed to build the training dataset are presented. Implementation details are also discussed for reproducibility of the results. The Python code is made available in GitHub[1]. Section V elaborates on the results of the model. Considering only the areas with a nonzero complexity value, the best model achieves a Mean Absolute Error of 8% in predicting the complexity value forty minutes in the future. Finally, Section VI summarizes the paper and suggests several improvements to system deployment as well as new applications that could benefit from this work.

II. Complexity Metric

We first introduce the mathematical notations used in the paper.

A. Mathematical Notation

Scalars are represented by lower-case letters, e.g., x, x_i, y_{ij}. Vectors are represented by bold lower-case letters, e.g., $\mathbf{x}$, $\mathbf{b}$. All vectors are column vector. Matrices are represented by bold upper-case letters, e.g., $\mathbf{A}$, $\mathbf{W}$. $\|\cdot\|_F$ is the Frobenius norm. Re is the real part function. $\lambda(\mathbf{A})$ is the spectrum of the matrix $\mathbf{A}$. The concatenation of two vectors $\mathbf{x}$ and $\mathbf{y}$ into a single column vector is denoted by $[\mathbf{x}^T, \mathbf{y}^T]^T$ where ^{T} is the transpose operator. σ is the logistic activation function $\sigma : x \mapsto \frac{1}{1+\exp(-x)}$.

B. Complexity Metric based on Linear Dynamical System

In this work, the intrinsic complexity metric is based on a linear dynamical system model that was recently introduced in [25]. This metric computes a measure of complexity in the neighborhood of an aircraft at a given time (see Figure 2). A filter is applied to consider only the flights that may interact with the reference aircraft. For example, an aircraft that is vertically separated with the reference aircraft by 1000 ft (in RVSM) will not interact with the reference aircraft, and, thus, should not be considered in the metric computation. After this

[1] https://github.com/lshigarrier/PFE

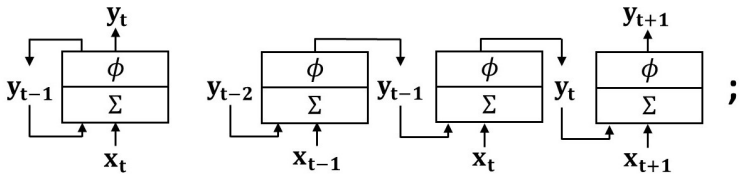


Figure 3: Basic RNN layer (left) and the same layer unrolled through time (right). Σ represents linear mapping and ϕ represents the activation function.

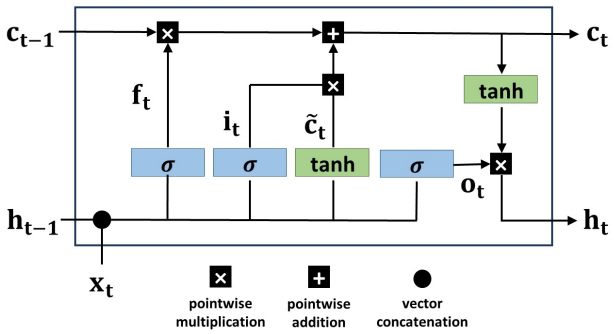


Figure 4: A Long Short-Term Memory (LSTM) layer. The current input $\mathbf{x}_t$ is combined with the previous hidden state $\mathbf{h}_{t-1}$ to compute the three gates used to update the cell state $\mathbf{c}_{t-1}$ and compute a new hidden state $\mathbf{h}_t$.

filtering, the selected aircraft positions are extended by adding p forward positions and p backward positions as depicted on Figure 2 with $p = 2$. The aim of this extension is to take into account uncertainties on the true aircraft's positions.

Let n_{ac} be the number of aircraft samples retained for the computation of the metric. Consider the matrices $\mathbf{P}$ and $\mathbf{V}$, which denote, respectively, the positions and velocities of the n_{ac} aircraft, i.e.,

$$\mathbf{P} = \begin{bmatrix} x_1 & x_2 & \dots & x_{n_{ac}} \\ y_1 & y_2 & \dots & y_{n_{ac}} \\ z_1 & z_2 & \dots & z_{n_{ac}} \end{bmatrix}; \mathbf{V} = \begin{bmatrix} v_{x_1} & v_{x_2} & \dots & v_{x_{n_{ac}}} \\ v_{y_1} & v_{y_2} & \dots & v_{y_{n_{ac}}} \\ v_{z_1} & v_{z_2} & \dots & v_{z_{n_{ac}}} \end{bmatrix},$$

where x_i, y_i, z_i are the coordinates and $v_{x_i}, v_{y_i}, v_{z_i}$ the speed components of the i^{th} sample. A linear dynamical system $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{b}$ is fitted to the positions $\mathbf{P}$ and speeds $\mathbf{V}$ of these aircraft, such that $\mathbf{V} \simeq \mathbf{A}\mathbf{P} + \mathbf{b}$ (here, $\mathbf{b}$, representing the mean of the speed vector field, is broadcast to be added to the matrix $\mathbf{AP}$). The matrix $\mathbf{A}$ and the vector $\mathbf{b}$ are computed as the solutions of the Least Mean Squares problem

$$\min_{\mathbf{A},\mathbf{b}} \|\mathbf{V} - (\mathbf{A}\mathbf{P} + \mathbf{b})\|_F^2. \tag{1}$$

The complexity metric, $c(\mathbf{A})$, is then defined as the absolute sum of the negative real part of the eigenvalues of the matrix $\mathbf{A}$

$$c(\mathbf{A}) = \sum_{\mathrm{Re}(\lambda(\mathbf{A}))<0} |\mathrm{Re}(\lambda(\mathbf{A}))|. \tag{2}$$

Recall that the evolution equation of a linear dynamical system has the form $\mathbf{P}(t) \sim \exp(\mathbf{A}t)$. By diagonalising the matrix $\mathbf{A}$, we can see that the asymptotic behaviour of the system depends uniquely on the eigenvalues of $\mathbf{A}$. Positive real parts of the eigenvalues correspond to diverging behaviour along the associated eigendirections while negative real parts correspond to convergence to a critical point (Figure 1). The imaginary parts correspond to rotation behaviour. Hence, our metric $c(\mathbf{A})$ measures the strength of the converging or shrinking behaviour of the dynamical system. Since our system is the closest linear dynamical system fitting the current positions and velocities of the aircraft, a strong converging behaviour corresponds to rapidly converging trajectories, which we associate with high complexity. This metric can be computationally efficient using known techniques from Numerical Linear Algebra (NLA) [26] and Randomized NLA (RandNLA) [27].

III. Encoder-Decoder Long Short-Term Memory Network

In this Section, we present the sequential machine learning models that will process the series of aircraft state vectors. Sequence models are first introduced, then integrated into an encoder-decoder network architecture. Finally, a one-dimensional convolutional layer used in the initial processing of aircraft state vectors is described.

A. Sequential Models

Recurrent Neural Networks (RNN) are a class of neural networks designed to process sequential data such as text streams, audio clips, and time-series data [28], [29], [30]. In these recurrent models, network weights are shared across time, building an internal memory that can capture sequential knowledge. RNNs have been extensively used in Natural Language Processing, including automatic translation [28] and speech-to-text [29].

In a basic RNN layer (see Figure 3), the recurrent cell receives the input $\mathbf{x}_t$ as well as its own output from the previous time step $\mathbf{y}_{t-1}$. The output $\mathbf{y}_t$ is computed as

$$\mathbf{y}_t = \phi(\mathbf{W} \cdot [\mathbf{x}_t^T, \mathbf{y}_{t-1}^T]^T + \mathbf{w}_0), \tag{3}$$

where $\mathbf{W}$ is the weights matrix, $\mathbf{w}_0$ is the bias vector and ϕ is an activation function e.g., $\tanh$ or ReLU. Observe that the output $\mathbf{y}_t$ depends not only on the current input $\mathbf{x}_t$ but also on the entire sequence of previous inputs $\mathbf{x}_0, \dots, \mathbf{x}_{t-1}$ using the same number of learnable parameters regardless of the length of the input sequence. However, the basic RNN layer has difficulty learning long-term dependencies when the length of the sequence is large. This problem has been referred to as the *vanishing gradient* problem [31]. To solve it, several architectures of RNN layers with long-term memory have been introduced such as Long Short-Term Memory (LSTM) [32] and Gated Recurrent Unit (GRU) [33] networks.

Long Short-Term Memory networks (Figure 4) were first introduced in [32], and then improved over the years [34],

[35]. LSTMs detect long-term dependencies by using three gates that control the gradient propagation in the recurrent network's memory, i.e., the *forget gate* $\mathbf{f}_t$, the *input gate* $\mathbf{i}_t$, and the *output gate* $\mathbf{o}_t$. In contrast to the basic RNN layer, the LSTM has two state vectors denoted $\mathbf{h}_t$ (the hidden state) and $\mathbf{c}_t$ (the cell state), which can be seen, respectively, as the short-term state and the long-term state. The LSTM's governing equations are given by

$$\text{(input gate)} \quad \mathbf{i}_t = \sigma(\mathbf{W}_i \cdot [\mathbf{x}_t^T, \mathbf{h}_{t-1}^T]^T + \mathbf{w}_i), \tag{4}$$

$$\text{(forget gate)} \quad \mathbf{f}_t = \sigma(\mathbf{W}_f \cdot [\mathbf{x}_t^T, \mathbf{h}_{t-1}^T]^T + \mathbf{w}_f), \tag{5}$$

$$\text{(output gate)} \quad \mathbf{o}_t = \sigma(\mathbf{W}_o \cdot [\mathbf{x}_t^T, \mathbf{h}_{t-1}^T]^T + \mathbf{w}_o), \tag{6}$$

$$\text{(candidate cell state)} \quad \tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c \cdot [\mathbf{x}_t^T, \mathbf{h}_{t-1}^T]^T + \mathbf{w}_c), \tag{7}$$

$$\text{(new cell state)} \quad \mathbf{c}_t = \mathbf{f}_t \times \mathbf{c}_{t-1} + \mathbf{i}_t \times \tilde{\mathbf{c}}_t, \tag{8}$$

$$\text{(new hidden state)} \quad \mathbf{h}_t = \mathbf{o}_t \times \tanh(\mathbf{c}_t), \tag{9}$$

where $\mathbf{W}_i$, $\mathbf{w}_i$, $\mathbf{W}_f$, $\mathbf{w}_f$, $\mathbf{W}_o$, $\mathbf{w}_o$, $\mathbf{W}_c$, and $\mathbf{w}_c$ are the learnable weights and biases of the LSTM network. The current input $\mathbf{x}_t$ and the previous hidden state $\mathbf{h}_{t-1}$ are used to compute the controllers of three *gates*: the input gate controller $\mathbf{i}_t$, the forget gate controller $\mathbf{f}_t$ and the output gate controller $\mathbf{o}_t$ (Eqs. (4-6)). These gate controllers are simply computed by a linear mapping followed by a logistic activation. In parallel, a candidate cell state $\tilde{\mathbf{c}}_t$ is computed using the current input and the previous hidden state (Eq. 7). Eq. 7 can be seen as the LSTM equivalent of the basic RNN layer. In Eq. 8, the current cell state $\mathbf{c}_t$ is then updated by adding the previous cell state $\mathbf{c}_{t-1}$ (filtered by the forget gate) and the candidate cell state $\tilde{\mathbf{c}}_t$ (filtered by the input gate). The hidden state $\mathbf{h}_t$ is finally computed by activating the cell state $\mathbf{c}_t$ with the $\tanh$ function (or another activation function, such as ReLU), which is then passed though the output gate (Eq. 9). With the input gate $\mathbf{i}_t$, the LSTM is able to store information in the cell state $\mathbf{c}_t$, to extract it with the output gate $\mathbf{o}_t$ and to discard it with the forget gate $\mathbf{f}_t$.

B. Encoder-Decoder LSTMs

The first introduction of an encoder-decoder model was provided for an automatic translation task [28]. This model has also been applied to various tasks, such as speech recognition [36] and video captioning [37]. The Encoder-Decoder LSTM is a recurrent neural network designed to address sequence-to-sequence problems. Sequence-to-sequence prediction problems are challenging because the number of items in the input and output sequences can vary. The Encoder-Decoder architecture is comprised of two models: one for encoding the input sequence into a fixed-length vector, and a second for decoding the fixed-length vector and outputting the predicted sequence. The innovation of this architecture is the use of a fixed-sized internal representation in the heart of the model, which may be referred to as *sequence embedding*. Both the encoder and decoder models are defined as recurrent neural networks, i.e., LSTMs.

The input of the decoder consists of two elements: the output of the encoder and the previously predicted (i.e., decoded) output sequence term. During training, the previously decoded sequence is provided as the ground-truth sequence at the previous timestep. During inference, the true output sequence is obviously not known. Hence the input of the decoder network has to be redefined. A simple method consists in recursively using the prediction of the last timestep as the input for the next one. However, this method may result in accumulation of prediction errors and lead to poor performance, especially if the output sequence is long enough. A solution to this issue is to change the task of the decoder network such that the network does not predict an element of the output sequence at each timestep, but rather a distribution over the possible outputs. Then, using a beam search algorithm [38], it is possible to maintain, at each timestep, a small number of possible output sequences using the partial probability of each sequence as the metric. At the last timestep, the model outputs the sequence with the highest probability.

This framework is more suited for classification tasks where the output distribution is discrete and finite. Even if it is possible to choose a family of continuous distributions to represent the set of possible output distributions of the decoder network for regression, a simpler approach has been retained here. The objective of the model is to find the output sequence $\mathbf{Y}$ that maximizes the conditional probability $P(\mathbf{Y}|\mathbf{X})$ given the input sequence $\mathbf{X}$. More formally, we are solving the following optimization problem:

$$\arg\max_{\mathbf{Y}} P(\mathbf{Y}|\mathbf{X}) = \arg\max_{\mathbf{Y}} \prod_t P(\mathbf{y}_t|\mathbf{y}_1, \ldots, \mathbf{y}_{t-1}, \mathbf{X}).$$

The network directly outputs a prediction $\mathbf{y}_\tau = \arg\max_{\mathbf{y}_\tau} P(\mathbf{y}_\tau|\mathbf{y}_1, \ldots, \mathbf{y}_{\tau-1}, \mathbf{X})$ for the τ^{th} element of the output sequence. Hence, at time step τ, the current estimation of the probability of the partial sequence $\mathbf{y}_1, \ldots, \mathbf{y}_{\tau-1}$ is $\prod_t^\tau \max_{\mathbf{y}_t} P(\mathbf{y}_t|\mathbf{y}_1, \ldots, \mathbf{y}_{t-1}, \mathbf{X})$. This greedy approach can be seen as approximating $\max_{\mathbf{Y}} P(\mathbf{Y}|\mathbf{X}) = \max_{\mathbf{Y}} \prod_t P(\mathbf{y}_t|\mathbf{y}_1, \ldots, \mathbf{y}_{t-1}, \mathbf{X})$ by $\prod_t \max_{\mathbf{y}_t} P(\mathbf{y}_t|\mathbf{y}_1, \ldots, \mathbf{y}_{t-1}, \mathbf{X})$. This approach is equivalent to the beam search algorithm with a beam width of 1.

C. One-dimensional (1-D) Convolutional Layer

The first layers of the encoder network are defined as 1-D convolutional layers to process the input sequence of aircraft states. A 1-D convolutional layer is composed of f different filters. A filter contains a kernel of learnable parameters with dimension $d \times m$, where m is the dimension of the input sequence, and d, the width of the kernel, is a hyperparameter of the convolutional layer. Each filter aggregates a subsequence of the input sequence of length d such that each element of the output sequence of the layer contains information from several successive timesteps. More precisely, we have:

$$y_{ij} = \phi\Big(\sum_{k=-d}^{d} \sum_l w_{kl}^j x_{i+k,l}\Big),$$

where y_{ij} is the j^{th} vector element of the i^{th} sequence term outputted by the 1-D convolutional layer, and x_{kl} is the l^{th}

Figure 5: Simulated trajectories of the dataset plotted in the horizontal plane.

vector element of the k^{th} sequence term of the input; w_{kl}^{j} is the k^{th} weight of the kernel of the j^{th} filter associated with the l^{th} dimension of the input sequence, and ϕ is an activation function. If $i+k$ is lower than zero or greater than the input sequence's length, then $x_{i+k,l} = 0$, for all l.

The convolution operation is performed only along the time dimension of the input sequence; this is why this convolutional layer is referred to as "one-dimensional". With the 1-D convolutional layers, we expect the encoder network to identify dependencies in a very short time window, such that the LSTM layers could process an intermediate sequence in which each timestep contain dynamical information from the previous and following timesteps. This may facilitate the computation of the embedding for the whole sequence. Without this convolutional layers, the LSTM layers would process one timestep after the other without any information from the next timesteps.

IV. Methodology and experiments

We now elaborate on the implementation details of our model, starting with the description of the dataset and the construction of the training set. Then, we provide the architecture details of the encoder-decoder model for reproducibility of our results.

A. Data acquisition and preprocessing

In order to test the capacity of the model to predict the complexity of the traffic, we use a dataset of simulated trajectories. In our simulation, no deconflicting actions were applied, such that each flight follows its flight plan without interacting with the other flights. Hence, we obtain the true complexity before mitigation, as opposed with historical data where the controllers have already applied measures to lower the complexity. The dataset represents a regular day of traffic over the French airspace. A trajectory is defined as a sequence of aircraft states. Each aircraft state consists of the following 8 elements.

$$\textit{State} = [\text{timestamp, aircraft ID, latitude, longitude, altitude, ground speed, heading, rate of climb}].$$

The trajectories are created by inputting flight plans to a simulator, where the flight plans consist of a sequence of waypoints and a cruise flight level. The atmosphere is assumed to be in ISA conditions with no wind. The final dataset, plotted in Figure 5, is composed of $8,011$ simulated trajectories spread along $3,025$ timesteps (1 timestep corresponds to 15 seconds). A complexity value is then computed for each aircraft at each timestep as described in Section II.

The objective of our supervised learning model is a sequence-to-sequence regression task. To achieve this, a training set must be built using pairs of training inputs/outputs, composed of sequences of aircraft states paired with sequences of complexity values over the entire airspace. To build the training outputs, we defined, for each time step t, a $n \times n$ matrix $\mathbf{H}_t$ that will be referred as a complexity matrix. This matrix is superimposed over the airspace, such that each element of the matrix covers a small rectangle area whose dimensions depend on the airspace size and on the parameter n. To avoid a too high output dimension, we do not consider the altitude axis. Hence, each area covers all flight levels and is only defined by the latitude and longitude of its center point. Then, for a given area, the corresponding element of the matrix takes the maximum value of the complexity metric of the aircraft inside this area at timestep t. The complexity values are then scaled by a logarithmic function $x \mapsto \log(1+x/Th)$, where Th is a fixed threshold, to obtain a

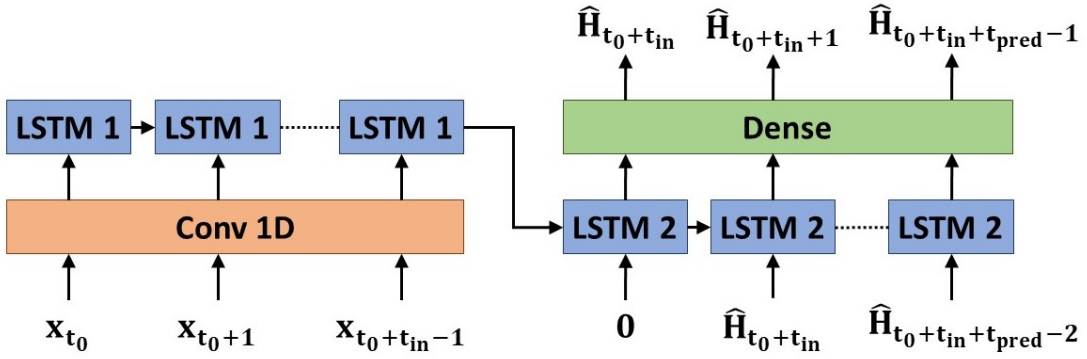


Figure 6: Encoder-decoder LSTM model for complexity prediction. The encoder network (left) uses a sequence of aircraft states $\mathbf{x}_{t_0}, \cdots, \mathbf{x}_{t_0+t_{in}-1}$ to compute an *encoding sequence*. The decoder network (right) uses the encoding sequence and the complexity matrix outputted at the last timestep $\hat{\mathbf{H}}_{t-1}$ to compute a prediction.

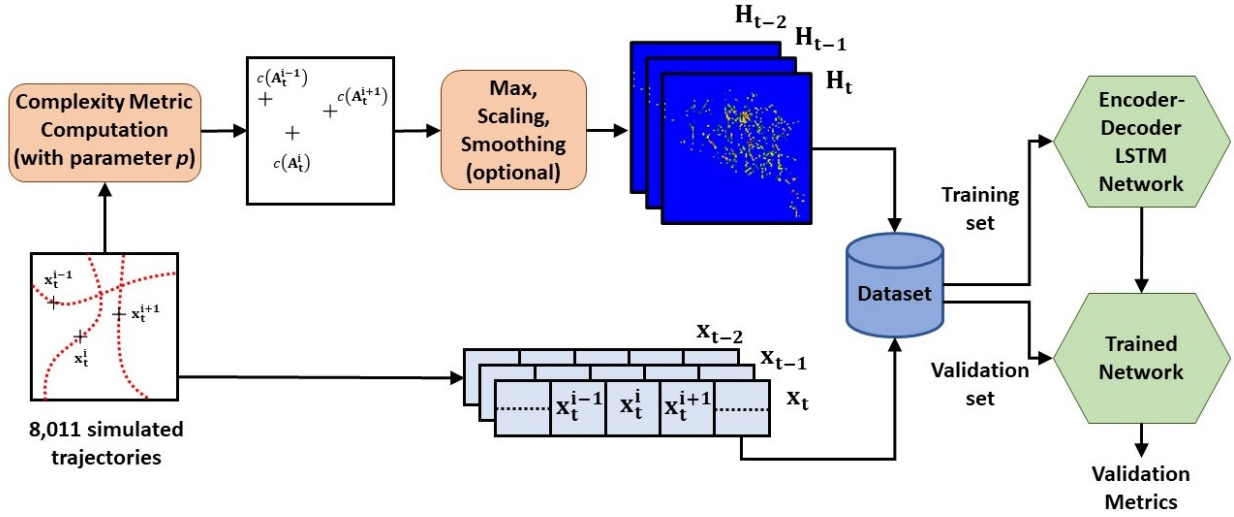


Figure 7: Depiction of the entire framework. The dataset is composed of trajectories where $\mathbf{x}_t^i$ is the state vector of the i^{th} trajectory at time t. The complexity values are computed as described in Section II. The dataset, composed of the aircraft states and the complexity matrices, is divided into training and validation sets. The encoder-decoder LSTM model is trained (validated) with the training (validation) sets.

more uniform distribution. The $\mathbf{H}_t$ matrices are then flattened as n^2 vectors to be considered by the machine learning model.

At a given time t, the input vector $\mathbf{x}_t$ is built by concatenating all the aircraft states currently in the airspace, ordered by increasing longitude (and by increasing latitude if the longitude is the same). This means that a given flight will not occupy the same position in $\mathbf{x}_t$ for every t. However, a given position in $\mathbf{x}_t$ will be occupied by aircraft flying at roughly the same geographical coordinates such that the positional information is preserved in the structure of the inputs. To get a fixed-size input vector, the dimension of $\mathbf{x}_t$ is set to the maximum number of simultaneous aircraft states in the dataset multiplied by 8 (the length of an aircraft state). If the number of states is lower than the maximum, the remaining elements of $\mathbf{x}_t$ are padded with zeros.

We can know describe the training set as pairs of input sequences $\mathbf{X}_{t_0}$ and output sequences $\mathbf{Y}_{t_0}$ where $\mathbf{X}_{t_0} = \begin{bmatrix}\mathbf{x}_{t_0} & \cdots & \mathbf{x}_{t_0+t_{in}-1}\end{bmatrix}$ and $\mathbf{Y}_{t_0} = \begin{bmatrix}\mathbf{H}_{t_0+t_{in}} & \cdots & \mathbf{H}_{t_0+t_{in}+t_{pred}-1}\end{bmatrix}$, where t_{in} is the length of the input sequence, while t_{pred} is the time horizon of the prediction (length of the output sequence). In the following, the dataset will be randomly divided between training examples (90% of the dataset) and validation examples (10% of the dataset).

B. Architecture of the model

The proposed encoder-decoder LSTM model predicts a sequence of complexity matrices using a sequence of concatenated aircraft states as input (see Figure 6). Let us first consider the encoder network. The input sequence is fed to one or several 1-D convolutional layers. The output of the 1-D convolutional layers is fed to several LSTM layers. The hidden state of the last LSTM layer is the *encoding sequence*, which is the only encoder information that will be passed to the decoder network. The decoder network is composed of several LSTM layers (the hidden states of these layers are initialized with the encoding sequence) followed

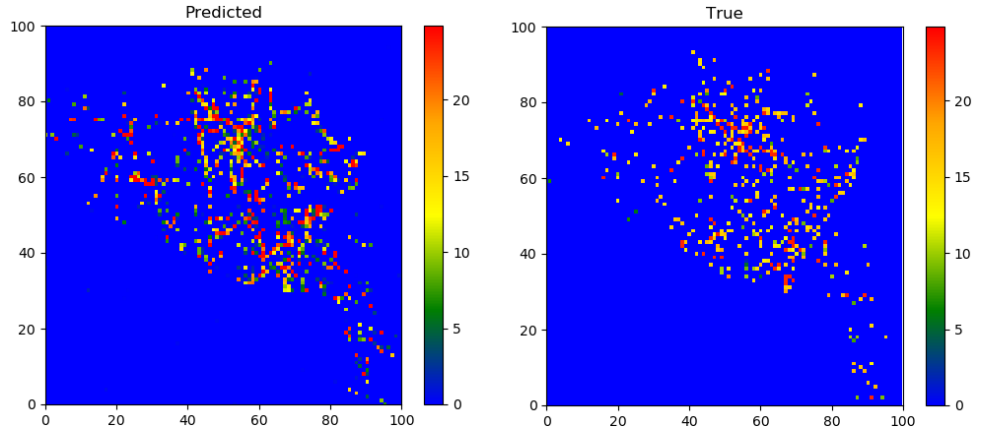


Figure 8: A heatmap of the complexity matrices of one sequence of trajectories from the training set. The longitude is represented horizontally and the latitude vertically. True values after 40 minutes (left) and predicted values (right).

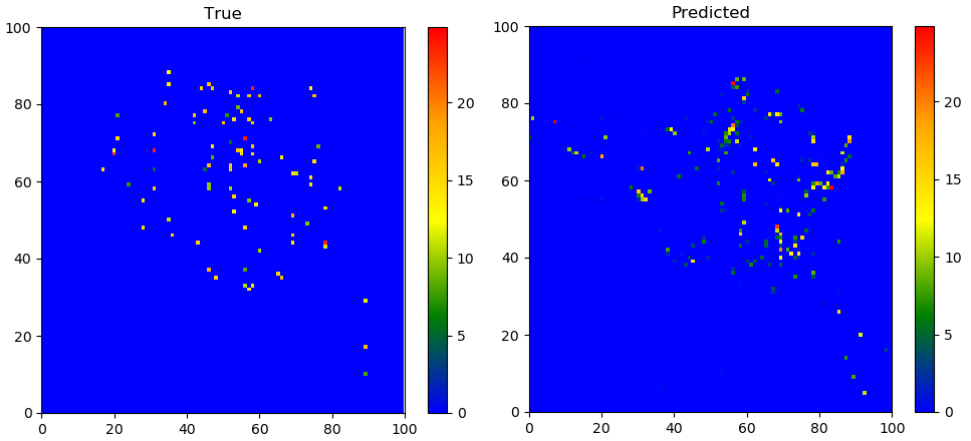


Figure 9: A heatmap of the complexity matrices of one sequence of trajectories from the validation set. The longitude is represented horizontally and the latitude vertically. True values after 40 minutes (left) and predicted values (right)

by several dense layers, which output a vector of dimension n^2 corresponding to the matrix of complexity values.

Another characteristic of the proposed model is that the decoder network is implementing a technique called "teacher forcing". This means that, during training, the decoder network takes as an additional input the true output sequence, but shifted by one timestep. In other words, the decoder network also receives as input the output it should have predicted at the previous timestep. During the inference process, the decoder network simply considers the prediction it has made at the previous timestep.

Several variations of the dataset and the network are tested in this work. First, we consider the dataset where the complexity matrices $\mathbf{H}_t$ are smoothed using a Gaussian kernel. Gaussian smoothing was performed using a 3×3 approximate Gaussian kernel $\frac{1}{16}\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix}$ to smooth the values of the complexity metric across the airspace. Smoothed data is then used to train the model. We then investigated the effect of the length of aircraft trajectory in the computation of the linear dynamical system metric. Specifically, we added p points corresponding to future positions and p points corresponding to past position of each aircraft. This experiment aims at modelling the uncertainty of the aircraft's positions along the curvilinear abscissa. In particular, we studied whether the choice of the parameter p affects the performances of the model. The entire framework is summarized in Figure 7.

The hyperparameters of the implemented model are as follows. The convolutional layer has a kernel of width $d = 3$ and $f = 512$ filters. The encoder's LSTM layer has a hidden dimension of 128. The number of time steps for the encoder network is fixed at $t_{in} = 160$. Since one time step is 5 seconds, this corresponds to 40 minutes, i.e., the model has access to the last 40 minutes of traffic to compute its prediction. The decoder's LSTM layer has a hidden dimension of 128 and is followed by one dense layer with output dimension $10,000$ (since the complexity matrix is a 100×100). The dataset (input and output) is linearly normalized between 0 and 1. We use the ReLU activation function for the last dense layer. The sequence predicted by the decoder network is set to $t_{pred} = 160$, which corresponds to the next 40 minutes of complexity values. During end-to-end model training, we used the Adam optimizer [39] with a learning rate $\epsilon = 10^{-3}$, and hyperparameters $\rho_1 = 0.9$, and $\rho_2 = 0.999$. The loss

function is the Mean Squared Error defined as

$$MSE(\hat{\mathbf{H}}, \mathbf{H}) = \frac{1}{n^2} \sum_{i=1}^{n^2} (\hat{h}_i - h_i)^2. \tag{10}$$

The mini-batch size is set to 128 and the model is trained over 100 epochs. The $8,011$ simulated trajectories of our dataset are spread across $3,025$ time steps. Since our model requires pairs of sequences of respective lengths t_{in} and t_{pred}, we built $3,025 - t_{in} - t_{pred} = 2,705$ sequences. Among these $2,705$ sequences, $2,434$ were part of the training set, and 271 were used to validate the model.

V. Results and Discussion

Figure 8 shows the true and predicted complexity matrices, as heat maps, for a sequence from the training set. The horizontal axis corresponds to the longitude and the vertical axis is the latitude. The color indicates the complexity value associated to each area of the airspace. The complexity values have been rescaled to their original distribution. We can visually check that the model is able to predict the future complexity of the airspace, 40 minutes ahead of time, with a reasonable accuracy. Figure 9 shows the true and predicted complexity heat maps for a validation sequence, i.e., that the network did not encounter during training. It can be seen that the model is less accurate with validation examples. This may be due to overfitting, although the validation loss curve seems to converge to the same value as the training loss. Several regularization methods were tested, including $L2$ regularization, dropout, and Batch Normalization, but they significantly decreased the overall performance. The small size of the dataset (2,705 sequences) may explain why the model struggles to generalize, as well as the absence of the vertical information in the target output.

To quantify the performance of the model, we used the absolute error between the predicted density and the true density over the validation set. Given that the validation set contains 271 examples, each example containing $10,000$ density values, we obtain a total of $2,710,000$ values of absolute errors, among which $182,982$ errors are nonzero (which is expected since the airspace is mostly empty). After removing the points with zero error, we plot the histogram of the non-zero absolute errors for three variants of the model, along with the corresponding Mean Absolute Error (MAE). Figures 10a and 10b show the histograms for the model with $p = 10$ (the same model used earlier in this Section) and $p = 2$, respectively. Figure 10c shows the distribution of the non-zero errors when the output matrices are smoothed with a Gaussian kernel. Comparing the two variants of the linear dynamical system metric (with $p = 2$ or $p = 10$), the two distributions feature two modes other than zero. The distributions have a decreasing overall trend but with a certain number of errors around or above 1. However, with $p = 2$, the errors are spread across a smaller range and the MAE is smaller. There is also a smaller number of non-zero errors overall. This result is coherent with the literature findings that smaller sequences are somewhat easier to predict that longer ones, notably due to known gradient vanishing or exploding issues. When the complexity measures are smoothed, the MAE is five times smaller than non-smoothed output, reaching a value of 0.08. The total number of non-zero errors is higher, which is due to the fact that there are more non-empty areas among the smoothed outputs. The prediction task with a smoothed dataset seems to be easier for the encoder-decoder LSTM model.

VI. Conclusion

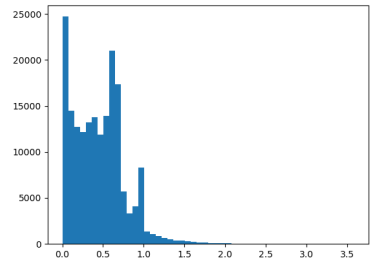
In this paper, we tackled the problem of predicting air traffic congested areas. This objective was achieved by combining Air Traffic Flow Prediction methods with a more rigorous definition of the congestion relying on an intrinsic complexity metric based on a linear dynamical system. The proposed model is an Encoder-Decoder LSTM network that uses a sequence of (discrete) aircraft states to predict the complexity of the air traffic in all areas of an airspace, in a time horizon of 40 minutes. The model achieves better performance when the output of the dataset is Gaussian smoothed prior to training, achieving a Mean Absolute Error of 0.08 on the validation set.

The impact of this work is significant. The Encoder-Decoder LSTM model can provide information to the Flow Management Position operator about the upcoming hot spots. Then, the FMP operator can decide to regulate flights involved in the hot spots or modify the configuration of the control room in order to avoid the formation of the predicted hot spots. The model can also be fed as an input to a mitigation system, which would suggest actions on trajectories (e.g., changing heading, speed, or altitude) to prevent the formation of congested areas. The ultimate goal is to bridge the gap between the Flow Management Position's strategic planning and the controller's tactical monitoring as part of the Extended ATC Planning concept. This work is an important step towards suggesting upstream strategies to remove the hot spots that are too local and unpredictable at the strategic time frame, but still generate complex situations at the tactical level.

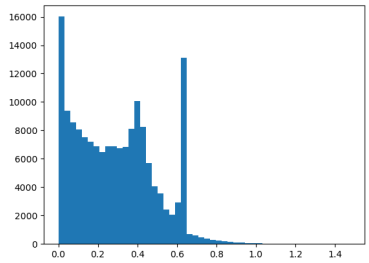
Future research directions include the quantification of the uncertainty in the predictions. The air transportation system is a highly dynamic system with inherent uncertainty and stochasticity. In order to be accepted in an operational context, the model should be able to detect if its predictions are no longer reliable and inform the FMP operator. The model should also be further refined and tested with larger datasets as well as using historical data. Moreover, the model can be improved by incorporating weather information such as wind and temperature which constitute the main causes of trajectory prediction uncertainties. Another possible approach consists in predicting the probability of formation of hot spots above pre-determined waypoints rather than predicting the precise locations of these hot spots.

Acknowledgment

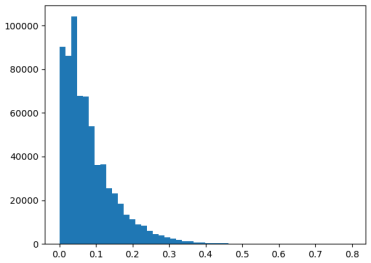
The authors would like to thank Ian Levitt and Andrew Cheng from the US. Federal Aviation Administration (FAA) William J. Hughes Technical Center, and Elsa Birman from the CRNA Est (Reims ACC) for their valuable input concerning ATM research and operational expertise. Nidhal C.



(a) with trajectory extension of $p = 10$ states before and after the reference state (to take into account uncertainty in the measurements). No smoothing applied. $182,982$ non-zero values with MAE = 0.4



(b) with trajectory extension of $p = 2$ states before and after the reference state. No smoothing applied. $161,130$ values with MAE = 0.3



(c) with no trajectory extension and Gaussian smoothing. $678,240$ values with MAE = 0.08

Figure 10: Distribution of non-zero absolute errors between prediction and ground truth over the validation set.

Bouaynaya is funded by the US. National Science Foundation grant NSF ECCS-1903466.

REFERENCES

[1] ICAO, *Long-Term Traffic Forecasts, Passenger and Cargo*, 2018.
[2] SESAR-Joint-Undertaking, *Airspace Architecture Study*, 2019.
[3] M. Prandini, L. Piroddi, S. Puechmorel, and S. L. Brázdilová, "Toward Air Traffic Complexity Assessment in New Generation Air Traffic Management Systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 3, pp. 809–818, 2011.
[4] E. M. Pfleiderer, C. A. Manning, and S. Goldman, "Relationship of complexity factor ratings with operational errors," *Oklahoma City (OK): Civil Aerospace Medical Institute, FAA*, 2007.
[5] SESAR-Joint-Undertaking, *SESAR Solutions Catalogue, Third Edition*, p. 61, 2019.
[6] J. Tang, "Conflict Detection and Resolution for Civil Aviation: A Literature Survey," *IEEE Aerospace and Electronic Systems Magazine*, vol. 34, no. 10, pp. 20–35, 2019.
[7] S. Kauppinen, C. Brain, and M. Moore, "European medium-term conflict detection field trials," *AIAA/IEEE Digital Avionics Systems Conference - Proceedings*, vol. 1, pp. 1–12, 2002.
[8] B. Sridhar, T. Soni, K. Sheth, and G. B. Chatterji, "Aggregate Flow Model for Air-Traffic Management," *Journal of Guidance, Control, and Dynamics*, vol. 29, no. 4, 2006.
[9] Y. Cao and D. Sun, "Link Transmission Model for Air Traffic Flow Management," *Journal of Guidance, Control, and Dynamics*, vol. 34, no. 5, pp. 1342–1351, 2011.
[10] D. Chen, M. Hu, Y. Ma, and J. Yin, "A network-based dynamic air traffic flow model for short-term en route traffic prediction," *Journal of Advanced Transportation*, no. 50, pp. 2174–2192, 2016.
[11] Y. Lin, J.-w. Zhang, and H. Liu, "Deep learning based short-term air traffic flow prediction considering temporal–spatial correlation," *Aerospace Science and Technology*, vol. 93, 2019.
[12] H. Liu, Y. Lin, Z. Chen, D. Guo, J. Zhang, and H. Jing, "Research on the Air Traffic Flow Prediction Using a Deep Learning Approach," *IEEE Access*, vol. 7, pp. 148019–148030, 2019.
[13] G. Gui, Z. Zhou, and J. Wang, "Machine Learning Aided Air Traffic Flow Analysis Based on Aviation Big Data," *IEEE Transactions On Vehicular Technology*, vol. 69, no. 5, pp. 4817–4826, 2020.
[14] I. V. Laudeman, S. G. Shelden, R. Branstrom, and C. L. Brasil, "Dynamic density: An air traffic management metric," Tech. Rep. April 1998, NASA, Moffett Field, CA, 1998.
[15] B. Sridhar, K. S. Sheth, and S. Grabbe, "Airspace Complexity and its Application in Air Traffic Management," in *2nd USA/EUROPE Air Traffic Management R&D Seminar*, pp. 1–9, 1998.
[16] B. Hilburn, "Cognitive Complexity in Air Traffic Control: A Literature Review," *Eurocontrol*, vol. 12, no. 13, pp. 93–129, 2004.
[17] G. B. Chatterji and B. Sridhar, "Neural network based air traffic controller workload prediction," *Proceedings of the American Control Conference*, vol. 4, pp. 2620–2624, 1999.
[18] G. B. Chatterji and B. Sridhar, "Measures for air traffic controller workload prediction," *1st AIAA, Aircraft, Technology Integration, and Operations Forum*, 2001.
[19] P. Kopardekar and S. Magyarits, "Dynamic density: Measuring and predicting sector complexity," *AIAA/IEEE Digital Avionics Systems Conference - Proceedings*, vol. 1, pp. 1–9, 2002.
[20] A. J. Masalonis, M. B. Callaham, and C. R. Wanke, "Dynamic density and complexity metrics for realtime traffic flow management: Quantitative analysis of complexity indicators," *5th USA/Europe Air Traffic Management R & D Seminar, Budapest, Hungary*, vol. 139, 2003.
[21] P. Flener, J. Pearson, M. Ågren, C. Garcia-Avello, M. Çeliktin, and S. Dissing, "Air-traffic complexity resolution in multi-sector planning," *Journal of Air Transport Management*, vol. 13, no. 6, pp. 323–328, 2007.
[22] M. A. Ishutkina and E. Feron, "Describing Air Traffic Complexity Using Mathematical Programming," in *AIAA 5th Aviation, Technology, Integration, and Operations Conference*, 2005.
[23] D. Delahaye and S. Puechmorel, "Air traffic complexity based on dynamical systems," in *49th IEEE Conference on Decision and Control*, pp. 2069–2074, 2010.
[24] B. G. Nguyen, *Classification in functional spaces using the BV norm with applications to ophthalmologic images and air traffic complexity*. PhD thesis, Université de Toulouse 3 Paul Sabatier, 2014.
[25] A. Garcia, D. Delahaye, and M. Soler, "Air traffic complexity map based on linear dynamical systems," *Unpublished*, 2020.
[26] L. Lin, Y. Saad, and C. Yand, "Approximating Spectral Densities of Large Matrices," *SIAM Review*, vol. 58, no. 1, pp. 34–65, 2016.
[27] P. Drineas and M. W. Mahoney, "Lectures on randomized numerical linear algebra," in *The Mathematics of Data, IAS/Park City Mathematics Series*, pp. 1–48, IAS/Park City Mathematics SeriesAMS/IAS/SIAM, 2018.
[28] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems*, 2014.
[29] G. Saon, G. Kurata, T. Sercu, K. Audhkhasi, S. Thomas, D. Dimitriadis, X. Cui, B. Ramabhadran, M. Picheny, L. L. Lim, B. Roomi, and P. Hall, "English conversational telephone speech recognition by humans and machines," *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, vol. 2017-August, pp. 132–136, 2017.
[30] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 07-12-June, pp. 3156–3164, 2015.
[31] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks.," in *ICML (3)*, vol. 28 of *JMLR Workshop and Conference Proceedings*, pp. 1310–1318, JMLR.org, 2013.
[32] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[33] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pp. 1724–1734, 2014.
[34] H. Sak, A. Senior, and F. Beaufays, "Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition," *arXiv e-prints*, 2014.
[35] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent Neural Network Regularization," *arXiv e-prints*, pp. 1–8, 2014.
[36] R. Prabhavalkar, K. Rao, T. N. Sainath, B. Li, L. Johnson, and N. Jaitly, "A Comparison of sequence-to-sequence models for speech recognition," *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, vol. 2017-August, pp. 939–943, 2017.
[37] S. Venugopalan, M. Rohrbach, J. Donahue, R. Mooney, T. Darrell, and K. Saenko, "Sequence to sequence - Video to text," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2015 International Conference on Computer Vision, ICCV 2015, pp. 4534–4542, 2015.
[38] A. Graves, "Sequence Transduction with Recurrent Neural Networks," *arXiv preprint arXiv:1211.3711*, 2012.
[39] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–15, 2015.