**Les Cahiers du GERAD** ISSN: 0711–2440

# Flight-connection prediction for airline crew scheduling to construct initial clusters for OR optimizer

Y. Yaakoubi,
S. Lacoste-Julien, F. Soumis

G–2019–26

April 2019



La publication de ces rapports de recherche est rendue possible grâce au soutien de HEC Montréal, Polytechnique Montréal, Université McGill, Université du Québec à Montréal, ainsi que du Fonds de recherche du Québec – Nature et technologies.

Dépôt légal – Bibliothèque et Archives nationales du Québec, 2019
– Bibliothèque et Archives Canada, 2019

The publication of these research reports is made possible thanks to the support of HEC Montréal, Polytechnique Montréal, McGill University, Université du Québec à Montréal, as well as the Fonds de recherche du Québec – Nature et technologies.

Legal deposit – Bibliothèque et Archives nationales du Québec, 2019
– Library and Archives Canada, 2019

**GERAD** HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

**Tél. : 514 340-6053**
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

# Flight-connection prediction for airline crew scheduling to construct initial clusters for OR optimizer

**Yassine Yaakoubi** [a,c]
**Simon Lacoste-Julien** [b,d]
**François Soumis** [a,c]

[a] *Department of Mathematics and Industrial Engineering, Polytechnique Montréal (Québec) Canada, H3C 3A7*

[b] *Department of Computer Science and Operations Research, Université de Montréal (Québec) Canada, H3C 3J7*

[c] *GERAD, Montréal (Québec), Canada, H3T 2A7*

[d] *MILA, Montréal (Québec), Canada, H2S 3H1*

yassine.yaakoubi@polymtl.ca
slacoste@iro.umontreal.ca
francois.soumis@gerad.ca

**April 2019**
**Les Cahiers du GERAD**
**G–2019–26**

**Abstract:** We present a case study of using machine learning classification algorithms to initialize a large scale commercial operations research solver (GENCOL) in the context of the airline crew pairing problem, where small savings of as little as 1% translate to increasing annual revenue by millions of dollars in a large airline. We focus on the problem of predicting the next connecting flight of a crew, framed as a multiclass classification problem trained from historical data, and design an adapted neural network approach that achieves high accuracy (99.7% overall or 82.5% on harder instances). We demonstrate the usefulness of our approach by using simple heuristics to combine the flight-connection predictions to form initial crew-pairing clusters that can be fed in the GENCOL solver, yielding a 10x speed improvement and up to 0.2% cost saving.

**Keywords:** Next flight prediction, crew pairing, crew scheduling, neural networks, column generation, constraint aggregation

**Acknowledgments:** This work was supported by IVADO and a Collaborative Research and Development Grant from the Natural Sciences and Engineering Research Council of Canada (NSERC) and AD OPT, a division of Kronos Inc. The authors would like to thank these organizations for their support and confidence.

# 1 Introduction

Airlines need to construct crew pairings to cover their flights. A pairing is a sequence of flights starting and finishing at a base and satisfying complex collective agreement constraints. For major airlines which handle more than 10k flights on a weekly basis, this becomes an important and difficult problem to solve. Efficient solutions are required since savings as low as 1% represent many dozens of millions saved every year. The complexity of the problem lies in the large number of possible pairings, and the selection of the set of pairings of minimal cost, which is a large integer programming problem impossible to solve with standard solvers (Elhallaoui et al., 2005; Kasirzadeh et al., 2017).

In our review of related work, we address some advanced optimization techniques that reduce the number of variables and the number of constraints to solve it. The main drawback of these techniques, however, is that they require days to compute, while airlines are often given all the scheduling data only a few days before having to build the schedule. The objective of this paper is to use machine learning (ML) techniques to improve the algorithmic efficiency and solve this problem in a more feasible time horizon. Unfortunately, solving the problem with ML alone seems out of reach. Learning from previous months' solutions, the structure of the 10k best **flight-connection variables**.[1] is a complex object to identify. For each flight, there are a large number of possible followers and very complex costs and constraints on the sequence of flights in a pairing. Finding these variables with high accuracy for the solution would require more data than what is available.

We propose to use ML to obtain in a matter of seconds good initial information to start an operations research (OR) optimizer, thus reducing the solution time by an order of magnitude. The information provided by ML does not need to be an entirely feasible solution (Elhallaoui et al., 2005). Consequently, it can contain some weak points, which the OR optimizer can fine tune into a feasible solution.

We investigate the performance of machine learning algorithms in Section 5 to solve the **flight-connection** prediction problem in which the goal is to predict the next flight that an airline crew should follow in their schedule with only partial information (framed as multiclass classification, cf. Section 3). We adapt a neural network solution in several ways to improve the accuracy of the predictions: proposing improved feature encodings of the input, reducing the number of classes to predict using domain-appropriate constraints, and testing solutions that can abstain to make a prediction (cf. Section 4).

We motivate the above problem in the context of solving the airline crew pairing problem. The idea is to exploit a large volume of flight data, that is, the existing solution counting thousands of flights over several months, and apply supervised learning algorithms to build a predictor for the flight-connection problem, in order to obtain good initial information to speed up an OR optimizer (called GENCOL) that needs to be run on new schedule data. Using our algorithm, we can process tens of thousands of flights, in contrast to current techniques reported in the literature. To the best of our knowledge, finding good initial information with ML algorithms in order to optimize an airline crew schedule of this magnitude has not been addressed before. As a proof of concept, we propose in Section 6 simple heuristics to form initial crew-pairing clusters that can be fed in the GENCOL solver, yielding a 10x speed improvement and up to 0.2% cost saving which would translate in millions of dollars saved for a large airline.

# 2 Related work

## 2.1 Machine learning for scheduling

Priore et al. (2014) provides an overview of the dynamic scheduling for manufacturing systems using machine learning techniques. The classical problem involving job scheduling is to allocate the time, for

[1] The following flight that a crew is going to perform after each flight

each job, for each specific machine, for satisfying a set of predefined constraints. The machine learning approach uses a set of previous system simulations (as training samples) in order to determine which rule is optimal for the current system state. The described approaches include a variety of techniques such as inductive learning, case-based reasoning, support vector machines, reinforcement learning, and other approaches. Unlike the dynamic scheduling for manufacturing systems, we do not have access to these predefined constraints, as each airline company has its own collective agreement and we would like to build a model that can be extended to *other* airline companies.

Other work has used neural networks (NNs) to solve scheduling problems. Wang et al. (2016) applies standard NN to predict the anticipated bus traffic, whereas Doherty and Mohammadian (2003) describes an application of NN to activity scheduling time horizon. These approaches are not used in combination with an OR optimizer, though, and the flight-connection problem contains a larger number of classes as well as more complicated constraints to satisfy.

## 2.2 Previous work on crew pairing

Those problems are known to be quite hard to solve. The complexity of such a problem is due to the number of variables and constraints involved in the problem definition. In short, they are large-scale integer programming problems. In the literature, there are two common methods frequently employed: (i) branch-and-price (B&P) (Freling et al., 2004; Günlük et al., 2005; Brunner and Bard, 2013); and (ii) Lagrangian relaxation (Ceria et al., 1998; Caprara et al., 1999; Li et al., 2009).

The most prevalent method since the 1990s has been the set covering problem with column generation inserted in branch-&-bound (see Desrochers and Soumis (1989)). This method, along with others, is described in a survey on crew pairing problems by Cohn and Barnhart (2003); see also Deveci and Demirel (2018) for a more recent survey. According to this latest survey, column generation was the most frequently used approach. We describe the literature in more details in Appendix A.1.

Column generation uses a main problem and a sub-problem. On the one hand, the main problem shall be referred to as the restricted master problem; here, restricted indicates that we are using a subset of all possible pairings. The objective function seeks to determine an optimal solution for the current columns. This problem is solved with the Simplex algorithm. On the other hand, the sub-problem generates promising pairings that are more likely to improve the solution of the main problem. In the sub-problem, the constraints of continuity and local constraint ensure that flights in the pairing will be chained in time and space. Furthermore, rules and collective agreements have to be met as well; examples include working hours per day, flight duration per day, and the number of days away. This problem is solved by dynamic programming.

To reduce the number of constraints considered simultaneously, the work reported by Elhallaoui et al. (2005) introduced a *dynamic constraint aggregation* technique that decreases the amount of the set partitioning constraints in the restricted master problem and reduces the solution time (see Appendix A.3). This is achieved by combining a few of them as per a correspondence relationship. The authors outlined a theoretical outline of the dynamic constraint aggregation algorithm. Range et al. (2014) describe that dynamic constraint aggregation begins with an initial aggregation partition. This technique aggregates in a single covering constraint each cluster of flights expected to be consecutive in the optimal solution. The algorithm modifies the clusters dynamically to reach an optimal solution if some expectations were wrong. In previous work, the initial clusters were constructed with heuristic rules based on the user's knowledge.

In this work, we use ML techniques to learn from previous solutions in order to construct clusters of flights. To produce such initial clusters, we ought to determine which flight-connection variables are likely to be equal to one. These clusters are not a feasible solution, but the dynamic constraint aggregation method will repair it with phase 1 and phase 2 of the simplex to reach a good feasible solution.

# 3 Problem setting

## 3.1 Crew pairing motivation

In our solution method for the overall crew pairing problem (cf. Section 6), we want to provide the OR optimizer with an initialization which uses a partition of the flights into clusters. Each cluster represents a sequence of flights with a strong probability to be consecutive in the same pairing in the solution. To construct each cluster, we will need the following:

- Information on where and when each crew begins the pairing. This makes it possible to identify whether a flight is the beginning of a pairing;
- For each incoming flight, predict what the crew is going to do next: layover, flight, or end of a pairing. If it is the second case (flight), then we further predict which flight the crew will undertake.

Since the end of a pairing depends on the maximum number of days permitted by the airline company, we will solely rely on this number as a hyperparameter. In other words, there is no need to predict the end of the pairing. Therefore, with regards to the pairing construction, we propose to decompose the second task into two sub-tasks. The first is predicting whether the crew will make a layover; the second is predicting the next flight, under the assumption that the crew will always take another flight.

On the one hand, layovers are highly correlated with the duration of the connection, although there is no fixed threshold for the number of hours a crew must stay in an airport to make a layover. On the other hand, predicting the next flight is a much more complicated prediction problem. Indeed, for each incoming flight, there may be hundreds of flights departing in the next hour and thousands of flight codes. We call this the **flight-connection problem** which is the focus of our ML approach described in the next sections.

## 3.2 Flight-connection prediction dataset

We obtained several months of historical crew pairing data from an anonymous airline company covering approximately 200 cities and tens of thousands of flights per month. We will transform this data to build a flight-connection prediction subproblem (a multiclass classification problem) where the goal is to predict the next flight that an airline crew should follow in their schedule given only partial information about the beginning of their schedule. To avoid error propagation through the whole sequence of flights (and because the OR solver is able to use partially correct plans), we will only use information about the crew's *incoming flight* (we will not use earlier information in the pairing) to predict the next flight.

The classification problem is thus the following: given the information about an incoming flight in a specific connecting city, choose among all the possible departing flights from this city (which can be restricted to the next 48 hours) the one that the crew should follow. These departing flights will be identified by their flight code (about $K$=2k possible flight codes in our dataset, $K$ will represent the number of classes). Different flights may share the same flight codes in some airline companies, as flights performed multiple times a week usually use the same flight code. Nevertheless, a flight is uniquely identified by its flight code, departing city and day; information which in practice can be deduced from the information about the incoming flight and our 48 hours window.

Each flight will give the following 5 features that we can use in our classification algorithm:

- City of origin and of destination (∼200 choices);
- Aircraft type (5 types);
- Duration of flight (in minutes)
- Time (with date) of arrival (for an incoming flight) or of departure (for a departing flight).

Our transformed dataset contains about 300k flight prediction examples (organized in different months), giving for each example these features for both the incoming flight as well as the possible departing flights from the same city. This dataset is used for training and validation. We keep a separate 10k flights for testing which are derived from a different weekly crew pairing schedule. Appendices B.1 and B.2 contain more details about the pre-processing.[2]

# 4 Algorithms

In this section, we describe the different methods and models that we investigated to solve the flight-connection problem.

## 4.1 Basic Neural Network (NN) model

As neural networks have shown great potential to extract meaningful features from complex inputs and obtain good accuracies through different classification tasks, we propose to use them and compare their results with classical machine learning methods. In both cases, the input contains information on the previous flight as well as the next departing flight. For more information on the features used in the input, see Appendix B.1.

We use a standard neural network with two dense hidden layers, each with 1,000 neurons and `relu` as the activation function. The output layer is another dense layer with $K$=2k neurons and `softmax` as the activation function. We use $K$ neurons in the output layer because we want to predict the flight code. We also use dropout (Srivastava et al., 2014) with a probability of 0.2 to prevent over-fitting as well as batch normalization (Ioffe and Szegedy, 2015) between the activation function and the dropout.

## 4.2 Adding an embedding layer

The city code and aircraft type features mentioned in Section 3.2 are categorical features which are simply treated as numeric values in our basic NN model. This means that cities with nearby codes are treated similarly by the NN even though the codes are somewhat arbitrary. A more meaningful encoding for the different categorical features is to use a one-hot encoding. By fully connecting each one-hot encoding of a categorical feature to a separate hidden layer, we basically get an embedding layer of dimension $d$ for this feature ($d$ is a hyperparameter). The $C \times d$ parameter matrix (where $C$ is the number of possible categorical values) represents the $d$-dimensional encoding for each of the $C$ values, and this encoding is learned during the NN training. The embedding layer approach thus learns a $d$-dimensional representation of each city, and one could explore which city is similar to another one in this space. By concatenating the embedding layer for each categorical features with the other numeric features (such as hours and minutes), we get an $n_d$-vector, where $n_d$ is the dimensionality of the representation of one flight that is fed into the NN. For the results given in Table 3, we feed the NN with the concatenation of the $n_d$ encoding of the incoming flight as well as the first next flight (using more next flights did not improve results but just took longer to learn).

## 4.3 Evaluation metrics

We now present the evaluation metrics that we will consider; the feasibility one will motivate further model refinements described next.

- Categorical accuracy:

$$\frac{1}{n}\sum_{i=1}^{n} \mathbb{1}\{\arg\max_j(y_{i,j}) \in \arg\max_j(p_{i,j})\} \tag{1}$$

  where $i$ is the sample index, $j$ refers to the class index, and $y_{i,\cdot}$ denotes the sample label encoded as a one-hot vector. $p_{i,j}$ is the classifier score for class $j$ for input $i$.

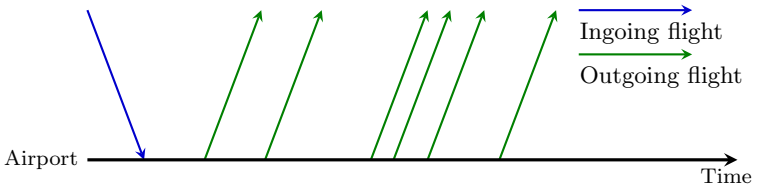
[2]The code and the dataset are available at: https://www.gerad.ca/fr/papers/G-2019-26

- top-$k$ categorical accuracy:

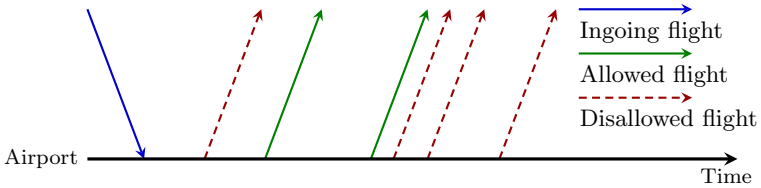
$$\frac{1}{n}\sum_{i=1}^{n}\mathbb{1}\{\arg\max_j(y_{i,j}) \in \text{top-}k_j(p_{i,j})\} \tag{2}$$

- Different aircraft accuracy: Throughout the months, in 88% to 95% of the instances, the crew arrives at an airport and follows the aircraft, i.e. takes the next departing flight that the same aircraft makes (note that the classifier cannot know which next flight uses the same aircraft though as we removed this information). To consider more difficult next-flight prediction instances, we therefore report the accuracy only on the instances where the crew changes aircraft.
- Feasibility: Given that the aircrew arrived at a specific airport at a given time, we can use *a priori* knowledge to define which flights are possible. For example, as shown in Figure 1, it is neither possible to make a flight that starts ten minutes after the arrival, nor it is possible five days later. Furthermore, it is rare that the type of aircraft changes between flights since each aircrew is formed to use one or two types of aircraft at most. The reader is referred to (Kasirzadeh et al., 2017) for further details on the likelihood of these scenarios. We describe in Appendix B.3 a simple short list of conditions that are always satisfied for the next flight taken by the crew, and we define a mask $m_{i,j} = 1$ only when flight $j$ satisfies these conditions for input $i$, and 0 otherwise. For the basic classifiers, we can evaluate the proportion of time they predict a feasible next flight with the "feasibility" metric:

$$\frac{1}{n}\sum_{i=1}^{n}\mathbb{1}\{\arg\max_j(p_{i,j}) \in \arg\max_j(m_{i,j})\} \tag{3}$$




(a) Different possibilities for an incoming flight




(b) Different **feasible** possibilities for an incoming flight

**Figure 1: Illustration of an incoming flight scenario**

## 4.4 Masked output neural network

To reduce the number of possible outputs, we can use the feasibility function $m_{i,j}$ to restrict our classifier to only predict a class among the feasible next flights for a specific incoming flight $i$. We can use such a mask to define a new output layer $L_m$, such that in the output layer, for the softmax function, we only take into account the probabilities for the feasible next flights.

## 4.5 Transformed classification problem

By using the feasibility information about next flights, we can design a more useful encoding for the output classes by normalizing their representation across different instances.

Therefore, for each incoming flight, we consider all departing flights in the next 48 hours as a set. Then, we consider only those that are feasible according to our masking constraints (defined in

Appendix B.3) to filter this set. Then, we sort these flights based on the time of departure and limit the maximal number of possible flights to 20, as it is sufficient in the airline industry. Thus, we predict the rank of the true label among that set with a 20-dimensional softmax output layer.

We use the embedding layer described earlier to construct a feature representation for each of these 20 flights. Then, we concatenate them to have a matrix $n_d \times 20$ input for the next layers, where $n_d$ is the embedded representation of information on one flight. Consequently, we do not only reduce the number of possible next flights but also construct a similarity-based input, where the neighboring factors have similar features, which in turn allows the usage of convolutional neural networks. The intuition here is that we can consider each next flight as a different time step, enabling the use of convolutional architecture across time (Hatami et al., 2018; Borovykh et al., 2017).

## 4.6 Post-processing step: Abstention methods

Instead of taking all the predictions into account, it is preferable in our case to discard a percentage of these predictions to enhance accuracy. We consider augmenting our classifier with an "abstain" option (Nadeem et al., 2009). Since the OR solver takes more time to break a connection in a cluster (links) than to build one, removing low confidence links (flights) using this "abstain" option can be advantageous. We observed that our solver is able to carry out the optimization problem much faster when taking into account 99% of our predictions instead of 100% (see Appendix A.3 for the OR solver description). We describe below three abstention methods that we investigated in our experiments (Section 5.3.4).

### 4.6.1 Abstain when low confidence

The simplest abstention technique would be to ignore the samples whose winning class confidence falls below a certain threshold. Indeed, Hendrycks and Gimpel (2017) show that the lower the softmax probability of the most probable class, the greater the likelihood that the prediction is incorrect. Therefore, we use a threshold for the probabilities given by the NN. We also consider an RBF kernel SVM, to which we give as input the probabilities given by the NN, as well as the rectangular input of the NN, where the categorical features are one-hot encoded. The binary label is 1 when the NN predicted correctly, 0 otherwise. Once trained, we put a threshold on the probabilities given by the SVM, exactly as we did for the probabilities given by the NN.

### 4.6.2 Abstain when high entropy

One issue with the approach of Hendrycks and Gimpel (2017) is that modern neural networks are often notoriously miscalibrated (Guo et al., 2017). Methods such as temperature scaling (Guo et al., 2017), can be applied to correct this miscalibration. Unfortunately, even when we consider using calibration for selective classification, we still encounter problems in the presence of class imbalance (which is the case in the flight-connection dataset). (Fumera et al., 2000) The idea here, in short, is that we choose not to predict if the predictive entropy of the output probability vector $p$ (given by $H(p) = -\sum_i p_i \log p_i$) is too high. (Hendrycks and Gimpel, 2017) Entropy is an uncertainty measure that takes into consideration the confidence scores of all the classes, in contrast to the confidence abstention method described in the previous paragraph which only takes into account the winning class confidence.

### 4.6.3 Estimate confidence with dropout

The NN literature considers various techniques to estimate the uncertainty of a prediction. In one technique, the prediction tasks can be carried $N$ times while applying dropout in the entire layers of the neural networks (Gal and Ghahramani, 2016), yielding $N$ probability vectors for each test sample. A rough estimate of certainty of prediction is obtained by computing the mean of these $N$ probability vectors and subtracting their component-wise estimated standard deviation (computed from the same $N$ vectors). This gives a rough lower bound on the certainty of our prediction. As in Section 4.6.1, if the maximum value of these confidences is too low, we decide to abstain.

# 5 Experiments

In this section, we illustrate the numerical examples employing the dataset described in subsection 3.2. Furthermore, this section describes the specifications of the employed hardware and software to realize the algorithm, in addition to the implementation details, and the main results of the experiments.

## 5.1 Hardware and software

The experiments were executed on a 40-core machine with 384 GB of memory. Each method is executed in an asynchronously parallel set up of 2-4 GPUs. That is, it can evaluate multiple models in parallel, with each model on a single GPU. When the evaluation of one model is completed, the methods can incorporate the result and immediately re-deploy the next job without waiting for the others to be finalized. We use four K80 (12 GB) GPUs with a time allocation of 10 hours. All algorithms were implemented in Python using Keras (Chollet et al., 2015) and Tensorflow (Abadi et al., 2015) libraries. For classical machine learning methods, we use Scikit-learn (Pedregosa et al., 2011).

## 5.2 Hyperparameter selection

### 5.2.1 Random search

As a first phase exploration of the hyperparameters space, we use random search (Bergstra and Bengio, 2012) with k-fold cross-validation on the training set to choose the best hyperparameters for each method we consider. We use different months for different folds (6 folds) to simulate the more realistic scenario where we make a prediction on a new time period. We keep the weekly problem of 10k flights for testing.

### 5.2.2 Bayesian optimization

After having identified the best predictor with random search, we refine further the choice of hyperparameters by using Bayesian optimization with k-fold cross-validation to measure the configuration quality. Bayesian optimization builds a probabilistic model of the function mapping from hyperparameter settings to the model performance and provide a systematic way to explore the space more efficiently (Hutter et al., 2011).

We optimize the hyperparameters listed in Table 1 (Dernoncourt and Lee, 2016) with an implementation of Gaussian process-based Bayesian optimization provided by the GPyOpt Python library version 1.2.1 (authors, 2016). In our approach, the optimization was initialized with 50 random search iterations, followed by up to 450 iterations of standard Gaussian process optimization. Here, the total return is used as the surrogate function and Expected Improvement ($\mathcal{EI}$) as the acquisition function.

**Table 1: Hyperparameters used in optimization**

| Parameters | Search space | Type |
|---|---|---|
| Optimizer | Adadelta; Adam; Adagrad; Rmsprop | Categorical |
| Learning rate | 0.001, 0.002, . . . , 0.01 | Float |
| Dimensions of the embeddings | 5, 10, 15, . . . , 50 | Integer |
| Number of dense layers | 1, 2, 3, 4, 5 | Integer |
| Neurons per layer | 100, 200, . . . , 1000 | Integer |
| Dropout rate | 0.1, 0.2, . . . , 0.9 | Float |
| Convolutional layers | 0, 1, 2, 3 | Integer |
| Filters $n$ | 50, 100, 250, 500, 1000 | Integer |
| Filter size h | 3,4,5 | Integer |

## 5.3 Results

### 5.3.1 Classical machine learning methods

Using features described in Appendix B.1, we considered several classical machine learning methods for our comparison, such as a logistic regression model and Multinomial Naive Bayes methods. As demonstrated in Table 2, we have non-linear decision boundaries. The data points are not easily separated by a single hyperplane, this is due to the fact, that the train classification accuracy is greater than test classification accuracy, in which we are slightly over-fitting the data.

**Table 2: Performance of classical machine learning algorithms**

| Method | Train accuracy (%) | Test accuracy (%) | Different aircraft accuracy (%) | Time (s) |
|---|---|---|---|---|
| Decision trees | 99.99 | 97.44 | 61.84 | 70 |
| One-Vs-Rest Decision trees | 99.99 | 92.66 | 66.53 | 320 |
| Logistic regression | 11.89 | 11.71 | 1.90 | 1990 |
| Multinomial Naive Bayes | 6.61 | 6.32 | 0.62 | 3 |

### 5.3.2 Neural networks

After testing with classical machine learning methods, we show results using NN. As speculated, we can see in Table 3 that the computational effort (CPU training time) of executing the model without the embedding layer is much higher, and provides a smaller accuracy rate.

When using the mask to guide the gradient descent during training jointly with the usage of the embedding layer, our model takes less than one-third of the training time and gives a better result put in comparison with the standalone embedding layer.

**Table 3: Comparison of results for different methods**

| Method | Test accuracy (%) | Different aircraft accuracy (%) | Top-3 accuracy (%) | Feasibility (%) | Time (s) |
|---|---|---|---|---|---|
| NN without embeddings | 88.03 | 60.52 | 98.95 | 92.25 | 4000 |
| NN + embeddings | 99.11 | 70.85 | 99.47 | 98.68 | 1850 |
| Masked output NN + embeddings | 99.20 | 71.24 | 100 | 100 | 500 |
| Transformed input without convolutional layers | 99.18 | 70.32 | 100 | 100 | 2300 |
| Transformed input with convolutional layers | 99.35 | 71.79 | 100 | 100 | 700 |
| Transformed input with convolutional layers (after GP) | 99.68 | 82.53 | 100 | 100 | 600 |

### 5.3.3 Gaussian process

We perform the Gaussian process on "Transformed input with convolutional layers" to search for the best configuration of hyperparameters. After a few iterations, we are able to get an accuracy of 99.35%. Then, random search boosts the total return very quickly up to 99.62% after 18 iterations and thus remains until the end of the random search cycle (iteration number 50). Using Gaussian process, we can show that we constantly improve our process of searching for the best architecture that maximizes the overall return. In our case, we stopped at iteration number 500 with the best architecture providing an accuracy of 99.68%. One should notice, that there was no limitation that prevents our algorithmic procedure from exploring, even more, the configuration space; in other words, the algorithm was stopped manually (see Appendix B.4 for more details). The final training was done with the best-identified architecture found by Bayesian optimization.

### 5.3.4 Abstention

Figure 2 shows the accuracy-abstention tradeoff curves for the abstention methods proposed in Section 4.6 on the best architecture on the test set giving a 99.62% accuracy initially; using these methods, we can discard some of our predictions and enhance the accuracy. For example, if we discard only 1% of our predictions, the accuracy increases from 99.62% to 99.90% abstaining when low confidence and to 99.94% estimating confidence with dropout. For the next Section 6, we will use a 1% rejection rate.

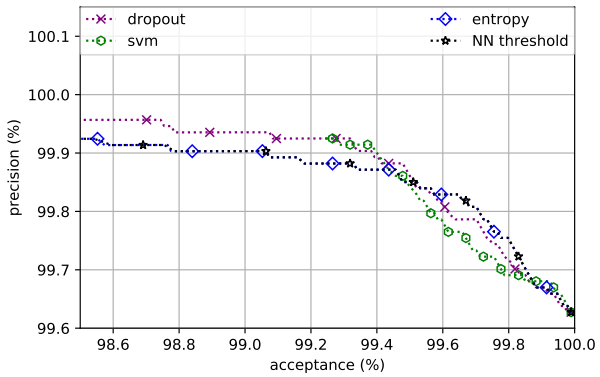



**Figure 2: Results of the different abstention methods. NN threshold and Entropy use probabilities given by the NN. SVM takes into consideration a confidence score given by a RBF-SVM. Dropout uses the lower confidence bound of the prediction confidence. For more information on these methods, see Section 4.6**

# 6 Integration of the flight-connection predictor into the crew pairing

GENCOL[3] (GENeration de COLonnes) is a commercial toolbox that produces real-world airline crew scheduling solutions. The toolbox uses column generation as an optimization strategy that allows a large number of variables to be considered in solving large-scale problems. As more than 20 airlines companies use GENCOL for their crew scheduling, we explore in this section whether our machine learning approach can yield gains on this state-of-the-art solver (through private communication with the company). We describe in more details the column generation approach in Appendix A.2.

Upon the finalization of the flights-connection prediction model, we can use the same architecture to solve two other prediction problems on the test set (10k flights): (i) predict if each of the scheduled flights is the beginning of a pairing or not; and (ii) predict whether each flight is performed after a layover or not. In reality, the three predictors share the same representation. To solve these independent classification problems, we sum the three prediction problems' cross-entropy losses when learning, therefore performing a multi-output classification.

We use a greedy heuristic to build the crew pairing. Specifically, we consider each flight that the model predicts at the beginning of a pairing as a first flight. Given this incoming flight, we predict whether the crew is making a layover or not. In both cases, we consider the incoming flight and predict the next one. The pairing ends when the crew has arrived at the base and when the maximal number of days permitted per pairing is approached.

We can use the above heuristic to construct a weekly solution for the testing data, obtaining a crew pairing that can be fed as an initial cluster for the GENCOL solver. Unfortunately, if one flight in the pairing is poorly predicted, as the flights are predefined, the crew will finish its pairing away from the base. Therefore, we consider different heuristics to correct the pairings:

[3] http://www.ad-opt.com/optimization/why-optimization/column-generation/

- **Heuristic 1:** includes pairings where the crews finish away from the base, but it erases what occurs after the last time the crew arrives at the base, or deletes the whole pairing if it never passes to the base again.
- **Heuristic 2:** deletes all pairings where the crews finish away from the base;
- **Heuristic 3:** like heuristic 1, erases what occurs after the last time the crew arrives at the base but also cuts the rotation into smaller ones if we pass multiple times at the base.

As a baseline, we consider a benchmark called "GENCOL init." which is a standard initial solution approach obtained with the GENCOL solver. In this approach, the weekly problem is solved by a "rolling time horizon" approach with one-day windows. This means that the week is divided into overlapping time slices of equal length (slightly more than one day). Then a solution is constructed greedily in chronological order by solving the problem *restricted* to each time slice sequentially, taking into account the solution of the previous slice through additional constraints.

We can feed the "GENCOL init." solution to GENCOL as an initial cluster for the constraint aggregation approach to further improve the solution globally, obtaining a solution that we call "GENCOL-DCA from GENCOL init." in Table 4. For more information on the constraint aggregation approach, see Appendix A.3.

As we can see in Table 4, two of the three heuristics that we propose outperform the benchmark as well as the solution from GENCOL-DCA with GENCOL initialization, and we can conclude that for the pairings that finish away from the base, it is better to allow the solver to cover the flights than to propose smaller pairings. Therefore, instead of performing the optimization process for 40 hours to obtain GENCOL init. and then for another 5 hours to obtain GENCOL-DCA, our proposed method gives better costs after a few seconds to predict the flight connections, and optimized results after five to six hours.

**Table 4: Final crew pairing costs after running GENCOL-DCA with different initialization clusters**

| Approach | Savings (%) | Time of execution |
|---|---|---|
| GENCOL init. | 0 | 40:00 |
| GENCOL-DCA from GENCOL init. | 1.21 | 05:00 |
| GENCOL-DCA from heuristic 1 init. | 1.45 | 06:30 |
| GENCOL-DCA from heuristic 2 init. | 1.38 | 05:00 |
| GENCOL-DCA from heuristic 3 init. | 0.97 | 06:30 |

We note that the heuristics that we propose to build crew pairings are still limited, even though they do give better costs in two of the three cases. Further study and experimentation are required to explore other heuristics to construct the crew pairings. Further study is also needed to solve *monthly* problems by rolling horizon with one-week windows. The fast machine learning predictor will permit to construct in a few seconds clusters for each window of a week, customized according to the flight schedule of this week. We expect improvements in particular when the monthly schedule is irregular from week to week. It is the case near every month: Christmas, Easter, Thanksgiving, National Holiday, Mother and Father days, big sports events, etc. It was out of the question to use five times 40 hours to produce customized clusters for each window with GENCOL init.

# 7 Conclusion

It is rather difficult to assign the crew workers to a range of tasks while taking into account all the variables and constraints associated with the process. In this paper, we incorporate different algorithms and methods, which include neural networks in conjunction with abstention techniques, such as dropout, in order to prevent overfitting, or masks to ensure guidance of the gradient descent. These networks allow our solver to achieve high accuracy for the flight-connection problem.

This paper introduces a new paradigm for solving a large combinatorial problem: "Start with ML — Finish with OR optimizer". This paradigm, first, use ML to learn from solutions of similar instances to produce predictions on some parts of the solution of the new instance. This information feeds the OR optimizer to finish the work taking account of the exact cost function and the complex constraints. This approach reduces significantly the solution time without losing on the quality of the solution. This new paradigm can be applied on many types of problems solved recursively.

# A Crew pairing problems and solution methods

We first provide background on the crew pairing problem and standard approaches to solve it. We then describe the most frequently used approach Deveci and Demirel (2018): Column Generation. Finally, we present the state-of-the-art dynamic constraint aggregation approach which improves upon column generation by reducing the number of constraints in the problem and permits to solve large-scale pairing problems. The commercial solver GENCOL that we use in our experiments uses both column generation and the constraint aggregation method.

## A.1 Crew pairing problem

According to Desaulniers et al. (1997), for each category of the crew and each type of aircraft fleet, the construction problem of crew pairing finds a set of pairings at minimal cost so that each planned flight is performed by a crew. The methodology for solving the problem depends on the size of the airline's network structure , rules, collective agreements, and cost structure (Yen and Birge, 2006).

Three time-horizons are generally studied: a daily, a weekly and a monthly horizon (Brunner and Bard, 2013). The daily problem assumes that the flights are identical, or relatively quite similar, for every day of the planning horizon. The problem also assumes that minimum cost pairings are generated for flights scheduled for a day. A cyclic solution is produced; that is to say, the number of crews present in every city in the evening is the same as in the morning. Accordingly, the weekly and monthly solutions can be produced by juxtaposing the daily solution. When the schedules of flights are not exactly the same every day, the pairings that do not fit with the flight rules are removed, as they no longer apply and can be re-optimized to create new pairings covering flights that are not covered previously. The weekly problem assumes that flights are identical (or somewhat similar) each week, and the pairing problem is solved for scheduled flights for a typical week.

In the literature, the crew pairing problem has been modeled as a set covering or a set partitioning problem, in which each column is a constraint associated with each flight and all feasible pairings are treated as variables (Desaulniers et al., 1997; Qiu, 2012; Kasirzadeh et al., 2017). This model contains millions of columns that thus do not fit in computer memory. Anbil et al. (1992) introduced a comprehensive method with a cost-minimization goal. In this collaborative research between American Airlines Decision Technologies and IBM, numerous possible pairings (columns) were generated as an a priori and a considerable amount of them is fed into the *linear program* solver. At every repetition, several of the non-basic pairs are rejected and new pairs are being inserted. The procedure keeps its execution until all the pairs have been taken into consideration Qiu (2012).

Current research in the literature indicates that heuristic algorithms have three main shortcomings. Firstly, they do not consider all scheduled flights simultaneously and must perform many iterations before they can achieve a solution with a high-quality Baker (1981). Secondly, the algorithms do not consider all viable pairings (Bianco et al., 1992). Thirdly, the divergence from the optimal solution is significant, resulting in new solutions which can be far from the global optimal solution (Voß, 1999). As a result, more sophisticated approaches have been proposed over the years. Scholars introduced the column generation method containing only a subset of the columns for this problem (Desrochers and Soumis, 1989); as they solved a set covering problem with a Simplex algorithm and generated columns by solving a shortest path problem.

## A.2 Column generation method

Column generation is an iterative method that solves, for each iteration, a restricted master problem and one or numerous sub-problems (Sadykov et al., 2016). Column generation is well-recognized for resolving *relaxation* of a (mixed) integer *linear program (LP)* in a range of applications, particularly in the fields of crew scheduling and vehicle-routing. For this type of applications, several problems can be framed as set-partitioning-problem-based integer problems (Bouarab et al., 2017). The reader is referred to Desrosiers et al. (1984) and Desrosiers et al. (1995), the founding papers of the domain.

When the number of columns is large, to the extent that they are no longer considered at the initial stage, column generation decomposition can be employed to solve a restricted master problem, that is, a linear program limited to a subgroup of columns (variables), resulting in a plausible and viable primal solution, along with dual variables. See (Bouarab et al., 2017) for further details. According to Bouarab et al. (2017), the dual vector is used by an oracle algorithm. This algorithm resolves one or several sub-problems, which in turn produces columns with negative reduced costs, and expands the restricted master problem. The procedure is executed until no more variables with negative reduced costs can be recognized.

## A.3 Constraint aggregation to speed up the column generation

The literature shows that column generation is frequently employed to solve problems comprising of set partitioning (SP) constraints, like crew-scheduling and vehicle routing problems. Column generation turns out to be ineffective when the number of constraints is huge and the columns have more than eight to twelve nonzero elements. This is due to the fact that resolving the restricted master problem needs excessive solution times resulting from great degeneracy (Bouarab et al., 2017).

In the event of the set partitioning problem, dynamic constraint aggregation and multi-phase constraint aggregation deal with degeneracy in a different way.

### A.3.1 Dynamic constraint aggregation

To address the aforementioned limitation of column generation method, Elhallaoui et al. (2005) introduced a dynamic constraint aggregation technique that decreases the number of set partitioning constraints in the restricted master problem by aggregating in a single constraint each cluster of tasks expected to be consecutive in the optimal solution. The algorithm modifies dynamically the clusters to reach the optimal solution if some expectations were wrong.

Elhallaoui et al. (2005) offered a theoretical outline of the dynamic constraint aggregation algorithm. On combined bus drivers and vehicle scheduling problems, they obtained time reduction of up to 90% on the master problem and up to 80 % on the total time. Benchimol et al. (2012) presented an application of this method on the crew assignment problem. In a few test circumstances, the entire solution time for the linear relaxation of the problem is decreased by around 80% compared to the normal column generation technique.

Range et al. (2014) discussed that dynamic constraint aggregation begins with an initial aggregation partition that may be a feasible solution or not. The initial tasks and constraint aggregation can be calculated by means of a heuristic solution or by solving an approximation of the problem. Due to its reduced size, this aggregated restricted master problem is clearly simpler to resolve, put in comparison with the non-aggregated one.

One major drawback of the dynamic constraint aggregation, as highlighted by Elhallaoui et al. (2005), is that it does not take the aggregation into consideration when resolving the sub-problem. Consequently, many variables may be produced, even if there exists a *compatible variable* with a negative reduced cost. When this occurs, the partitions which are incompatible are disaggregated, to permit these variables to enter the aggregated restricted master problem. This increases the size of the aggregated master problem and slows down the algorithm.

### A.3.2 Multi-phase dynamic constraint aggregation

To decrease the impact of the aforementioned shortcoming, Elhallaoui et al. (2010) offered an enhanced version of the dynamic constraint aggregation algorithm, named the "multi-phase dynamic constraint aggregation" algorithm. In this algorithm, a partial pricing strategy, favoring the compatible columns with the constraint aggregation is employed. In phase k, we add in the subproblem a constraint forbidding to generate a pairing breaking clusters more than k times. Similar to dynamic constraint aggregation, the algorithm begins with an initial aggregation partition that is produced from logical reasoning or a heuristic solution (Saddoune et al., 2012). Such an initial aggregation partition is proper if its clusters combine components that have a high likelihood of being in a column of an optimum linear relaxation solution.

The multi-phase dynamic constraint aggregation algorithm is comprised of two kinds of iteration: (i) major; and (ii) minor ones. A major iteration consequently contains a range of minor iterations and an update of the aggregation partition. In this update, the partition is disaggregated and aggregated in line with the situations that are achieved to produce a novel partition (Bouarab et al., 2017). A minor iteration begins by partly resolving the restricted master problem.

# B Reproducibility-related information

## B.1 The flight-connection prediction dataset

The data provided by the airline company consists of six months of crew-pairing data for approximately 200 cities and tens of thousands of flights per month. For each month, we are given two files. The first one considers engine rotations, whereas the second one contains the crew rotations. It is the second data file that is used in the learning phase and reconstructed during the testing phase.

The dataset considered in this paper consists of approximately 1k different source-destination pairs and **2k different flight codes**. Figure 3 shows the data format for a single pairing. Apart from this, we know the distance between the two airports is 761 Km, from which we can extrapolate the duration of the flight, based on an average aircraft flight speed, which is equivalent to roughly 1 hour and 26 minutes.

```
O T 1821 0 CC 1 CAT 1 () "a" "" _2_456_
2018/08/09 2018/09/03 X 2018/08/10 2018/08/20
2018/08/29
{
RPT ;
ABC05914  ORD 1 02:10 BUF;
ABC06161  BUF 1 09:50 ORD;
RLS;
}
```

**Figure 3: Data format for aircrew pairings. Here, this entry describes a pairing that takes place every Tuesday, Thursday, Friday, and Saturday (_2_456_) between August 9, 2018 and September 3, 2018, except for the dates: 08/10, 08/20, and 08/29. The pairing contains flights between Buffalo Niagara International Airport (BUF) and O'Hare International Airport (ORD).**

Once the dataset is pre-processed, the data shall contain the following information:

- **On the crew:** ID, departure or connecting city, the date, day, hour, and minute from which the crew is ready to begin the next flight, worked hours and minutes in this day, and lastly duration of the shift so far;
- **On the connection:** Duration in hours and minutes between the previous and the subsequent flights;
- **On the aircraft:** Type of the aircraft number corresponding to the aircraft pairing;

- **On the previous flight:** This includes code and duration;
- **On the next flights:** This covers code, next (arrival) city, the date, day, hour, minute and duration of each one of the flights to follow.

In total, there are 22 features one can use to predict the next flight. However, in practice, it is not always possible to know each of those features. For example, the time passed on the shift is unknown if one only considers the previous flight and attempts to predict the next one. This is basically the case in our particular problem. The same applies for the time worked during the day and the crew identification. Moreover, one can consider the previous flight as one of the features; however, one would use the time from which the crew is ready and the departure time of the next flight instead. In fact, if the method works on the latter, a benchmark can be obtained for new companies where the classification or the prediction does not necessarily depend on how the flight codes have been constructed. The benchmark, in this case, would use information on aircraft, time, and cities. This is quite typical for most airline companies.

Therefore, the prediction model for the next flight takes into consideration the following features:

- Engine and its rotation number.
- The city of origin (departure city).
- The connecting city (arrival city of the previous flight).
- The time taken by the previous flights (duration of the flight).
- The time from which the crew is ready to take a flight arriving at the connecting city, regarding, day of the year, week, hours, and minutes.
- Information on the next departing flight(s)

When presenting publicly an example of the dataset used during the phase, we begin by encoding all the available features. These features are encoded from categorical to indexes, which are values provided for by the dictionary. Eventually, we delete all code flights, city names, year and month details, and engine specifications.

Even more, our use of an embedding layer makes the rebuilding of the graph infeasible. Indeed, to find the airline company, one needs to use the numerical codes and information about flight pairs, bases reconstruction to count the number of flights per base, the number of days and the number of flights. These numerical codes are not retrievable from the dataset we intend to make public.

The embedding layer turns positive integers (indexes) into dense vectors of fixed size. Technological advancements in the natural language processing in Artificial Intelligence are based on the application of dense vectors for the representation of language relations, that is, embedded words for encoding connections between phrases and words (Mikolov et al., 2013).

Therefore, from our perspective, the flight code encoded to an integer is projected to a hyperspace having 25 dimensions or more. An advantage of this approach is to anonymize further the dataset, an important condition to release the dataset in the public domain from our company partners. The dataset contains the output of the embedding layer, therefore an embedded representation of the input. We get these representations during the last training epoch with a small tuned learning rate. This way, the representation of a city is not unique throughout the whole dataset. However, two representations of the same city are still similar.

## B.2 Features representation

For each incoming flight, we use the following features:

- Information on the aircraft: type of the aircraft (categorical)
- Information on the flight: duration of previous flight (numeric, in minutes) and time of arrival to the connecting city (Date, Day, Hour, Minutes)
- Information on the city: codes of the departure and the connecting city (both categorical)

Then, for each outgoing flight, we use the same features. The only difference is that we use the codes of the connecting and arrival city (instead of departure and connecting), the duration of next flight and time of departure from the connecting city. For each flight, we use a one-hot encoding for the categorical features. When using the embedding layer, we project each one of these features into an embedded representation. Then, we concatenate all embedded features with the numeric ones to create an $n_d$-vector where $n_d$ is the embedded representation of information of one flight.

## B.3 Filtering conditions to construct the mask

To construct a mask for feasible flights, we use the following conditions:

- The departure time of the next flight should follow the arrival time of the previous flight to the connecting city;
- The departure time of the next flight should not exceed 48 hours following the arrival time of the previous flight to the connecting city;
- The departure city of the next flight should be identical to the connecting city in the previous flight;
- The aircraft type should be the same. Indeed, crew scheduling is separable by crew category and aircraft type or family (Kasirzadeh et al., 2017).

It would have also been interesting to consider the aircraft pairing number, which indicates the routes of different aircraft. This, however, is not considered in our case.

## B.4 Hyperparameter space and search

We list below the hyperparameters that are considered during our initial random search phase for the different learning methods:

- Decision trees and One-Vs-Rest Decision trees:
  - criterion $\in$ { gini, entropy };
  - splitter $\in$ { best, random };
  - $max\_depth \in$ { 2, 5, 10, 15, 20 }.
- Logistic regression:
  - solver: $\in$ { sag, lbfgs };
  - C $\in$ { 0.001, 0.01, 0.1, 1, 10, 100 };
- Multinomial Naive Bayes:
  - $\alpha \in$ { 0.001, 0.01, 0.1, 1 };
  - fit_prior $\in$ { True, False };
- Neural networks : In addition to the hyperparameter space description in Table 1, we give more details about the optimizers used for training neural networks:
  - Adadelta: $\rho = 0.95$, decay $= 0$;
  - Adam: $\beta_1 = 0.9$, $\beta_1 = 0.999$, decay $= 0$;
  - Adagrad: decay $= 0$;
  - Rmsprop: $\rho = 0.9$, decay $= 0$.

For the best performing model (neural network with transformed input with convolutional layers, fifth line in Table 3), we further fine tuned the hyperparameter choice using Bayesian optimization as described in Section 5.3.3. To get an idea of the progress made during the Bayesian optimization, we plot the obtained cross-validation accuracy as a function of the number of iterations of the optimizer in Figure 4. After 300 iterations, the performance stagnates, and thus we can stop the exploration.

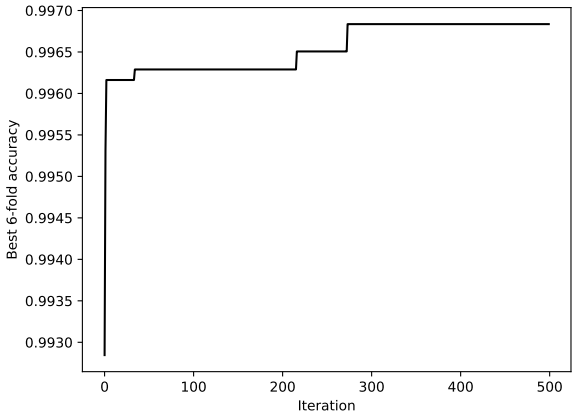



**Figure 4: The convergence plot of the Gaussian process**

# References

Martín Abadi et al. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

Ranga Anbil, Rajan Tanga, and Ellis L. Johnson. 1992. A global approach to crew-pairing optimization. IBM Systems Journal 31, 1, 71–78.

The GPyOpt authors. 2016. GPyOpt: A Bayesian Optimization framework in Python. http://github.com/SheffieldML/GPyOpt.

Edward K. Baker. 1981. Efficient heuristic algorithms for the weighted set covering problem. Computers & Operations Research 8, 4, 303–310.

Pascal Benchimol, Guy Desaulniers, and Jacques Desrosiers. 2012. Stabilized dynamic constraint aggregation for solving set partitioning problems. European Journal of Operational Research 223, 2, 360–371.

James Bergstra and Yoshua Bengio. 2012. Random Search for Hyper-parameter Optimization. J. Mach. Learn. Res. 13, Article 25 (Feb. 2012), 281–305 pages.

Lucio Bianco, Maurizio Bielli, Aristide Mingozzi, Salvatore Ricciardelli, and Massimo Spadoni. 1992. A heuristic procedure for the crew rostering problem. European journal of operational research 58, 2, 272–283.

Anastasia Borovykh, Sander Bohte, and Kees Oosterlee. 2017. Conditional time series forecasting with convolutional neural networks. In Lecture Notes in Computer Science/Lecture Notes in Artificial Intelligence. Springer, USA, 729–730.

Hocine Bouarab, Issmail El Hallaoui, Abdelmoutalib Metrane, and François Soumis. 2017. Dynamic constraint and variable aggregation in column generation. European Journal of Operational Research 262, 3, 835–850.

Jens O. Brunner and Jonathan F. Bard. 2013. Flexible weekly tour scheduling for postal service workers using a branch and price. Journal of Scheduling 16, 1, 129–149.

Alberto Caprara, Matteo Fischetti, and Paolo Toth. 1999. A heuristic method for the set covering problem. Operations research 47, 5, 730–743.

Ceria et al. 1998. A Lagrangian-based heuristic for large-scale set covering problems. Mathematical Programming 81, 2, 215–228.

François Chollet et al. 2015. Keras.

Amy Mainville Cohn and Cynthia Barnhart. 2003. Improving crew scheduling by incorporating key maintenance routing decisions. Operations Research 51, 3, 387–396.

Franck Dernoncourt and Ji Young Lee. 2016. Optimizing neural network hyperparameters with Gaussian processes for dialog act classification. 2016 IEEE Spoken Language Technology Workshop (SLT), 406–413.

Guy Desaulniers, Jacques Desrosiers, Yvan Dumas, S Marc, B Rioux, Marius M Solomon, and François Soumis. 1997. Crew pairing at air france. European journal of operational research 97, 2, 245–259.

Martin Desrochers and François Soumis. 1989. A Column Generation Approach to the Urban Transit Crew Scheduling Problem. Transportation Science 23, 1–13.

Jacques Desrosiers, Yvan Dumas, Marius M. Solomon, and François Soumis. 1995. Chapter 2 Time constrained routing and scheduling. In Handbooks in Operations Research and Management Science. Elsevier, Canada, 35–139.

Jacques Desrosiers, François Soumis, and Martin Desrochers. 1984. Routing with time windows by column generation. Networks 14, 4, 545–565.

Muhammet Deveci and Nihan Çetin Demirel. 2018. A survey of the literature on airline crew scheduling. Engineering Applications of Artificial Intelligence 74, 54–69.

Sean T. Doherty and Abolfazl Mohammadian. 2003. Application of Artificial Neural Network Models to Activity Scheduling Time Horizon. Transportation Research Record 1, 1854 (1 12 2003), 43–49.

Issmail Elhallaoui, Abdelmoutalib Metrane, François Soumis, and Guy Desaulniers. 2010. Multi-phase dynamic constraint aggregation for set partitioning type problems. Mathematical Programming 123, 2, 345–370.

Issmail Elhallaoui, Daniel Villeneuve, François Soumis, and Guy Desaulniers. 2005. Dynamic aggregation of set-partitioning constraints in column generation. Operations Research 53, 4, 632–645.

Richard Freling, Ramon M Lentink, and Albert PM Wagelmans. 2004. A decision support system for crew planning in passenger transportation using a flexible branch-and-price algorithm. Annals of Operations Research 127, 1-4, 203–222.

Giorgio Fumera, Fabio Roli, and Giorgio Giacinto. 2000. Reject Option with Multiple Thresholds. Pattern Recognition 33, 2099–2101.

Yarin Gal and Zoubin Ghahramani. 2016. Dropout As a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48 (ICML'16). JMLR.org, New York, NY, USA, Article 10, 1050–1059 pages.

Oktay Günlük, László Ladányi, and Sven De Vries. 2005. A branch-and-price algorithm and new test problems for spectrum auctions. Management Science 51, 3, 391–406.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. 2017. On Calibration of Modern Neural Networks. In Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research), Doina Precup and Yee Whye Teh (Eds.), Vol. 70. PMLR, International Convention Centre, Sydney, Australia, 1321–1330.

Nima Hatami, Yann Gavet, and Johan Debayle. 2018. Classification of time-series images using deep convolutional neural networks, In Tenth International Conference on Machine Vision (ICMV 2017). Proc.SPIE 10696, 10696 – 10696 – 8.

Dan Hendrycks and Kevin Gimpel. 2017. A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks. Proceedings of International Conference on Learning Representations 5, Article 1, 12 pages.

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2011. Sequential Model-based Optimization for General Algorithm Configuration. In Proceedings of the 5th International Conference on Learning and Intelligent Optimization (LION'05). Springer-Verlag, Berlin, Heidelberg, Article 17, 507–523 pages.

Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37 (ICML'15). JMLR.org, Lille, France, Article 9, 448–456 pages.

Atoosa Kasirzadeh, Mohammed Saddoune, and François Soumis. 2017. Airline crew scheduling: models, algorithms, and data sets. EURO Journal on Transportation and Logistics 6, 2, 111–137.

Jing-Quan Li, Pitu B. Mirchandani, and Denis Borenstein. 2009. A Lagrangian heuristic for the real-time vehicle rescheduling problem. Transportation Research Part E: Logistics and Transportation Review 45, 3, 419–433.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems. NIPS, Nevada, USA, 3111–3119.

Malik Sajjad Ahmed Nadeem, Jean-Daniel Zucker, and Blaise Hanczar. 2009. Accuracy-Rejection Curves (ARCs) for Comparing Classification Methods with a Reject Option. In Proceedings of the third International Workshop on Machine Learning in Systems Biology (Proceedings of Machine Learning Research), Sašo Džeroski, Pierre Guerts, and Juho Rousu (Eds.), Vol. 8. PMLR, Ljubljana, Slovenia, 65–81.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, P. Prettenhofer, Ron Weiss, V. Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Mathieu Brucher, Mathieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research 12, 2825–2830.

Paolo Priore, Alberto Gómez, Raúl Pino, and Rafael Rosillo. 2014. Dynamic scheduling of manufacturing systems using machine learning: An updated review. Artificial Intelligence for Engineering Design, Analysis and Manufacturing 28, 1, 83–97.

Shengli Qiu. 2012. Airline crew pairing optimization problems and capacitated vehicle routing problems. Ph.D. Dissertation. Georgia Institute of Technology.

Troels Martin Range, Richard Martin Lusby, and Jesper Larsen. 2014. A column generation approach for solving the patient admission scheduling problem. European Journal of Operational Research 235, 1, 252–264.

Mohammed Saddoune, Guy Desaulniers, Issmail Elhallaoui, and François Soumis. 2012. Integrated airline crew pairing and crew assignment by dynamic constraint aggregation. Transportation Science 46, 1, 39–55.

Ruslan Sadykov, François Vanderbeck, Artur Alves Pessoa, Eduardo Uchoa, and Issam Tahiri. 2016. Recent results for column generation based diving heuristics.. In ColGen. INRIA, Buzios, Brazil, 1–33.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research 15, 1929–1958.

Stefan Voß. 1999. Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization. Kluwer Academic Publishers, USA.

Peng Wang, Gang Zhao, and Xingren Yao. 2016. Applying back-propagation neural network to predict bus traffic. In Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), 2016 12th International Conference on. IEEE, ICNC-FSKD, China, 752–756.

Joyce W. Yen and John R. Birge. 2006. A stochastic programming approach to the airline crew scheduling problem. Transportation Science 40, 1, 3–14.