Les Cahiers du GERAD ISSN: 0711–2440

Machine learning in airline crew pairing to construct initial clusters for dynamic constraint aggregation

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

G–2020–13

February 2020

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, 2020
– Bibliothèque et Archives Canada, 2020

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, 2020
– Library and Archives Canada, 2020

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

Machine learning in airline crew pairing to construct initial clusters for dynamic constraint aggregation

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

[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
francois.soumis@gerad.ca
slacoste@iro.umontreal.ca

February 2020
Les Cahiers du GERAD
G–2020–13

Abstract: The crew pairing problem is generally modelled as a set partitioning problem where the flights have to be partitioned in pairings. A pairing is a sequence of flight legs separated by connection-time and rest-periods that starts and ends at the same base. This problem becomes difficult to solve when the number of flights increases because the number of feasible pairings (number of variables) grows exponentially. This paper introduces a new paradigm for solving this large combinatorial problem: "Machine Learning → Mathematical Programming". This paradigm uses Machine Learning to learn from solutions of similar instances to produce predictions on some parts of the solution of the new instance. This information feeds the Mathematical Programming optimizer that connects these parts of the solution by modifying them as needed, taking account of the exact cost function and the complex constraints. We show that the clusters produced by ML-based heuristics are better suited for crew pairing problems, resulting in an average reduction of solution cost between 6.8% and 8.52% and cost of global constraints between 69.79% and 78.11%, when compared to pairings obtained with a standard initial solution.

Keywords: Machine learning, mathematical programming, airline crew scheduling, crew pairing

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 IBS Software. The authors would like to thank these organizations for their support and confidence.

1 Introduction

Crew scheduling is of crucial importance for airlines since the crew cost represents their second-highest source of spending, after fuel costs. The first step of crew scheduling is the construction of the pairings. A pairing is a sequence of flight legs separated by connections and rest-periods that starts and ends at the same crew base (an airport where crews are stationed). A pairing may also contain legs that some crew members take as passengers to be relocated, called deadheads. A pairing contains multiple duties — sequences of flights and deadheads that form a day of work. Two consecutive duties inside a pairing are separated by a layover. Pairings must comply with airline regulations as well as collective agreements. The cost of a pairing approximates the crew's salary as well as other expenditures, such as hotel costs. Since pilots and copilots are trained to operate on a single type of aircraft, the crew pairing problem (CPP) can be decomposed by aircraft type. The CPP is to find a set of pairings at minimal cost so that a pairing covers each planned flight.

This problem is known to be quite hard to solve. The complexity is due to the number of variables and constraints involved in the problem definition. In short, it is a large-scale integer programming problem. The most prevalent method since the 1990s has been the branch-and-price: column generation embedded into a branch-and-bound structure [15]. This method is described with others in a survey on CPPs by Cohn and Barnhart [11]; see also Deveci and Demirel [18] for a more recent survey.

In the process of developing a solver capable of dealing with much larger problems than what had been dealt with so far, GENCOL, a commercial operations research solver was integrated into a rolling horizon approach. Because the original solver can handle only a few thousand flights per window, windows are constrained to be two-days long in the case of a large CPP problem. The length of the windows is too short to produce good solutions to monthly problems, as the constraints cover long periods of time.

To permit to solve large-scale pairing problems, a dynamic constraint aggregation approach was introduced, which improves upon column generation by reducing the number of constraints in the problem. Indeed, to reduce the number of constraints considered simultaneously, the work reported by Elhallaoui et al. [22] introduced the dynamic constraint aggregation (DCA) technique that decreases the number of the set partitioning constraints in the restricted master problem and the degeneracy. The GENCOL-DCA solver starts with an aggregation, in clusters, of flights having a good probability of being done consecutively by the same crew, in an optimal solution. The initial aggregation partition is produced with a heuristic solver producing pairings. It corresponds to fixing some flight connection variables to one temporarily, which permits to replace all the flight-covering constraints of the flights in a cluster by a single constraint. GENCOL-DCA uses reduced costs to identify flight connection variables that can be unfixed to improve the solution by breaking clusters. The drawback of this method is that it takes more time to produce the initial clusters than re-optimize with GENCOL-DCA.

This paper proposes to combine multiple methods implemented, developed and tested on small datasets, in order to obtain an efficient algorithm for large-scale CPPs. We use Machine Learning (ML) to propose a good initial partition for a large CPP: monthly problems with up to 50 000 flights. ML alone cannot solve CPP: Solving a 50 000 flights problem needs to find 50 000 crew connections. Even if ML was 99.9% accurate for each connection, the probability of finding a good feasible solution is $(.999)^{50000} \approx 10^{-22}$. We use ML to produce clusters of flights having a high probability of being done consecutively by the same crew, in an optimal solution. A new algorithm combining many advanced Operations Research techniques will be used to assemble and modify these clusters, when necessary, to produce a good solution. This new approach, starting with Machine Learning and finishing the optimization with Mathematical Programming will permit to solve globally larger problems and will avoid the loss of optimality resulting of heuristic decomposition in small time slices in the rolling horizon approach.

In this paper, we show that modern ML techniques can help to automatically learn highly efficient crew pairing schedules and develop initial clusters for DCA. We describe both the Machine Learning

and the Operations Research methodology and detail a proof-of-concept study on a huge monthly problem with up to 50 000 flights.

The remainder of this article is structured as follows. A literature review on crew pairing is presented in Section 2. Section 3 presents an overview of multiple methods that will be combined to obtain an efficient algorithm for very large CPPs and describes the new algorithm. Next, Section 4 describes the Machine Learning predictor and introduces the cluster construction methodology, including different heuristics to avoid constructing infeasible pairings. Section 5 reports computational results. Finally, a short conclusion is drawn in Section 6.

2 Crew pairing

This section reviews a collection of relevant literature on crew pairing to provide an overview of how researchers suggest approaching this problem. We focus on approaches developed to solve large-scale industrial problems.

According to Desaulniers et al. [13], 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 [42].

In the literature, the CPP has been traditionally modelled as a set covering problem (SCP) or a set partitioning problem (SPP), with a covering constraint for each flight and a variable for each feasible pairing [13, 34, 28]. Formally, let F be a set of legs that must be operated during a given period and let Ω be the set of all feasible pairings that can be used to cover these legs. Let a_{fp}, $f \in F$, $p \in \Omega$, be a constant equal to 1 if pairing p contains leg f, and 0 otherwise, and let c_p be the cost of this pairing. For each $p \in \Omega$, define x_p as a binary variable that takes value 1 if pairing p is selected, and 0 otherwise. Using a set-partitioning formulation, the CPP can be modelled as follows:

$$\underset{x}{\text{minimize}} \quad \sum_{p \in \Omega} c_p x_p \tag{1}$$

$$\text{subject to} \quad \sum_{p \in \Omega} a_{fp} x_p = 1 \qquad \forall f \in F \tag{2}$$

$$x_p \in \{0, 1\} \qquad \forall p \in \Omega \tag{3}$$

The objective function (1) minimizes the total pairing costs. Constraints (2) ensure that each leg is covered exactly once, and constraints (3) enforce binary requirements on the pairing variables.

Marsten et al. [29] present a first commercial system with a specialized Mixed Integer Linear Program (MILP) solver for the set partitioning formulation. They present results for various companies and propose a heuristic decomposition for larger problems. Anbil et al. [2] introduced a comprehensive method with a cost-minimization goal. In this research, presenting collaborative research between American Airlines Decision Technologies and IBM, numerous possible pairings (columns) were generated as an apriori, and a considerable amount is fed into the MILP solver. At every repetition, several of the non-basic pairings are rejected, and new pairings are being inserted. The procedure keeps its execution until all the pairings have been taken into consideration [34].

Overall, the conclusion that can be drawn from literature is 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 [3]. Secondly, the algorithms do not consider all viable pairings [5]. Thirdly, the divergence from the optimal solution is significant, resulting in new solutions which can be far from the globally optimal solution [40]. To overcome the above limitations, more sophisticated approaches have been proposed over the years, which are presented in Section 3.

The most prevalent method since the 1990s to solve this large SCP is the column generation inserted in branch-&-bound [13, 4]. This method is described with others in a recent survey [18] concluding

that column generation is the most frequently used approach. The column generation method iterates between pairing generation (sub-problem (SP)) and pairing selection (master problem (MP)). For the SP, many authors have used shortest-path in a network with resource constraints, and the MP is a SPP [16].

In practice, three time-horizons are generally studied: a daily, a weekly, and a monthly horizon [9]. 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, where 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.

For large fleets, globally solving the weekly problem or globally re-optimizing the monthly problem can be too long. The rolling horizon approach is used to speed up the solution process [36]. The horizon is divided into time slices of equal length (except maybe the last one), each one overlapping partially with the previous one. 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, and the next one if it is a re-optimization, through additional constraints. When the size of the problem increases, the time slices need to be shorter to obtain problems that can be resolved within a reasonable time. When the time slices become shorter than the pairings, the quality of the solution is deeply impacted. The pairings partially outside of the time slice cannot be moved to another base, and many deadheads are used to balance the workload between bases. 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. The monthly problem globally deals with a full month. Recent studies have concentrated on weekly and monthly problems. Owing to the vacation period and differences in flight schedules, the monthly time horizon is the most accurate [9]. To solve a monthly CPP, a one-week window with two-days overlapping is usually used.

In recent years, personnel scheduling has become more advanced, using different multidimensional approaches and techniques. A frequently used method is constraint programming, where we seek a good feasible solution that satisfies a specific set of constraints [33]. The most common objective when using constraint programming is to minimize the weighted quantity of late jobs. Several scheduling algorithms are used to optimize the set of schedules which occur in an airport or airline company, from flight scheduling to personnel and equipment scheduling, and researchers often tend to use heuristics instead of exact solution techniques [8]. Nevertheless, solutions provided by Machine Learning methods are often infeasible and need to be refined further by a Mathematical Programming optimizer, taking into account the exact cost function and the complex constraints.

3 Solution methods

We first describe the most complex methods that will be combined to obtain an efficient algorithm for very large CPPs. We then present the structure of the new algorithm, describing the other procedures that compose it and the relations between the procedures.

3.1 Column Generation method

Column Generation (CG) is an iterative method that solves, for each iteration, a restricted master problem and one or numerous sub-problems [37]. Column generation is well-recognized for solving *relaxation* of a MILP 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-type problems [7]. The reader is referred to Desrosiers et al. [17] and Desrosiers et al. [16], the founding papers of the domain.

When the number of columns is large, to the extent that they can no longer be 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. 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.

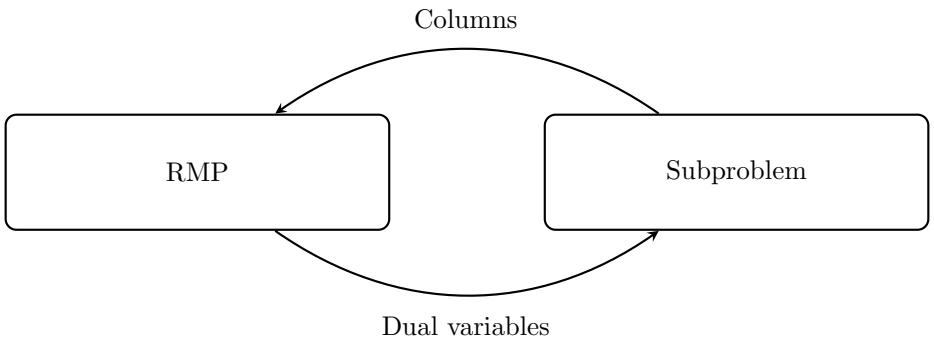


Figure 1: Standard column generation decomposition [7]

Like the simplex method, the column generation method is a convergent and exact method [19]. The speed of convergence depends on the number of iterations necessary to get the proof of convergence. The time by iteration depends on the time to solve the restricted master problem and the sub-problem. It turns out that in practice, a compromise must be found between the speed of convergence and the complexity of the generator algorithm. The generator algorithm can solve a restricted subproblem, or use approximate methods to reduce the computational time [24].

Nevertheless, one should note that the convergence of the iterative process requires an exact algorithm in the last iterations, in order to prove that there is no negative reduced cost column involved. A commonly used method for accelerating column generation is the simultaneous generation of several improving columns at every iteration [14]. This method makes it possible to reduce the number of iterations necessary to generate the columns of the optimal basis.

3.2 Dynamic Constraint Aggregation method

When the number of flights in a CPP increases, the resolution time by column generation becomes large. The larger problems appear in a short-haul network where the crews make many flights per day. In this case, there are many flights per pairing, and the SPP has more dense columns that yield high degeneracy of the simplex algorithm. In the late 1990s, Pan [32] developed some acceleration techniques for SPPs. Elhallaoui et al. [22] improves the Pan's work and adapts it for column generation, which resulted in the DCA algorithm. The DCA algorithm speeds up the master problem by reducing the number of constraints and degeneracy. The DCA method starts with an aggregation, in clusters, of flights having a high probability of being done consecutively by the same crew, in an optimal solution.

Let Q be the set of clusters. The aggregation corresponds to temporarily fix to one the flight connection variables between the flights in a cluster, thus permitting to replace all the flight-covering constraints of the flights in a cluster by a single constraint. This constraint elimination removes degenerate constraints. On the other hand, the variables are partitioned in compatible and incompatible variables. A variable is said to be compatible with the set of clusters Q if the set of flights covered by the corresponding schedule is the union of some clusters in Q. Otherwise, this variable is declared incompatible. Observe that the constraints in a cluster are identical for the compatible variables and in particular the non-degenerate basic variables, justifying keeping only one of them. The Aggregated Reduced Master Problem (ARMP) with only $|Q|$ constraints and only the compatible variables is

solved efficiently with the primal simplex. This smaller non-degenerate problem permits to improve the solution rapidly.

To reach optimality, the incompatible variables with negative reduced cost, need to be considered. The ARMP provides dual variables only for the aggregated set-partitioning constraints. To compute the dual variables for all set-partitioning constraints of the original model, a procedure based on shortest-path calculations is invoked by DCA to identify negative reduced cost variables. To add a group of incompatible variables to the ARMP, the clusters need to be adapted. Some clusters are broken to make the added variables compatible while others are merged when they are connected by a variable equal to one. This dynamic management of clusters permits to reach an optimal solution with a smaller and less degenerate master problem.

3.3 Improved Primal Simplex

Elhallaoui et al. [20] and Omer al. [31] proposed Improved Primal Simplex (IPS), a generalization of DCA, applicable to any linear programming problem. Both IPS and DCA consider a problem with a reduced number of constraints and are restricted to compatible variables. While DCA groups in clusters the constraints that are identical on the non-degenerate variables and keeps only one copy of the constraints in each cluster, IPS keeps a maximum subset of independent constraints on the non-degenerate variables. Even though many dependent constraints are not just copies in linear programming, the vast majority of dependent constraints are copies for SPPs. In IPS, the compatible variables are the variables in the span of the non-degenerate variables. The compatible variables of DCA are a special case of this definition.

In addition to being applicable to any linear programming problem, the main innovation of IPS is the introduction of the Complementary Problem (CP). The CP is formulated with the following notations. Let Q be the set of constraints in the ARMP, $\bar{Q}$ the remaining constraints, C the set of compatible variables and I the set of incompatible variables. The sub-matrix of A with columns indexed by J is denoted $A_{.J}$ and the sub-matrix of A with rows indexed by I and columns indexed by J is denoted A_{IJ}, then the CP is formulated as follows:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \bar{c}_I^T x_I \\ \text{subject to} \quad & \bar{A}_{\bar{Q}I} x_I = 0 \\ & w_I^T x_I = 1 \\ & x_I \geq 0 \end{aligned} \tag{CP}$$

This CP finds a normalized combination of incompatible variables with minimum reduced cost. The first constraint requests that the solution x_I^* is in the subspace of the compatible variables where $\bar{A} = B^{-1}A$ is the matrix of the simplex tableau. This ensures that entering the surrogate variable $\bar{A}_{.I}x_I^*$ in the basis of the ARMP permits a non-degenerate pivot and improves the solution of $\bar{c}_I^T x_I^*$, which is negative. The sequence of pivots on the non-zero variables x_I^* produces the same improvement [31].

The second constraint is a normalization constraint to obtain a bounded solution instead of an extreme ray. The selection of a good normalization is discussed in [31]. This problem is very primal degenerate, but can be solved efficiently with the dual simplex. The dual is not degenerate because the reduced costs in the objective are real numbers. The probability of having two subsets of variables with the same cost is very low.

This CP produces variables to enter in the ARMP to improve the solution and dual variables for all set-partitioning constraints of the model. There is a huge number of dual solutions for the primal solution of the ARMP. The optimal dual solution of the CP is an interior point in the complementary subspace associated with constraints in $\bar{Q}$. Unlike the extreme dual solution of the original problem, this more central solution does not zigzag from an extreme point to another of the dual polyhedron. The distance between dual solutions used at successive iterations of column generation is reduced by a factor of more than 2 and reduces the number of iterations by a factor of 4 [7].

3.4 The new algorithm for the Set Partitioning Problem

In this section, we present the structure of the new SPP solver Commercial-GENCOL-DCA, providing a short description of the procedures that compose it and the relations between the procedures. Figure 2 presents the structure of the new algorithm.

In box 1, the aggregation of flights in a set of Q clusters is done with Machine Learning and will be explained in Section 4. The mathematical programming part of the algorithm is the Branch-&-Bound using CG at each node of the branching tree (see Section 3.1)

In boxes 2 and 3, the compatible and incompatible variables are identified with the network techniques of DCA (see Sections 3.2 and 3.3). This applies easily in CPPs where the flights can be ordered by beginning-time in the clusters and in the path corresponding to the variables. It suffices to follow each path corresponding to a variable. The variable is incompatible if the path enters or exits in the middle of a cluster. It is easy to update C and I when Q is modified by splitting clusters. Pointers from flights to columns covering these flights permit to rapidly re-classify previous incompatible columns covering flights in these clusters by verifying if they enter in the middle of the divided clusters. It is less time consuming than computing B^{-1} and $\bar{A}$ to verify if $\bar{A}\bar{Q}_{.j} = 0$ or not.

In box 4, when the solution of CP has negative reduced cost, the non-null columns of this solution are added to the ARMP. Some clusters are broken to make compatible the added variables. The incompatible variables becoming compatible are also added to the ARMP.

In box 5, when the solution of CP has null reduced cost, the sub-problem needs to be solved to generate columns with negative reduced cost, if possible. To be more efficient, we go to the sub-problem when the reduced cost is above a negative threshold. Instead of adding existing incompatible columns with a small potential of solution improvement, we give priority to generating columns with a better potential for improvement. At each column generation iteration, this threshold is increased up to zero to reach the optimality criterion. There is a sub-problem per crew base and per starting day for pairings. Each sub-problem has a shortest path problem structure with resource constraints modelling the rules, ensuring that the paths are feasible pairings.

In box 6, the variables compatible with the actual partition are added in ARMP. The incompatible ones are added to the CP. The variables are easily classified using the method described for boxes 2 and 3. After adding new variables, ARMP is solved in priority without modifying the partition. ARMP has priority, as long as the reduced cost of its least-reduced cost variable is less than that of the least reduced cost of incompatible variables multiplied by a predetermined multiplier (less than 1).

The column generation combining DCA and IPS is accelerated with the multi-phase dynamic constraint aggregation (MPDCA). Elhallaoui et al. [21] present results on bus and airline crew scheduling with 2000 tasks where the time for the LP solution is reduced by a factor of 4.5 compared to DCA. In this algorithm, a partial pricing strategy, favouring the compatible columns with the aggregation is employed. In phase k, we add in the sub-problem a resource constraint forbidding to generate a pairing breaking clusters more than k times. This method reduces the search space and CPU time. Furthermore, it gives priority to less dense columns in $\bar{A}\bar{Q}_{.I}$ and produces less fractional solutions by combining less dense columns. We see in the computational results LP solutions with less than 1400 fractional variables for a problem with 10 000 constraints.

In box 7, we use a partial exploration of the branching tree. With the column generation solving sub-problem at integrality and having access to a huge number of columns, the LP relaxation provides a very good lower bound. Furthermore, MPDCA provides a primal solution with a small number of fractional variables. The lower bound and the primal solution gives useful information to explore the branching tree efficiently. We first use the column fixing algorithm as follows (for instance, see [36]): At each node of the search tree, several columns taking a fractional-value in the current ARMP solution are selected, and their values are set to one. These columns are selected in decreasing order of their values because columns with larger values, in general, increase less the value of the lower bound when they are set to one than columns with smaller values. To impose a pairing p, we remove from the ARMP

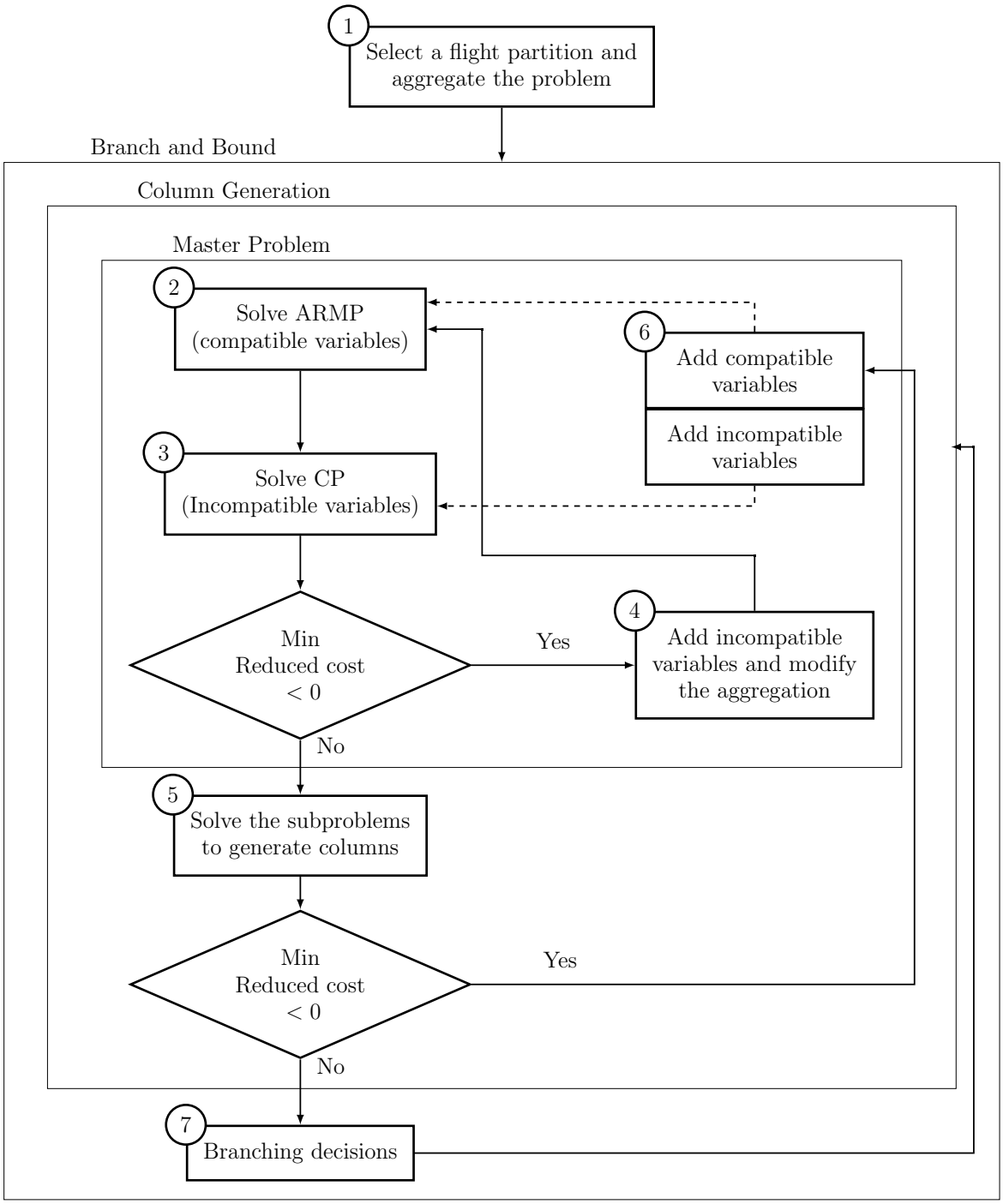


Figure 2: The new algorithm for the SPP type

and CP all columns containing the flights covered by p, and from the sub-problem networks all arcs representing those flights. These decisions rapidly reduce the size of the problems to solve. When there are no variables large enough, we use the arc fixing strategy. We fix to one the arcs with a large value. Unfortunately, such diving heuristic may sometimes make bad decisions and produce a poor integer solution. To reduce the weakness of the diving branching, we use the Retrospective Branching [35]. This algorithm detects and revises, without backtracking, poor decisions made previously in the search tree. These decisions are selected from a list of risky decisions that is maintained during the branching process. A risky decision is a column, or an arc fixed even if their value was smaller than a threshold. When the relative gap q_i between the values z_i and z_0 of the solutions computed at a node i in the search tree and at the root node (i.e., $q_i = \frac{z_i - z_0}{z_0}$) exceeds a relative estimate gap for a good integer solution, the algorithm ejects risky decisions from the current solution without backtracking. This ejection is performed by adding a constraint on the number of risky decisions in the solution.

Several strategies of partial pricing on the variables to enter in the ARMP and in the basis can be used to enter first the most promising variables. As such, a heuristic can be applied in the pricing steps as long as the last iteration uses a complete model. Due to this, partial pricing is appealing as it can apply different sets of compatible and incompatible variables to lower the density of the current pricing problem. For example, one can use partial cost, residual capacity, or rank-incompatibility. Partial cost bias uses a threshold value to discard incompatible variables temporarily based on their partial reduced cost. Residual capacity guarantees a minimum step size. The last type of bias is the rank incompatibility in which all the incompatible values are assigned a rank according to their extent of incompatibility. The pricing model first takes consideration of the low ranked incompatible values. The aim of this bias is not to stray too far from the existing definition of compatibility. The above idea was first developed by Elhallaoui et al. [21] and used in MPDCA.

4 Machine Learning model

This section describes the Machine Learning predictor constructed to provide the probabilities of flights being performed consecutively by the same crew. The prediction problem is first stated and formulated in Section 4.1. The proposed prediction model is disclosed in Section 4.2. Section 4.3 presents the input transformations needed to get an effective prediction model. Upon the finalization of the model training and in order to build pairings, we use different heuristics presented in Section 4.4. Finally, we present additional improvements to the crew pairing construction methodology in Section 4.5.

4.1 Prediction problem formulation

In our solution method for the overall CPP, we provide the optimizer with an initial partition of flights into clusters. Each cluster represents a sequence of flights with a high probability to be consecutive in the same pairing in the solution. To construct each cluster, we need the following:

- Information on where and when a crew begins a pairing. This information makes it possible to identify whether a flight is the beginning of a pairing;
- For each incoming flight to a connecting city, 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 in a pairing 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. We call this the **flight-connection problem**, which is the focus of our ML approach described in the next sections (see Figure 3).

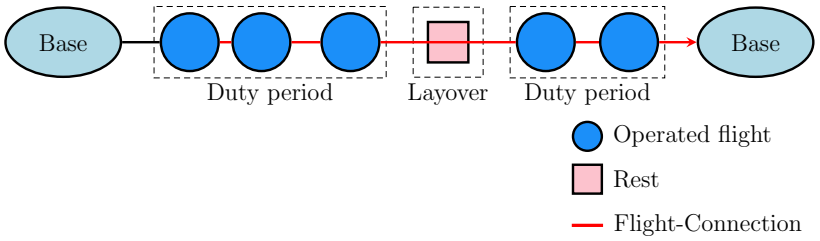


Figure 3: Illustration of a crew pairing. Here the flight-connection variable (in red) is defined to determine the next flight that a crew is going to perform, given an incoming flight

We will transform this data to build a flight-connection prediction sub-problem (a supervised ML multiclass classification problem) where the goal is to predict the next flight that an airline crew should follow in their schedule given the previous 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 and departing city and day (about 2 000 possible flight codes in our dataset).

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

- City of origin and city 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).

4.2 Neural Network architecture

As Neural Networks (NN) have shown great potential to extract meaningful features from complex inputs and obtain good accuracies through different classification tasks, we use in our work the predictor described in Yaakoubi et al. [41]. In more detail, we use a multi-layer convolutional neural network. The output layer is a dense layer with `softmax` as the activation function. We also use dropout [38] to prevent overfitting as well as batch normalization [27] between the activation function and the dropout.

Using standard encoding, the city code and aircraft type features are treated as numeric values. This means that cities with close values are treated similarly by the NN even though the codes are somewhat arbitrary. A more meaningful encoding for such categorical features is to use one-hot encoding. By fully connecting each one-hot encoding of a categorical feature to a separate hidden layer, we 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 feature 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.

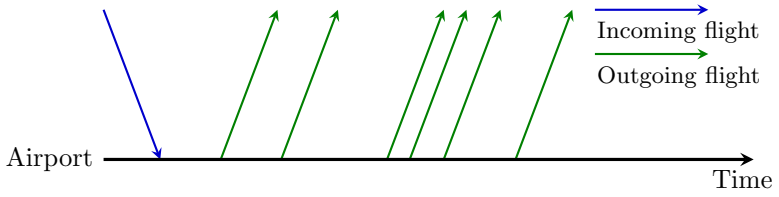
4.3 Transformed input

Throughout the months, in 74% to 76% of the connections, the crew arrives at an airport and follows the aircraft, i.e., takes the next departing flight that the same aircraft makes. To consider more difficult next-flight prediction instances, we, therefore, report "the accuracy only on the instances where the crew changes aircraft."

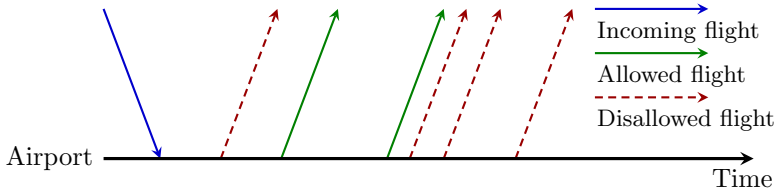
Given that the aircrew arrived at a specific airport at a given time, we can use *a priori* knowledge to define which flights are possible. For example, as shown in Figure 4, it is not possible to make a flight that starts ten minutes after the arrival, nor it is possible five days later. Furthermore, 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 [28] for further details on the likelihood of these scenarios.

In our work, we use the following conditions, that need to be always satisfied for the next flight performed by the crew:

- 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 [28].



(a) Different possibilities for an incoming flight



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

Figure 4: Illustration of an incoming flight scenario

We can, therefore, design a more useful encoding for the output classes by normalizing their representation across different instances. 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 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 not only reduce the number of possible next flights but also construct a similarity-based input, where the neighbouring 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 [25, 6].

4.4 Cluster construction

Upon the finalization of the flights-connection prediction model training, we can use the same architecture to solve two other prediction problems on the test set (50 000 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. To build the crew pairing, one can use the following heuristics:

- Heuristic 1 (H1): We use a greedy heuristic to build a 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 maximal number of days permitted per pairing is approached.
  We can use the above heuristic to construct a solution for the testing data, obtaining a crew pairing that can be fed as an initial cluster for the solver. Unfortunately, if one flight in the pairing is poorly predicted, as the flights are predefined, the crew can finish its pairing away from the base. To correct the pairings, we define a heuristic (H1) in which all pairings where the crews finish away from the base are deleted.
- Heuristic 2 (H2): like H1, consider each flight that the model predicts at the beginning of a pairing as a first flight and for each incoming flight, consider the incoming flight and predict the next one, but deletes all pairings where the crews finish away from the base and all pairings where the predictor abstains.
  Indeed, instead of taking all the predictions into account in constructing the initial clusters for DCA, one can also discard a percentage of these predictions to enhance accuracy. We consider augmenting our classifier with an "abstain" option [30]. Since MPDCA adds in phase k in the sub-problem a constraint forbidding to generate a pairing breaking clusters more than k times, breaking a connection in a cluster takes more time than building one. Therefore, removing low confidence links (flights) using this "abstain" option can be advantageous.

4.5 Further improvements

Start in a base. While training the NN predictor to recognize whether a flight is the beginning of a pairing or not, it is possible that it misclassifies that a flight departing from a non-base city is indeed the beginning of a pairing. It is imperative to correct such false prediction in order to avoid pairings starting away from the base. Even though it is possible to construct a predictor to which only flights departing from the bases are given, it is more efficient and robust to use all flights in the training step. That way, the predictor learns a better representation of the input.

No Layover below a threshold. While training the NN predictor to recognize whether there is a layover or not between two flights, and since this decision is independent of finding the next flight, we can use a threshold on the number of hours between the previous and the next flight, below which it does not make sense to make a layover. This threshold should be defined considering previous solutions or by a practitioner.

Adaptable DCA partition generation. Because the monthly problem is solved using a rolling-horizon approach with one-week windows and two days of overlap period, constructing an initial partition for the entire month and using the subset in each window to feed DCA can be a major flaw. Such initial partition will have many inconsistencies with the solution of the previous window, particularly during the overlap period, such as legs belonging to two different clusters. We propose to adapt the proposed

clusters to the solution of the previous window using the heuristics proposed in Section 4.4 to construct the clusters of the current window in accordance with the solution found for the previous window and any inconsistency with the previous window is avoided, so the proposed partition is adapted to the current resolution. In addition, instead of only considering the flights that the model predicts at the beginning of a pairing as a first flight, incomplete clusters from the solution of the previous window starting during the overlap period are completed. For the next section, we denote by Adapt-H1 and Adapt-H2 the heuristics proposing such adaptation to the partitions produced with H1 and H2.

5 Computational experiments

In this section, we report the results of the computational experiments we conducted using the proposed new algorithm for large-scale CPPs to compare the performance of the proposed heuristics described in Section 4.4 with that of using a cyclic weekly solution as initial clusters. First, we present the instances used for training and testing in Section 5.1, then the hyperparameters and hardware used in the experimentation in Section 5.2. Finally, Sections 5.3 and 5.4 report our ML prediction results and the crew-pairing problem resolution using Commercial-GENCOL-DCA.

5.1 Instances description

Our instances originate from a major airline and consist of six monthly crew pairing solutions for approximately 200 cities and approximately 50 000 flights per month. Each instance contains a one-month flight schedule, crew pairings, as well as a list of airports and bases. This information is used to create input for the learning phase. The dataset consists of approximately 1 000 different source-destination pairings, 2 000 different flight codes and pairings start from 7 different bases. Figure 5 shows the data format for a single pairing.

As shown in Figure 5, this entry describes a pairing that takes place every Tuesday, Thursday, Friday, and Saturday 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). 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 5: Data format for aircrew pairings

5.2 Parameters setting

The parameters setting is relevant to the hyperparameter optimization, evaluation process, and hardware description. To find the best configuration of hyperparameters, we use Bayesian optimization with k-fold cross-validation on the training set to measure the configuration quality. These are presented in Table 1. We use different months for different folds (6 folds) to simulate the more realistic scenario where we make a prediction over a new period of time.

More specifically, we optimize the hyperparameters listed in Table 1 [12] with an implementation of Gaussian process-based Bayesian optimization provided by the GPyOpt Python library version 1.2.1 [39]. Bayesian optimization constructs a probabilistic model of the function mapping from hyperparameter settings to the model performance and provides a systematic way to explore the space more efficiently [26]. 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

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 [10] and Tensorflow [1] libraries.

5.3 Results on Next-Flight-Prediction

We perform the Gaussian process 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 continuously 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 no limitation prevents our algorithmic procedure from exploring, even more, the configuration space; in other words, the algorithm was stopped manually. The final training was done with the best-identified architecture found by Bayesian optimization.

Using the "abstain" option, the accuracy increases from 99.62% to 99.94%, estimating confidence with dropout: The prediction tasks can be carried N times while applying dropout in the entire layers of the neural networks [23], 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 lower bound on the certainty of our prediction. If the maximum value of these confidences is too low, we decide to abstain. For the next subsection, we will use a 0.5% rejection rate for Heuristic 2 (H2).

5.4 Results on crew pairing problems

GENCOL init We consider a standard monthly solution called "GENCOL init", obtained with the GENCOL solver (without DCA). In this approach, the problem is solved by a "rolling time horizon" approach. Because the GENCOL solver (without DCA) is able to solve up to a few thousand flights per window, we are constrained to use two-days windows and one-day overlap period. This means that the month is divided into overlapping time slices of equal length. 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.

Baseline Using the constraint aggregation approach, GENCOL-DCA is fed with the pairings of the solution "GENCOL init" as initial clusters to solve the monthly pairing problem, thus obtaining a solution that we consider as a baseline for comparison. Then, for each experiment, we use one of the heuristics previously defined to construct an initial DCA partition and feed it to Commercial-GENCOL-DCA as initial clusters to solve the monthly pairing problem.

Table 2: Computational results per window

Win.	Alg.	LP-N0	Diff. (%)	#VF-N0	#Nodes	Best-LP	Diff. (%)	INT	Diff. (%)	T time (s)
	Baseline	10122035		2719	159	10025487.73		10276344.14		7891
	H1	9904828	-2.15	2870	203	9891390.17	-1.34	10136106.73	-1.36	10405
1	H2	9889525	-2.30	3014	249	9877050.50	-1.48	10300447.14	0.23	11455
	Adapt H1	9904828	-2.15	2870	203	9891390.17	-1.34	10136106.73	-1.36	10691
	Adapt H2	9889525	-2.30	3014	249	9877050.50	-1.48	10300447.14	0.23	13642
	Baseline	11396498		2519	162	11225022.17		11501016.42		27778
	H1	10589590	-7.08	4771	446	10426771.03	-7.11	13080746.04	13.74	63186
2	H2	10528757	-7.61	5048	381	10386235.55	-7.47	11133714.80	-3.19	63462
	Adapt H1	10319719	-9.45	5848	419	10215252.88	-9.00	10865255.80	-5.53	94004
	Adapt H2	10271980	-9.87	6182	492	10188780.86	-9.23	10990621.86	-4.44	126761
	Baseline	10740127		2330	177	10641496.00		11107990.12		30421
	H1	10590614	-1.39	5340	366	10439285.84	-1.90	10926909.75	-1.63	67217
3	H2	9850067	-8.29	4561	307	9659421.20	-9.23	10272961.85	-7.52	52748
	Adapt H1	9596326	-10.65	4838	294	9432461.69	-11.36	10051850.40	-9.51	53736
	Adapt H2	9685782	-9.82	4574	308	9483718.96	-10.88	10160044.85	-8.53	68597
	Baseline	9764727		2606	229	9591592.13		9968081.73		33898
	H1	9164173	-6.15	3808	314	9015056.00	-6.01	13618635.11	36.62	37142
4	H2	8007573	-17.99	5264	358	7872923.59	-17.92	9619055.54	-3.50	79271
	Adapt H1	8063084	-17.43	4763	394	7868177.74	-17.97	8791799.94	-11.80	56291
	Adapt H2	7574006	-22.44	6473	463	7516289.41	-21.64	8333643.43	-16.40	152781
	Baseline	8095063		3150	188	7899149.01		8102703.93		35948
	H1	9375047	15.81	3571	303	9193080.40	16.38	10730905.63	32.44	34123
5	H2	6305624	-22.11	4558	362	6146413.51	-22.19	6746656.99	-16.74	50138
	Adapt H1	6076461	-24.94	5333	417	5953932.68	-24.63	6453605.80	-20.35	74622
	Adapt H2	5706073	-29.51	6008	480	5646359.35	-28.52	6464805.10	-20.21	133395
	Baseline	6778925		2513	187	6610562.32		6939096.41		29368
	H1	6611841	-2.46	3960	324	6432282.33	-2.70	8915306.53	28.48	39626
6	H2	4905952	-27.63	4882	422	4814868.16	-27.16	5297148.54	-23.66	61760
	Adapt H1	4763520	-29.73	4940	318	4697990.54	-28.93	4947363.87	-28.70	55263
	Adapt H2	4763509	-29.73	5713	437	4726686.40	-28.50	5114636.73	-26.29	94092
	Baseline	9482896		2640	183	9332218.23		9649205.46		27551
	H1	9372682	-0.57	4053	326	9232977.63	-0.45	11234768.30	18.05	41950
Mean	H2	8247916	-14.32	4555	346	8126152.09	-14.24	8894997.48	-9.06	53139
	Adapt H1	8120656	-15.73	4765	340	8009867.62	-15.54	8540997.09	-12.88	57435
	Adapt H2	7981813	-17.28	5327	404	7906480.91	-16.71	8560699.85	-12.61	98211

Computational results per window are reported in Table 2 for all algorithms, namely, baseline, H1, H2, Adapt-H1 and Adapt-H2, where Adapt-H1 and Adapt-H2 are adaptation-based versions of H1 and H2 (see Section 4.5). For each window and each algorithm, we provide the LP value at the root node of the search tree N0 (LP-N0), the computational time at N0 (N0 time), the number of fractional variables (# VF-N0) in the current MP solution at N0, the number of branching nodes resolved (# Nodes), the best LP value found (Best-LP), the pairing cost of the best feasible solution (INT) and finally the total computational time (T time); times are in seconds. Furthermore, for all ML algorithms, and for LP-N0, Best-LP and INT, we indicate the relative difference between the result obtained with this algorithm and that with "GENCOL-DCA" (Baseline).

We start by comparing Baseline, H1 and H2. Observe first that H2 gives better LP-N0 and Best-LP values with an average reduction factor of 14.32% and 14.24% respectively, compared to Baseline, while H1 gives lees significant reductions in LP-N0 and Best-LP values with a reduction factor of only 0.57% and 0.45% respectively. The same can be observed for the cost of the best feasible solution (INT), where H2 has an average reduction factor of 9.06% while H1 gives worse results with an increase factor of 18.05%. H1 does not perform well because the proposed clusters are not adapted to the solution of the previous window found by the optimizer, while this problem seems to be partially avoided when using the abstention method. This is explained by the ability of H2 to discard poorly predicted clusters by using the "abstain" option. Since the optimizer 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.

On the other hand, note that for H1 and H2, the average total computational times per window are between 9.57% and 133.85% larger than those of Baseline. This time increase is due to the large number of fractional variables at the root node of the search tree with an increase factor between 5.55% and 129.18%. This is explained by the fact that, when base constraints are too restrictive, the root node solutions contain a more significant number of fractional-valued pairing variables in order to split the worked time between the bases evenly. This causes an increase in the number of branching nodes required to obtain a good integer solution. Indeed, H1 and H2 present an increase factor in the number of branching nodes between 27.67% and 175.30%.

Next, we compare Baseline, Adapt-H1 and Adapt-H2. For the root node (N0), observe that both adaptation-based heuristics give better LP-N0 values for all windows providing an average reduction factor of 15.73% and 17.28% respectively. Likewise, both heuristics provide better Best-LP values with an average reduction factor of 15.54% and 16.71% and better feasible solutions with a reduction factor of 12.88% and 12.61%. This is explained by the ability of Adapt-H1 and Adapt-H2 to propose custom-made clusters in an online manner adapted to the solution of the previous window, completing clusters that start in the overlap period and proposing new unseen clusters for the non-overlap period.

On the other hand, note that the average computational time for Adapt-H1 is similar to H2 while that of Adapt-H2 is on average 76.46% larger. This is due to the larger number of nodes, caused by the larger number of fractional variables at the root node of the search tree. This can be explained by the fact that Adapt-H2 discards between 30 and 50 clusters per window, providing fewer clusters than Adapt-H1. Therefore, the adaptation scheme is capable of proposing suitable clusters and the shifting scheme to use the next available flight if the predicted next flight is covered by another crew makes the abstention option unnecessary.

Computational results on monthly solutions are reported in Table 3. We report the total solution cost, the cost of global constraints and the number of deadheads. For all heuristics algorithms, we also indicate the relative difference between the result obtained with this algorithm and that with GENCOL-DCA (Baseline).

Table 3: Computational results on monthly solution

	Solution cost	Diff. vs Baseline (%)	Cost of global constraints	Diff. vs Baseline (%)	Number of deadheads
GENCOL init	30 681 120.5		9 465 982.28		1725
Baseline	20 639 814.6	-32.73 (vs GENCOL init)	2 127 086.77	-77.53 (vs GENCOL init)	992
H1	21 118 006.16	2.32	2 202 610.31	3.55	1136
H2	19 235 343.4	-6.80	642 599.29	-69.79	1059
Adapt-H1	18 881 977.89	-8.52	465 687.94	-78.11	1014
Adapt-H2	19 104 804.62	-7.44	490 787.75	-76.93	1097

We start by comparing GENCOL init and Baseline. Observe first that a significantly better solution was found using Commercial-GENCOL-DCA, compared to the initial solution GENCOL init. Indeed, the solver significantly reduced both the solution cost and the cost of global constraints with a reduction factor of 32.73% and 77.53% respectively, while reducing the number of deadheads by 42.49%. This

attests to the optimizer's capacity to tackle larger windows finding better solutions and improving industrial-scale solutions.

Next, we compare Baseline, H1 and H2. While the solution cost and the cost of global constraints found with H1 are slightly worse than Baseline with an increase factor of 2.32% and 3.55%, H2 outperforms Baseline reducing the solution cost by 6.8%. Furthermore, H2 reduced the cost of global constraints by 69.79%, which supports and justifies the large number of fractional variables at the root node of the search tree and the number of nodes, yielding larger computational times. We believe that this trade-off is acceptable since the improvement in the cost of global constraints is significant and that the larger computational times are due to the tight constraints on the number of worked hours per base. Relaxing these constraints may yield better results than Baseline while reducing the computational time. Finally, note that the poor results of H1 and good results of H2 are explained by the ability of H2 to tackle and revise poor predictions and discard poorly constructed clusters.

Then, we compare Baseline, Adapt-H1 and Adapt-H2. The solutions found by both heuristics present better statistics than Baseline, H1 and H2. Adapt-H1 yields better solutions than any other heuristic, with a reduction factor in the solution cost and cost of global constraints of 8.52% and 78.11%, respectively. Adapt-H2 presents similar results while providing a reduction factor of 7.44% and 76.93%. It is also worth noting that, for all heuristics, the number of deadheads used is slightly larger with an increase factor between 2.22% and 14.52%, compared to Baseline. The cost of deadheads is accounted for in the solution cost. Because the solution cost is a multi-objective function, we believe that using slightly more deadheads permitted to get better solutions, enhancing both the solution cost and the cost of global constraints.

6 Conclusion

In this paper, we present Commercial-GENCOL-DCA, a new efficient SPP solver that relies on column generation and constraint aggregation. We developed ML-based heuristics capable of constructing adapted initial clusters for the optimizer, taking into account multiple past solutions. This work is the first attempt to embed into a column generation framework recently developed ML methods. We also proposed an adaptation mechanism to propose clusters in an online manner, taking into account the solution of the previous window.

We compared the performance of Commercial-GENCOL-DCA either using a standard initial solution or with clusters proposed by ML-based heuristics with an existing standard solution used in the airline industry. The main computational results show that Commercial-GENCOL-DCA yields better results than the old GENCOL solver using a rolling-horizon approach with narrow windows. In addition, ML-based heuristics, along with either abstention or adaptation yields better results with significantly smaller costs reducing by 8.52% the solution cost and by 78.11% the cost of global constraints. This improvement comes with an increase in computational time.

We believe that the proposed solution is able to handle larger problems as well as learn from multiple airline companies to construct initial clusters for a new company since it does not use flight codes in any part of the pre-processing, learning or prediction process. We believe it can easily be adapted to other types of optimization problems, such as railway or bus shift scheduling. Finally, our long-term goal is to develop efficient new learning techniques that could handle and learn from the flight-based network structure, where nodes correspond to time-space coordinates and arcs represent tasks performed by crew members (legs, deadheads, connections, rests, etc.) capable of incorporating global and local constraints in the ML-predictor learning process.

References

[1] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow. org 1, 2 (2015), 19.

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

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

[4] Cynthia Barnhart, Ellis L Johnson, George L Nemhauser, Martin WP Savelsbergh, and Pamela H Vance. 1998. Branch-and-price: Column generation for solving huge integer programs. Operations research 46, 3 (1998), 316–329.

[5] 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 (1992), 272–283.

[6] 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.

[7] 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 (2017), 835–850.

[8] Philippe De Bruecker, Jorne Van den Bergh, Jeroen Belien, and Erik Demeulemeester. 2015. A two-stage mixed integer programming approach for optimizing the skill mix and training schedules for aircraft maintenance. Working Papers Department of Decision Sciences and Information Management 516588. KU Leuven, Faculty of Economics and Business, Department of Decision Sciences and Information Management.

[9] 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 (2013), 129–149.

[10] François Chollet et al. 2015. Keras.

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

[12] 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) 2016 (2016), 406–413.

[13] 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 (1997), 245–259.

[14] Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon. 2002. Accelerating Strategies in Column Generation Methods for Vehicle Routing and Crew Scheduling Problems. Springer US, Boston, MA, Chapter 14, 309–324. https://doi.org/10.1007/978-1-4615-1507-4_14

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

[16] 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.

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

[18] Muhammet Deveci and Nihan Çetin Demirel. 2018. Evolutionary algorithms for solving the airline crew pairing problem. Computers & Industrial Engineering 115 (2018), 389–406.

[19] Olivier du Merle, Daniel Villeneuve, Jacques Desrosiers, and Pierre Hansen. 1999. Stabilized column generation. Discrete Mathematics 194, 1 (1999), 229–237. https://doi.org/10.1016/S0012-365X(98)00213-1

[20] Issmail Elhallaoui, Abdelmoutalib Metrane, Guy Desaulniers, and François Soumis. 2011. An Improved Primal Simplex Algorithm for Degenerate Linear Programs. INFORMS Journal on Computing 4 (2011), 569–577. https://doi.org/10.1287/ijoc.1100.0425

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

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

[23] 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, 10 pages.

[24] Michel Gamache, François Soumis, Gérald Marquis, and Jacques Desrosiers. 1999. A Column Generation Approach for Large-Scale Aircrew Rostering Problems. Operations Research 47, 2 (1999), 247–263.

[25] 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.

[26] 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, 24 pages.

[27] 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, 9 pages.

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

[29] Roy E. Marsten and Fred Shepardson. 1981. Exact Solution of Crew Scheduling Problems Using the Set Partitioning Model: Recent Successful Applications. Networks 11 (1981), 165–177.

[30] 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.

[31] Jérémy Omer, Samuel Rosat, Vincent Raymond, and François Soumis. 2015. Improved primal simplex: a more general theoretical framework and an extended experimental analysis. INFORMS Journal on Computing 27, 4 (2015), 773–787.

[32] P-Q Pan. 1998. A basis-deficiency-allowing variation of the simplex method for linear programming. Computers & Mathematics with Applications 36, 3 (1998), 33–53.

[33] Michael L. Pinedo. 2016. Scheduling. Springer International Publishing, New York. https://doi.org/10.1007/978-3-319-26580-3

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

[35] Frédéric Quesnel, Guy Desaulniers, and François Soumis. 2017. A new heuristic branching scheme for the crew pairing problem with base constraints. Computers & Operations Research 80 (2017), 159–172.

[36] Mohammed Saddoune, Guy Desaulniers, and François Soumis. 2013. Aircrew pairings with possible repetitions of the same flight number. Computers & Operations Research 40, 3 (2013), 805–814.

[37] 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.

[38] 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 (2014), 1929–1958.

[39] The GPyOpt authors. 2016. GPyOpt: A Bayesian Optimization framework in python. http://github.com/SheffieldML/GPyOpt.

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

[41] Yassine Yaakoubi, François Soumis, and Simon Lacoste-Julien. 2019. Flight-connection prediction for airline crew scheduling to construct initial clusters for OR optimizer. Les Cahiers du GERAD G–2019, 26 (2019), 22.

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