POLYPUBLIE

Gaining insight into crew rostering instances through ML-based sequential assignment

Philippe Racette, François Soumis, Frédéric Quesnel, Andrea Lodi

Document en libre accès dans PolyPublie
Open Access document in PolyPublie

Date	2024
Type	Article de revue / Article
Version	Version finale avant publication / Accepted version
URL de PolyPublie PolyPublie URL	https://publications.polymtl.ca/58706/
Évaluation par les pairs Peer reviewed	Révisé par les pairs / Refereed
Conditions d'utilisation Terms of use	© Tous droits réservés / All rights reserved
Référence Citation	Racette, P., Soumis, F., Quesnel, F., & Lodi, A. (2024). Gaining insight into crew rostering instances through ML-based sequential assignment. Top, 42 pages. https://doi.org/10.1007/s11750-024-00678-8

Publication officielle
Official publication

Titre de la revue Journal title	Top
Maison d'édition Publisher	Springer
URL officielle Official URL	https://doi.org/10.1007/s11750-024-00678-8
Mention légale Legal notice	This version of the article has been accepted for publication, after peer review (when applicable) and is subject to Springer Nature's AM terms of use, but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. The Version of Record is available online at: https://doi.org/10.1007/s11750-024-00678-8

Ce fichier a été téléchargé à partir de PolyPublie, le dépôt institutionnel de Polytechnique Montréal
This file has been downloaded from PolyPublie, the institutional repository of Polytechnique Montréal
https://publications.polymtl.ca

Gaining insight into crew rostering instances through ML-based sequential assignment

Philippe Racette[1*,3], Frédéric Quesnel[2], Andrea Lodi[3,4], and François Soumis[1]

[1]Department of Mathematics and Industrial Engineering and GERAD, Polytechnique Montréal, Montréal, Québec, Canada
[2]Department of Analytics, Operations and Information Technology, Université du Québec à Montréal, Montréal, Québec H2X 3X2, Canada
[3]Canada Excellence Research Chair in Data Science for Real-Time Decision-Making, Polytechnique Montréal, Quebec H3C 3A7, Canada
[4]Cornell Tech and Technion - IIT, New York City, New York, United States

*Corresponding author. E-mail: philippe.racette@gerad.ca

Abstract

Crew scheduling is typically performed in two stages. First, solving the crew pairing problem generates sequences of flights called pairings. Then, the pairings are assigned to crew members to provide each person with a full schedule. A common way to do this is to solve an optimization problem called the crew rostering problem (CRP). However, before solving the CRP, the problem instance must be parameterized appropriately while taking different factors such as preassigned days off, crew training, sick leave, reserve duty or unusual events into account. In this paper, we present a new method for the parameterization of CRP instances for pilots by scheduling planners. A machine learning-based sequential assignment procedure (*seqAsg*) whose arc weights are computed using a policy over state-action pairs for pilots is implemented to generate very fast solutions. We establish a relationship between the quality of the solutions generated by *seqAsg* and that of solutions produced by a state-of-the-art solver. Based on those results, we formulate recommendations for instance parameterization. Given that the *seqAsg* procedure takes only a few seconds to run, this allows scheduling workers to reparameterize crew rostering instances many times over the course of the planning process as needed.

Keywords: crew rostering, crew scheduling, discrete optimization, evolutionary algorithm, machine learning, reinforcement learning

Statements and Declarations

- **Competing Interests: None**

MSC code: 90-08 Computational methods for problems pertaining to operations research and mathematical programming

Acknowledgements

This work was supported financially by the Natural Sciences and Engineering Research Council of Canada and IBS under grant no. CRDPJ-477127-14. The authors are grateful for this support.

1 Introduction

Crew scheduling is a complex problem crucial to the operations of airline companies. From an economic point of view, it is the second source of expense for airline carriers, after fuel costs (see Schaefer et al. (2005)). As such, it has typically garnered interest from researchers seeking to improve its efficiency.

Crew scheduling is typically solved in two steps. First, it is necessary to solve the *crew pairing problem* (CPP). This problem involves creating pairings, defined as sequences of flights (also called airlegs) and layovers starting and ending at the same airport. They represent a few days of work for a crew member. The next step is to assign pairings to pilots (the case we consider) or the cabin crew. One possibility is to solve the *crew rostering problem* (CRP). The outcome is a full schedule per pilot over the month considered, and the entire collection of schedules makes up the roster for the month.

When solving the CRP, the scheduling worker must plan the CRP instance. This process involves a very large number of decisions of two types. First, there are decisions made sequentially and related to granting pilot requests. Requests made by pilots can be of many kinds and include asking for certain days off, further training (e.g., training to fly airplanes of different types), medical leave or work-related health screenings. Next, it is necessary to determine, of all available pilots at the company, how many will be on reserve for the upcoming month. Reserve duty refers to the situation where a pilot is on call for the month to fill in when pairings cannot be assigned. This is judged an undesirable situation for industry-specific reasons, such as the salary given for few hours worked. The number of pilots on reserve duty may change during planning as new requests are made by pilots.

Making these decisions occurs as the scheduling worker also aims to take pilot preferences regarding time off and pairing assignments into account. Note that those preferences do not need to be satisfied. They also consider special events on certain months such as Thanksgiving or Christmas holidays. This requires determining a set of parameters indicating what percentage of the requests should be granted as preassigned days off (i.e., specific days off explicitly requested by pilots in advance) and how many pilots should be on reserve.

Adequately parameterizing the instance for the upcoming month means finding good values for these parameters such that a maximum of requests is granted and a minimum of pilots is required to carry out flights on regular duty (i.e., not on reserve) while not affecting the latter's satisfaction or the feasibility of the schedules generated. Granting too many requests or putting too many crew members on reserve forces pilots actively covering pairings during the month to serve unwanted pairings. It may also mean that some pairings will remain unassigned during peak flight periods for the airline. This situation must be avoided as a priority while still taking pilot satisfaction into account. Overall, this state of matters means that planning a CRP instance is not a trivial task. Indeed, the parameterization process can involve hundreds of manual decisions when determining how to create a definitive instance. In practice, it may also have to be repeated many times until an instance is deemed satisfactory.

The impact of an incorrectly parameterized instance becomes obvious when many pilots become unavailable for one reason or another. Typically, such situations would include having many pilots gone for training or on personal leave, or an unusually high number of pilots having been granted time off during the same period, such as yearly recurring holidays. In those problematic cases, the CRP can be very difficult to solve, or even infeasible.

To prevent this situation, one option is to adjust the parameters each time a decision is made and to solve the CRP instance to see if pilot satisfaction or feasibility are affected. However, this method quickly becomes extremely time-consuming, as the mathematical programming solver used must generate a full roster each time the necessity of a change to the instance becomes apparent. For that reason, insight into the CRP instance allowing to change parameters without solving the entire CRP could be very useful. Unfortunately, obtaining this kind of reliable, instance-specific insight can be hard, especially for

inexperienced planners.

In recent years, machine learning (ML), including reinforcement learning (RL), has proven to be useful in extracting valuable information from existing data, including data pertaining to highly constrained problems like the CRP. In this paper, we propose integrating ML tools with optimization techniques to analyze CRP instances and improve their parameterization.

This paper proposes a new way of aiding scheduling planners to parameterize CRP instances. In particular, we do the following:

- propose an ML-based sequential assignment procedure called *seqAsg* that quickly builds a feasible roster. The *seqAsg* procedure relies on pairing assignment weights computed via an RL model;
- establish the relationship between the quality for both the solution generated by *seqAsg* and the solution obtained by a state-of-the-art solver;
- make recommendations for scheduling planners based on the quality of the solutions generated by *seqAsg*.

Together, these different contributions create a comprehensive method for checking iteratively, each time the scheduling worker makes a decision, if some changes need to be made to the instance. The *seqAsg* procedure takes only a few seconds to run even on large instances, making it suitable for instance analysis.

We now present the structure of the paper. A literature review on the CRP and the integration of ML and operations research (OR) is presented in Section 2. In Section 3, we give a formal definition of crew scheduling in general and of the CRP especially. In Section 4, we present the data used in the computational experiments. In Section 5, we describe the *seqAsg* algorithm, as well as the function approximators used to generate its weights. In Section 6, we measure the quality of the solutions generated by our algorithm, demonstrate the relationship between them and those obtained by the solver, and develop guidelines for scheduling workers. We conclude with a general discussion in Section 7.

2 Literature review

In this section, we review certain techniques that have been used to solve the CRP and crew scheduling problems more generally. We also cover the literature on several aspects of ML, particularly where ML methods have been relevant to solving combinatorial optimization (CO) problems.

2.1 Crew scheduling and column generation

The crew scheduling literature includes a large variety of different methods aimed at solving crew scheduling problems successfully. For a general overview of the main approaches involved, see Gopalakrishnan and Johnson (2005).

One well-studied problem in the crew scheduling literature is the crew pairing problem described in Desaulniers et al. (1997). This problem concerns building feasible pairings of high quality while covering all flights planned for the month. Once pairings are built, it is possible to form high-quality schedules using crew rostering. There is also the possibility of solving the crew bidding problem described in Boubaker et al. (2010). In that context, schedules are built from the provided pairing list, and pilots and flight attendants bid on them based on seniority.

In many crew scheduling problems, there are too many feasible pairings or schedules and it is impossible to list them all explicitly. For that reason, column generation is often used. Column generation involves iteratively solving a master problem and several subproblems (e.g., one per pilot), with columns here referring to feasible schedules. The available columns at an iteration form the *restricted* master problem (RMP) that is solved to find the optimal solution for this restricted set of columns. This provides a

dual solution whose variables are used to define the reduced costs provided to the subproblems. The schedules found by solving the subproblems can then be added to the RMP. When there are none, the current solution is optimal. See Lübbecke and Desrosiers (2005) for more on column generation.

Additional methods have been developed to tackle these scheduling problems as well. In Saddoune et al. (2012), the authors suggest an approach that solves both the crew pairing and crew rostering problems simultaneously with column generation. Similarly, in Zeighami and Soumis (2019), the authors bring forward an integrated model using Benders' Decomposition. In Zeighami et al. (2020), the integrated model focuses on Lagrangian relaxations. This family of approaches can be advantageous to solving pilot and copilot pairing and rostering problems with the objective of creating similar pairings even if rosters are different. These integrated modelling approaches can better accommodate personnel preferences.

The CRP has been the focal point of a growing body of literature as well. There are many variants of the CRP, and each favors different objective criteria. Some common criteria are the solution cost, crew satisfaction, fairness, seniority, or a combination of these points of consideration. While we present in this section some examples of CRP formulations applied to research problems, a more comprehensive review of the different CRP approaches can be found in Kohl and Karisch (2004).

With the work of Caprara et al. (1998), we can see that the authors use Lagrangian relaxations to solve the CRP in the context of the railway industry. They use the relaxations to assign all the fixed set of required pairings with the smallest number of crew members possible. In Gamache et al. (1999), the authors present a column generation approach and show that they can solve the CRP substantially faster (up to three orders of magnitude) while remaining within 0.6% of the optimal solution objective. In Cappanera and Gallo (2004), the authors feature a multicommodity formulation of the CRP for airline crew members and show the effectiveness of their method with instances of various sizes and natures using CPLEX. Another paradigm is introduced in El Moudani et al. (2001), where the authors consider a bi-criterion objective including both the cost and crew satisfaction as important variables to optimize. Using a genetic algorithm, they provide a set of solutions of non-inferior quality that can help with the full resolution of the CRP.

We also mention the aspect of the CRP we cover in more depth in this paper, i.e., the parameterization process that occurs when scheduling planners create CRP instances. This process has little published literature addressing it, as the sort of work that it entails is often highly customized depending on the airline and even individual planners. This being said, a notable exception is Beulen et al. (2020). In their work, the authors train a neural network to determine when crew members' flight requests should be granted. These decisions occur during CRP planning. They obtain that their method grants 22% more requests than had previously been the case for a major European airline. Nonetheless, the existing work on instance parameterization is very scarce and we draw attention to it with this paper.

2.2 Machine learning in operations research

When it comes to the different ways of aiding the resolution of CO problems with ML, there are three different paradigms. The first one is called *end-to-end* learning and involves the entire resolution of the problem through an ML learning solution without the help of a CO solver.

For example, in Vinyals et al. (2015), the authors introduce the concept of pointer networks, which are inspired by the attention mechanism for natural language processing described in Bahdanau et al. (2014). They use pointer networks to solve various CO problems. In Bello et al. (2016), the authors use a policy gradient RL algorithm to solve the traveling salesman problem (TSP), while in Nazari et al. (2018), the work presented expands on this contribution to apply a deep RL algorithm to the vehicle routing problem (VRP). Attention mechanisms are also used in Kool et al. (2019), where the authors develop a model with attention layers to improve the existing RL heuristics applied to the TSP and the VRP. For a more comprehensive review of the use of end-to-end learning on OR problems, including a

large variety of different tasks, we refer the reader to Kotary et al. (2021) and Karimi-Mamaghan et al. (2022).

The next type of ML-OR integration is *sequential* learning. In this case, a trained ML model produces information that is then passed to the solver before resolution. For example, in Bonami et al. (2022), the ML algorithm determines whether a mixed integer linear program should be linearized in CPLEX v. 12.10.0 before solving starts. Using a support vector machine model, they reduce computing times by a percentage ranging from 28% to 92% depending on the problem's size. In crew scheduling, sequential algorithms have primarily focused on giving elements of initial solutions to the solver. In Yaakoubi et al. (2020a), the authors use supervised learning to provide initial clusters of airlegs likely to go together in a pairing. They improve the CPP solution time by a factor of 10 with a 0.2% improvement of the objective. In Yaakoubi et al. (2020b), the authors use a similar approach with an integral primal simplex algorithm presented in Elhallaoui et al. (2011). As to crew rostering, in Quesnel et al. (2022), the authors use supervised learning to learn which pairings are likely to be in a flight attendant's final schedule and to reduce the size of the subproblem network. They obtain an objective of similar quality for the CRP in less than half the time.

The last type of ML application to OR algorithms is *integrated* learning, also known as online learning. In this framework, the ML algorithm intervenes throughout the OR algorithm. Among other examples, in Furian et al. (2021), the authors suggest a branch-and-price algorithm that makes use of online learning to determine binary decision variables for node selection and make predictions on fractional variables during branching. Their method makes significant gains in computing time for the VRP. In Morabit et al. (2023) and Morabit et al. (2021), the authors use ML to improve column generation by selecting promising columns or arcs among those generated by or present in the subproblems, respectively, at each iteration of the resolution of a subproblem. They save 40% and 30% of the computing time for the CPP, respectively. A similar approach is developed in Tahir et al. (2021), where only high-probability flight connections are included in the subproblems to be solved. They outperform state-of-the-art benchmarks for the CPP both in terms of objective and speed.

With regard to the CRP, some work has been done from an end-to-end point of view in work trying to address the problem with ML, while it has more often been treated from a sequential or integrated learning paradigm as in some of the contributions cited above. Nonetheless, some research, mostly using different kinds of heuristics, is worth mentioning. The work in de Armas et al. (2017) builds a multi-start heuristic alternating random and local searches to find fast high-quality solutions on small, industry-realistic CRP instances. For this purpose, they draw inspiration from Juan et al. (2013). In Zhou et al. (2020), the authors seek to optimize a CRP objective taking into account both fairness and satisfaction, and they present an ant colony algorithm that is then compared against other heuristics on the instances found in Kasirzadeh et al. (2017). In Zhou et al. (2022), the authors expand on this work and test their method on a small instance comprised of 6 pilots. These different contributions also build on the work, among others, found in Lučic and Teodorovic (1999), Lučić and Teodorović (2007), Boufaied et al. (2016) and Maenhout and Vanhoucke (2010). These contributions all consider different heuristics to solve different variants of the CRP more efficiently.

However, in light of the previous work mentioned and to the best of our knowledge, ML techniques have not yet been used to compare the solutions of such methods with state-of-the-art CRP solvers on large, realistic instances from an industry angle, and while also providing an ML solution in a few seconds only on such instances. Typically, we observe that either the instances considered are small, that they are not realistic in a real-life context, or, if the instances are both large and realistic, that they take several minutes to solve, even when the technique cuts the solving time in half or more. This paper can then be seen as presenting a way to obtain very fast solutions on large, difficult, and industry-realistic instances through ML so as to infer useful information regarding the CRP's planning phase.

We note that the work presented in this paper can be viewed as belonging to both the end-to-end and

integrated learning paradigms. ML solutions are generated without the use of a traditional CRP solver: however, the ML weights learned by the algorithm we present in Section 5 intervene at each step of the *seqAsg* procedure.

3 Crew rostering problem

We now describe the variant of the crew rostering problem considered in this paper. It is inspired by the work presented in Kasirzadeh et al. (2017), with some adaptations to fit the current version of the CRP coded in the solver we use in Section 6. Consider the set of pairings W and the set of pilots K. The goal is to assign the pairings in W to pilots in K to build one feasible schedule per pilot. The schedules are built over a monthly horizon. Also, each pilot in K is attached to one of d bases, and the problem is solved separately by base.

Schedules must comply with collective agreements and airline regulations. In this paper, we consider the following schedule restrictions:

- each pairing must be assigned to exactly one pilot,
- each pilot must receive a full schedule (complete sequence of pairings and days off),
- each pilot must be assigned a minimum of T^{off} days off, which may include preassigned days off,
- pilots cannot work over $T^{credits}$ hours in a month,
- pilots cannot work more than T^{work} consecutive *duties* (calendar days with at least one ongoing pairing),
- there must be at least T^{min} hours between two consecutive pairings, and
- all preassigned days off must be respected.

The objective function for the CRP, given below by (3.1), maximizes an expression with three terms. The first term maximizes pilot satisfaction. The satisfaction a pilot derives from their schedule is a function of the pilot's preferences. Prior to solving the CRP, pilots state their preferences for certain flights (that can be part of any pairing) and certain vacations (three consecutive days off). Furthermore, they assign a weight to each preference. A schedule's satisfaction score for a given pilot is the weighted sum of the pilot's preferences that the schedule meets. The other two terms subtract penalties for failing to cover some pairings, or for not respecting preassigned days off. The penalty for an unassigned pairing (i.e., a pairing that has not been assigned) increases by C^{flight} points for each airleg included in the pairing (e.g., the penalty for an unassigned pairing with six airlegs is $6 * C^{flight}$ points). Uncovered preassigned days off are heavily penalized, at C^{day} points per day. This value is selected to be sufficient to ensure that all such preassigned days off are covered when solving the CRP.

We note that, due to the penalties described above, while the requirements related to pairing assignments (which are also referred to as coverage requirements) and preassigned days off are part of the CRP, the violation of these constraints does not make a schedule (or the corresponding roster) infeasible and is instead penalized. However, any other violated restriction makes a schedule infeasible.

3.1 Mathematical formulation

The CRP is defined as follows. Let Ω and Ω^k be the set of feasible schedules, and feasible schedules for pilot $k \in K$, respectively. Let x_p^k be a variable equal to 1 if schedule $p \in \Omega^k$ is assigned to pilot k, and 0 otherwise. Let Q^k be the set of preassigned days off for pilot $k \in K$. Let y_w and y_q be the slack variables ensuring the coverage of pairing $w \in W$ and preassigned day off $q \in Q^k, k \in K$, respectively. Let c_p^k be the cost (satisfaction score) of schedule $p \in \Omega^k, k \in K$. Let $n_w, w \in W$ be the number of airlegs in

unassigned pairing $w \in W$. Finally, let a_{wp} and a_{qp} be binary constants indicating whether pairing w or preassigned day off q are in schedule $p \in \Omega^k, k \in K$, respectively. The CRP reads as

$$\max \sum_{k \in K} \sum_{p \in \Omega^k} c_p^k x_p^k - C^{flight} \sum_{w \in W} n_w y_w - C^{day} \sum_{k \in K} \sum_{q \in Q^k} y_q \tag{3.1}$$

subject to:

$$\sum_{k \in K} \sum_{p \in \Omega^k} a_{wp} x_p^k + y_w = 1, \quad \forall w \in W \tag{3.2}$$

$$\sum_{p \in \Omega^k} a_{qp} x_p^k + y_q = 1, \quad \forall q \in Q^k, \forall k \in K \tag{3.3}$$

$$\sum_{p \in \Omega^k} x_p^k = 1, \quad \forall k \in K \tag{3.4}$$

$$x_p^k \in \{0, 1\}, \quad \forall k \in K, \forall p \in \Omega^k \tag{3.5}$$

Equations (3.2) and (3.3) ensure that each pairing and preassigned day off are covered exactly once, respectively. Equation (3.4) ensures that each pilot has exactly one schedule.

Formulation (3.1) – (3.5) typically contains a large number of variables. Therefore, it is often solved using a heuristic branch-and-price algorithm. In that framework, relaxations of (3.1) – (3.5) are solved using column generation. The column generation algorithm is embedded in a heuristic branch-and-bound algorithm. Such a commonly used heuristic fixes some pairing variables at each node until an integer solution is obtained.

4 Data

Our data is derived from the data sets published in Kasirzadeh et al. (2017). It consists of seven instances of different sizes for which the airlegs (i.e., flights), pilot bases, and airports are given. The instances are labeled I1 to I7 in increasing number of flights, and each instance has $d = 3$ bases (see Table 1). For each instance, a state-of-the-art CPP solver builds pairings (see Quesnel et al. (2020a)). This data set was originally chosen due to the diversity of the seven instances. It includes instances with different proportions of short-haul and medium-haul pairings lasting from 1 to 5 days. These proportions are such that the difference in composition between instances is very large. For example, for instance I2, the percentage of pairings of length ≥ 4 days is 1.8% for the base with the highest percentage. For instance I1, this goes up to 66.7%. Given this fact, each instance being quite different from the others, an ML algorithm trained on one instance and generalizing well to the other six shows robustness on a variety of instance types.

As the data set does not contain pilot-related information, pilot-related data was randomly generated using a procedure adapted from Quesnel et al. (2020b) that aims to create preferences similar to that of an international airline based in North America. First, a percentage of pilots (uniformly chosen between 5% and 15%) receive randomly four, six, or seven consecutive preassigned days off. Then, each pilot is given three or four three-day preferred vacations. These vacations are generated uniformly over the month with the requirement that they be non-overlapping with preassigned days off or with each other. As to flight preferences, pilots in real-life scenarios tend to choose airlegs that mark the beginning or the end of a pairing, as mentioned in Quesnel et al. (2020b). Each pilot is given between zero and ten such flights, inclusively.

The weights in the objective that are associated with pilot preferences are as follows. For each preferred airleg or vacation, a weight ranging between 10 and 400 points is added to the objective. Preference

Instance	Nb pilots	Nb airports	Nb airlegs	Nb pairings
I1	27	26	1013	157
I2	27	35	1500	331
I3	37	41	1854	359
I4	117	49	5613	1038
I5	239	34	5743	1222
I6	180	52	5886	1080
I7	322	54	7765	1473

Table 1 Description of the instances used

weights are multiples of 25 points, with the exception of some flights worth 10 points. Preferred vacations must also be worth at least 75 points. All preference weights are generated randomly following two uniform distributions: one for flights and the other for vacations. All possible values for a flight or preferred vacation have equal probabilities. As mentioned in Section 3.1, the total satisfaction score of a pairing-pilot assignment is the weighted sum of all preferences for flights included in the pairing.

Preference weights are generally below 150 points for specific airlegs, whereas they often go to the upper end of the weight range for three-day vacations. Given that pairings are made on average of over six airlegs (and almost always have at least two), this means the penalty for missing a pairing is almost always considerably higher than not assigning a preferred pairing to a pilot. The cost of not assigning a pairing to anyone is also usually higher than the loss of satisfaction from not assigning a three-day vacation. The opposite case occurs only rarely, i.e., when a very short pairing is not assigned to allow for the allocation of a highly prized vacation. On these few occurrences, this is considered an acceptable compromise so as to avoid taking away what may be a pilot's greatest source of satisfaction for the month.

As we aim to learn from data sets with an ML-based procedure, we need more than seven instances to train the ML model. As such, we build a new method for obtaining more data. For each instance, we create different scenarios with each representing a different set of pilots and associated preferences. We generate 30 different scenarios for each instance. The scenarios are obtained with the help of software that takes a fixed set of pairings, a fixed number of pilots, possible preferred flights, and preassigned days off as input (i.e., those of the original instance). A file containing the parameters for the different uniform probability distributions is also provided, as shown in Table 2. Omitted from this table is information related to the distributions for preference weights, as their software implementation is less straightforward. Drawing from the possible flights, preferred vacations, and sequences of preassigned days off, as well as the parameter file, the software then creates a random scenario by assigning certain flights and vacations as preferred to each pilot and giving a percentage of them preassigned days off. In practice, this means that scenarios are close to being permutations of each other. However, this is not quite the case, as not all preferences appearing in one scenario do in the other and vice versa due to sampling.

Parameter	Possible values
Fraction of pilots with preassigned days off	$\{0.05, 0.10, 0.15\}$
Number of consecutive preassigned days off	$\{4, 6, 7\}$
Number of preferred vacations	$\{3, 4\}$
Number of days per preferred vacation	3
Maximum number of legs per employee	10

Table 2 Parameters for scenario generation

Once the scenarios are generated, we can build an input file for each one. Each input file is passed to the GENCOL software (version 4.5), where we implement a branch-and-price algorithm for the CRP. GENCOL is a state-of-the-art solver similar to the ones used in commercial planning software. The solutions produced by GENCOL are used as a point of reference in Sections 6.2 and 6.3. The CRP constants used in our experiments, as defined in Section 3, are given in Table 3.

Constant	Value	Description
T^{off}	10	Min. number of days off
T^{credit}	85	Max. credited hours per pilot
T^{work}	6	Max. number of consecutive duties
T^{min}	12	Min. post-pairing rest time (hours).
C^{flight}	100	Penalty for unassigned flight
C^{day}	1000000	Penalty for preassigned days off

Table 3 CRP constants

The penalty for preassigned days off is high to the extent shown as preassigned days are enforced by collective agreement regulations. A high penalty ensures that these coverage constraints are met.

In a real-life application, data could be obtained by looking at historical rosters over several months or years for different instances. In such a situation, the rosters would be more closely related than the scenarios we generate since the majority of the pilots remain from one month to another. For this reason, our prediction task is likely harder than it would be in real life. Additionally, in cases where no historical data exists and where ML weights learned at a previous time as in Section 5 are not available, it may be possible to use the scenario generation procedure outlined above on a single instance. We explore this possibility further in Section 6.2.

5 Algorithm

Our method gains insight into a CRP instance by quickly generating a CRP solution and then analyzing it. The solution generation procedure has two steps. First, a sequential assignment procedure (called *seqAsg*) quickly produces a schedule for each pilot. This solution may be infeasible. In that case, a heuristic correction procedure is applied. We describe the sequential assignment procedure in Section 5.1 and the feasibility heuristic in Section 5.2. The algorithm *seqAsg* relies on a set of weights that are produced by an ML model trained on the data described in Section 4. We describe the ML procedure used for this purpose in Section 5.3.

5.1 Sequential assignment

The sequential assignment procedure aims at providing a schedule for each pilot quickly (i.e., within a few seconds). Here, an assignment is defined as adding a specific pairing, day off (whether preassigned or not), or preferred vacation, to the schedule of a pilot ready to work on a certain day. An assignment problem seeks to assign all required pairings and preassigned days off on a certain day to pilots and to give all pilots required to receive an assignment on that day exactly one pairing, day off, or preferred vacation. Some pilots available to work, but not required to, may also be assigned an activity. The sequential assignment procedure iteratively solves an assignment problem for each day of the month, in chronological order. Each problem assigns pairings and days off to pilots. The sequential assignment procedure can thus be viewed as a constructive heuristic.

The assignment problem for day t can be described as follows. Let W^t be the set of pairings to be covered. Let P^t be the set of pilots able to resume work on day t and whose last assignment ends on

day t. Let Q^t be the set of pilots able to resume work on day t and whose last assignment ends on the day just before t. Let A^t be the set of possible assignments. Days off and preferred vacations are not part of W^t as it is not strictly required to cover them, with the exception of preassigned days off, which we address later. However, they can still be assigned to pilots: these assignments are part of A^t. The distinction between P^t and Q^t is important, as the pilots in the former may receive an assignment on day t but are not obligated to, while the pilots in the latter must receive an assignment. Pilots that are not in either set are excluded from the problem.

To build the set A^t of valid assignments, we consider the following criterion. The assignment of activity (pairing, day off, three-day vacation) j to pilot i, denoted (i, j) is valid if :

- the pairing starts after the pilot has completed their last assignment (with sufficient post-pairing rest in the case of a pairing arrival if the previous activity and j are both pairings),
- if j is a pairing, it does not cause the pilot to work more than T^{work} consecutive working days, and
- it does not overlap with a preassigned day off, in the case of pairings or preferred vacations.

Let assignment variable $x_{ij}, (i, j) \in A^t$ be equal to 1 if pairing, day off, or three-day vacation j is added to the partially built schedule for pilot i. Let w_{ij} be the weight of such an assignment, corresponding to a measure of its desirability. Let slack variable y_j be equal to one if pairing $j \in W^t$ remains unassigned.

The assignment problem at day t then reads as

$$\max \sum_{(i,j)\in A^t} w_{ij} x_{ij} \tag{5.1}$$

subject to:

$$\sum_{i:(i,j)\in A^t} x_{ij} + y_j = 1, \quad \forall j \in W^t \tag{5.2}$$

$$\sum_{j:(i,j)\in A^t} x_{ij} \leq 1, \quad \forall i \in P^t \tag{5.3}$$

$$\sum_{j:(i,j)\in A^t} x_{ij} = 1, \quad \forall i \in Q^t \tag{5.4}$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in A^t \tag{5.5}$$

Here, constraint (5.2) ensures that each pairing is covered, while constraints (5.3) and (5.4) ensure that pilots available at day t are assigned to exactly one pairing, day off or preferred vacation or remain idle for those pilots whose last arrival was on day t, as expressed by the inequality in constraint (5.3). Note that the slack variables y are not penalized in the objective function.

In our tests, we solve an assignment problem that differs slightly from (5.1) - (5.5). Indeed, we add constraints to ensure that as many pairings as possible are assigned. Let G be a bipartite graph representing the feasible assignments of pilots to pairings, and let there be a maximum matching in G. The additional constraint then reads:

$$\sum_{j\in W^t} y_j = |W^t| - \nu(G) \tag{5.6}$$

where $\nu(G)$ is the size of the maximum matching discussed here.

It is worth observing that the weights w_{ij} are learned through ML, which we justify in Section 5.3.3. These weights should reflect not only the pilot preferences, but also the need to assign each pairing to a

pilot, as well as the need for each pilot to have a valid schedule. For example, a bigger weight may be given to a pairing that is harder to assign. The ML model trained to learn these weights is described in detail in Section 5.3.

Another way to represent the sequential assignment problem for day t is to set up a bipartite graph for each day. In that graph, arcs link pilots to the pairings or days off that can be assigned to them, with arc weights determining the desirability of a pilot-pairing assignment. In that case, W^t, P^t and Q^t represent pairing nodes, nodes for pilots available to work after the previous assignment ended on day t and pilots available to work after the previous assignment ended the day before t, respectively, while A^t represents possible assignment arcs. Pilots that are not available on day t are also represented by nodes that are not linked to any arc, and arc capacities are all exactly one. As this formulation is an example of a max (network) flow problem whose demands are all integers, solving the linear relaxation of the problem always provides an integer solution. This considerably simplifies its resolution. Figure 1 represents this conceptualization of the sequential problem solved by the *seqAsg* procedure. To be complete, each graph would need additional flow arcs: input arcs for pilots and output arcs with flow set to 1 exactly for pairing nodes. As input arcs vary by pilot, we do not represent input or output arcs. In Figure 1, we also represent imaginary pilots to account for pairings that are left unassigned. Satisfying the requirement that these pairings be assigned by selecting arcs associated with virtual pilots is equivalent to setting the slack variables for the unassigned pairings to 1. The input flow on nodes for those virtual pilots is equal to the number of pairings that cannot be assigned on the day considered.

Each assignment problem is solved with the goal of maximizing the weighted sum of the selected assignments. Once this is done, the pairings, days off, or preferred vacations associated with the selected assignments are added to the schedules of the corresponding pilots. Then, the procedure moves on to the following day and solves the next assignment problem with the pilots available on that day (i.e., those not currently carrying out a previously assigned pairing or preferred vacation or still recovering during post-pairing rest). Those steps are then repeated until the end of the month, at which point each pilot has a complete schedule.

The *seqAsg* procedure does not strictly enforce the limit for the total number of credit (i.e., working) hours per pilot over the entire month, as assignment decisions are done on a day-to-day basis. We choose this approach as enforcing a hard limit on credit hours in the sequential assignment procedure could prevent assigning a highly desirable pairing in a later graph. The drawback of this is that following the *seqAsg* procedure, some schedules may exceed T^{credit} hours of flight time. We employ two strategies to prevent this. First, the weights computed by the ML model consider the pilot's accumulated work time so as to discourage assignments that exceed the credit limit or cause future high-quality assignments to do so. Second, a feasibility heuristic later corrects any remaining violations.

Producing a schedule with the *seqAsg* procedure is much faster than solving the CRP using branch-and-price. The quality of the obtained solution is highly dependent on the performance of the learning algorithm presented in Section 5.3.

5.2 Feasibility heuristic

The next component of the methodology presented in this section is the heuristic that ensures each schedule within an ML-generated solution is feasible. While the majority of such schedules will generally be feasible, a minority of them are expected to require some corrections. There are two possible types of infeasibility. First, some pilots may have been assigned too many credit hours. Second, it is possible that not enough days off have been assigned to some pilots.

In order to make each schedule feasible, each source of infeasibility is addressed in the following order. First, the algorithm tackles pilots with too many credits. To correct the schedules for these pilots, pairings are removed until there is no excess credit. Specifically, the algorithm first tries to repair the

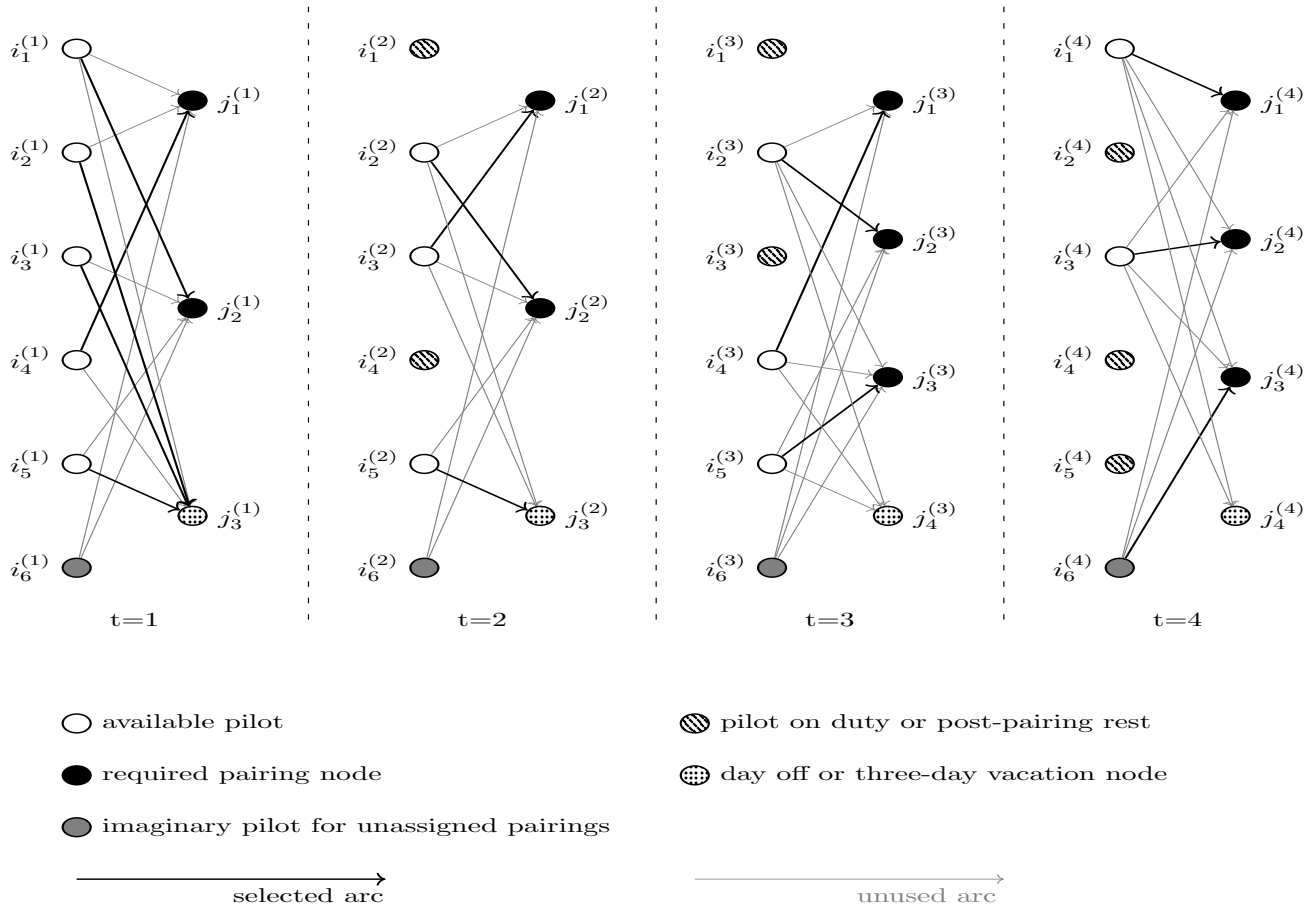


Fig. 1 Visual representation of *seqAsg*. Sequential assignment of pairings and days off for available pilots

schedule by removing one or two pairings in a way that minimizes the corresponding penalty for leaving a pairing unassigned (i.e., $n_w * C^{flight}$, as presented in Section 3.1). If that fails, it instead removes pairings one by one, in increasing order of penalty, until there remains no excess credit. Next, if after these operations, some schedules still contain too few days off (i.e., fewer than T^{off}), pairings are removed from those schedules, in increasing order of penalty, until there are enough days off allocated to the pilot. After this step, the schedule for each pilot is feasible, as is the entire roster. An example of pseudo-code for the heuristic can be found in Appendix A.

Also, at the end of this process, it may be possible to re-assign some unassigned pairings that have been excluded from a pilot's schedule. Indeed, in a minority of cases, such pairings may fit within another schedule and be directly inserted into it. This can happen when both steps of the heuristic have been applied to make a schedule feasible so that short, unassigned pairings can fit. However, this is an uncommon situation, especially for instances that were well planned, where very few pairings are left unassigned. When some short pairings can in fact be reinserted into a schedule, there are too few of them to get a significantly better solution. For these reasons, this step has not been implemented.

5.3 ML procedure

The *seqAsg* procedure relies on good values of the weights w_{ij} to correctly estimate the quality of the assignment matching pilot i with pairing, day off, or preferred vacation j. Those weights are computed using either a linear or neural network function approximator. The function approximator makes it possible to define an RL policy whose parameterization is learned by an evolutionary algorithm named

CMA-ES. We present the general framework for the ML model and RL policy in Section 5.3.1, detailed information about the ML models selected in Section 5.3.2, the reward function used during learning in Section 5.3.3, and describe the CMA-ES algorithm in Section 5.3.4.

5.3.1 ML framework

In order to learn a policy estimating the quality of the assignment of a pairing, day off, or preferred vacation to a pilot and to determine the parameters of the sequential assignment problem presented in Section 5.1, we define a learning environment. We also show the sequence of steps to complete one ML training iteration within it.

The environment is defined by states representing the assignment graph at the current time step and a matrix of feature vectors. These feature vectors provide information about the partially completed schedule for each pilot. Actions are defined by selecting one solution for the assignment problem on the current day: the action space is then the set of all possible solutions for that problem. State-action transitions set up the next assignment problem by making feature vector updates based on the pairings, days off, and preferred vacations assigned to the pilots, adding allowable assignment arcs to the new graph and computing their weights. The new state is then an updated assignment graph and feature matrix. As the assignment problem on day t is fully determined by the state of the environment, the action on that day will always be to select the solution produced by *seqAsg*, as it maximizes the total quality for the assignments made.

To perform an action within the environment, we need a function approximator to compute assignment weights on the arcs of the graph for day t. This function approximator intervenes during state-action transitions as new assignment graphs are set up and ensures that the *seqAsg* procedure can solve the assignment problem, thus performing the action needed. Function approximators must be trained and their parameters learned.

We draw a distinction between the arc weights in each assignment problem and the function approximator's parameters. The arc weights are recomputed at each step of the sequential assignment procedure, i.e., each time we set up the graph for the next assignment problem. The function approximator's parameters, on the other hand, are only updated after a batch of parallel *seqAsg* runs (see below), and only during training iterations. They are used to compute the arc weights.

We also distinguish the function approximator's parameters from the CRP instance's parameters. The CRP instance's parameters, which we want to help scheduling workers adjust as is appropriate and possibly a very large number of times, are related to the number of pilots who are granted specific requests or placed on reserve. They are separate from the function approximator's parameters.

Once the learning environment is defined as above and a function approximator has been chosen, training the function approximator follows a well-defined sequence of steps. Before we introduce these steps, we clarify that we call a *parallel run* a group of *seqAsg* runs for different ML weight vectors tested simultaneously on one scenario; a batch is the set of parallel runs done on each training scenario. Then, we have that

- First, as per the CMA-ES algorithm (see Section 5.3.4), a fixed number of noisy parameter vectors are generated by introducing noise into the approximator's parameters at the current time step.
- Next, each set of noisy parameters is associated with one full run of the *seqAsg* procedure. During each run, one set of noisy parameters is used with the relevant feature matrix to compute assignment weights. This builds a distinct solution associated with each parameter set.
- Then, a reward function (see Section 5.3.3) scores each roster. The best noisy parameter vectors are those that, through *seqAsg*, generated the rosters ranked the highest by the reward function. This completes a parallel run of *seqAsg*, which is repeated for each training scenario.

- Finally, the original approximator's parameter vector is updated by recombining the best noisy parameters averaged over the entire batch. The search space for the next iteration is updated as well.

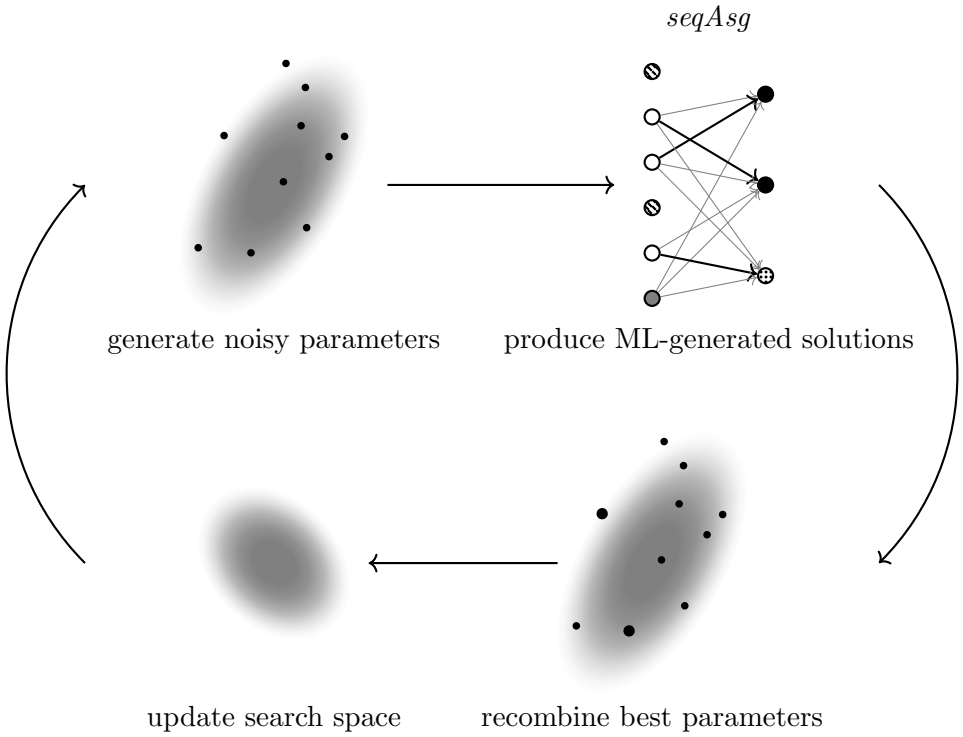


Fig. 2 ML steps per training iteration

The overall sequence is depicted in Figure 2. We consider that one full pass over the data for the instance takes m iterations, or one per scenario. Additionally, this sequence of steps only applies to ML training. When testing the models learned during training, only the *seqAsg* procedure comes into play by taking the models to generate solutions.

5.3.2 ML models

We consider two types of function approximators. One is based on a so-called linear function approximator, where a weighted recombination of features scaled to be between zero and one is used. The other is a neural network approximator. Whichever function approximator is used, it is called each time the quality of the assignment of a pairing, day off, or preferred vacation to a pilot must be calculated. However, the architecture and parameters of the function approximator are fixed during a run of *seqAsg*: only the input given to the approximator varies between possible assignments.

Neural networks are made of nodes (i.e., neurons) and arcs connecting them. There are parameters on the arcs that determine the signals that the nodes on each layer receive. The signals received by the nodes on each layer are processed by an activation function and the output (final) layer gives an action to be made in the environment. The parameters must be learned, for example using stochastic gradient descent or an evolutionary algorithm. See Higham and Higham (2019) for an in-depth discussion of neural networks.

For the neural network approximator, we use a hyperparameter search to determine the number of hidden layers and hidden units on each layer. The rectified linear unit (ReLU) activation function was chosen to process the weighted input signal at each neuron of the hidden and output layers. This function is

defined as: $f : \mathbb{R} \to \mathbb{R}$, $f(x) = \max(0, x)$ and thus keeps the positive part of the signal.

Each function approximator requires features as input. Traditionally, ML input is a collection of data points. For the purpose of the ML model, a data point is a pilot-pairing pair, with information about the partially built schedule for the pilot at time step t.

Features are related to pilot preferences, pairings, days off, and the overall composition of the partially built schedule for each pilot. The features passed as input are as follows:

- the cumulative number of days off assigned to pilot i before day t (numerical),
- the number of consecutive days off or work days for pilot i before day t, respectively (numerical),
- the satisfaction brought to pilot i by assignment j (numerical),
- the number of credit hours or duties associated with an assignment (both zero for non-pairing assignments), respectively (numerical),
- a binary feature checking if the assignment is a day off (preassigned or not) or preferred vacation (categorical),
- two binary features testing if there are enough credits left for an assignment in the schedule for pilot i at day t, and for future preferred pairings of weights ≥ 75, with or without a 15-hour buffer, respectively (categorical),
- two binary features checking if an assignment conflicts with the next preference for pilot i, or ends the day just before it (categorical), and
- a feature assessing whether a pairing, day off, or preferred vacation is valid within pilot i's schedule, i.e., no conflict or the assignment is worth at least as much as the next preference (categorical).

A bias term is also included as an additional feature.

5.3.3 Reward function

Let r_k for pilot $k \in K$ be the value of the total satisfaction accumulated over a full schedule and let v_k be the penalty incurred if the schedule of pilot $k \in K$ is infeasible. Then, the total reward R for a roster generated by *seqAsg* during an ML training iteration can be expressed as

$$R = \sum_{k \in K} (r_k - v_k), \quad k \in K, \text{ the set of pilots for a given instance.} \tag{5.7}$$

In our tests, we used $v_k = 1000$, which is a good compromise between favoring improvements in satisfaction and reducing the infeasibility of the generated rosters. Another possibility would be to penalize infeasibility by removing pairings using the correction heuristic presented in Section 5.2 and subtracting the penalties associated with them. Reward function (5.7) is generally more conservative with respect to infeasibility, as most infeasible schedules can be corrected by excluding pairings with a total cost below 1000. However, we apply the correction included in the feasibility heuristic in Section 5.2 on test CRP instances to see how the function approximators perform in reality. This means that reward function (5.7) is only used during training.

There are two other reasons why unassigned pairings are not taken into account by this reward function. First, if the training instances are adequately parameterized, meaning that the right balance has been found between asking some pilots to report for duty and placing others on reserve or granting them special requests, unassigned pairings are unlikely to happen during training. This is because there is then a sufficient number of pilots to cover all required pairings. On the other hand, if some of the training data comes from instance variations that are in fact unbalanced (in the sense that too many pilots have

been excused from regular duty), then the number of unassigned pairings can be simultaneously high and variable from one iteration to the next and one scenario to another, causing a significant unpredictability of the learning process during the first few training iterations. In practice, we have found that (5.7) strikes a good balance between limiting the infeasibility of the generated solutions and maintaining a high level of satisfaction for pilots.

One could wonder why learned weights are necessary for the *seqAsg* procedure to produce good-quality solutions. Indeed, we can imagine solving *seqAsg* with arc weights corresponding to the CRP assignment values. We first note that most of those weights would be zero since most possible assignments do not contribute to fulfilling a pilot's wishes. It is then likely that *seqAsg* would act in a myopic way, choosing assignments that directly fulfill pilot's wishes without considering future possibilities. However, it is often advantageous to consider possible future rewards. For instance, it may be advantageous to select a less valued but shorter pairing if it allows the pilot to take a preferred vacation directly afterwards. These considerations are taken into account by our ML features. Furthermore, the ML updates implemented at the end of each training iteration are based on reward function (5.7), which means that the RL weights are improved based on the quality of an entire roster. In other words, while assignments are made one day at a time and in sequence, they take into account future information.

5.3.4 CMA-ES algorithm

This section describes the CMA-ES algorithm in more detail and justifies its choice among other RL algorithms. An exhaustive description can be found in Hansen and Ostermeier (2001), while different adaptations of the algorithm are detailed in Bäck et al. (2013). The covariance matrix adaptation evolution strategy (CMA-ES) is a type of evolutionary algorithm. This means that, at each iteration, it generates λ real-valued vectors obtained from the vector at the current iteration. These modified vectors are obtained by adding noise following a multinormal distribution. The multinormal distribution and a global variance acting as a step size change dynamically at each iteration, forming a new covariance matrix each time. This means that the search space for better real-valued vectors evolves dynamically. Then, the modified vectors are evaluated with a reward function like (5.7).

Once this is done, the best μ sets of modified parameters are kept and their weighted average is taken as the new parameter vector. The covariance matrix and global step size are also changed. In order to do so, the concept of evolution path is useful. The global step size is adjusted while taking into account both its previous updates (as accounted by one evolution path) and whether the mutations were unusually small or large, as measured by a χ_n distribution for an isotropic normal distribution. The covariance matrix is adjusted based on previous changes for that matrix (as accounted by another evolution path) and the sample noisy covariance matrix at that iteration.

The CMA-ES algorithm requires setting the value of certain hyperparameters, including constants determining the magnitude of updates made to the covariance matrix, the initial global step size and the initial evolution paths. All values for the hyperparameters have been empirically validated in Hansen and Ostermeier (2001). The only constant we set ourselves is the dimension of the search space: it is determined by the architecture of the function approximator chosen for the ML model.

While it is more traditional to learn an RL policy's parameters using some version of stochastic gradient descent, the evolutionary algorithm presented above was chosen for two reasons. First, evolutionary algorithms have been found to be competitive on a variety of RL problems (see Salismans et al. (2017)) and are therefore a sensible option. Also, in order to do gradient updates using standard actor-critic methods for RL, the policy needs to be stochastic. If it is deterministic, as in our case, there is the possibility of either using natural gradients (e.g., with natural evolution strategies, an alternative to CMA-ES quite close to standard policy gradients but doing gradient updates over the parameters generated by a probability distribution) or with methods such as those presented in Silver et al. (2014). We tested NES

and did not get better results than with CMA-ES; furthermore, our evolutionary algorithm is a simpler alternative to deterministic actor-critic methods.

6 Results

In this section, we present results for the *seqAsg* method. We show that the solutions generated with the algorithm described in Section 5 can help scheduling planners gain insight into CRP instances and parameterize them more appropriately. In Section 6.1, we give details of the implementation of our ML model and the optimization solver used for the CRP. In Section 6.2, we consider the quality of the solutions produced by the model learned at the end of the ML training process. In Section 6.3, we establish the relationship between the quality of an ML-generated solution and the parameterization of a CRP instance. We do so by observing how the CRP objective in the optimization solver is associated with the quality of the ML-generated solution. Finally, in Section 6.4, we compare the *seqAsg* procedure with ML weights to another very fast heuristic created for this purpose.

6.1 Methodology

We now provide the methodology used to train the ML model presented in Section 5.3. For each instance, 25 out of the 30 available scenarios are kept for training, while the remaining 5 are set aside for testing after the entire ML process is complete. Training is performed on each instance separately to get the ML model for that instance, but we test the model obtained on all 7 instances. For example, if we train an ML model on instance I1 on the 25 first scenarios for that instance, we test the learned model by running the *seqAsg* procedure on the 5 last scenarios for all 7 instances. We do the same for each instance. In the absence of undue overfitting, all generalizable ML models should yield similar results on the test scenarios.

For both approximators, we use early stopping to limit overfitting. In our experiments, we stopped training after 120 iterations. For the neural network approximator, it was found that the algorithm learned best with a 2-layer net. The hidden layers had 13 and 10 neurons, respectively, for a total of 320 parameters. The *seqAsg* procedure was implemented in Python v. 3.7.1 through the package *pulp* v. 2.6.0. The *pulp* module in turn calls the integrated CBC solver v. 2.10.3 to solve the assignment problem defined by (5.1)-(5.5). CPLEX 12.9 was also considered and exhibited similar performance as CBC.

To compare different CRP solutions, we define the following quantities. Let S_k^{best} be the best feasible schedule for pilot $k \in K$. S_k^{best} is obtained by solving the column generation subproblem for each pilot, without dual information (all dual variables equal to 0). In other words, it is found by considering the cost of each arc in the pilot's subproblem rather than the reduced cost as would typically be the case. Let $S^{best} = \sum_{k \in K} S_k^{best}$ be an upper bound on the total satisfaction, and thus, on the CRP optimal value. We use S^{best} as a reference point to compare different solutions of the same instance. In addition, S^{best} is easily computed (in less than one second) and is independent of any solution method, making it ideal to compare different CRP solutions.

In practice, S^{best} is almost never achieved because of conflicting preferences among employees. However, we observe that for well-parameterized problems, GENCOL finds solutions whose satisfaction is very close to S^{best}. In fact, it has been our experience that when a CRP instance is well parameterized, the overwhelming majority of the pairings and days off accounting for pilot satisfaction in the schedules with total satisfaction S^{best} are retained for each pilot when the CRP is solved by GENCOL. However, this is not the case for poorly parameterized instance variations.

Consider a CRP solution of value $z = S - U - V$, with S its satisfaction (first term of 3.1), U the penalty due to missed coverage (second term of 3.1), and V the penalty due to missed preassigned days off (third

term of 3.1). We define the *satisfaction loss* as

$$L^{sat} = 1 - \frac{S}{S^{best}}$$

Likewise, we express the *coverage loss* and the *preassignment loss* relative to S^{best} :

$$L^{cover}_{ML} = \frac{U}{S^{best}}$$

$$L^{preasg} = \frac{V}{S^{best}}$$

Note that L^{preasg} does not appear in the results because it is always zero in our observations. Furthermore, L^{cover}_{ML} is defined here only for ML solutions. While one could also define a variable for the loss of objective due to pairings unassigned to any pilot in standard GENCOL rosters, this measure would be more useful to get a more detailed breakdown of any objective loss in GENCOL solutions, for example by examining any correlation between the cover loss in ML solutions and GENCOL rosters. Since scheduling planners are primarily interested in predicting the global loss of objective to choose whether to re-determine how many pilots work regular duty, are on reserve, or are assigned special activities and requests, we focus on using L^{cover}_{ML} to infer about GENCOL's behavior.

Let S_{ML} and L^{sat}_{ML} be the total satisfaction and the satisfaction loss for ML-generated solutions. Likewise, let S_{GENCOL} and L^{sat}_{GENCOL} be the total satisfaction and the satisfaction loss for solutions generated by GENCOL.

6.2 ML-generated solution quality

We now report on ML-generated solution quality. We do so with the intent of verifying if a relationship between L^{sat}_{ML} and L^{sat}_{GENCOL} is apparent for the instances described in Section 4. A more detailed and expanded analysis of the conclusions found in this section is presented next in Section 6.3 with many different instance variations.

Table 4 shows the average value of L^{sat}_{ML}, which includes the penalties applied for pairings unassigned to any pilot, for each validation instance and each ML model trained. Each row corresponds to a different ML model (one for each training instance) and each column to a different set of validation scenarios (split by instance). In each case, the ML model learned over the training instance is used in the *seqAsg* method to generate five different solutions per validation instance (one for each validation scenario). This means that columns represent the results obtained for different ML models, trained on different instances, for a same validation instance. We report these results for both the linear and the neural network function approximators. For each validation instance, we also report the mean value of L^{sat}_{ML}, as well as the mean value of L^{sat}_{GENCOL} for all seven instances. Table 4 also shows the average pairing-to-pilot ratio. What constitutes a high ratio is instance-specific. Finally, we report the training time in minutes to learn the function approximator's parameters, per instance. Smaller instances allow training the approximator faster and are therefore preferred to larger instances during training when performance across instances is comparable during testing. All values in the table are accompanied by the standard deviation for the sample of scenarios involved in their computation.

From the results in Table 4, a few observations stand out. First, in all but one instance (where the difference is negligible), the neural net performs better than the linear approximator. Next, there is little variation in performance for any given validation instance. This suggests there is little overfitting in general, as the performance on a validation instance is generalizable no matter which ML model is

used, i.e., no matter which instance was used for training. Following up on the discussion brought up in Section 4, we see how in the absence of historic data, scenarios could be generated on a single instance and how a model as we describe could be trained on them. There is generally little variation between performance on different scenarios (within an instance variation), just as is the case between instances. This means that the average values reported in Table 4 are close to those for individual validation scenarios. Additionally, since our instances are varied, Table 4 suggests that the *seqAsg* procedure is consistent even on instances whose compositions, as understood from the lens of instance size and the length of its pairings, differ substantially.

There is a drastic difference in performance between validation instances. In particular, instances I1 to I4 are associated with a loss of satisfaction that is greater than for instances I5 to I7. In the case that performance variability is due to the quality of an instance's parameterization, this would mean that good instances (i.e., balanced in terms of pilots assigned for regular duty as opposed to special tasks and requests, well-parameterized) are associated with solutions in the GENCOL solver whose objective is close to S^{best}. These results confirm the interest in trying to figure out a relationship between L^{sat}_{ML} and L^{sat}_{GENCOL}.

One possibility is that some of our instances are poorly parameterized and do not include enough pilots ready to carry out pairings. This may cause uneven performance across instances after the ML-based sequential assignment procedure is done. In that case, the higher the number of pairings each pilot must cover on average in their schedule, the more difficult the problem is. However, testing for this hypothesis requires more than the data presented here, and the notion of instance variation introduced in Section 4 is useful to cover a larger span of ML solutions than is available in Table 4, as mentioned above.

Another interesting fact is made explicit by the pairing-to-pilot ratio by instance. It can be seen from Table 4 that if two instances have different values for that quantity, the instance with the lower ratio does not necessarily have ML solutions with lower values of L^{sat}_{ML}. This is made evident by comparing the results for instance $I1$ to instances $I5$-$I7$. This is consistent with industry situations, as the ideal number of pilots to schedule for duty will vary based on the situation and with such factors as the length of the pairings or the presence of bottlenecks (i.e., certain days of the month when the pilots available are particularly solicited). In any case, the unevenness in the association between pairing-to-pilot ratio and ML solution quality between instances suggests that L^{sat}_{ML} is better than the pairing-to-pilot ratio as an indicator of the behavior observed in L^{sat}_{GENCOL}.

With regard to training time, there is a positive association between instance size and computing time. This observation combined with the reusability of weights learned on small instances for larger instances suggests that if training time is an issue, learning on smaller instances may be more advisable. Nonetheless, an important caveat is that training only needs to happen once, while the *seqAsg* procedure can be run an unlimited number of times as needed by the scheduling workers: for that reason, it is less of an issue if training takes time on larger instances and this method may also be preferred by certain companies.

6.3 Linking ML-generated solutions with solver performance

Based on the observations in Section 6.2, we now seek to establish a robust relationship between the solution generated with the *seqAsg* procedure and the performance of the GENCOL solver. This can give insight into a CRP instance's parameterization. We do so by considering many instance variations, aimed at providing a good range of different levels of difficulty (i.e., we vary how many pilots are available for a fixed set of pairings) for each instance. The results for these experiments are shown in Table 5. This table shows, for each instance variation, how many pilots are part of it as well as the corresponding pairing-to-pilot ratio. We also report the sum of best possible satisfactions (S^{best}), the ML-generated solution total satisfaction (S_{ML}) and mean satisfaction loss (L^{sat}_{ML}), the GENCOL total satisfaction (S_{GENCOL}) and mean loss (L^{sat}_{GENCOL}) and the loss due to unassigned pairings (L^{cover}_{ML}). Finally, we report the

average computing time for GENCOL over the five scenarios. For instance variations associated with a computing time in GENCOL greater than 8 hours, we have sometimes used less than five scenarios (e.g., only one if the computing time exceeds one week). We have not noticed major differences in the variability of the solutions in those cases.

Training instance	Validation instance							Pairings/pilot	Time (min)
	I1	I2	I3	I4	I5	I6	I7		
Linear function approximator									
I1	49.3 (11.3)	71.0 (19.2)	68.4 (6.0)	54.9 (3.8)	19.6 (1.7)	31.5 (3.2)	14.8 (1.5)	5.81	70
I2	44.1 (3.6)	65.7 (13.7)	68.4 (10.9)	56.0 (4.1)	14.6 (2.2)	29.0 (3.8)	13.6 (1.2)	12.26	82
I3	47.1 (5.4)	62.34.6)	62.9 (6.5)	52.5 (4.1)	22.0 (1.6)	27.2 (3.5)	16.4 (2.3)	9.70	80
I4	46.2 (1.8)	61.8 (8.5)	60.8 (7.2)	50.7 (5.0)	15.6 (1.7)	24.2 (3.1)	14.6 (1.2)	8.87	220
I5	43.8 (5.3)	64.1 (2.8)	60.1 (4.3)	56.5 (6.2)	12.2 (1.8)	22.9 (2.5)	11.0 (1.0)	5.11	579
I6	43.1 (4.4)	65.9 (9.3)	64.4 (8.8)	55.3 (7.0)	12.2 (1.6)	24.2 (1.8)	11.0 (1.2)	5.98	310
I7	41.4 (3.3)	63.0 (11.5)	61.5 (4.1)	61.2 (6.7)	14.6 (2.3)	23.9 (2.6)	10.7 (1.0)	4.57	717
Mean (L^{sat}_{ML})	**45.0 (5.8)**	63.4 (10.6)	62.4 (7.6)	55.3 (6.6)	15.8 (4.1)	26.1 (4.0)	13.2 (2.5)		
Mean (L^{sat}_{GENCOL})	**17.4 (2.3)**	17.2 (5.2)	17.6 (2.4)	10.0 (1.6)	2.4 (0.9)	4.2 (0.9)	2.1 (1.6)		
Neural function approximator									
I1	50.3 (3.5)	55.8 (8.9)	57.8 (14.1)	53.8 (3.7)	16.4 (0.9)	26.3 (1.8)	13.6 (0.9)	5.81	127
I2	46.2 (2.0)	53.0 (11.8)	59.2 (7.2)	53.7 (9.8)	14.8 (1.3)	26.5 (3.4)	12.2 (1.5)	12.26	134
I3	44.1 (3.9)	51.8 (8.4)	55.2 (8.5)	45.8 (6.3)	17.5 (1.8)	24.0 (1.4)	12.8 (1.3)	9.70	132
I4	40.6 (4.5)	51.6 (9.9)	56.7 (6.6)	53.7 (6.4)	14.4 (1.8)	23.3 (2.5)	13.3 (1.0)	8.87	345
I5	46.7 (9.8)	51.5 (10.5)	58.7 (6.5)	53.2 (6.3)	11.4 (1.2)	20.0 (2.8)	10.5 (1.3)	5.11	794
I6	43.8 (6.5)	51.9 (7.4)	66.5 (6.7)	52.5 (11.7)	12.4 (1.0)	20.2 (2.4)	11.1 (1.2)	5.98	492
I7	44.7 (4.9)	56.8 (8.6)	65.7 (7.8)	54.1 (3.2)	11.9 (1.4)	21.5 (2.8)	11.3 (1.2)	4.57	1174
Mean (L^{sat}_{ML})	45.2 (10.0)	53.2 (8.8)	58.5 (9.9)	52.4 (6.0)	14.1 (2.5)	23.1 (3.5)	12.1 (2.0)		
Mean (L^{sat}_{GENCOL})	17.4 (2.3)	**17.2 (5.2)**	**17.6 (2.4)**	**10.0 (1.6)**	**2.4 (0.9)**	**4.2 (0.9)**	**2.1 (1.6)**		

Table 4 Average L^{sat}_{ML} (%) for each training set/test set combination (two approximators)

Also, we note that the speed to obtain the ML solutions is not mentioned. This is because, in our experience, the *seqAsg* procedure has always taken less than 15 seconds (under 10 seconds for smaller instances). It also increases only slightly with instance size. Given that for large instances especially, such a speed approaches immediate resolution, we consider the time taken by *seqAsg* to provide an ML solution negligible in those cases.

From Table 5, a number of observations can be made. First, predictably, both ML-generated and GENCOL solution objectives decrease as the total number of pilots available for an instance is reduced. On the other hand, in the same situation, computing times and the loss of satisfaction due to unassigned pairings increase. This is remarkable as a more limited number of pilots means that there are fewer subproblem networks during column generation, which reinforces the idea that not scheduling enough pilots for regular duty increases the complexity of crew rostering. Additionally, at first glance, it seems like the decrease for the GENCOL objective follows the decrease in ML solution quality in an approximatively linear fashion, and this is true across all 7 instances. In particular, the values for L^{sat}_{ML} and L^{sat}_{GENCOL} range from 3.4% (I5-01) to 86.1% (I6-07) and from 1.3% (I5-01) to 33.8% (I6-07), respectively, and seem to covary.

Furthermore, computing time increases rapidly as fewer pilots are available, while L^{cover}_{ML} starts exceeding zero for most instances once enough pilots have been removed from scheduling duty. This is significant, as when the problem increases in complexity, it becomes harder to generate a solution that meets all pairing coverage constraints. We also observe that when the number of pilots scheduled for regular duty is sufficient, L^{cover}_{ML} is zero and only increases slowly at first, but can dominate quickly (for example as in the cases of instance variations I3-07, I6-07 and I7-07). The extent to which this finding is significant depends on the scheduling planner's needs. In the case that a planner is less interested in getting a precise estimate of the value for L^{sat}_{GENCOL} as the instance is currently parameterized, then a condition on L^{cover}_{ML} could enforce that it be zero to keep L^{sat}_{GENCOL} high. Indeed, it seems that when our sequential

Variation	Nb pilots	Pairings/pilot	S^{best}	S_{ML}	L_{ML}^{sat} (%)	S_{GENCOL}	L_{GENCOL}^{sat} (%)	Time (s)	L_{ML}^{cover} (%)
I1-01	52	3.0	39603	36736	7.2	38861	1.9	0.5	0.0
I1-02	45	3.5	34173	31075	9.1	33379	2.3	0.7	0.4
I1-03	39	4.0	29820	26013	12.7	28795	3.4	0.7	0.5
I1-04	35	4.5	26350	21737	17.5	24844	5.7	1.1	1.8
I1-05	31	5.0	23521	17345	26.3	21248	9.7	1.5	4.4
I1-06	29	5.5	22106	14900	32.6	19824	10.3	2.0	6.9
I1-07	26	6.0	19721	10126	48.7	15596	20.9	3.2	14.3
I2-01	66	5.0	55333	52352	5.4	53959	2.8	7.1	0.0
I2-02	55	6.0	45323	42347	6.6	44336	2.2	8.5	0.4
I2-03	47	7.0	40013	37226	7.0	39210	2.0	9.3	0.0
I2-04	41	8.0	34653	32051	7.5	33562	3.1	16	0.5
I2-05	37	9.0	30320	27180	10.4	28984	4.4	19	0.9
I2-06	33	10	27670	22757	17.7	25977	6.1	31	4.6
I2-07	30	11	25289	17769	29.7	22796	9.7	51	11.0
I3-01	90	4.0	70239	64223	8.6	69050	1.7	16	0.0
I3-02	72	5.0	56660	51393	9.3	55449	2.1	19	0.0
I3-03	60	6.0	47369	42136	11.0	46301	2.3	24	0.2
I3-04	51	7.0	40729	35001	14.1	39492	3.0	37	1.4
I3-05	45	8.0	35157	28540	18.8	32749	6.8	108	4.2
I3-06	40	9.0	31878	20727	33.9	27755	12.9	128	13.7
I3-07	36	10	28106	10057	64.2	21250	24.4	807	33.2
I4-01	260	4.0	198915	186702	6.1	195627	1.7	223	0.0
I4-02	208	5.0	159531	149490	6.3	156993	1.6	268	0.0
I4-03	173	6.0	133135	121948	8.4	131448	1.3	291	0.0
I4-04	160	6.5	124436	113766	8.6	121972	2.0	438	0.0
I4-05	148	7.0	112619	99242	11.9	110526	1.9	556	0.5
I4-06	138	7.5	103774	88277	14.9	101282	2.4	767	1.4
I4-07	130	8.0	99654	77259	22.5	94912	4.8	1268	4.6
I4-08	122	8.5	93092	59246	36.4	86432	7.2	2474	12.2
I4-09	115	9.0	89097	35563	60.0	77428	13.1	4047	29.5
I5-01	407	3.0	300987	290631	3.4	297201	1.3	324	0.0
I5-02	349	3.5	260584	251506	3.5	256942	1.4	365	0.0
I5-03	306	4.0	226636	216858	4.3	223568	1.4	421	0.0
I5-04	272	4.5	203401	191348	5.9	200160	1.6	467	0.1
I5-05	244	5.0	182711	164968	9.7	177850	2.7	656	0.6
I5-06	222	5.5	165992	136962	17.5	159451	3.9	936	3.5
I5-07	204	6.0	150899	95930	36.4	135540	10.2	>8 hrs	16.2
I6-01	269	4.0	204453	194498	4.9	201366	1.5	124	0.0
I6-02	240	4.5	180873	171003	5.5	178278	1.4	150	0.0
I6-03	215	5.0	161289	147172	8.8	158692	1.6	190	0.0
I6-04	196	5.5	146884	129457	11.9	143157	2.5	244	0.0
I6-05	179	6.0	135290	106414	21.3	128947	4.7	470	2.4
I6-06	166	6.5	125294	65755	47.3	112185	10.4	>8 hrs	21.4
I6-07	154	7.0	117058	16282	86.1	77483	33.8	>8 hrs	50.5
I7-01	491	3.0	367439	352302	4.1	361987	1.5	455	0.0
I7-02	421	3.5	316157	300478	5.0	311402	1.5	516	0.0
I7-03	368	4.0	274031	257204	6.1	269206	1.8	707	0.0
I7-04	327	4.5	244622	220567	9.0	239787	2.0	897	0.1
I7-05	295	5.0	218413	177926	18.5	209831	3.9	1552	2.5
I7-06	268	5.5	201112	128838	35.9	182775	9.1	>8 hrs	14.9
I7-07	246	6.0	184138	63781	65.4	136215	26.0	>8 hrs	36.8

Table 5 Relationship between ML-generated and final solutions in the solver by instance and pairing-to-pilot ratio

assignment procedure generates a fully feasibly roster and all CRP constraints are met, L_{GENCOL}^{sat} is close to as low as it can possibly get.

To further explore the relationship between L_{ML}^{sat} and L_{GENCOL}^{sat}, we use the data in the table and fit a regression model with $\log(L_{ML}^{sat})$ as an explanatory variable and $\log(L_{GENCOL}^{sat})$ as a predicted variable. It is then possible to draw conclusions from the results of this analysis, which we perform for the four largest instances (I4-I7). Results are presented in Table 6. The first row of the table gives general information about the linear regression model fit to the data. It shows the explanatory and predicted variables, the regression method (ordinary least squares, also written OLS), the number of data points

part of the model fit, the degrees of freedom of the model used to compute the F-statistic and t-statistics elsewhere in the table, the number of degrees of freedom in the model, and the covariance type (i.e., robust to heteroscedasticity or not). The second row includes information about the quality of the model fit: the R-squared and adjusted R-squared coefficients, the F-statistic and associated probability that such a statistic would be reached if the model coefficients were all zero, as well as other goodness of fit statistics such as the log-likelihood and the AIC and BIC (Aikake information criterion and bayesian information criterion, respectively). The next two rows give information about the regression parameters: their values, standard errors, t-statistics, the probability of reaching such values of the (absolute value of the) t-statistic if the coefficients are zero, and the 95% confidence intervals for the coefficient values.

By analyzing Table 6, certain observations stand out. First, the values for R-squared and adjusted R-squared are similar and very high: this R-squared value suggests that 91.9% of the variance in $\log(L^{sat}_{GENCOL})$ is explained by $\log(L^{sat}_{ML})$. In other words, the loss of satisfaction in our ML solutions explains almost all the variance in loss of satisfaction in GENCOL according to a linear relationship. Furthermore, the F-statistic suggests the non-trivial model is statistically significant, with both t-statistics suggesting that the intercept and the explanatory variable coefficients are each different from zero. Finally, the association between $\log(L^{sat}_{ML})$ and $\log(L^{sat}_{GENCOL})$ is positive, as would be expected. Altogether, this summary of the regression analysis in Table 6 allows us to conclude that a strong relationship between L^{sat}_{ML} and L^{sat}_{GENCOL} exists and that the model's fit is excellent.

Expl.	**Pred.**	**Method**	**Obs.**	**Df (residuals)**	**Df (model)**	**Cov.**
$\log(L^{sat}_{ML})$	$\log(L^{sat}_{GENCOL})$	OLS	30	28	1	nonrobust
R^2	**Adj. R^2**	**F-stat.**	**Pr(F-stat.)**	**Llk**	**AIC**	**BIC**
0.919	0.916	315.7	8.82e-17	-13.359	30.72	33.52
Variable	**Coef.**	**Std err.**	**t-stat**	**Pr(t-stat)**	**LC**	**UC**
intercept	-1.7971	0.206	-8.713	0.000	-2.220	-1.375
$\log(L^{sat}_{ML})$	0.9486	0.053	17.768	0.000	0.839	1.058

Table 6 Regression analysis for OLS fit between $\log(L^{sat}_{ML})$ and $\log(L^{sat}_{GENCOL})$, instances I4-I7

To better visualize the effect of an increasing value of L^{sat}_{ML}, we represent the observations from Table 5 with Figure 3. In this figure, for each instance variation presented in Table 5, we associate one data point in the scatterplot. Each point represents the $(L^{sat}_{ML}, L^{sat}_{GENCOL})$ coordinate pair for the instance variation. The horizontal axis represents the value of L^{sat}_{ML} in percentage for the solution generated by *seqAsg* and for that instance variation. The vertical axis gives the corresponding value of L^{sat}_{GENCOL} for the same data point. Both axes are on a base 10 logarithmic scale to remain consistent with the model variables and for better visibility. We then fit the data using the model presented in Table 6 and trace the regression line for the best OLS fit. The best OLS fit shows a clear positive and very strong association between $\log(L^{sat}_{ML})$ and $\log(L^{sat}_{GENCOL})$, thereby reinforcing the conclusions drawn from the regression analysis summary presented in Table 6. Another important point of consideration is that, based on the observed data, the variance of the model seems mostly uniform across the range of different values of L^{sat}_{ML} for the data. This suggests that there is no heteroscedasticity, as previously assumed, and that the data was fit properly.

Both Table 6 and Figure 3 seem to support the notion of a linear association between $\log(L^{sat}_{ML})$ and $\log(L^{sat}_{GENCOL})$ that informs as to the expected loss of satisfaction when solving the CRP in GENCOL. We now evaluate the usefulness of obtaining information in such a way. As mentioned in the introduction, CRP scheduling planners need to make many, sometimes hundreds of decisions throughout a month when creating the instance for the next month. Absent any suitable analysis tool, they must check whether their newly modified instance is likely to give a good objective after resolution in GENCOL. However,

Table 5 suggests the time required to do that becomes prohibitive for large instances once L^{sat}_{ML} becomes too large. This means that building an instance that includes enough pilots scheduled to fly (i.e., not granting too many requests or placing too many pilots on reserve) takes an unreasonable amount of time for instances that are not adequately parameterized. In this way, the *seqAsg* procedure allows testing the adjustments made by the scheduling workers within seconds while giving them insight into whether the instance is overly difficult to solve. For example, it allows them to check if it can be solved to most pilots' satisfaction while granting as many requests as possible, including during special events.

Another aspect worth reiterating is the speed at which we produce feasible CRP solutions fit to help with schedule planning. It is unparalleled in previous research so far, as past ML research (see Section 2) has mostly focused on solving the CRP either by enhancing the work of a solver or by using complex heuristics. While these methods often match or even slightly exceed objective quality for well-known benchmark instances, they always take at least several minutes to run (a duration that may be shorter than standard resolution but still far exceeds the 15 seconds or less for our method). Furthermore, it has been our observation that the 10 to 15-second duration does not increase much with instance size. There are also instances larger than even our largest instances (up to 50000 flights, see Yaakoubi et al. (2020b)) that may take hours to solve through traditional means or heuristics. The ML weights only having to be learned once, the *seqAsg* procedure is largely immune to this issue and it would likely be of similar speed in that context as well, therefore allowing scheduling planners to do their work in a fraction of the time it would normally take. For this reason, while our solutions are not perfect and are not taken as an end product in themselves, they are very fast and are still strong enough to provide useful information: this is not something addressed by any alternative method yet.

With these considerations in mind, a rigorous evaluation of our ML-based method's usefulness needs to take into account potential methods that would also be very fast in providing important information to scheduling planners. We do so in Section 6.4.

6.4 Fast GENCOL heuristic

We now seek to assess the appropriateness of our method, especially with regard to the use of ML as opposed to other types of algorithms. Not having found in the literature any approach similar to *seqAsg* in terms of speed, as a comparison point, we provide a simple heuristic that can be directly implemented in GENCOL. We define this heuristic as follows.

First, we focus on changing certain parameters in the GENCOL solver to converge towards a solution very quickly. Since the goal is to offer a heuristic as an alternative to *seqAsg*, we seek to bring computational time below 60 seconds and in most cases, below 30 seconds for the linear relaxation of the problem (in comparison to *seqAsg*'s 10-15 seconds for the full solution depending on the instance). To achieve this goal, we change the value of certain parameters in GENCOL. The parameters changed in this way fall into three categories: parameters related to resource dominance during the subproblem phase of column generation, parameters related to stopping criteria (e.g., number of rolling iterations and relative improvement of the objective over these iterations), as well as parameters related to branching. Once the correct configuration of parameters is found, we run the input file for one scenario for each instance variation associated with the four largest instances. We then note the objective value and time taken by the solver once the final solution is obtained. We also take note of the objective value and computing time once the linear relaxation is solved, i.e., before branching starts. We refer to these values as those obtained once branching node zero is solved.

The results for the GENCOL heuristic are shown in Table 7.

Let L^{sat}_{heur} be the satisfaction loss for solutions produced by the accelerated GENCOL heuristic. Table 7 shows the loss of satisfaction for the heuristic (L^{sat}_{heur} against S_{best} for the solutions obtained after solving node zero or after the full resolution of the integer problem, as well as the loss of satisfaction L^{sat}_{GENCOL}

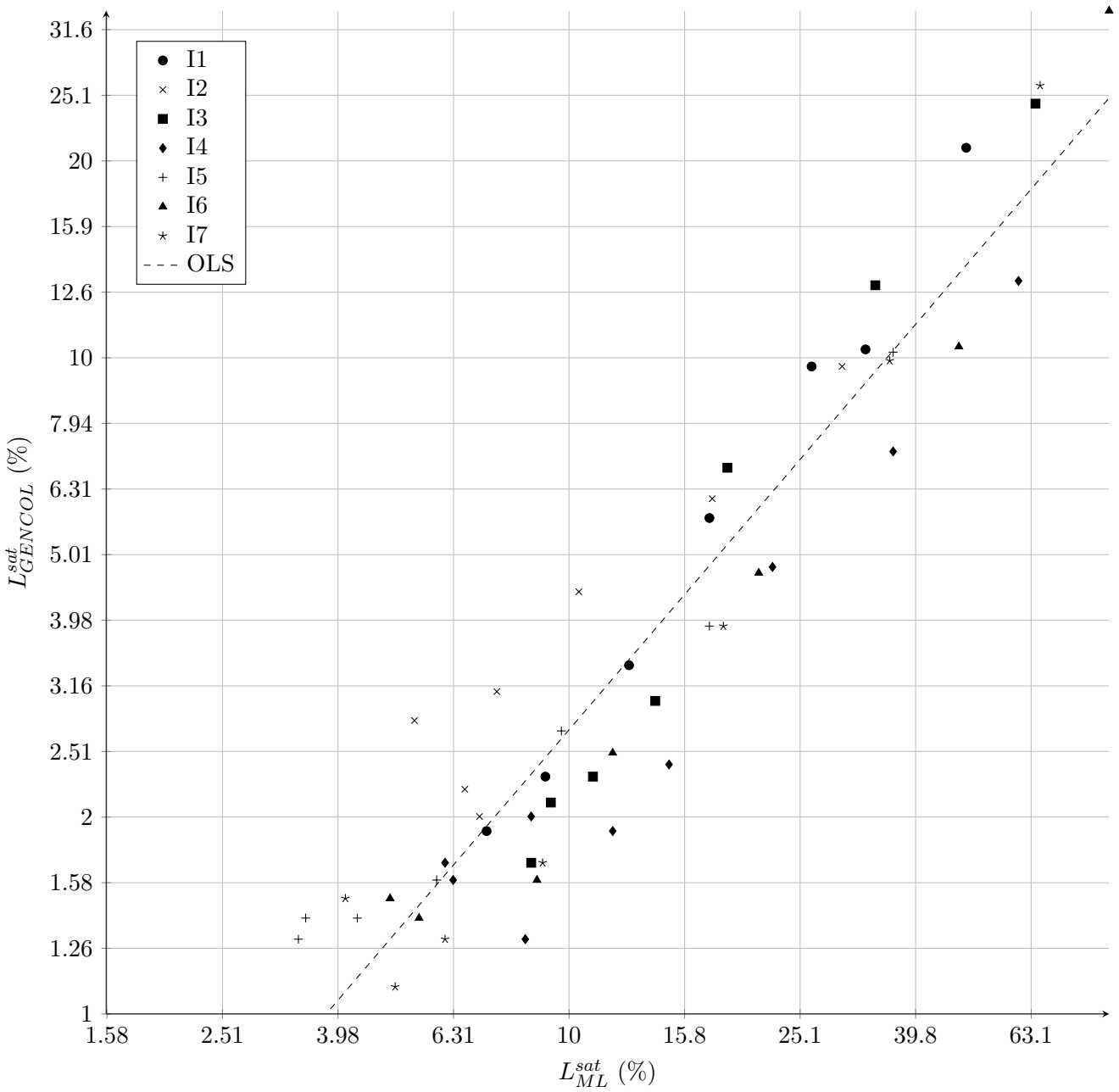


Fig. 3 Log-log relationship between L^{sat}_{ML} and L^{sat}_{GENCOL} for many instance variations

for the full solution obtained without the heuristic (i.e., with the standard configuration of the solving parameters in GENCOL). The first column shows the instance variations considered. We present the objective value, solving time and loss of satisfaction L^{sat}_{heur} both for the relaxed and the full solution in the six following columns, while the last two show L^{sat}_{GENCOL} and S_{best} by instance variation.

A closer examination of the data reported in Table 7 suggests the value of L^{sat}_{heur} for the relaxed heuristic solutions is associated more closely with L^{sat}_{GENCOL} than the loss of satisfaction observed in the final heuristic solutions. The relaxed solutions are also considerably faster to get. For example, in the final heuristic solution for instance variation I4-06, we observe a value of L^{sat}_{heur} of 30.4% while L^{sat}_{GENCOL} stands at 1.49%, whereas for instance I7-07 we have that L^{sat}_{heur} and L^{sat}_{GENCOL} are 32.8% and 26.0% respectively. For this reason, we focus on comparing the goodness of fit for OLS models based on the *seqAsg* predictions and the heuristic applied to relaxed CRP solutions.

Variation	Node zero			Final solution			L^{sat}_{GENCOL}	S^{best}
	Objective	Time	L^{sat}_{heur} (%)	Objective	Time	L^{sat}_{heur} (%)		
I4-01	167439	13.0	16.3	183575	63.5	8.27	1.87	200120
I4-02	133925	20.4	16.1	142515	82.1	10.0	1.04	158405
I4-03	111861	19.3	16.1	111370	97.6	16.4	1.45	133305
I4-04	105662	20.7	15.4	97770	118	21.8	4.36	124965
I4-05	94681	28.3	16.9	84000	127	26.3	1.83	113995
I4-06	91139	45.2	12.2	72260	145	30.4	1.49	103810
I4-07	77084	29.9	22.6	68315	150	31.4	5.89	99575
I4-08	72734	50.9	19.9	50575	162	44.3	6.75	90765
I4-09	62532	38.7	30.9	43825	211	51.6	12.1	90515
I5-01	259385	18.6	13.7	288545	127	4.05	1.52	300725
I5-02	214201	21.4	18.1	246455	129	5.81	1.27	261650
I5-03	193413	28.6	14.7	213435	146	5.85	1.06	226695
I5-04	166305	28.6	15.9	177670	148	10.2	1.56	197795
I5-05	161103	39.7	13.0	163495	156	11.7	3.03	185250
I5-06	133506	36.8	19.4	141090	188	14.8	3.87	165595
I5-07	112967	52.7	25.9	122205	300	19.8	10.2	152445
I6-01	178789	10.7	11.7	185770	42.8	8.27	1.49	202525
I6-02	155475	9.6	14.7	170075	47.0	6.65	1.47	182200
I6-03	130926	10.4	18.8	142050	53.7	11.9	0.82	161230
I6-04	121072	14.1	17.7	130410	69.1	11.4	3.44	147130
I6-05	99175	13.1	27.0	115890	85.2	14.7	5.44	135815
I6-06	85966	20.3	31.7	94540	142	24.8	10.4	125780
I6-07	59519	22.6	48.6	66595	130	42.4	33.8	115700
I7-01	312847	28.6	14.9	353900	163	3.77	1.23	367775
I7-02	258663	34.7	17.6	295685	187	5.81	1.34	313915
I7-03	227988	40.0	16.7	251965	197	7.94	1.19	273695
I7-04	197805	44.0	18.5	223450	219	7.93	1.68	242695
I7-05	174219	49.8	21.3	197500	250	10.8	3.72	221375
I7-06	144591	66.9	28.1	163185	590	18.8	9.1	201030
I7-07	100131	55.8	46.3	125185	563	32.8	26.0	186310

Table 7 Comparison between standard resolution and a fast heuristic in GENCOL

The observations made above were a surprising finding, given that one might expect that a full solution would convey more useful information to scheduling workers than a relaxed solution. This initially made the accelerated resolution in GENCOL the natural candidate to consider. Once we noticed that the relationship between L^{sat}_{heur} and L^{sat}_{GENCOL} seemed weak in that specific case, we realized that the change in branching parameters put in place to converge towards a full solution very quickly in GENCOL caused a massive drop of performance between the values of the objective after node zero and the final objective. This means that a full solution, far from adding information, requires sacrificing performance to such an extent that the resulting objective is less informative about L^{sat}_{GENCOL}. This explains why we have also included the version of the heuristic concerned with the objective for the linear relaxation of the problem, and why it performs better.

We also performed a regression analysis on the results of Table 7. Results are presented in Table 8, in the same fashion as Table 6. Comparing the two tables allows us to make several observations. First, while the heuristic model captures most of the variance in the explained variable, $\log(L^{sat}_{GENCOL})$, our model does much better by capturing most of the variance unexplained by the GENCOL heuristic. This is a meaningful difference, as the precise prediction of GENCOL's behavior allows us to detect more accurately the small losses in pilot satisfaction that can be expected for the instance under consideration. Given that even small differences in satisfaction can substantially impact pilots, this is not trivial. Furthermore, both models are significant and both the intercept and the explanatory variable are significant

in each case as well. However, the goodness of fit for the *seqAsg* OLS model is better than for the model based on the heuristic. The R-squared is 0.919 for the former and 0.754 for the latter; all three values for the log-likelihood, the AIC, and the BIC favor the model based on the ML solutions as the best fit on prospective new data.

Expl.	**Pred.**	**Method**	**Obs.**	**Df (residuals)**	**Df (model)**	**Cov.**
$\log(L^{sat}_{heur})$	$\log(L^{sat}_{GENCOL})$	OLS	30	28	1	nonrobust
R^2	**Adj. R^2**	**F-stat.**	**Pr(F-stat.)**	**Llk**	**AIC**	**BIC**
0.754	0.745	85.87	5.03e-10	-32.075	68.15	70.95
Variable	**Coef.**	**Std err.**	**t-stat**	**Pr(t-stat)**	**LC**	**UC**
intercept	-8.8831	1.138	-7.809	0.000	-11.213	-6.553
$\log(L^{sat}_{ML})$	2.4509	0.264	9.267	0.000	1.909	2.993

Table 8 Regression analysis for OLS fit between $\log(L^{sat}_{heur})$ (node zero) and $\log(L^{sat}_{GENCOL})$, instances I4-I7

It is worth detailing some of the reasons that may explain why our sequential assignment procedure performs better than the GENCOL heuristic that we have implemented. In particular, we have that the columns generated by the heuristic are not guided by any auxiliary information. When a time constraint is added to the entire process, this means that the solver must largely find a solution that is based on a random and limited set of schedules. Consequently, as discussed previously, with branching in place to reach an integer solution, there is a great amount of loss of quality due to this element of randomness. Regarding the variant of the heuristic leveraging the linear relaxation of the problem, we have not only that the set of available columns is restricted, but also that the value of the objective does not account for the inevitable fluctuations in loss of quality due to branching during the standard form of resolution in GENCOL. In contrast, *seqAsg* provides an integer solution for the same CRP instance as the one given as input to GENCOL: it is then not surprising that it can capture some of the fluctuation related to the transformation from a relaxed solution into an integer solution. This makes it unlikely that even extensive efforts would lead to a configuration of the GENCOL parameters such that the heuristic produces solutions comparable to *seqAsg* in terms of predictions.

Based on these observations, it is possible to see that while a simple heuristic can to some extent quickly give an approximate idea of L^{sat}_{GENCOL} for a given CRP instance, such a heuristic lacks in precision and is less strong than a more elaborate approach such as the analysis of solutions generated by an ML procedure like *seqAsg*. Another point of consideration is that such an ML procedure is more robust to changing instance conditions or even problem tasks: one can see how different CO tasks would be amenable to an approach that builds a solution fast and sequentially with the correct ML sequential assignment procedure. It is not clear, on the other hand, that a simple heuristic of the sort that we have shown would necessarily be available or reliable for these other tasks. These remarks point to the fact that the method we have introduced in this paper shows not only promise for an understudied problem (the CRP), but may also extend to other scheduling or more general OR tasks wherever it is most practical to implement. It is then possible to gain insight into final solutions obtained through standard resolution at a much discounted time.

7 Discussion

In this paper, we have presented an innovative way of gaining insight into CRP instances. After reviewing how ML and OR problems are interrelated, we have put forward an algorithm built upon a sequential assignment procedure, *seqAsg*, whose purpose is to generate quick solutions rather than solving the full CRP in a standard solver. We showed how this procedure relies on an RL policy defined by parameters

updated through an evolutionary algorithm, CMA-ES. We have then shown, using seven data instances and a number of their variations, how the quality of the solutions generated by this procedure can be used to gain insight into the solutions produced by the GENCOL solver. By generating many variations of the data instances with gradually decreasing numbers of pilots, we showed the existence of a positive linear association between two types of loss of satisfaction. These were L^{sat}_{ML} and L^{sat}_{GENCOL}, in ML and solver-generated solutions, respectively. In both cases, the loss of satisfaction was measured against the sum of satisfactions for the best feasible schedules for pilots, S^{best}. We have then established that as L^{sat}_{ML} increases, the quality of the objective for a CRP instance handled by the solver quickly decreases while computing time dramatically increases. In that sense, the loss of satisfaction in the ML solutions generated by *seqAsg* directly translates into information as to the behavior of the GENCOL solver.

Next, we demonstrated how the relationship between ML and solver loss of satisfaction could be used in practice to help scheduling workers make adjustments to CRP instances as needed without prohibitive time requirements. The robustness of our method was also contrasted to other methods, such as hand-made heuristics developed by scheduling workers and a heuristic yielding fast solutions in GENCOL itself. Indeed, we have seen that our method is generalizable to many different contexts and outperforms the insights that can be derived from simply obtaining solutions in GENCOL in a few seconds. This allowed us to provide tangible recommendations for the use of the *seqAsg* procedure during the iterative construction of a CRP instance.

Overall, our contribution provides a concrete way of solving the CRP more efficiently, while using ML and RL techniques in a unique manner. Indeed, our procedure allows us to solve the CRP quickly, thus producing a fully feasible solution for suitable numbers of pilots while also allowing us to gain insight into the relevant instance. This allows for crew scheduling workers to reparameterize the problem at a low time cost, therefore obtaining a more sophisticated solution in the GENCOL solver when it is invoked at the end of the parameterization process.

Our method is also unique in that it provides very fast solutions, i.e., solutions obtained in an order of magnitude less time than other existing heuristics. While this involves some loss of ML solution quality, this loss is limited so that the solutions are strong enough to provide useful information almost immediately. This constitutes a unique contribution.

We conclude by mentioning that while this paper offers a way to improve CRP instances, there is still much to be gained by speeding up the resolution of the problem in an optimization solver like GENCOL. As *seqAsg* can also be used to determine feasible schedules for each pilot, this approach lends itself well to the dynamic constraint aggregation approach (DCA). Indeed, DCA procedures require clusters of pairings for each pilot, such as provided by an ML-generated solution, to launch. This solution can be generated by *seqAsg*. Thus, the solutions given by our ML-based procedure could not only build insight into the CRP instance, but also facilitate its resolution once the instance is deemed parameterized in a satisfactory way. Another possibility would be to exploit the information given by schedules with total satisfaction S^{best} in combination with the analysis tool provided by this paper. In this way, well-parameterized CRP instances can be readily identified, and valuable information regarding likely pairings and days off for each pilot can be provided to a learning algorithm. As seen in Section 6.2, this information is closely related to what is observed in GENCOL solutions for the CRP, potentially making it highly valuable.

Finally, it might be interesting to see under which conditions very fast solutions such as those provided by a procedure similar to *seqAsg* might be not only good enough to gain certain types of useful insights, but also to be used as final solutions in themselves. With the increasing popularity of GNNs that learn about an entire graph's topology, it might be possible to provide better features to quickly train algorithms of the sort presented in this paper in such a way as to bridge the gap in objective performance compared to other heuristics. These possibilities could be explored in future work.

References

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint*, https://doi.org/10.48550/arXiv.1409.0473.

Bello, I., Pham, H., Le, Q. V., Norouzi, M., & Bengio, S. (2016). Neural combinatorial optimization with reinforcement learning. *arXiv preprint*, https://doi.org/10.48550/arXiv.1611.09940.

Beulen, M., Scherp, L., & Santos, B. F. (2020). Dynamic evaluation of airline crew's flight requests using a neural network. *EURO Journal on Transportation and Logistics*, *9*(4), 100018, https://doi.org/10.1016/j.ejtl.2020.100018.

Bonami, P., Lodi, A., & Zarpellon, G. (2022). A classifier to decide on the linearization of mixed-integer quadratic problems in CPLEX. *Operations Research*, *70*(6), 3303–3320, https://doi.org/10.1287/opre.2022.2267.

Boubaker, K., Desaulniers, G., & Elhallaoui, I. (2010). Bidline scheduling with equity by heuristic dynamic constraint aggregation. *Transportation Research Part B: Methodological*, *44*(1), 50–61, https://doi.org/10.1016/j.trb.2009.06.003.

Boufaied, C., Trabelsi, R., Masri, H., & Krichen, S. (2016). A construction of rotations-based rosters with a genetic algorithm. In *2016 IEEE Congress on Evolutionary Computation (CEC)* (pp. 2389–2394).: IEEE.

Bäck, T., Foussette, C., & Krause, P. (2013). *Contemporary evolution strategies*, volume 86. Springer: Berlin.

Cappanera, P. & Gallo, G. (2004). A multicommodity flow approach to the crew rostering problem. *Operations Research*, *52*(4), 583–596, https://doi.org/10.1287/opre.1040.0110.

Caprara, A., Toth, P., Vigo, D., & Fischetti, M. (1998). Modeling and solving the crew rostering problem. *Operations research*, *46*(6), 820–830, https://doi.org/10.1287/opre.46.6.820.

de Armas, J., Cadarso, L., Juan, A. A., & Faulin, J. (2017). A multi-start randomized heuristic for real-life crew rostering problems in airlines with work-balancing goals. *Annals of Operations Research*, *258*, 825–848, https://doi.org/10.1007/s10479-016-2260-y.

Desaulniers, G., Desrosiers, J., Dumas, Y., Marc, S., Rioux, B., Solomon, M. M., & Soumis, F. (1997). Crew pairing at Air France. *European journal of operational research*, *97*(2), 245–259, https://doi.org/10.1016/S0377-2217(96)00195-6.

El Moudani, W., Alberto Nunes Cosenza, C., de Coligny, M., & Mora-Camino, F. (2001). A bi-criterion approach for the airlines crew rostering problem. In *Evolutionary Multi-Criterion Optimization: First International Conference, EMO 2001 Zurich, Switzerland, March 7–9, 2001 Proceedings 1* (pp. 486–500).: Springer.

Elhallaoui, I., Metrane, A., Desaulniers, G., & Soumis, F. (2011). An improved primal simplex algorithm for degenerate linear programs. *INFORMS Journal on Computing*, *23*(4), 569–577, https://doi.org/10.1287/ijoc.1100.0425.

Furian, N., O'Sullivan, M., Walker, C., & Çela, E. (2021). A machine learning-based branch and price algorithm for a sampled vehicle routing problem. *OR Spectrum*, *43*, 693–732, https://doi.org/10.1007/s00291-020-00615-8.

Gamache, M., Soumis, F., Marquis, G., & Desrosiers, J. (1999). A column generation approach for large-scale aircrew rostering problems. *Operations Research*, *47*(2), 247–263, https://doi.org/10.1287/opre.47.2.247.

Gopalakrishnan, B. & Johnson, E. L. (2005). Airline crew scheduling: State-of-the-art. *Annals of Operations Research*, *140*, 305–337, https://doi.org/10.1007/s10479-005-3975-3.

Hansen, N. & Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, *9*(2), 159–195, https://doi.org/10.1162/106365601750190398.

Higham, C. F. & Higham, D. J. (2019). Deep learning: An introduction for applied mathematicians. *Siam review*, *61*(4), 860–891, https://doi.org/10.1137/18M1165748.

Juan, A. A., Faulin, J., Ferrer, A., Lourenço, H. R., & Barrios, B. (2013). MIRHA: multi-start biased randomization of heuristics with adaptive local search for solving non-smooth routing problems. *TOP*, *21*, 109–132, https://doi.org/10.1007/s11750-011-0245-1.

Karimi-Mamaghan, M., Mohammadi, M., Meyer, P., Karimi-Mamaghan, A. M., & Talbi, E.-G. (2022). Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art. *European Journal of Operational Research*, *296*(2), 393–422, https://doi.org/10.1016/j.ejor.2021.04.032.

Kasirzadeh, A., Saddoune, M., & Soumis, F. (2017). Airline crew scheduling: models, algorithms, and data sets. *EURO Journal on Transportation and Logistics*, *6*(2), 111–137, https://doi.org/10.1007/s13676-015-0080-x.

Kohl, N. & Karisch, S. E. (2004). Airline crew rostering: Problem types, modeling, and optimization. *Annals of Operations Research*, *127*(1-4), 223–257, https://doi.org/10.1023/B:ANOR.0000019091.54417.ca.

Kool, W., van Hoof, H., & Welling, M. (2019). Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, https://doi.org/10.48550/arXiv.1803.08475.

Kotary, J., Fioretto, F., Van Hentenryck, P., & Wilder, B. (2021). End-to-end constrained optimization learning: A survey. *arXiv preprint arXiv:2103.16378*, https://doi.org/10.48550/arXiv.2103.16378.

Lučic, P. & Teodorovic, D. (1999). Simulated annealing for the multi-objective aircrew rostering problem. *Transportation Research Part A: Policy and Practice*, *33*(1), 19–45, https://doi.org/10.1016/S0965-8564(98)00021-4.

Lučić, P. & Teodorović, D. (2007). Metaheuristics approach to the aircrew rostering problem. *Annals of Operations Research*, *155*, 311–338, https://doi.org/10.1007/s10479-007-0216-y.

Lübbecke, M. E. & Desrosiers, J. (2005). Selected topics in column generation. *Operations Research*, *53*(6), 1007–1023, https://doi.org/10.1287/opre.1050.0234.

Maenhout, B. & Vanhoucke, M. (2010). A hybrid scatter search heuristic for personalized crew rostering in the airline industry. *European journal of operational research*, *206*(1), 155–167, https://doi.org/10.1016/j.ejor.2010.01.040.

Morabit, M., Desaulniers, G., & Lodi, A. (2021). Machine-learning–based column selection for column generation. *Transportation Science*, *55*(4), 815–831, https://doi.org/10.1287/trsc.2021.1045.

Morabit, M., Desaulniers, G., & Lodi, A. (2023). Machine-learning-based arc selection for constrained shortest path problems in column generation. *INFORMS Journal on Optimization*, *5*(2), https://doi.org/10.1287/ijoo.2022.0082.

Nazari, M., Oroojlooy, A., Snyder, L. V., & Takáč, M. (2018). Deep reinforcement learning for solving the vehicle routing problem. *Advances in neural information processing systems*, *31*.

Quesnel, F., Desaulniers, G., & Soumis, F. (2020a). A branch-and-price heuristic for the crew pairing problem with language constraints. *European Journal of Operational Research*, *283*(3), 1040–1054, https://doi.org/10.1016/j.ejor.2019.11.043.

Quesnel, F., Desaulniers, G., & Soumis, F. (2020b). Improving air crew rostering by considering crew preferences in the crew pairing problem. *Transportation Science*, *54*(1), 97–114, https://doi.org/10.1287/trsc.2019.0913.

Quesnel, F., Wu, A., Desaulniers, G., & Soumis, F. (2022). Deep-learning-based partial pricing in a branch-and-price algorithm for personalized crew rostering. *Computers & Operations Research*, *138*, 105554, https://doi.org/10.1016/j.cor.2021.105554.

Saddoune, M., Desaulniers, G., Elhallaoui, I., & Soumis, F. (2012). Integrated airline crew pairing and crew assignment by dynamic constraint aggregation. *Transportation Science*, *46*(1), 39–55, https://doi.org/10.1287/trsc.1110.0379.

Salismans, T., Ho, J., Chen, X., Sidor, S., & Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint*, https://doi.org/10.48550/arXiv.1703.03864.

Schaefer, A. J., Johnson, E. L., Kleywegt, A. J., & Nemhauser, G. L. (2005). Airline crew scheduling under uncertainty. *Transportation science*, *39*(3), 340–348, https://doi.org/10.1287/trsc.1040.0091.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *International conference on machine learning* (pp. 387–395). PMLR.

Tahir, A., Quesnel, F., Desaulniers, G., Elhallaoui, I., & Yaakoubi, Y. (2021). An improved integral column generation algorithm using machine learning for aircrew pairing. *Transportation Science*, *55*(6), 1411–1429, https://doi.org/10.1287/trsc.2021.1084.

Vinyals, O., Fortunato, M., & Jaitly, N. (2015). Pointer networks. *Advances in Neural Information Processing Systems*, *28*.

Yaakoubi, Y., Soumis, F., & Lacoste-Julien, S. (2020a). Flight-connection prediction for airline crew scheduling to construct initial clusters for OR optimizer. *arXiv preprint*, https://doi.org/10.48550/arXiv.2009.12501.

Yaakoubi, Y., Soumis, F., & Lacoste-Julien, S. (2020b). Machine learning in airline crew pairing to construct initial clusters for dynamic constraint aggregation. *EURO Journal on Transportation and Logistics*, *9*(4), 100020, https://doi.org/10.1016/j.ejtl.2020.100020.

Zeighami, V., Saddoune, M., & Soumis, F. (2020). Alternating lagrangian decomposition for integrated airline crew scheduling problem. *European Journal of Operational Research*, *287*(1), 211–224, https://doi.org/10.1016/j.ejor.2020.05.005.

Zeighami, V. & Soumis, F. (2019). Combining Benders' decomposition and column generation for integrated crew pairing and personalized crew assignment problems. *Transportation Science*, *53*(5), 1479–1499, https://doi.org/10.1287/trsc.2019.0892.

Zhou, S.-Z., Zhan, Z.-H., Chen, Z.-G., Kwong, S., & Zhang, J. (2020). A multi-objective ant colony system algorithm for airline crew rostering problem with fairness and satisfaction. *IEEE Transactions on Intelligent Transportation Systems*, *22*(11), 6784–6798, https://doi.org/10.1109/TITS.2020.2994779.

Zhou, T., Chen, X., Wu, X., & Yang, C. (2022). A hybrid multi-objective genetic-particle swarm optimization algorithm for airline crew rostering problem with fairness and satisfaction. In *International Conference on Machine Learning for Cyber Security* (pp. 563–575).: Springer.

Appendix A Pseudo-code for the feasibility heuristic

In this section, we detail further the feasibility heuristic presented in Section 5.2. We give the following pseudo-code to describe the concrete steps taken to make one pilot's schedule feasible.

Pseudo-code for the feasibility heuristic

$pairings \leftarrow pairings.sortbylength()$
$short_pairings \leftarrow pairings.keep_shorts()$ // keeps pairings with few duties only
$short_pairs \leftarrow short_pairings.make_combinations()$ // all pairs of pairings with few duties
$credits \leftarrow pairings.lengths.sum()$

Function $trim_schedule(pairings, rests)$ // removes pairings until credits do not exceed the limit
for $pairing$ **in** $short_pairings$ **do** // if one short pairing is enough, remove it and make updates
    if $credits - pairing.length \leq T^{credit}$ **then**
        $pairings.remove(pairing)$
        $rests \leftarrow rests + pairing.duties$ **and return**
for $pair$ **in** $short_pairs$ **do** // if one pair of short pairings is enough, remove them and make updates
    if $credits - pair.length.sum() \leq T^{credit}$ **then**
        $pairings.remove(pair)$
        $rests \leftarrow rests + pair.duties.sum()$ **and return**
while $credits > T^{credit}$ **do** // otherwise, remove pairings until below limit
    $pairing \leftarrow pairings.next_long_enough()$
    if $pairing$ **is not** NULL **then** // remove first pairing that is long enough, if any
        $pairings.remove(pairing)$
        $rests \leftarrow rests + pairing.duties$
    else // if none, remove the last (and longest) pairing instead
        $pairing \leftarrow pairings.last()$
        $pairings.remove(pairing)$
        $rests \leftarrow rests + pairing.duties$
return

Function $add_rest(pairings, rests)$ // removes pairings until there are enough days off
$pairings \leftarrow pairings.sortbyduties()$
while $rests < T^{off}$ **do**
    $pairing \leftarrow pairings.next()$
    $pairings.remove(pairing)$
    $rests \leftarrow rests + pairing.duties$
return

if $credits > T^{credit}$ **then** // call $trim_schedule$ if too many credits
    $pairings$, $rests \leftarrow trim_schedule(pairings, rests)$
if $rests < T^{off}$ **then** // call add_rests afterwards if still not enough days off
    $pairings$, $rests \leftarrow add_rest(pairings, rests)$