An improved integral column generation algorithm using machine learning for air-crew pairing

A. Tahir, F. Quesnel, G. Desaulniers, I. El Hallaoui, Y. Yaakoubi

G–2020–71
December 2020
An improved integral column generation algorithm using machine learning for aircrew pairing

Adil Tahir a
Frédéric Quesnel a
Guy Desaulniers a
Issmail El Hallaoui a
Yassine Yaakoubi a

a GERAD & Department of Mathematics and Industrial Engineering, Polytechnique Montréal (Québec) Canada, H3C 3A7

adil.tahir@gerad.ca
frederic.quesnel@gerad.ca
guy.desaulniers@gerad.ca
issmail.elhallaoui@gerad.ca
yassine.yaakoubi@gerad.ca

December 2020
Les Cahiers du GERAD
G–2020–71
Copyright © 2020 GERAD, Tahir, Quesnel, Desaulniers, El Hallaoui, Yaakoubi

The authors are exclusively responsible for the content of their research papers published in the series Les Cahiers du GERAD. Copyright and moral rights for the publications are retained by the authors and the users must commit themselves to recognize and abide the legal requirements associated with these rights. Thus, users:

• May download and print one copy of any publication from the public portal for the purpose of private study or research;
• May not further distribute the material or use it for any profit-making activity or commercial gain;
• May freely distribute the URL identifying the publication.

Si vous pensez que ce document enfreint le droit d’auteur, contactez-nous en fournissant des détails. Nous supprimerons immédiatement l’accès au travail et enquêterons sur votre demande.
Abstract: The crew pairing problem (CPP) is solved in the first step of the crew scheduling process. It consists of creating a set of pairings (sequence of flights, connections, and rests forming one or multiple days of work for an anonymous crew member) that covers a given set of flights at minimum cost. Those pairings are assigned to crew members in a subsequent crew rostering step. In this paper, we propose a new integral column generation algorithm for the CPP, called Improved Integral Column Generation with Prediction ($I^2CG_p$), which leaps from one integer solution to another until a near-optimal solution is found. Our algorithm improves on previous integral column generation algorithms by introducing a set of reduced subproblems. Those subproblems only contain flight connections that have a high probability of being selected in a near-optimal solution and are, therefore, solved faster. We predict flight connection probabilities using a deep neural network trained in a supervised framework. We test $I^2CG_p$ on several real-life instances and show that it outperforms a state-of-the-art integral column generation algorithm as well as a branch-and-price heuristic commonly used in commercial airline planning software, both in terms of solution cost and computing times. We highlight the contributions of the neural network to $I^2CG_p$.

Keywords: Crew pairing, machine learning, integral column generation, deep neural network

Acknowledgements: This work was supported financially by the Natural Sciences and Engineering Research Council of Canada and AD OPT under grant no. RGPIN–2018–05284. The authors are grateful for this support.
1 Introduction

Creating efficient crew schedules is of great importance for airlines. Indeed, Gopalakrishnan and Johnson (2005) note that “Next to fuel costs, crew costs represent the largest single cost factor for the airlines”. Creating good-quality schedules is challenging due to the large size of real-life instances and to the combinatorial nature of the problem. For this reason, aircrew schedule optimization has been an active field of research for over half a century.

Aircrew rostering is usually performed according to a two-step procedure: crew pairing followed by crew rostering. The goal of the crew pairing problem (CPP) is to find a set of pairings covering a given set of flights at minimum cost. A pairing is a sequence of flights, deadheads (flights crew members take as passengers to be relocated), connections and rests forming one or several days of work that can legally be performed by a crew member. A pairing must also start and end at the same crew base (airport in a city where crew members live) and comply with airline regulations and collective agreements. An individual day of work in a pairing is called a duty. The crew rostering problem assigns those pairings to crew members to create a feasible schedule. Solving both problems simultaneously in an integrated fashion would lead to better schedules. Such integrated approaches have been proposed (see, e.g. Mercier and Soumis 2007, Saddoune et al. 2012, Zeighami and Soumis 2019), but are seldom used in practice for two main reasons. First, it is common among airlines to have two distinct departments in charge of creating pairings and rosters for historical and practical reasons. Second, integrated approaches do not scale well to the size of real-life instances.

This paper focuses on the CPP. Over the years, solution methods for the CPP have been greatly improved by the development of specialized algorithms, as well as advances in generic optimization techniques, namely column generation. Simultaneously, real-life instances have become more challenging. Indeed, instances have become larger due to airline mergers and the increase in air travel. Furthermore, many airlines want to leverage the improvement of computers to solve more complex variants of the CPP, which require novel solution methods. For this reason, aircrew optimization remains an active field of research.

The CPP is usually formulated as a set partitioning problem (sometimes with a few side constraints). State-of-the-art solution methods for the CPP use column generation (e.g. Saddoune, Desaulniers, and Soumis 2013, Soykan and Erol 2014, Quesnel, Desaulniers, and Soumis 2020) to solve its linear relaxation. The column generation algorithm iteratively solves a restricted master problem (RMP) and one or multiple subproblems (SP). The RMP is a restricted version of the CPP linear relaxation in which a small subset of all feasible pairings is considered. The goal of the SPs is to find new pairings of negative reduced cost to add to the RMP. The column generation algorithm iteratively solves the RMP and the SPs until no negative reduced cost columns are returned by the SPs, at which point, the current RMP solution is optimal.

The CPP is usually solved using a branch-and-price heuristic. This algorithm explores a branch-and-bound tree and solves a CPP linear relaxation at each node using column generation. In most real-life CPP instances, the branch-and-bound tree is too large to be fully explored; a heuristic is then applied to explore a small fraction of the tree in order to obtain a good-quality solution in a reasonable time. See Joncour et al. (2010) for a review of such heuristics, and Quesnel, Desaulniers, and Soumis (2020) for a comparison of some of those heuristics specifically for the CPP. One of the main weaknesses of branch-and-price algorithms is that the column generation algorithm optimizes a linear program whereas an integer solution is desired. In practice, this means that the SP's usually produce columns that are advantageous for the linear relaxation of the CPP, but not necessarily for the CPP itself. This leads to long computing times, as more branching decisions are required to reach integrality.

Recently, Tahir, Desaulniers, and El Hallaoui (2019) developed an integral column generation algorithm (ICG) for the set-partitioning problem. The main idea behind this method is to solve the RMP using an integral simplex algorithm such as the Integral Simplex Using Decomposition (ISUD) (Zaghrouti, Soumis, and El Hallaoui 2014). ICG has many advantages over branch-and-price
algorithms. First, since the RMP solutions are always integer, ICG can be applied to the problem itself, rather than its linear relaxation. Another advantage of ICG is that a feasible solution of the problem is produced at each iteration. It is, therefore, possible to provide the planner with many good-quality solutions at no extra cost. It is also possible to stop ICG early and still obtain an integer solution. Tahir, Desaulniers, and El Hallaoui (2019) use ICG to obtain good-quality solutions to instances of the CPP containing up to 1700 flights in less than 3800 seconds. Their method outperforms a standard heuristic branch-and-price method both in terms of computing time and solution cost.

The SPs for the CPP are usually formulated as shortest-path problems with resource constraints (SPPRC) on a directed graph (e.g. Anbil, Forrest, and Pulleyblank 1998, Sandhu and Klabjan 2007, Zeighami and Soumis 2019, Quesnel, Desaulniers, and Soumis 2017). The goal of the SPPRC is to find the shortest path between the source and the sink of a graph, that respects a set of path constraints. Resources are used to enforce those path constraints. A resource is a commodity that is consumed on arcs and is bounded on nodes. For instance, a resource can be used to limit the duration of a pairing, the number of flights per day,... The SPPRC can be solved by a dynamic programming algorithm (often called labeling algorithm). More details on the SPPRC are presented in Section 4.2.1.

The SP networks for the CPP can be classified into two categories: duty-based and flight-based networks. In duty-based networks, the nodes represent feasible duties and arcs connect duties that can be operated sequentially. Anbil, Forrest, and Pulleyblank (1998) use such a network to create good-quality solutions to instances of the CPP containing up to 837 flights in less than one hour. One drawback of duty-based networks is that they require enumerating a large number of feasible duties beforehand, making them unsuitable for large instances. For this reason, flight-based networks are usually used to solve large instances. In this type of network, nodes correspond to space-time coordinates and arcs represent tasks (flights, deadheads, connections, rests, ...) that can be performed inside a pairing. Sandhu and Klabjan (2007) propose a column generation algorithm that uses a flight-based network to solve the integrated aircraft fleeting (optimization of individual aircraft routes) and crew pairing problem. They obtain solutions in less than 30 hours for instances containing up to 1000 flights. Other examples of algorithms using flight-based networks to solve the CPP are those proposed by Saddoune, Desaulniers, and Soumis (2013), and Zeighami and Soumis (2019). One aim of this paper is to develop solution methods for large CPP instances. Therefore, we formulate the SPs as SPPRCs on flight-based networks.

Even using state-of-the-art methods, solving large CPP instances can take a considerable amount of time. A large fraction of this time is spent solving the SPs. The time required to solve each SP increases with the number of resources and the number of arcs in the networks. In flight-based networks, most of the arcs in the SPs are arcs connecting pairs of flights that can be operated sequentially, called flight connection arcs. However, only a small fraction of those arcs are used in any given integer solution.

A potential solution to this challenge can be found in vehicle routing problem literature. It consists of reducing the size of the networks by excluding some arcs that are unlikely to be selected in a good solution. For instance, Fukasawa et al. (2006) develop a greedy heuristic that exclude arcs based on their length while maintaining the connectivity of the network. Desaulniers, Lessard, and Hadjar (2008) use a similar strategy, but select arcs based on their reduced cost, while ensuring that each node has a minimum number of incoming and outgoing arcs. Other strategies can be found in, for instance, Chabrier (2006) and Contardo and Martinelli (2014). A common point of all those strategies is that they all rely on heuristic rules carefully designed by experts. Those rules are relatively easy to design for VRPs because the network structure is simple and the costs are placed directly on the arcs, which is not the case for the CPP. It is very unlikely that such successful strategies for vehicle routing problems are also successful for the CPP. Designing new strategies for the CPP would require a significant amount of effort. Furthermore, since airlines have very different requirements, a new arc selection strategy would have to be devised for each new CPP variant.

In this paper, we propose a partial pricing scheme for the CPP in which a set of reduced SPs are solved first to reduce the computational time. Those reduced SPs contain only a fraction of all flight connections. We exclude flight connection arcs based on their estimated probabilities of being selected...
in the final solution. Flight connections with a zero estimated probability are excluded from the reduced SPs. The reduced SPs also contain an additional resource that tracks the likelihood of partial paths, thus preventing highly probable paths from being discarded while solving the SPs. Full SPs (with all flight connections) are used when a column generation iteration fails to sufficiently improve the current best solution, making the algorithm resilient to prediction mistakes. This partial pricing scheme is embedded in an ICG algorithm.

To predict flight connection probabilities, we use a machine learning (ML) model, namely, a deep neural network trained on historical optimal or near-optimal CPP solutions. This prediction model is based on the work of Yaakoubi, Lacoste-Julien, and Soumis (2019), with some key differences in the definition of features and in the way the input is built. Rather than accurately predicting the “true” flight connection probabilities (which would require an unrealistic amount of data), our ML model imitates the behavior of a near-optimal solver (imitation learning). The prediction model then acts as an expert in our solution method, that rapidly guides the optimizer towards a near-optimal solution.

The proposed method is tested on several instances of the CPP. The ICG with partial pricing algorithm is compared to ICG without partial pricing, as well as a standard branch-and-price algorithm. We show that our method produces solutions of lower cost than the other methods tested, in less computing times. The contributions of this paper can be summarized as follows:

- We develop a ML model that accurately predicts flight connection probabilities for the CPP. The main advantage of this method is that it converges to a near-optimal solution in few column generation iterations because the reduced SPs generate columns with a high probability of being selected in such a solution.
- We propose a generic partial pricing scheme that leverages those flight connection probabilities. We show that it outperforms standard pricing schemes when used in an ICG framework.

The remainder of this article is structured as follows. We outline the CPP in Section 2. In Section 3, we describe the ML model used to predict flight connection probabilities and present computational results regarding its accuracy. In Section 4, we describe our CPP solution method, and present computational results on multiple large-size instances of the problem. Conclusions are drawn in Section 5.

## 2 Problem description

In this section, we briefly describe the CPP variant considered in this paper which is identical to the one studied by Quesnel, Desaulniers, and Soumis (2017) except for the global constraints (see below). This variant captures the most widespread CPP features found among airlines.

Let \( F \) be a set of flights for a given time period (usually a month), and let \( \Omega \) be the set of all legal pairings. A pilot is authorized to operate only one aircraft type at any given time, and must undergo a multiple-week training program to change to another aircraft type. For this reason, the CPP is usually decomposed by aircraft type. Therefore, we assume that the same aircraft type is used to operate all flights in \( F \). When creating pairings for cabin crews as stated above, the goal of the CPP is to select a set of feasible pairings covering all flights of \( F \) at minimum cost. The cost of pairing \( p \in \Omega \), denoted \( c_p \), is determined by a non-convex function that depends on multiple pairing features. It corresponds to its total work time \( T_p \) (explained below), plus penalties for deadheads, short connections and short rests. Let \( H_p \) and \( W_p \) be the sets of deadheads and connections/rests in pairing \( p \), respectively. Let \( \delta_w \) be the length of \( w \), measured in minutes, where the nature of \( w \) (flight, deadhead, connection, rest, duty, or pairing) is inferred from the context. Let \( \phi^{DH}(\delta) \) be the penalty function for a deadhead of length \( \delta \). Similarly, let \( \phi^{C}(\delta) \) be the penalty function for a connection or a rest of length \( \delta \). The cost \( c_p \) is given by

\[
c_p = T_p + \sum_{f \in H_p} \phi^{DH}(\delta_f) + \sum_{w \in W_p} \phi^{C}(\delta_w).
\]
We now explain each term of (1). In accordance with industry practices, we define $T_p$ as the maximum between two quantities: a quarter of the total length of $p$, and the sum of $p$'s duties paid time. The paid time of a duty corresponds to the total time spent operating an aircraft, plus half the deadhead time, with a minimum guaranteed paid time of 4 hours (or 240 minutes). Let $D_p$ be the set of duties in $p$, and let $F_d$ and $H_d$ be the sets of flights and deadheads in duty $d$, respectively. $T_p$ is given by:

$$T_p = \max \left\{ \frac{\delta_p}{4}, \sum_{d \in D_p} \max \left\{ 240, \sum_{f \in F_d} \delta_f + \frac{1}{2} \sum_{f \in H_d} \delta_f \right\} \right\}$$

The second term of (1) penalizes deadheads. Deadheads are undesirable because it corresponds to time during which a non-working crew member is paid while taking a seat that could otherwise be given to a paying customer. Each deadhead incurs a fixed penalty of $\gamma^{DH}$, and a variable penalty of $\lambda^{DH}$ per minute. The penalty for deadhead $f \in F$ is thus

$$\phi^{DH}(\delta_f) = \gamma^{DH} + \lambda^{DH} \delta_f.$$  

The last term of (1) penalizes short rests and connections, which airlines try to avoid because they make pairings vulnerable to small disruptions. Let $\bar{t}^C$ and $\bar{t}^R$ be the minimum connection and rest times, respectively. Similarly, let $\hat{t}^C$ and $\hat{t}^R$ be the target connection and rest times, respectively. A connection shorter than $\hat{t}^C$ incurs a penalty of $\lambda^C$ for each minute it falls short of $\hat{t}^C$. Similarly, a rest shorter than $\hat{t}^R$ incurs a penalty of $\lambda^R$ for each minute it falls short of $\hat{t}^R$. Connections and rests of lengths exceeding their respective targets are not penalized. The penalty function for a connection or rest of length $\delta$ is thus:

$$\phi^C(\delta) = \begin{cases} 
\lambda^C(\hat{t}^C - \delta) & \text{if } t^C \leq \delta < \hat{t}^C \\
\lambda^R(\hat{t}^R - \delta) & \text{if } t^R \leq \delta < \hat{t}^R \\
0 & \text{otherwise.}
\end{cases}$$

Note that $t^R > \hat{t}^C$ and that connections longer than $t^R$ are not allowed as they are considered rests.

Airlines usually have a large set of pairing legality rules, imposed by regulatory agencies and collective agreements. Those rules can vary greatly between airlines. Our CPP model considers a subset of the most common rules among airlines. They are stated as follows:

- A pairing must start and end at the same crew base.
- A pairing must be no longer than $\bar{d}$ days.
- A pairing must contain at most $\bar{d}^{duty}$ duties.
- A duty must last at most $\bar{t}^D$ minutes.
- Each duty must contain no more than $\bar{t}^W$ minutes of work (defined as the total flying time, plus half the deadhead time).
- Each duty must contain at most $f^{MAX}$ flights.
- The connection time between two consecutive flights in a duty must be at least $t^C$ minutes long.
- The rest period between two consecutive duties must be at least $t^R$ minutes long.

Most CPP variants also include global constraints. For instance, it is common to use base constraints to spread the work time amongst the bases according to their crew member availability (e.g. Saddoune, Desaulniers, and Soumis 2013, Zeighami and Soumis 2019, Quesnel, Desaulniers, and Soumis 2017). Alternatively, we use crew availability constraints that limit the number of ongoing pairings assigned to each base on each day to ensure that enough crews are available at each base to operate their assigned pairings. In fact, these constraints slightly overestimate crew requirements since it is sometimes possible for a crew member to finish a pairing on one day and start another pairing on the same day. Nevertheless, these constraints are more precise than the traditional base constraints.
We now present a mathematical formulation of the CPP. Let $B$ and $\mathcal{J}$ be the set of bases and days in the optimization period, respectively. Let $x_p$ be a binary variable taking value 1 if pairing $p \in \Omega$ is chosen, and 0 otherwise. Let $a_{fp}$ be a parameter equal to 1 if flight $f \in \mathcal{F}$ is in pairing $p$ and 0 otherwise. Let $d_{bj}$ be the number of crews available at base $b \in B$ on day $j \in \mathcal{J}$, and let $u_{pbj}$ be a binary parameter equal to 1 if pairing $p$ starts at base $b$ and is ongoing on day $j$, and 0 otherwise. Our CPP variant is formulated as:

$$\min \sum_{p \in \Omega} c_p x_p$$  \hspace{1cm} (5)

subject to

$$\sum_{p \in \Omega} a_{fp} x_p = 1 \quad \forall f \in \mathcal{F}$$  \hspace{1cm} (6)

$$\sum_{p \in \Omega} u_{pbj} x_p \leq d_{bj} \quad \forall j \in \mathcal{J}, b \in B$$  \hspace{1cm} (7)

$$x_p \in \{0, 1\} \quad \forall p \in \Omega$$  \hspace{1cm} (8)

The objective function (5) minimizes the total pairing cost. The set-partitioning constraints (6) ensure that each flight is covered exactly once. Constraints (7) are the crew availability constraints and constraints (8) enforce binary requirements on the pairing variables. Note that formulation (5)–(8) is very close to a set-partitioning problem because the number of side constraints is relatively small compared to the number of flights.

### 3 A machine learning prediction model for flight connections

The solution method described in Section 4 requires accurate predictions on flight connection probabilities (i.e. the probability that flight $i \in \mathcal{F}$ is followed by flight $j \in \mathcal{F}$ in a pairing that is part of an optimal or near-optimal solution). We train a machine learning model to predict flight connection probabilities with a high level of accuracy. It is a deep convolutional neural network similar to the one used by Yaakoubi, Lacoste-Julien, and Soumis (2019) in a similar context. It is trained in a supervised learning framework, using data from solutions of 42 CPP instances.

We formalize the prediction task in Section 3.1. The features are described in Section 3.2. The neural network structure and the hyper-parameters of the model are presented in Section 3.3. Finally, we present results for our prediction model in Section 3.4.

#### 3.1 Prediction task

Let $i, j \in \mathcal{F}$. The ultimate goal of our prediction model can be stated as follows: *Given that $i$ belongs to a pairing starting at base $b \in B$, estimate the probability that $i$ is immediately followed by $j$ in an optimal solution.* Let $p_{ijb}$ denote this estimated probability. In the remainder of this paper, estimated probabilities are simply referred to as probabilities. The probabilities for base $b$ and for all pairs of flights can be stored in the *connection probability matrix* $P^b$, where $P^b_{ij} = p_{ijb}$.

It is not obvious that a different connection probability matrix is required for each base. To illustrate why it is the case, consider the example depicted in Figure 1. Let $i, j, k \in \mathcal{F}$ such that:

- $i$ is a flight arriving at airport $a$ on day 4 at 12:00;
- $j$ is a flight from $a$ to $b_1 \in B$ that departs on day 4 at 13:00;
- $k$ is a flight from $a$ to $b_2 \in B$ that departs on day 4 at 13:00.

If the pairing containing flight $i$ starts at base $b_1$, there is a high probability that $i$ is followed by $j$ so that the crew returns to its starting base, thus possibly ending the pairing. Conversely, it is unlikely that $i$ is followed by $k$ since it sends the crew further away from their base. By a similar reasoning one deduces that if flight $i$ belongs to a pairing starting at base $b_2$, the $(i, k)$ connection is more likely than the $(i, j)$ connection.
Our prediction model does not compute directly connection probabilities. This is because \( p_{ijb} \) is often influenced by other flights with similar features as \( j \). For instance, if in addition to the flights in the example above, there was another flight \( l \) from \( a \) to \( b_1 \) departing on day 4 at 12:50, then it is likely that \( p_{ilb_1} > p_{ijb_1} \) since the \((i,l)\) connection is more efficient.

Our prediction model takes into account this interplay between similar flights. Let \( D_{ib} \) be a sequence of flights that are possible successors of flight \( i \in \mathcal{F} \), given that \( i \) belongs to a pairing starting at base \( b \in \mathcal{B} \), ordered in increasing order of departure time, and indexed by \( k \in \{1, \ldots, |D_{ib}|\} \). We formulate the prediction task as follows: Given that \( i \in \mathcal{F} \) belongs to a pairing starting at base \( b \in \mathcal{B} \), estimate the probability that \( i \) is followed by the \( k \)th flight of \( D_{ib} \). Let \( p_{ikb} \) denote this probability. This task can be seen as a multiclass classification problem where the classes for flight \( i \) are the flights of \( D_{ib} \). It is trivial to translate those probabilities into the desired format, by assigning \( p_{ijb} = p_{ikb} \), where \( j \) is the \( k \)th element of \( D_{ib} \), and \( P_{ij} = 0 \) if \( j \notin D_{ib} \).

### 3.2 Data and features

We use flight connection data from solutions of 42 CPP instances (1,285,751 flight connections in total). Those instances were created using the dataset published by Kasirzadeh, Saddoune, and Soumis (2017), that contains a set of flights for seven aircraft types and for a one-month period. We split the month into seven overlapping time windows of variable lengths (from four to seven days). Each instance contains the flights corresponding to one aircraft type and a given time window. Because of the overlapping between consecutive time windows, some flights appear in two different instances. We ensure that the test data does not contain duplicates of flights seen during training and validation. A solution for each instance was obtained using the ICG method proposed by Tahir, Desaulniers, and El Hallaoui (2019). We call those solutions the reference solutions to avoid confusion with the solution of the \( P^2CG_n \) method.

We initially defined \( D_{ib} \) as the sequence of all feasible flight connections for flight \( i \) within the next 48h following its landing. However, doing so results in \( D_{ib} \) containing on average 22 possible successors, with a maximum of 149. Preliminary results showed that the amount of data at our disposal is not sufficient to adequately train a prediction model on this number of classes. For this reason, we filter \( D_{ib} \) using spatial features (base airport, previous airport, and connecting airport) as well as temporal features (day and hour of departure from the previous airport, duration of the previous flight, and day and hour of arrival to the connecting city). For the training set, for each flight \( i \) connected to a flight \( j \) in the past solutions, we create a surjective mapping from the features of flight \( i \) to a list of possible next airports, containing the next airport of flight \( j \). For the test set, for each flight \( k \), the features of said flight are used to extract from the mapping the possible next airports, which is used to limit the number of candidate potential next flights. We detail in Table 1 and Figure 2 an example of using the filters, where we have two base airports (namely \( A \) and \( B \)) and 4 airports. It is assumed in the example that only the following features are used to define the mapping (filtering): the previous

![Figure 1: Example illustrating the need to compute different probabilities for each base.](image-url)
airport, the connecting airport, and the time of arrival to the connecting airport. To further simplify the example, the time of arrival has only two possible values (AM or PM).

Table 1: A sequence of pairings composing the training set. Curved arrows refer to flights, and dashed arrows refer to connections and rest periods. AM and PM are separated with a vertical line, and all pairings start in AM.

<table>
<thead>
<tr>
<th>Pairings</th>
<th>Sequence of flights</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A ↷ 1 → 1 ↷ 2 → 2 ↷ B</td>
</tr>
<tr>
<td>2</td>
<td>A ↷ 3 → 3 ↷ 1 → 1 ↷ A</td>
</tr>
<tr>
<td>3</td>
<td>A ↷ 1 → 1 ↷ 2 → 2 ↷ 4</td>
</tr>
<tr>
<td>4</td>
<td>B ↷ 2 → 2 ↷ 3 → 3 ↷ 4</td>
</tr>
<tr>
<td>5</td>
<td>B ↷ 3 → 3 ↷ 4 → 4 ↷ 3</td>
</tr>
<tr>
<td>6</td>
<td>B ↷ 3 → 3 ↷ 4 → 4 ↷ B</td>
</tr>
</tbody>
</table>

Figure 2: An example of using the constructed mapping from Table 1 to limit the number of candidates in the test set flight-network.

The average number of successors after filtering is given for each aircraft type in Table 4. Note that this filtering is performed for the purposes of training and evaluating the prediction model. All feasible flight connections are considered in our CPP solution method (see Section 4.2).

One entry is created for each flight that has a successor in the reference solution. A small fraction (less than 10%) of the flights have no successor in their reference solution, either because they end a pairing or because they are followed by a deadhead. No entry is created for those flights. Let \( b \) be the starting base of the pairing containing flight \( i \) in the reference solution. The entry for flight \( i \) is a \( |D_{ib}| \times t \) matrix, denoted \( X_i \), where the \( k \)th row of \( X_i \) represents the \( k \)th flight of \( D_{ib} \), and \( t \) is
the number of features per successor. The \( k \)th row (corresponding to flight \( j \in \mathcal{D}_{ib} \)) contains: \( b \), the features of flight \( i \), and the features of flight \( j \).

The features of the flights, as well as their type (categorical or numerical), are given in Table 2. Each categorical feature is represented by a different integer value. The features are arranged according to the order they appear in Table 2, so that similar features are grouped together.

The entry for flight \( i \) is given a label \( y_i \), corresponding to the row index of the successor of \( i \) in the reference solution. Note that since a heuristic is used to obtain the reference solution, \( y_i \) may not be the “true” label, i.e. it may not be the successor of \( i \) in any optimal solution. It is unfortunately impossible to obtain such true labels since no exact method can currently solve large-scale CPPs in reasonable times. However, the reference solutions have small optimality gaps (0.12% on average), making \( y_i \) a very good approximation of the true label.

### Table 2: Flight features.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Departure airport</td>
<td>categorical</td>
</tr>
<tr>
<td>Arrival airport</td>
<td>categorical</td>
</tr>
<tr>
<td>Base</td>
<td>categorical</td>
</tr>
<tr>
<td>Aircraft type</td>
<td>categorical</td>
</tr>
<tr>
<td>Departure date</td>
<td>numerical</td>
</tr>
<tr>
<td>Departure time</td>
<td>numerical</td>
</tr>
<tr>
<td>Arrival date</td>
<td>numerical</td>
</tr>
<tr>
<td>Arrival time</td>
<td>numerical</td>
</tr>
<tr>
<td>Duration</td>
<td>numerical</td>
</tr>
</tbody>
</table>

#### 3.3 Network structure

The first layer of the neural network is an embedding layer that encodes the categorical features. This is to prevent categories of the same feature with similar codes to be treated similarly by the prediction model. In the embedding layer, each feature is assigned a fixed number of neurons (up to 40). Along with reducing the representation dimension of a categorical feature, this technique allows for an encoding of the input based on similarity, where flights that have similar characteristics will have similar encodings. This similarity can be effectively exploited by the convolutional neural network. The encoding is learned in the training phase.

The output of the embedding layer is fed to a deep convolutional neural network. To determine an appropriate network architecture, we explore a large hyper-parameter configuration search space. We assign a range of values to the different hyper-parameters of the neural networks. We use a Gaussian process based on Bayesian optimization (see Dernoncourt and Lee 2017) with cross-validation in \( k \)-fold (\( k=7 \)) on the training data to look for the best possible configuration in the defined space. At each iteration, the Gaussian process aims to determine an architecture that maximizes the average accuracy (for all 7 aircraft types) of the neural network. The value ranges for all hyper-parameters are given in Table 3. The output layer of that network has a softmax activation function so that the output can be interpreted as an array of probabilities.

After identifying the best hyper-parameter configuration, we built 7 neural networks with the same architecture (one for each aircraft type). Since we have access to a very limited amount of data, the neural network for each aircraft type is trained using the data from the other six aircraft types.

#### 3.4 Computational experiments

We now assess the performance of the prediction model. All experiments were performed on a 40-core machine with 384 GB of memory, allowing several models to be evaluated in parallel. We implemented the Gaussian process described above using the Python GPyOpt library (version 1.2.1). Among 2500 architectures tested, we chose the one achieving the highest average accuracy. Note that many hyper-parameter configurations yielded an accuracy close to the optimal one.
Table 3: Hyper-parameters used in the optimization.

<table>
<thead>
<tr>
<th>Hyper-parameter</th>
<th>Search space</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimizer</td>
<td>adadelta, adam, adagrad, RMSprop,</td>
<td>Categorical</td>
</tr>
<tr>
<td></td>
<td>SGD, Adamax, Nadam</td>
<td></td>
</tr>
<tr>
<td>Learning rate</td>
<td>0.001, 0.002, · · · , 0.01</td>
<td>Float</td>
</tr>
<tr>
<td>Dimensions of the embeddings</td>
<td>5, 10, 15, · · · , 40</td>
<td>Integer</td>
</tr>
<tr>
<td>Number of dense layers</td>
<td>1, 2, 3, 4, 5</td>
<td>Integer</td>
</tr>
<tr>
<td>Neurons per layer</td>
<td>10, 60, · · · , 1000</td>
<td>Integer</td>
</tr>
<tr>
<td>Dropout rate</td>
<td>0.1, 0.2, · · · , 0.9</td>
<td>Float</td>
</tr>
<tr>
<td>Convolutional layers</td>
<td>0, 1, 2, 3</td>
<td>Integer</td>
</tr>
<tr>
<td>Number of filters</td>
<td>50, 10, 250, · · · , 100</td>
<td>Integer</td>
</tr>
<tr>
<td>Size of convolutional layer filters</td>
<td>2, 3, 4, 5</td>
<td>Integer</td>
</tr>
</tbody>
</table>

In our prediction model, if \( i \in F \) belongs to a pairing starting at base \( b \), its predicted successor is defined as \( j' = \arg \max_j \{ P_{bj}^i \} \). A standard way to measure the performance of a prediction model is by measuring its accuracy, i.e. the frequency at which the model makes the correct prediction. By extension, the top-\( k \) accuracy is the frequency at which the correct prediction is in the top \( k \) of the model’s prediction, ranked by decreasing value of probabilities. Measuring the top-\( k \) accuracy makes sense for our application because our CPP solution method still benefits from inaccurate predictions for which the right successor is ranked close to the first rank.

We present the top-\( k \) accuracy of the neural network trained for each aircraft type in Table 4. For each aircraft type, we also report the total number of flights in the dataset, the average number of possible successors per flight (after filtering), as well as the top-\( k \) accuracy for \( k \in \{1, \ldots, 7\} \).

Table 4: Top-\( k \) accuracy of the neural network.

<table>
<thead>
<tr>
<th>Aircraft Type</th>
<th>Nb. Flights</th>
<th>Avg. Nb. Successors</th>
<th>top-( k ) accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>727</td>
<td>1228</td>
<td>2.74</td>
<td>89.27</td>
</tr>
<tr>
<td>09</td>
<td>1826</td>
<td>2.41</td>
<td>92.98</td>
</tr>
<tr>
<td>94</td>
<td>2248</td>
<td>2.85</td>
<td>89.30</td>
</tr>
<tr>
<td>95</td>
<td>6796</td>
<td>4.9</td>
<td>78.27</td>
</tr>
<tr>
<td>757</td>
<td>6890</td>
<td>7.04</td>
<td>66.21</td>
</tr>
<tr>
<td>319</td>
<td>7070</td>
<td>4.67</td>
<td>79.70</td>
</tr>
<tr>
<td>320</td>
<td>9368</td>
<td>4.45</td>
<td>76.60</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>4.81</td>
<td>81.76</td>
</tr>
</tbody>
</table>

We first note that our prediction model achieves a 100% top-7 accuracy. Furthermore, a 99% top-3 accuracy is achieved for all but one aircraft type. The top-1 accuracy is significantly lower (81.76% on average). One possible explanation for this is that in many cases, all flights in the top 3 are all very good candidates that could be selected in an optimal or near-optimal solution. In such a case, the prediction model has virtually no way of knowing which of these candidates happened to be selected in the reference solution. Unfortunately, we have no way to verify this hypothesis since no objective definition of what is a good candidate exists (otherwise there would be no need to build a machine learning model!).

Finally, notice that the prediction model performs better for the aircraft types with fewer flights. This is because smaller instances have fewer feasible connections on average, making the prediction task easier.

4 Integral column generation with prediction

We now describe the improved ICG with prediction (\(I^2CG_p\)) algorithm used to solve the CPP. It is an ICG algorithm that takes advantage of the prediction model developed in Section 3. The core of
$I^2CG_p$ is the $I^2CG$ method developed by Tahir, Desaulniers, and El Hallaoui (2019). This algorithm is presented in Section 4.1. $I^2CG_p$ improves on ICG by performing partial pricing on a set of reduced SPs in order to generate columns that are likely to be selected in an optimal or near-optimal solution. The full SPs are used when a column generation iteration fails to sufficiently improve the best known solution. We describe both types of SPs in Section 4.2. In Section 4.3, we present results comparing $I^2CG_p$ with state-of-the-art solution methods.

### 4.1 Integral column generation

A flowchart for the $I^2CG_p$ algorithm is shown in Figure 3. It takes an initial solution $S_0$ as input. For instance, one could initialize the algorithm with a solution that uses an artificial variable with a high cost to cover each task. The outermost loop is the column generation loop, in which the RMP and the SPs are iteratively solved until a stopping criterion is met. The RMP corresponds to formulation (5)–(8) in which $\Omega$ is replaced by $\Omega' \subseteq \Omega$, the columns that have been generated since the beginning of the algorithm. Note that contrarily to branch-and-price methods, the RMP includes integrality constraints. The RMP is solved using the $I^2SUD$ algorithm proposed by Tahir, Desaulniers, and El Hallaoui (2019), based on the work of Zaghrouti, Soumis, and El Hallaoui (2014).

$I^2SUD$ iteratively solves a reduced problem (RP) and a complementary problem (CP). We call this loop the RP–CP loop. The basic idea of the RP–CP loop is to leap from an integer solution to a better one by performing pivots on multiple pairing variables at once. Such a pivot is called an integer pivot and the variables involved in this pivot form an integer descent direction in the solution space.

Let $S$ be the best known integer solution for the RMP and let $c_S$ be its cost. The RP is a version of the RMP in which $\Omega'$ is replaced by $U$, the subset of columns of $\Omega'$ that are compatible with $S$. Let $A$ be the coefficient matrix of the set-partitioning constraints (6), and let $A_j$ be the $j$th column of $A$. A definition of compatibility is given by Foutlane, El Hallaoui, and Hansen (2019) based on the work of Elhallaoui et al. (2011) as follows:

**Definition 1** A subset $U$ of $N$ is said to be compatible with $S$, or simply compatible, if there exist two vectors $v \in \mathbb{R}^{|U|}$ and $\lambda \in \mathbb{R}^{|S|}$ such that $\sum_{j \in U} v_j A_j = \sum_{l \in S} \lambda_l A_l$. The combination of columns, possibly a singleton, $\sum_{j \in U} v_j A_j$ is also qualified as compatible.

Let $C_S$ be the set of individually compatible columns with regards to $S$ (including those of $S$). The RP is defined as:

$$\min \sum_{p \in C_S} c_p x_p$$

s.t. $\sum_{p \in C_S} a_{fp} x_p = 1 \quad \forall f \in F$ (10)

$$\sum_{p \in C_S} u_{pbj} x_p \leq d_{bj} \quad \forall j \in J, b \in B$$ (11)

$$x_p \in \{0, 1\} \quad \forall p \in C_S$$ (12)

In practice, the RP is relatively small and is easily solved by commercial solvers. In general, the solution of the RP is a local optimum since there may be integral descent directions ( pivots) involving more than one columns that are incompatible with $S$. The goal of the CP is to find such a direction ( albeit not necessarily an integral one).

Tahir, Desaulniers, and El Hallaoui (2019) show that the presence of side constraints in the RMP can break the quasi-integrality property of the set partitioning problem. This property is very advantageous for primal methods because it ensures the existence of an integer path (passing only through integer extreme points of the polyhedron) between any integer extreme point and one corresponding to an optimal solution. To correct this issue, they add an artificial variable to the CP whose coefficients
correspond to the right-hand side of the RP’s constraints. By adding this variable, an artificial integer path between any two integer extreme points is created, thus restoring the quasi-integrality property.

Let $v_p$ be the weight of $x_p$ in the best descent direction returned by the CP, and let $v^A$ be the weight of the artificial variable in the best descent direction. The CP is formulated as:

$$
\min_{v, v^A} \sum_{p \in P \setminus S} c_p v_p + \sum_{p \in S} c_p v_p - c_S v^A
$$

(13)
Finally, constraints (17) ensure all variables are non-negative. Constraints (16) bounds the problem. Constraints (15) ensure that pivoting on this direction yields a solution that respects the set-partitioning and the crew availability constraints, respectively. Constraint (16) bounds the problem.

The objective function (13) minimizes the reduced cost of the chosen direction. Constraints (14) and (15) ensure that pivoting on this direction yields a solution that respects the set-partitioning constraints and the crew availability constraints, respectively. Constraint (16) bounds the problem. Finally, constraints (17) ensure all variables are non-negative.

Let $v^*, v^{*A}$ be the optimal solution of the CP, and let $z^{*CP}$ be its cost. Tahir, Desaulniers, and El Hallaoui (2019) show that a descent direction exists if and only if $z^{*CP} < 0$, and they explain how to construct such a direction. Furthermore, they show that this direction is integral if and only if $v^*_p \in \{0, v^{*A}\}$ $\forall p \in \Omega'$. The CP can, therefore, return with three possible outcomes:

i. An integral descent direction $d^I$ is found.
ii. A fractional descent direction $d^F$ is found.
iii. No descent direction is found.

In the first case, a pivot is performed on the entering and leaving variables of $d^I$ to create a new integer solution $S'$. The RP is then solved again using $C_{S'}$ instead of $C_S$. In the second case, the RP–CP loop exits and a zooming procedure (see Zaghrouti, El Hallaoui, and Soumis (2018)) is applied to attempt finding an integer descent direction $d^Z$ in the neighborhood of $d^I$. If this zooming procedure is successful, a pivot is performed on the variables of $d^Z$ to create a new integer solution. The last case indicates that there exists no descent direction compatible with $S$. The algorithm exits the RP–CP loop since $S$ cannot be improved without additional columns.

Note that the $I^2CG_p$ algorithm always exits the RP–CP loop when the CP does not find an improving integral direction (cases ii and iii). We also tested a version of $I^2CG_p$ in which the RP is solved again when the zooming procedure is successful (i.e. it finds an integral descent direction). However, preliminary tests showed this to be inefficient because of a tailing-off effect after a few RP–CP iterations. By exiting the RP–CP loop whenever the CP returns a fractional direction, new columns are more frequently added, resulting in a faster improvement of the solution.

After exiting the RP–CP loop (and potentially applying the zooming procedure), the algorithm checks if the best solution has improved by at least $\epsilon^{MIN}$ in the last column generation iteration. If so, a new column generation iteration is started using the reduced SPs. Otherwise, the full SPs are used instead. If maxFail successive iterations are performed without a sufficient improvement, the algorithm stops and returns the best solution found.

The RMP is solved using a multi-phase strategy similar to the one used in Tahir, Desaulniers, and El Hallaoui (2019). The goal of this strategy is to prioritize generating improving directions whose columns have a low degree of incompatibility with the current solution. The degree of incompatibility of a column corresponds to the number of pairings that must be broken so that the column becomes compatible with the current solution. For instance, if the current solution contains pairings $p_1$ and $p_2$, covering the fights $\{1, 2, 3\}$ and $\{4, 5, 6\}$, respectively, the new pairing $p_3$ covering flights $\{1, 2, 5\}$ has a degree of incompatibility of 2. The pairing $p_4$ covering flights $\{1, 2, 3, 4\}$ has a degree of incompatibility of 1 since only $p_2$ must be broken to make $p_4$ compatible. In phase $l$, only columns with a degree of incompatibility at most $l$ are considered in the CP. Starting with $l = 0$, the value of $l$ is iteratively increased until a descent direction is found or the maximum incompatibility value $l^{MAX}$ is reached.

\[
\begin{align*}
\text{s.t.:} & \quad \sum_{p \in \Omega \setminus S} a_{fp} v_p + \sum_{p \in S} a_{fp} v_p - v^A = 0 \quad \forall f \in \mathcal{F} \\
& \quad \sum_{p \in \Omega \setminus S} u_{pbj} v_p + \sum_{p \in S} u_{pbj} v_p - d_{bj} v^A \leq 0 \quad \forall j \in \mathcal{J}, b \in \mathcal{B} \\
& \quad \sum_{p \in \Omega \setminus S} \alpha_p v_p + \sum_{p \in S} \beta_p v_p = 1 \\
& \quad v \geq 0 \quad v^A \geq 0
\end{align*}
\]
4.2 Subproblems

The goal of the SPs is to find pairings with negative reduced costs. The SPs are formulated as SPPRCs, in which feasible paths correspond to legal pairings. We use a partial pricing strategy to generate new columns. Two sets of SPs are used: reduced SPs and full SPs. The reduced SPs are faster to solve because they contain fewer arcs. For this reason, they are solved first, and the full SPs are solved only when a column generation iteration fails to sufficiently improve the best known solution. Before describing the full SPs in Section 4.2.2 and the reduced SPs in Section 4.2.3, we first present the SPPRC and its solution method in Section 4.2.1.

4.2.1 The shortest-path problem with resource constraints

Let $G = (V, E)$ be a directed graph with $V$ a set of vertices (or nodes) and $E$ a set of arcs. The SPPRC is an extension of the shortest-path problem in which the goal is to find the shortest path that satisfies a set of feasibility constraints between a source and a sink node in $G$.

The SPPRC enforces feasibility constraints using resources. A resource is a commodity that is consumed on arcs and is bounded on nodes. Figure 4 depicts a simplified version of the CPP SPs to illustrate how resources can be used to impose feasibility constraints. We show how to impose a maximum of 480 minutes (8 hours) of work time to any duty. For the sake of simplicity, we assume the work time corresponds to the total flight time in this example. This rule can be implemented in the SPPRC using a single resource. The resource consumption of each arc is displayed above it. Flight arcs have a consumption equal to their flight time. Connection arcs have a null consumption since no work is performed during a connection. Rest arcs have a consumption of -480 to reset the resource value to 0 at the end of a duty, in a way that is explained below. The numbers in parenthesis below each node represents the value of the resource for the path from the source to that node. We impose the same resource bounds (0 480] at each node. Note that a path may be extended to a node even if it violates its lower bound, by increasing the resource consumption to match the lower bound.

First, consider the path (source – 1 – 2 – 7 – 8). Such a path is not valid because it contains a duty with 600 minutes of work. The resource prevents such path in the network because its value at node 8 would exceed the node’s upper bound. Conversely, the path (source – 1 – 2 – 3 – 4 – 5 – 6 – sink) represents a feasible pairing under the simplified rules, and respects all the resource bounds in the network. After flight $f_2$, 7 units of time have been consumed. Taking arc (4 – 5) would then result in a resource value of -1 at node 5, but its value is increased to 0 to match the lower bound. This resets the resource value to 0 at the beginning of the new duty.

In general, the SPPRC can contain multiple resources. The path cost is treated as an unbounded resource, and complicated cost functions may require several resources. The SPPRC can be solved
using the labeling algorithm proposed by Desrochers et al. (1992). A label is an array associated with a node that contains the resource consumptions of a partial path ending on that node. Let $R$ be the set of resources and let $p$ be a partial path ending at node $i$, with $\alpha^r(p), r \in R$, its consumption of resource $r$. The label for this partial path is denoted $L^p_i = (\alpha^1(p), \alpha^2(p), \ldots, \alpha^{|R|}(p))$. The labeling algorithm extends labels from the source throughout the network. When label $L^p_i$ is extended through arc $(i, j)$, a new label $L^p_j$ is created on node $j$, with $\alpha^r(p')$ computed using the extension function $E_{ij}(L^p_i)$. Note that even though extension functions with constant consumptions on arcs are common (such as in the example above), this might not be the case in general. A dominance procedure is applied to remove inefficient (non-Pareto optimal) labels at each node. Standard dominance rules have been designed for the case in which all extension functions are nondecreasing function of the resources, which is the case for the SPs in this paper. The reader is referred to Irnich and Desaulniers (2005) for a comprehensive description of the SPPRC.

### 4.2.2 Full subproblems

There is one full SP for each base and each day. The network structure for those SPs is the same as in Quesnel, Desaulniers, and Soumis (2017). Figure 5 displays the network corresponding to the SP for base $b \in B$ and day $d \in J$. Let $F_{bd}$ be the set of flights whose departure is in the interval $[d, d + \bar{d} - 1]$. The network contains a source node, a sink node, and three nodes for each flight in $F_{bd}$: a departure node, an arrival node and a waiting node.

There are nine types of arc. A beginning of pairing arc connects the source to the departure node of every flight that departs base $b$ on day $d$. Similarly, an end of pairing arc connects every arrival node to the sink. Two arcs connect each departure node to its corresponding arrival node: a flight arc and a deadhead arc. A connection arc connects the arrival node of flight $i$ to the departure node of flight $j$ if such a connection is feasible and shorter than $t^R$. Similarly, a short rest arc connects the arrival node of flight $i$ to the departure node of flight $j$ if the corresponding rest is shorter than $t^R$. Longer rests are made possible via waiting queues. For this purpose, the waiting nodes are grouped by airport and connected in a chronological order by waiting arcs to form a waiting queue. Let $i$ be a flight arriving at airport $a \in A$ and let $j$ be the earliest flight departing from $a$ such that a rest longer than $t^R$ can occur between $i$ and $j$. A long rest arc connects the arrival node of flight $i$ to the waiting node of flight $j$. Any feasible rest longer than $t^R$ can be represented by a long rest arc followed by a (possibly empty) sequence of waiting arcs, and an empty arc. Such a path is represented by the bold arcs on Figure 5.

Five resources are necessary to constrain the paths. The number of flights per duty resource ensures that each duty contains at most $f^{MAX}$ flights. Similarly, the number of duties resource ensures the pairings contain at most $d^{duty}$ duties. The length of duty resource imposes a maximum of $T^D$ minutes on the length of any duty, and the work time of duty resource ensures no duty contains more than $T^W$ minutes of work time. Two additional resources are used to compute each cost term of (2).

The SPs contain one additional resource, called cumulative encoded probability (CEP) that encodes the probability that a partial path will be part of an optimal or near-optimal solution. Its value is in the interval $[1, \infty)$, with a lower CEP indicating a more probable path. Including the CEP resource in the dominance rule will favor keeping some highly probable paths that would have been otherwise dominated by more efficient but less likely paths.

For each connection and for each base, we define an encoded probability (EP). The EP for connection $i-j$ and base $b$, denoted $\Psi_{ij}^b$, is computed as follows. Recall that $D_{ib}$ is defined in Section 3 as the set of successors of flight $i \in F$. Let $D_{ij}^+ = \text{seq}(j \in D_i | P_{ij}^b > 0)$ be the sequence of successors with nonzero probabilities, ordered by decreasing value of $P_{ij}^b$. Let $r_{ij}^b$ be the rank of flight $j$ in $D_{ij}^+$ and let $\text{classMax}_b = \max_{i \in F} |D_{ij}^+|$. We set $\Psi_{ij}^b = r_{ij}^b$ if $j \in D_{ib}^+$ and $\text{classMax}_b + 1$ otherwise. This means that the most probable connection following flight $i$ has an EP of 1, the second most probable one has an EP of 2, and so forth.

Recall that a connection can be performed either via a connection arc, or via a sequence consisting of a long rest arc, possibly one or several waiting arcs, and an empty arc. In the latter case, the incoming
flight information is stored in the labels on the waiting nodes, so that the EP can be computed when extending a label through an empty arc.

The CEP resource is initialized at 1 at the source node. When label $L_i^p$ is extended through arc $(i, j)$, the CEP of the new label $L_i^{p'}$ is computed as follows. If arc $(i, j)$ is either a connection arc or an empty arc representing connection $k - l$, the CEP extension function is

$$\alpha^{CEP}(p') = \alpha^{CEP}(p) \times E_{kl}^b.$$ 

This multiplicative extension function makes the CEP behave similarly to probabilities. For instance, a path containing two connections of EPs 1 and 5 should be roughly as likely as a path containing a single connection of EP 5 since in the former case, the first connection has a very high probability. For all other arcs, the CEP extension function is

$$\alpha^{CEP}(p') = \alpha^{CEP}(p)$$

i.e. the CEP remains unchanged.
Note that because the CEP resource is unbounded, it does not restrict possible paths in the network. Thus, the standard dominance rules remain valid.

A more straightforward way to define the EP would be to use $P^b_{ij}$ instead of $\Psi^b_{ij}$. However, preliminary tests showed that doing so leads to large computing times. This is because $P^b_{ij}$ is continuous, whereas $\Psi^b_{ij}$ is discrete. This means that when comparing labels, the CEP values are almost never equal when $P^b_{ij}$ is used. Conversely, more labels have an equal CEP value when $\Psi^b_{ij}$ is used, causing a larger number of labels to be dominated.

In order to properly compute the reduced cost of a pairing, the dual values associated with constraints (6)–(7) must be given to the SPs. In traditional column generation, the dual variables are obtained from the last CPP linear relaxation solved. However, in $I^2CG_p$, the CPP linear relaxation is never solved, so the dual values are not directly available. Instead, approximate values of the dual variables are obtained from the last CP solution (for more details, see Tahir, Desaulniers, and El Hallaoui 2019).

4.2.3 Reduced subproblems

The reduced SPs have the same network structure and resources as the full SPs. However, they contain a relatively small subset of the connection arcs. Recall that $P^b$ is the connection probability matrix for base $b \in B$, where $P^b_{ij}$ represents the probability that, given that flight $i \in F$ belongs to a pairing starting at base $b$, $i$ is followed by flight $j \in F$ in an optimal or near-optimal solution. Because there is no way to analytically compute $P^b$, we approximate it using the prediction model developed in Section 3. Flight connection arc $(i, j)$ is included in the reduced SP for base $b \in B$ and day $d \in D$ if and only if arc $(i, j)$ is included in the corresponding full SP and $P^b_{ij} > 0$.

4.3 Computational results

We now present computational results for the $I^2CG_p$. The parameter values of the CPP are displayed in Table 5. $I^2CG_p$ was implemented in C++ using the standard library. We start by comparing $I^2CG_p$ with two state-of-the-art algorithms in Section 4.3.1. We then highlight the contributions of the machine learning model to our solution method in Section 4.3.2. All experiments were conducted on a Linux computer with an Intel Core i7-6700 CPU clocked at 3.4 GHz, using a single core and a single thread.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t^C$</td>
<td>30</td>
</tr>
<tr>
<td>$t^R$</td>
<td>570</td>
</tr>
<tr>
<td>$d$</td>
<td>5</td>
</tr>
<tr>
<td>$d^{duty}$</td>
<td>4</td>
</tr>
<tr>
<td>$D^P$</td>
<td>720</td>
</tr>
<tr>
<td>$P^W$</td>
<td>480</td>
</tr>
<tr>
<td>$f^{MAX}$</td>
<td>5</td>
</tr>
</tbody>
</table>

4.3.1 Main results

We compare $I^2CG_p$ with two competing solution methods. The first one is the $I^2CG$ algorithm proposed by Tahir, Desaulniers, and El Hallaoui (2019). It is similar to $I^2CG_p$ with the two following differences:

- It always solves the full SPs.
- The CEP resource is replaced by a degree of incompatibility resource that restricts paths to those with a degree of incompatibility less than or equal to the amount allowed in the current phase of the multi-phase strategy.
The second method is a branch-and-price heuristic called diving heuristic (DH) that is frequently used in commercial software. The DH heuristic solves the linear relaxation of a CPP at each node of a branch-and-bound tree. Each branching decision consists of fixing to 1 the value of the pairing variable with the highest fractional value. The branch-and-bound tree is explored in a diving fashion and the algorithm stops when an integer solution is reached (no backtracking is performed). The SPs used in the DH heuristic are equivalent to the full SPs presented in Section 4.2.2, without the CEP resource. Variants of DH allows fixing several variables at each node. This speeds up the algorithm at the expense of the solution’s cost. Indeed, fixing more variables at each node typically leads to solutions of higher cost. In this paper, we want to show that \( I^2CG_p \) produces solutions of lower cost than DH. Therefore, we compare \( I^2CG_p \) with the DH variant leading to the lowest-cost solutions on average.

The results for \( I^2CG \), DH, and \( I^2CG_p \) are presented in Table 6. For each method, we display the elapsed time in seconds (Time), the relative gap between the best integer solution found and the value of the linear relaxation of the problem (Gap), the number of column generation iterations (Itr.), and the total number of columns generated (Cols.). The numbers in bold indicate the lowest gap and elapsed time for each instance. For \( I^2CG \) and \( I^2CG_p \), we also display the number of integer solutions found during the solution process (IntS). For DH, this number is always 1.

We first compare \( I^2CG_p \) with DH. For all instances, \( I^2CG_p \) is faster than DH, except for a few small instances for which the computing times are similar. Moreover, this speedup is more important for larger instances. Indeed, we observe a speedup of 2 or more in all instances with more than 400 flights, and a speedup of more than 10 in many cases. There are three main reasons for this. First, \( I^2CG_p \) often solves the reduced SPs instead of the full ones, which takes less time. Second, \( I^2CG_p \) performs fewer column generation iterations than DH. Third, the CEP resource favors pairings that are likely part of an optimal or near-optimal solution.

Regarding the cost of the solutions, we obtain solutions with lower gaps with \( I^2CG_p \) than with DH except for 7 instances for which both methods reach optimality. On average, the solutions produced with \( I^2CG_p \) have an average gap of 0.09% whereas the solutions produced with DH have an average gap of 0.29%. This is because DH has no recourse when it makes a poor branching decision. Conversely, \( I^2CG_p \) can easily recover from a poor solution obtained in a previous iteration.

On average, the solutions produced by \( I^2CG_p \) have a slightly lower gap than those of \( I^2CG \), with \( I^2CG \) producing a better solution than \( I^2CG_p \) in only one case. This similarity between solution costs is expected because the two methods are very similar. However, the average computing time of \( I^2CG_p \) is almost half that of \( I^2CG \). Again, the speed-up is more important for large instances. This difference in computing times is explained by the use of reduced problems in \( I^2CG_p \). The effects of using the reduced SPs in \( I^2CG_p \) are discussed further in Section 4.3.2.

### 4.3.2 Effect of the ML model

The ML model presented in Section 3 plays two roles in \( I^2CG_p \). First, it determines connection probabilities, which are used by the CEP resource. Second, it identifies undesirable flight connections that are removed from the reduced SPs. However, a large number of undesirable flight connections are removed by the filtering procedure described in Section 3.2 rather than by the ML model itself. In order to isolate the contribution of the ML model, we solve all instances using a modified version of \( I^2CG_p \), called \( I^2CG_f \), in which only the flight connections filtered out by the filtering procedure are removed from the reduced SP. The CEP resource is modified thusly: We set \( E^b_{ij} = 2 \) if connection \((i, j)\) is filtered out from a SP of base \( b \) and 1 otherwise. This identifies the filtered out connections as less probable than the filtered in connections. Note that this modified CEP resource is only used in the full SPs because filtered out connections are not included in the reduced SPs.

Results for \( I^2CG_f \) are presented in Table 7 (the results of \( I^2CG_p \) are reproduced for easy comparison). While both methods produce solutions of very similar cost (we obtain identical gaps for all instances but three), \( I^2CG_p \) is significantly faster than \( I^2CG_f \). This is because, in \( I^2CG_f \), paths cor-
responding to pairings that are well-suited for a near-optimal integer solution are often dominated in the first column generation iterations. Conversely, the CEP resource in the $I^2CG_p$ SPs prevents those paths from being dominated. This leads to a larger number of advantageous pairings being generated early in the solution process.

## 5 Conclusion

In this paper, we propose a new ICG method for the CPP that uses a machine learning model to predict the probability of flight connections. The predictions of the ML model are used to create a set of reduced SPs that are faster to solve compared to traditional approaches. We also add a resource to the SPs, making them more likely to generate pairings that belong in a near-optimal solution. We
show that our method is significantly faster and produces solutions of lower costs than other state-of-the-art algorithms. We highlight the contributions of the ML model to the method. Some of the novel ideas presented in this paper could easily be adapted to create new solution methods for related problems, such as the CRP. Finally, we show that a neural network trained offline can provide valuable insight and help speed-up optimization algorithms. This paper thus contributes to the growing body of literature on the ways ML can be used to help solving optimization problems.

References


