INDERSCIENCE PUBLISHERS
Linking academia, business and industry through research

European J. of Industrial Engineering

ISSN online: 1751-5262 - ISSN print: 1751-5254
https://www.inderscience.com/ejie

Solving airline crew pairing problems through constraint partitioning

Maryam Radman, Kourosh Eshghi

DOI: 10.1504/EJIE.2023.10044607

Article History:

Received:	08 May 2020
Accepted:	29 December 2021
Published online:	15 December 2022

Solving airline crew pairing problems through constraint partitioning

Maryam Radman* and Kourosh Eshghi

Department of Industrial Engineering,
Sharif University of Technology,
Tehran, Iran
Email: maryam.radman@ymail.com
Email: eshghi@sharif.edu
*Corresponding author

Abstract: In this paper, a decomposition technique based on constraint partitioning is developed to solve the crew pairing problem (CPP) which has an overriding importance in the airline industry as it determines the crew cost. The method is based on the observation that in large-scale problems, the constraints can be partitioned to some sub-problems which involve special subsets of variables. The resultant structure is called the 'partitioned structure'. Therefore, in the proposed method, first, a feasible solution is generated for a reduced CPP with a 'partitioned structure' through the optimal solutions of its sub-problems. Then, at each step, the feasible solution is improved through adding/removing some pairings to/from it. The proposed algorithm is applied to a case study from the literature as well as some randomly generated test problems. One advantage of the proposed method is finding multiple feasible solutions with lower time than the method used to solve the case. [Submitted: 8 May 2020; Accepted: 29 December 2021]

Keywords: crew pairing problems; CPPs; constraint partitioning; decomposition technique; sub-problem; airline industry.

Reference to this paper should be made as follows: Radman, M. and Eshghi, K. (2023) 'Solving airline crew pairing problems through constraint partitioning', *European J. Industrial Engineering*, Vol. 17, No. 1, pp.29–59.

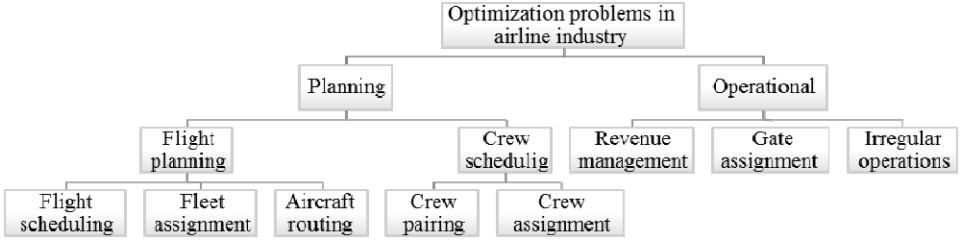
Biographical notes: Maryam Radman obtained her PhD in Industrial Engineering from the Sharif University of Technology, Tehran, Iran in 2021. Her research interests include mathematical programming, discrete optimisation, decomposition algorithms and queuing theory. She has published several peer-reviewed papers in reputed international journals in recent years.

Kourosh Eshghi is a Professor in the Department of Industrial Engineering at the Sharif University of Technology. He received his PhD in Mathematical Modelling form University of Toronto in 1997. His interests include graph theory, mathematical programming, meta-heuristic algorithms, decision theory, and disaster management. He is the author of five textbooks and has published over one hundred academic papers in national and international journals and conferences.

1 Introduction

There are various optimisation problems in the airline industry. As shown in Figure 1, these problems are broken down into planning and operational. The planning problems are further broken down into flight planning problems containing flight scheduling, fleet assignment and aircraft routing problems and crew scheduling problems containing crew pairing and crew assignment problems (CAPs). The operational problems are also sub-divided into revenue management, gate assignment and irregular operations problems. These issues are interrelated such that the output of one problem is the input of another problem (Deveci and Demirel, 2018a, 2018b).

Figure 1 Classification of optimisation problems in the airline industry



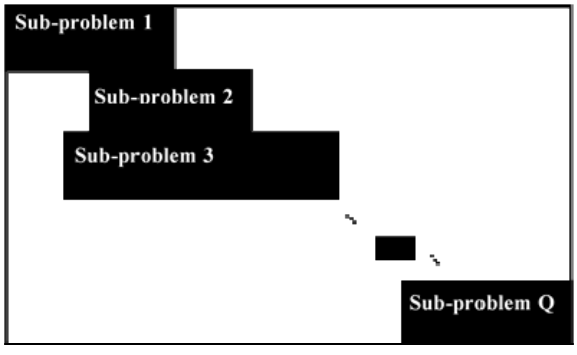
The crew scheduling problem is one of the main problems, constituting one of the biggest costs for an airline company, which is composed of the crew pairing problem (CPP) and the CAP. In the former, the minimal-cost pairings which cover all the scheduled flights of the airline companies are generated, while in the latter, the crews are assigned to the generated pairings. The main part of the crew cost is related to the CPP, while CAP aims at increasing the level of equality in the crew's schedule. The fiercely competitive airline market has further brought to the fore the importance of working out an acceptable solution to this problem.

In this paper, the CPP is modelled as a set covering problem (SCP) and a new decomposition technique based on constraint partitioning is developed to solve it. The proposed method is based on the observation that in real-world large-scale problems, each constraint does not involve all variables of the problem; but the constraints can be partitioned to some sub-problems which involve special subsets of variables (Kato and Sakawa, 2003; Elfeky, 2009). The resultant structure is called the 'partitioned structure'. As shown in Figure 2, this structure is simply obtained by partitioning the similar constraints of the problem in the same sub-problems. Therefore, each two sub-problems may have variables in common with each other.

In this paper, first, a framework is proposed to generate a feasible solution to an SCP with a 'partitioned structure', using the optimal solutions of its sub-problems. Since in this method, the original problem is solved through the smaller sub-problems containing fewer variables and constraints, its complexity diminishes. Then, the proposed framework is extended to solve a CPP. The proposed method starts from an initial set of feasible pairings which cover all the scheduled flights. Then, after partitioning the constraints of the problem, the feasible solution of the 'partitioned structure' is calculated. Afterwards, at each iteration, some of the pairings of the current solution are removed from the problem and the new pairings are added to the problem such that this feasible solution is strictly improved. There are two sub-problems in the proposed

method which form the basis for the addition and removal of the pairings. The removal sub-problem is a knapsack model which determines the pairings that should be removed. The addition sub-problem aims to generate new pairings in order to improve the feasible solution of the 'partitioned structure'. In addition, the implementation of the proposed algorithm is simplified when the knapsack problem is solved by a greedy method. The details of the proposed algorithm are explained in Sections 3 and 4.

Figure 2 The 'partitioned structure' of the problem



Notes: The constraints and the variables of the problem are represented in the rows and the columns of the matrix, respectively. Each two sub-problems may involve common variables.

The results of applying the proposed algorithm to a case study in Section 5 testify to its strengths vis-à-vis the previous method used to solve the case.

The method proposed in this paper, which is based on constraint partitioning, is novel and original as it has never been studied in the literature before. In addition, the application of the clustering methods to group the similar constraints in the same sub-problems can be considered another contribution of this study.

The performance of the proposed algorithm is evaluated on a case study, whose data was available in the literature of CPPs as well as on some randomly-generated test problems, involving up to 320 flights. The main advantages of the proposed algorithm vis-à-vis the previous method for the case study are the generation of various final solutions and the lower solution time.

The paper is organised as follows. The literature review of solving CPPs is presented in Section 2. In Section 3, the theoretical framework of the paper is presented. Section 4 explains the proposed algorithm. The experimental results are reported in Section 5. Finally, the concluding remarks are presented in Section 6.

2 Problem settings and discussion of related work

2.1 Problem definition

Before defining the problem, some terminological explanation should be in order. The time between two consecutive flights is called the connection time. A sequence of flights that start and end at the crew base (also called home base) is called a pairing. If the crew of a pairing attend as passengers at a flight in order to transport to the start city of another

flight or return to the home base, that flight is called a deadhead flight for them. Deadhead flights are not desirable for airline companies as they put a strain on the capacity of the flight for passengers and on crew utilisation. If a pairing satisfies the rules set by airline companies, labour unions and the government, it is called a feasible pairing. These rules set standards for the duration of pairings, total flight time (FT) of pairings, minimum and maximum connection time, etc.

The CPP aims to generate the minimal-cost subset of feasible pairings which cover all flights of the airline company. The CPP can be modelled as a set covering or partitioning problem. The set covering model of a CPP is as follows (Zeren and Özkol, 2016):

$$\min \sum_{j} c_j x_j \tag{1}$$

$$\sum_{j} a_{ij} x_j \geq 1 \qquad \forall i \in F \tag{2}$$

$$x_j \in \{0,1\} \qquad \forall j \in P \tag{3}$$

In the above model, F and P denote the set of flights and feasible parings, respectively. In this model, a_{ij} is a parameter which gets value 1 if flight i is covered by pairing j. If the j^{th} pairing is chosen, x_j gets value 1 and 0 otherwise. Equation (1) calculates the cost of pairings and equation (2) guarantees that each flight is covered by at least one pairing. If the right-hand side of equation (2) is equal to one for all flights, the set partitioning model of the CPP is obtained; otherwise, the flights for which equation (2) is satisfied as (>1) are deadhead flights.

2.2 Solution methods

An appropriate solution to this problem can reduce the operating costs of the crew and, consequently, accrue more profits for airline companies. Since the crew cost is one of the biggest costs (making up about 20% of the total cost), second only to the fuel cost, any shrinkage in this cost, however slight, can ultimately end up saving millions of dollars for airline companies (Wu et al., 2016).

The numbers of pairings are usually about 200,000 for small CPPs, about one billion for medium-sized ones and several billions for large-scale CPPs (Klabjan et al., 2001). Therefore, it is impossible to generate all feasible pairings in a CPP. As late as the early 1990s, heuristics were used to solve CPPs which gave way to the use of mathematical optimisation techniques (Kasirzadeh et al., 2017). Generally speaking, two types of techniques are used to solve this problem:

1 The first technique consists of two phases: pairing generation and optimisation. In the former, a large amount of feasible pairings are generated. In the latter, an optimisation set covering or partitioning problem is solved in order to choose the optimal subset of pairings generated in the first phase to cover all flights. This technique is called offline column generation. The optimisation problem in the second phase can be solved using exact or heuristic methods.

2 The second technique is the most commonly used technique in the CPP literature whereby the phase of pairing generation is simultaneously solved with the optimisation phase. This technique is called dynamic column generation or, in short,

column generation. Column generation is used to solve problems containing a large number of variables in such a way that it is not possible to generate all of them. It guarantees the optimal solution of the linear programming (LP) relaxation model of the problem. In each step of this method, a new variable is generated and added to the LP relaxed problem, leading to the strict improvement of its optimal solution. For a detailed description of the method, see Wu et al. (2016).

AhmadBeygi et al. (2009), Aksoy (2010), Dück et al. (2011), Rasmussen et al. (2011), Reisi and Moslehi (2013), Saddoune et al. (2013), Zeren and Özkol (2016), Wu et al. (2016) and Kasirzadeh et al. (2017) are examples of studies having applied the column generation method to solve the CPP problem.

In Klabjan et al. (2001), first, the LP relaxation model of the problem is solved using column generation then, some pairings with the best reduced cost are chosen for the second phase where a branch and bound method is applied to find the minimal-cost subset of pairings. Kornilakis and Stamatopoulos (2002) and Zeren and Özkol (2012) have developed offline column generation techniques using genetic algorithms for the optimisation phase.

In Aydemir-Karadag et al. (2013), three methods have been developed to solve the CPP. In the first method, some feasible pairings are generated using a heuristic method; then, their minimal-cost subset is chosen using CPLEX. In the second method, following the generation of the initial feasible pairings, a genetic algorithm is implemented to improve them; finally, CPLEX is applied to choose their minimal-cost subset. Column generation has been used as the third method whose initial pairings are the output of the two previous methods.

To solve the CPP, Erdoğan et al. (2015) have used a metaheuristic algorithm based on a neighbourhood search method to construct the pairings and have then applied an optimisation-driven heuristic, employing these pairings to solve a set partitioning problem to select their best possible subset.

Agustín et al. (2016) proposed a metaheuristic method based on biased randomisation to solve the CPP. In Demirel and Deveci (2017), a dynamic genetic algorithm has been developed where the length of the chromosome changes dynamically in each iteration and consequently in the optimisation run. In this method, after generating all legal parings, subsets of them are chosen. Then, this subset dynamically changes during the algorithm by exchanging pairings between itself and the main set of pairings. In Novianingsih and Hadianti (2018), a two-phase heuristic method is developed to solve the CPP. In the first phase, a feasible solution is constructed based on a greedy method which maximises the covered flights. In the second phase, the solution is improved in such a way that it avoids local optimal solutions.

In Deveci and Demirel (2018b), an offline column generation technique is developed. The depth-first search method is used for the pairing generation phase. For the optimisation phase, three heuristics, containing two genetic algorithms and a memetic algorithm, are used.

Aggarwal et al. (2018) proposed three modifications for pairing generation, namely

1 generating parallel pairing

2 improving the depth-first search method for generating legal pairings in the network structure

3 reducing the search space by filtering the generated legal pairings whose corresponding cost indicator is greater than a given number.

Masipa (2019) developed two heuristic solutions for generating legal pairings based on greedy and random tree searches, respectively. These approaches have also been adapted for a stochastic environment where weather delays may occur.

Haouari et al. (2019) developed a new nonlinear formulation for CPPs that contains a polynomial number of constraints and variables, compared to the exponential number of variables (pairings) of their traditional set-covering or partitioning formulation. Then, the proposed formulation is linearised using reformulation linearisation techniques. Moreover, to improve the computational performance of the proposed formulation, two enhancement strategies involving tightening the variable bounds and adding the valid inequalities have been developed. Finally, it is shown that the resulting model can be efficiently solved using commercial solvers (e.g., CPLEX) without complicated algorithmic implementations.

Quesnel et al. (2020) introduced a new variant of the problem, called CPPs with language constraints, to generate final pairings that are less violative of crew members' language requirements. The proposed problem is modelled as a mixed-integer program, solved by column generation, and shown to generate more suitable pairings than the traditional CPP.

Aggarwal et al. (2020a) designed a framework, called airline crew pairing optimisation framework, which consists of three parts:

1 legal crew pairing generator to generate the feasible pairings through a parallel architecture

2 initial feasible solution generator to select a set of feasible pairings of the previous part covering all flights in a reasonable time to initialise the next part

3 optimisation engine including linear and integer solution generator.

In addition, a sensitivity analysis of the proposed framework is performed on the cost quality and running time of the initial feasible solution.

In Aggarwal et al. (2020c), the initial feasible solution generator of the previous paper is explained in more details. This generator is a heuristic based on a divide-and-cover strategy and integer programming. In the proposed heuristic, the flight set is decomposed into smaller subsets. Then all legal pairings for each subset are generated, and a subset with the minimum cost is selected from them. Finally, the pairing subsets with the minimum cost are combined to form the initial feasible solution. In Aggarwal et al. (2020e), a learning mechanism is developed to support the column generation algorithm of the optimisation engine of Aggarwal et al. (2020a). This learning mechanism identifies a set of critical flights using the available flight connection information to construct new legal pairings of the pricing sub-problem. In Aggarwal et al. (2020b), the column generation algorithm of the optimisation engine of Aggarwal et al. (2020a) is extended to generate a lower-cost solution compared to a standard column generation algorithm. In fact, in the pricing sub-problem of the proposed column generation heuristic, four strategies are developed to prioritise the generation of pairings compared to the single strategy of a standard column generation algorithm in which a pairing with the most negative shadow price is generated.

Aggarwal et al. (2020d) proposed a genetic algorithm with improved initialisation phase and genetic operators, vis-a-vis previous works, and compared its efficiency to a column generation approach to solving CPPs.

Yaakoubi et al. (2020) proposed some improvements to a previously proposed baseline algorithm for CPPs. The presented modifications are based on machine learning techniques that cluster the flights with a high probability of being consecutive in a solution. The improved algorithm was implemented in a commercial solver called GENCOL-DCA. The new algorithm provides better results at a lower cost, which was reduced by 8.25%.

Desaulniers et al. (2020) proposed a column generation algorithm for solving CPPs. The important factor of this algorithm is using a dynamic constraint aggregation technique that exploits the degeneracy of the master problem, resulting in the addition of fewer constraints to it. The performance of the combined column generation is compared to that of a standard column generation algorithm.

3 Theoretical framework

In this section, first, a feasible solution is generated for an SCP with a 'partitioned structure' through the optimal solutions of its sub-problems; then, the conditions of adding a new variable to the SCP, which reduces the objective value of this feasible solution, are obtained. As shown in Figure 2, the 'partitioned structure' of an SCP is obtained by grouping its similar constraints into the same sub-problems. To illustrate this point, consider Figure 3. In addition, more details about exploiting this structure is explained in Subsection 5.1 (Partition_Alg).

Suppose P is an SCP with a 'partitioned structure' containing Q sub-problems. P_i ($\forall i = 1, 2, \ldots, Q$) is the i^{th} sub-problem of this structure and V_i and $Const_i$($\forall i = 1, 2, \ldots, Q$) are the set of variables and constraints of P_i, respectively. The mathematical model of sub-problem P_i($\forall i = 1, 2, \ldots, Q$) is as follows:

$$P_i : \min \sum\nolimits_{x_j \in V_i} c_j x_j \tag{4}$$

$$\sum\nolimits_{x_j} a_{kj} x_j \geq 1 \qquad k \in Const_i \tag{5}$$

$$x_j \in \{0, 1\} \qquad \forall x_j \in V_i \tag{6}$$

As it can be seen, the structure of sub-problem P_i is similar to an SCP; therefore, any existing algorithms for SCPs can also be used to solve P_i. In order to generate a feasible solution for P through the optimal solutions of P_i($\forall i = 1, 2, \ldots, Q$), a simple method is proposed in Figure 4. In this method, n denotes the number of the variables of P.

Let us denote the solution obtained from the above method by S^f. Note that, since in step 3, S is obtained by the union of sets S_1^* to S_Q^*, therefore, each variable receives the maximum value generated by S_1^* to S_Q^* for it. Therefore S^f is feasible for sub-problems P_1 to P_Q and, consequently, problem P. For instance, according to Figure 3, the optimal sets for sub-problems P_1 to P_3 are $S_1^* = \{x_1, x_2\}$, $S_2^* = \{x_4\}$ and $S_3^* = \{x_3, x_4\}$,

respectively. Therefore, $S^f = \{x_1, x_2, x_3, x_4\}$, which is obviously feasible for the whole problem.

Figure 3 (a) An SCP (b) The 'partitioned structure' of the SCP obtained by grouping its similar constraints into the same sub-problems (c) The sub-problems of the 'partitioned structure'

$$\boldsymbol{P} = \min \quad 2x_1 + 2x_2 + 3x_3 + x_4 + 3x_5 + 5x_6$$

$$\begin{aligned}
C1:&\quad x_1 && && && \geq 1\\
C2:&\quad && x_3 + && x_6 && \geq 1\\
C3:&\quad && x_3 + x_4 && && \geq 1\\
C4:&\quad && && x_4 + && x_6 \geq 1\\
C5:&\quad x_2 + && x_3 && && \geq 1\\
C6:&\quad x_2 && && && \geq 1\\
C7:&\quad && && x_4 + && x_5 \geq 1
\end{aligned}$$

$$x_j \in \{0,1\} \quad j = 1, \dots, 6$$

(a)

$$\boldsymbol{P} = \min \quad 2x_1 + 2x_2 + 3x_3 + x_4 + 3x_5 + 5x_6$$

$$\begin{aligned}
C1:&\quad x_1 && && && \geq 1\\
C6:&\quad && x_2 && && \geq 1\\
C5:&\quad && x_2 + && x_3 && \geq 1\\
C3:&\quad && && x_3 + x_4 && \geq 1\\
C7:&\quad && && x_4 + && x_5 \geq 1\\
C4:&\quad && && x_4 + && x_6 \geq 1\\
C2:&\quad x_3 + && && && x_6 \geq 1
\end{aligned}$$

(b)

$$\boldsymbol{P_1} = \min \quad 2x_1 + 2x_2 + 3x_3$$

$$\begin{aligned}
C1:&\quad x_1 && && \geq 1\\
C6:&\quad && x_2 && \geq 1\\
C5:&\quad && x_2 + && x_3 \geq 1
\end{aligned}$$

$$x_j \in \{0,1\} \quad j = 1,2,3$$

$$\boldsymbol{P_2} = \min \quad 3x_3 + x_4 + 3x_5$$

$$\begin{aligned}
C3:&\quad x_3 + x_4 && && \geq 1\\
C7:&\quad && x_4 + && x_5 \geq 1
\end{aligned}$$

$$x_j \in \{0,1\} \quad j = 3,4,5$$

$$\boldsymbol{P_3} = \min \quad 3x_3 + x_4 + 5x_6$$

$$\begin{aligned}
C4:&\quad && x_4 + && x_6 \geq 1\\
C2:&\quad x_3 + && && x_6 \geq 1
\end{aligned}$$

$$x_j \in \{0,1\} \quad j = 3,4,6$$

(c)

It should be noted that the authors have shown, for 45 well-known instances from OR-Library (http://people.brunel.ac.uk/~mastjjb/jeb/orlib/scpinfo.html) (problem sets 4, 5, 6, A, B, C and D), the percentage difference of feasible solution S^f from the optimal value is 2.36% on average. In other words, through proper clustering of the constraints of these instances, leading to the 'partitioned structure', the relative percentage difference is 2.36%, on average.

Figure 4 The steps of generating a feasible solution for an SCP with a 'partitioned structure' through the optimal solutions of its sub-problems

Step 1: Find the optimal solution of subproblem P_i $(\forall\ i = 1, \dots, Q)$ and define S_i^* $(\forall\ i = 1, \dots, Q)$ as the set of variables with value 1.
Step 2: Calculate $S = S_1^* \cup S_2^* \cup \dots \cup S_Q^*$.
Step 3: If $x_j \in S,\ (\forall\ j = 1, \dots, n)$, it gets value 1 and 0 otherwise.

In what follows, the effect of adding a new variable to problem P on feasible solution S^f is examined. Suppose that $P(n)$ denotes problem P with n variables whose constraints are partitioned to Q sub-problems $P_1(n)$ to $P_Q(n)$. The optimal solution of $P(n)$ is represented by $S^*(n)$ and its feasible solution generated by the above method is shown by $S^f(n)$.

Now, suppose that new variable x_{n+1} is added to $P(n)$. According to the steps of Figure 4, we have the following equation for the new problem $P(n + 1)$.

$$S^f(n+1) = \bigcup_{i=1}^{Q} S_i^*(n+1) \tag{7}$$

Suppose that v_{ij} is the binary value assigned to variable $x_j(\forall j = 1, \ldots, n+1)$ by $S_i^*(n+1)$ $(\forall i = 1, \ldots, Q)$. According to equation (7), we have:

$$f\left(S^f(n+1)\right) = \sum_{j=1}^{n+1} c_j \max\left(v_{1j}, \ldots, v_{Qj}\right) \tag{8}$$

Now, we look for the conditions on new variable x_{n+1}, so that by its entrance to problem $P(n)$, $f(S^f(n+1))$ gets a value less than $f(S^f(n))$. Therefore, we have:

$$f\left(S^f(n+1)\right) < f\left(S^f(n)\right) \Rightarrow \sum_{j=1}^{n+1} c_j \max\left(v_{1j}, \ldots, v_{Qj}\right) < f\left(S^f(n)\right) \tag{9}$$

We assume $f(S^f(n)) = A$ and $\max(v_{1j}, \ldots, v_{Qj}) = z_j$. Therefore, equation (9) equals:

$$\sum_{j=1}^{n+1} c_j z_j < A \tag{10}$$

Obviously, the new variable x_{n+1} should get value 1 by at least one of $S_i^*(n+1)$ $(\forall i = 1, \ldots, Q)$; otherwise $f(S^f(n+1)) = f(S^f(n))$ and equation (10) is not satisfied. In addition, it is assumed that c_{n+1} is known. Therefore, equation (10) equals:

$$\sum_{j=1}^{n} c_j z_j + c_{n+1} < A \Rightarrow \sum_{j=1}^{n} c_j z_j < A - c_{n+1} \tag{11}$$

In the above equation, A and c_{n+1} are parameters and $z_j(\forall j = 1, \ldots, n)$ is a variable. In order to find the best value for each $z_j(\forall j = 1, \ldots, n)$, we can consider an objective function in the form $\max \sum_{j=1}^{n} w_j z_j$ in which $w_j(\forall j = 1, \ldots, n)$ is the objective coefficient of z_j (the reason for considering this form for the objective function is explained later). Consequently, the following optimisation problem is obtained:

$$\max \sum_{j=1}^{n} w_j z_j$$

s.t.

$$\begin{aligned} &\sum_{j=1}^{n} c_j z_j < A - c_{n+1} \\ &z_j = \{0, 1\} \qquad\qquad \forall j = 1, \ldots, n \end{aligned} \tag{12}$$

Model (12) is a knapsack problem. After solving the above model:

Assumption 1: If $z_j = 1$ $(\forall j = 1, \ldots, n)$, then $\max(v_{1j}, \ldots, v_{Qj}) = 1$. As a result, x_j should be the member of at least one of optimal solutions $S_1^*(n+1)$ to $S_Q^*(n+1)$.

Assumption 2: If $z_j = 0$ $(\forall j = 1, \ldots, n)$, then $\max(v_{1j}, \ldots, v_{Qj}) = 0$. As a result, x_j should not be present in any of the optimal solutions $S_1^*(n+1)$ to $S_Q^*(n+1)$.

Note that our goal is to satisfy $\sum_{j=1}^{n} c_j z_j < A - c_{n+1}$ in equation (12), so only the results of the second assumption are significant. To understand why that is the case, assume that after solving model (12), $z_l = 1$, ($l = 1, \ldots, n$), but x_l is not a member of any S_i^* $(n + 1)$, $\forall j = 1, \ldots, Q$. Therefore, z_l equals zero and c_l is reduced from the left-hand side of $\sum_{j=1}^{n} c_j z_j < A - c_{n+1}$, implying that it still remains satisfied. In addition, according to this result, the reason for considering the maximisation form for the objective function of model (12) is that variables $z_j (\forall j = 1, \ldots, n)$ get value one as far as possible; therefore, the results of the second condition need not to be considered in the problem.

We define $X_0 = \{x_j | z_j = 0$ in the solution of model (12)$\}$, then according to the second assumption, the members of X_0 should get value zero in the optimal solutions of all sub-problems $P_i(n + 1)$, $\forall i = 1, \ldots, Q$. Therefore, they can be temporarily removed from problem $P(n + 1)$, because they have no effect on the optimal solutions of the sub-problems. Since variables X_0 are present in some of the constraints of $P(n + 1)$, their removal from the problem results in two cases:

- Case 1: Some constraints with no variables are created.
- Case 2: All constraints contain at least one variable.

The interpretation of the first case in a CPP is that a number of flights are not covered if pairings X_0 are removed from the problem. In this case, we consider F_0 as the set of uncovered flights. The members of F_0 give insights about producing the coefficient column of the new variable x_{n+1}. Consequently, the new variable x_{n+1} should satisfy the two following conditions in order to improve feasible solution S^f:

- Condition 1: The cost of x_{n+1} should be less than the pre-considered value c_{n+1}.
- Condition 2: The coefficient column of x_{n+1} should cover flights F_0.

In the second case, all flights are still covered if pairings X_0 are removed from the problem. Therefore, a new feasible solution for the CPP is obtained, whose objective function is less than $A - c_{n+1}$ according to equation (12).

A noteworthy point here is that airline companies have special rules for generating their pairings. As a result, one may not satisfy the above two conditions by generating only one pairing according to these rules. For instance, airline companies place a cap on the number of flights of a pairing. Therefore, if $|F_0|$ or the number of the members of the coefficient column of x_{n+1} is more than this limited value, one feasible pairing alone cannot satisfy the second condition.

Therefore, in the following, the effect of adding more than one new variable, i.e., h variables, to problem $P(n)$ is examined. As in the case of adding one new variable, we have the following equations:

$$f\left(S^f(n+h)\right) < f\left(S^f(n)\right) \Rightarrow \sum_{j=1}^{n+h} c_j \max\left(v_{1j}, \ldots, v_{Qj}\right) < f\left(S^f(n)\right) \tag{13}$$

$$\sum_{j=1}^{n+h} c_j z_j < A \tag{14}$$

In a similar way to the process of adding one new variable, $z_{n+l} = 1$ ($\forall l = 1, \ldots, h$). In addition, we assume $C = \sum_{l=1}^{h} c_{n+l}$. Therefore, equation (14) equals:

$$\sum_{j=1}^{n} c_j z_j < A - C \tag{15}$$

Consequently, the following knapsack model is obtained:

$$\max \sum_{j=1}^{n} w_j z_j$$

s.t.

$$\begin{aligned} &\sum_{j=1}^{n} c_j z_j < A - C \\ &z_j = \{0, 1\} \qquad\qquad \forall j = 1, \ldots, n \end{aligned} \tag{16}$$

As it can be seen, the above knapsack model for more than one variable is the same as model (12) for one variable. As a result, when adding several new variables, their conditions for improving the feasible solution S^f are as follows:

- Condition 1: The sum of the costs of $x_{n+l} = 1$ ($\forall l = 1, \ldots, h$) should be less than C.
- Condition 2: Each member of F_0 should be covered by at least one of the coefficient columns of $x_{n+l} = 1$ ($\forall l = 1, \ldots, h$).

4 The proposed algorithm

In this section, first, the proposed algorithm to solve a CPP is explained; then, the implementation of the proposed algorithm is simplified when the knapsack problem (16) is solved by a greedy method.

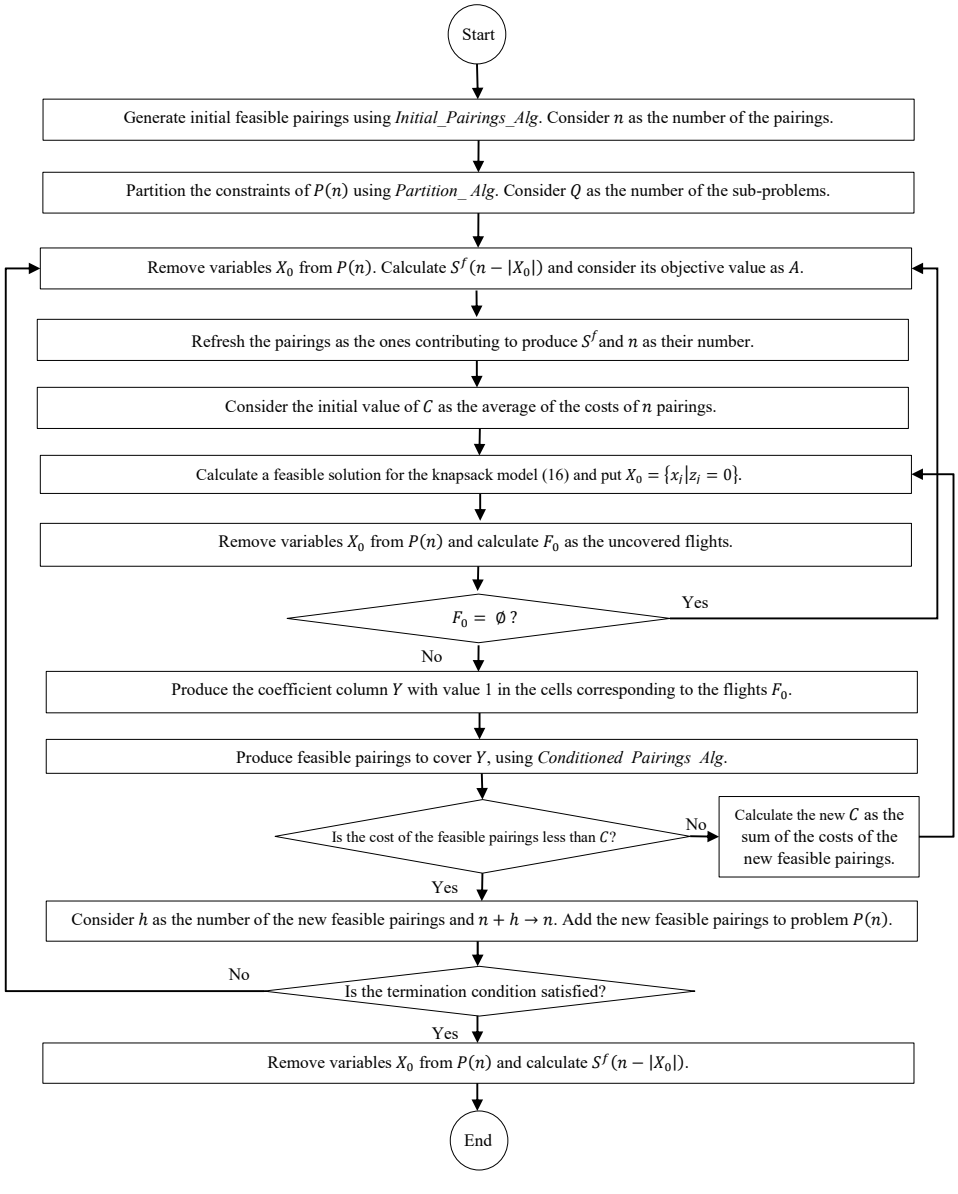
In general, the proposed algorithm starts from an initial set of feasible pairings which cover all the scheduled flights. Then, after partitioning the constraints of the problem, the feasible solution S^f is calculated. At each iteration of the algorithm, first the *removal sub-problem*, which is the Knapsack model (16), is solved to determine the set of pairings, X_0, to be removed from the problem. By removing X_0, the set of uncovered flights by the remaining pairings, F_0, is identified. Then, a problem called the *addition sub-problem*, which seeks to generate one or more pairings with a certain cost limit, C_h, to cover F_0, is solved. The process is repeated until the termination condition is satisfied. The flowchart of the proposed algorithm is shown in Figure 5.

According to the algorithm explained in Figure 5, first, an initial feasible solution is generated for the CPP, using the *Initial_Pairings_Alg*. The initial pairings are put in $P(n)$. Then, using the *Partition_Alg*, the constraints of $P(n)$ are grouped based on their similarity to each other, and the 'partitioned structure' is produced. The number of sub-problems in this structure is considered as Q.

Next, once pairings X_0 have been removed, the Q sub-problems are simultaneously solved and feasible solution S^f is generated by the union of these Q solutions as described

in Figure 4. As in generating S^f, the SCP is solved through the smaller sub-problems containing fewer variables and constraints, its complexity diminishes. The objective value of S^f determines parameter A in the knapsack model (16). In addition, the pairings which generate S^f are considered the new pairings of problem P. Note that at the beginning of the algorithm, $X_0 = \emptyset$, and after solving the knapsack problem for the first time, it is initialised. Parameter C in the knapsack problem is calculated as the average of the costs of the n pairings.

Figure 5 The flowchart of the proposed algorithm to solve a CPP based on constraint partitioning



In the next step, a feasible solution to the knapsack model is calculated and then, $X_0 = \{x_j | z_j = 0$ *in the solution of model* (16)$\}$ is formed. By removing pairings X_0 from $P(n)$, F_0, which consist of the uncovered flights, is calculated. If $F_0 = \varnothing$, a new feasible solution for the CPP with the objective value less than $A - C$ is obtained. Therefore, we return to the step of removing X_0 and calculating $S'(n - |X_0|)$. If $F_0 \neq \varnothing$, binary coefficient column Y with values one for the flights F_0 is formed.

By specifying

1 the maximum sum of the costs of the new pairings, C

2 the flights to be covered by them, Y, *Conditioned_Pairings_Alg* is used to generate new feasible pairings with the purpose of covering Y with the lowest possible cost.

If the sum of the costs of the newly-generated pairings is less than C, they are added to $P(n)$ and again the process is repeated for $P(n + h)$. Otherwise, C is reset as the costs of the new pairings in the knapsack problem. If the termination condition is satisfied, the final feasible solution S^f is calculated.

The steps involved in the three algorithms, i.e., *Initial_Pairings_Alg*, *Partition_Alg* and *Conditioned_Pairings_Alg*, in addition to some details regarding the implementation of the proposed algorithm including the termination condition and the algorithm used to solve the knapsack problem are explained in Section 5.

A simpler implementation of the proposed algorithm is also available since knapsack model (16) can be solved by a greedy method. It is shown in Section 5 that this new implementation of the algorithm can significantly reduce the solution time of the proposed algorithm.

In the greedy method to solve knapsack model (16), the variables are sorted in a descending order based on values w_j/c_j ($\forall j = 1, \ldots, n$). If the sorted variables are displayed by $z_{[j]}$ ($\forall j = 1, \ldots, n$), then, until $\sum_j z_{[j]} < A - C$, the variables get value 1. The details of this classic method are explained in Dantzig (1957). Assume in the resulted solution, X_1 and X_0 denote the set of the variables with value 1 and 0, respectively, therefore, we have:

$$\begin{aligned} &\text{if } \sum_{j=1}^{n-1} c_{[j]} < A - C \leq \sum_{j=1}^{n} c_{[j]} = A - \sum_{j=1}^{n} c_{[j]} \leq C < A - \sum_{j=1}^{n-1} c_{[j]} \\ &\Rightarrow X_1 = \{z_{[1]}, \ldots, z_{[n-1]}\}, X_0 = \{z_{[n]}\} \end{aligned} \tag{17}$$

Similarly, we have:

$$\begin{aligned} &\text{if } A - \sum_{j=1}^{n-1} c_{[j]} \leq C < A - \sum_{j=1}^{n-2} c_{[j]} \quad \Rightarrow \quad X_1 = \{z_{[1]}, \ldots, z_{[n-2]}\}, X_0 = \{z_{[n-1]}, z_{[n]}\} \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ &\text{if } A - c_{[1]} - c_{[2]} \leq C < A - c_{[1]} \quad \Rightarrow \quad X_1 = \{z_{[1]}\}, X_0 = \{z_{[2]}, \ldots, z_{[n]}\} \\ &\text{if } A - c_{[1]} \leq C < A \quad \Rightarrow \quad X_1 = \varnothing, X_0 = \{z_{[1]}, z_{[2]}, \ldots, z_{[n']}\} \end{aligned} \tag{18}$$

In the above equations, it is assumed that $A - \sum_{j=1}^{k} c_{[j]} = L_k$. The new implementation of the proposed algorithm is displayed in Figure 6. The main idea behind the second version

of the proposed algorithm is to perform a sensitivity analysis on the right-hand side value of the constraint of knapsack model (16). In fact, in the first version of the algorithm, changing C only affects this value, and the other components of the model remain unchanged. Therefore, it is possible to calculate the resulting changes in the current solution by performing a sensitivity analysis and without resolving the knapsack model, using equations (17) and (18).

According to Figure 6, the first four steps of this flowchart are similar to the previous version in Figure 5. In the fifth step, using the *Initial_Pairings_Alg*, some feasible pairings to cover all flights are generated, which are denoted by FP. Then, based on the results of equations (17) and (18), the initial member of X_0 is $z_{[n]}$ which should be removed from $P(n)$ to calculate F_0. Then an SCP is solved with pairings FP in order to cover F_0. The new pairings are considered *new_P*. If the sum of the costs of *new_P* is between L_n and L_{n-1}, then a new feasible solution with the objective value less than A is obtained. Otherwise, a new variable according to the ordering is added to $X_0(X_0 = \{z_{[n-1]}, z_{[n]}\})$ and the same procedure is repeated until the termination condition is satisfied.

5 Experimental results

Unfortunately, papers dealing with CPPs are often plagued with the lack of sample test problems to compare the performance of various algorithms on them (Aydemir-Karadag et al., 2013). In addition, the data of the case studies examined in the papers is private (Demirel and Deveci, 2017). In the papers reviewed, only one paper presented the time table of the scheduled flights of the airline company studied. Therefore, in order to evaluate the performance of the proposed algorithm, the real case studied in Agustín et al. (2016) is used in Subsection 5.1. In addition, larger instances with a similar structure to the case are randomly generated and their experimental results are reported in Subsection 5.2. In what follows, two versions of the proposed algorithm in Figures 5 and 6 are displayed by CPP_V1 and CPP_V2, respectively.

The proposed algorithm is coded in MATLAB and implemented on a computer with 2.27 GHz of CPU and 3 GB of RAM.

5.1 A case study

The case study of Agustín et al. (2016) consists of 41 flights on a five-day time horizon. The rules of the related company are as follows:

1 The maximum daily flight time (DFT) of a pairing is eight hours.

2 The maximum days of a pairing is limited to three days.

3 The minimum connection time between two flights is 45 minutes.

4 The home base of the airline company is Madrid (MAD).

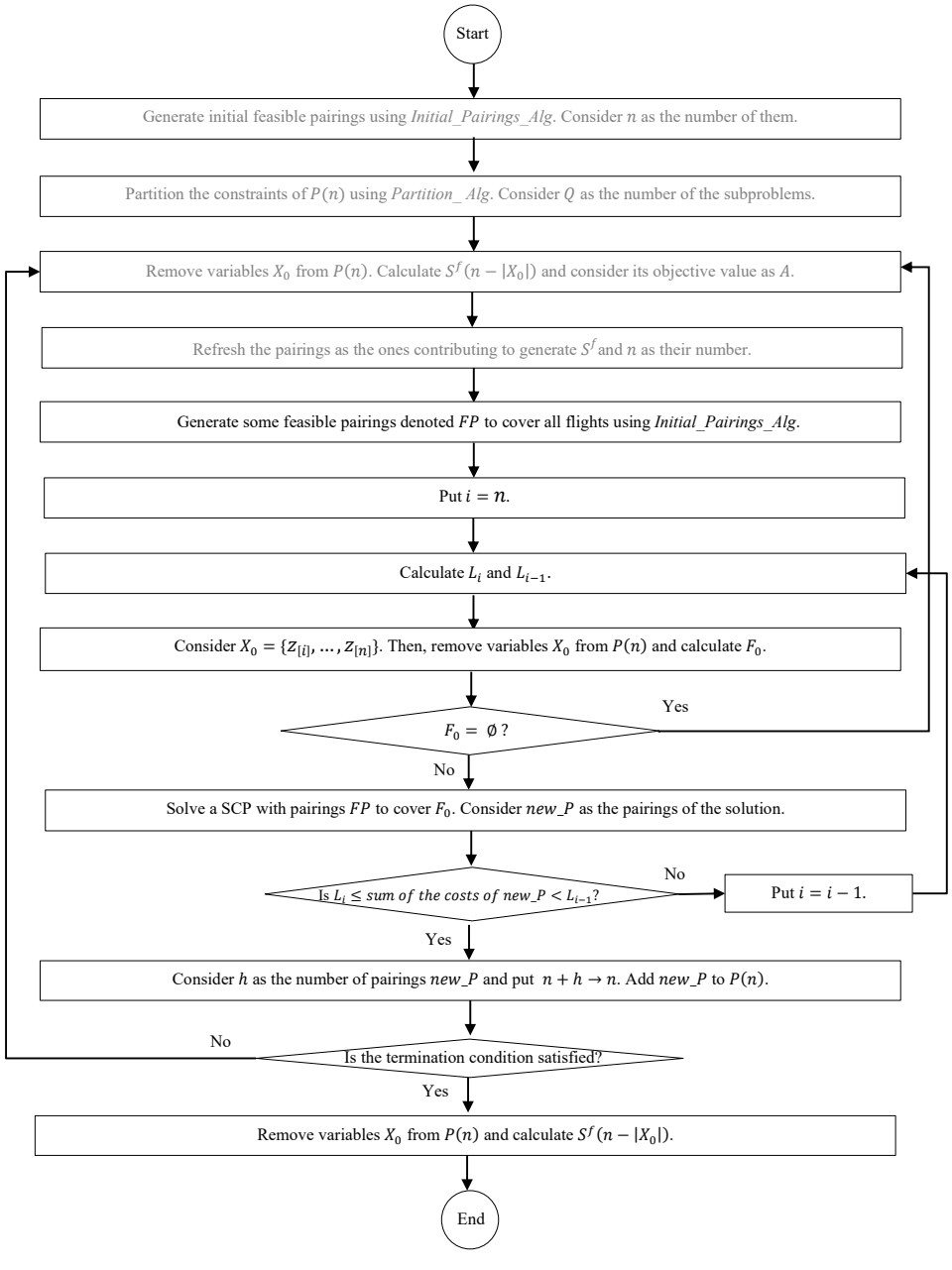
5 There is a deadheading possibility.

6 The days of a pairing must be consecutive.

7 The cost of a pairing is equal to its overall FT.

To view the details of the proposed algorithm for this case study and the time table of the flights, see Agustín et al. (2016). In the following, first, the details of implementing the proposed algorithm of this paper is explained; then, the experimental results are compared with those of the study done by Agustín et al. (2016).

Figure 6 The flow chart of the simpler implementation of the proposed algorithm when the knapsack model is solved by the greedy method of Dantzig (1957)



5.1.1 Initial_Pairings_Alg

For a better understanding of the steps involved this algorithm, which generates the initial feasible pairings to start the algorithm, some definitions should be in order, as presented in Table 1.

Table 1 The definitions of the terms used in the *Initial_Pairings_Alg*

Title	*Definition*
PairingList	The set of the feasible pairings generated by the algorithm
CurrentPairing	The pairing under generation
FlightTime	The total daily flight time of a pairing
CoveredFlights	A vector with m elements. Its j^{th} ($\forall j = 1,\ldots,m$) element gets value 1 if flight j is covered by the pairings of *PairingList*
FlightList	The flights which are not included in the *CurrentPairing*
CandidateList	The list of the flights if added to the current pairing, which still remains feasible

The steps involved in the algorithm are shown in Figure 7. As it can be seen, in the first and second steps of the algorithm, the variables are initialised. The first step is repeated once in the algorithm, while the second step is repeated when a new pairing is generated. In the third step, a flight is randomly selected of *CandidateList* and is added to *CurrentPairing*. In the next step, *FlightTime* is updated based on the time of the flight added in the previous step. In step 5, if a feasible pairing is obtained, it is added to *PairingList* and then *CoveredFlights* is updated in the next step. Otherwise, step 7 runs to create a new *CandidateList* based on the rules of the airline company. If *CandidateList* has no member, according to step 8, the generation of the current pairing stops and the process of generating a new pairing starts. Otherwise, the generation of *CurrentPairing* continues.

5.1.2 Conditioned_Pairings_Alg

This algorithm produces some pairings to cover a subset of flights, F_0. This algorithm is similar to *Initial_Pairings_Alg* with only one minor difference: *CoveredFlights* only contains flights F_0 in the first step. In addition, In order to increase the probability of the coverage of flights F_0, that is the goal of this algorithm, a greater chance, seven times, is considered to choose F_0 in the *CandidateList* than other flights.

5.1.3 Partition_Alg

In this algorithm, the 'partitioned structure' of the problem is generated. As mentioned before, there are Q sub-problems in this structure that may have some variables in common with each other. This structure is obtained simply by partitioning the similar constraints of the problem into the same sub-problems. For this part, a clustering method inspired by Omran et al. (2006) is used. Clustering methods group a set of objects such that objects in the same group are similar to each other but dissimilar to those in the other groups.

In the method proposed for this part, a PSO-based clustering method is used. In this method, as shown in Figure 8, some constraints are first chosen as the initial centres of

the clusters. During the PSO iterations, the best cluster centres are selected. In each iteration, by means of the Euclidean distance, the distance of each constraint from the centres is calculated to assign it to the cluster with the smallest distance. In the last iteration, the best cluster centres and the distribution of the constraints are determined. With this clustering method, similar constraints that contain almost the same group of variables are grouped to form the sub-problems.

Figure 7 The steps involved Initial_Pairing_Alg to produce the initial feasible pairings

Step 1: Put $PairingList = \emptyset, CoveredFlights = [0 \quad \dots \quad 0]_{1\times m}$.

Step 2: Put $CurrentPairing = \emptyset,\ FlightTime = 0, FlighList = all\ flights,$
$CandidateList = flights\ whose\ start\ city\ is\ the\ home\ base$

Step 3: Select a flight randomly of $CandidateList$ and call it $CurrentFlight$. Add it to $CurrentPairing$ and remove it from $FlightList$.

Step 4: Update $FlightTime$:
If the days of the two last flights of $CurrentPairing$ are the same, then:
$FlightTime = FlightTime + flight\ time(CurrentFlight)$
Otherwise, put $FlightTime = 0$.

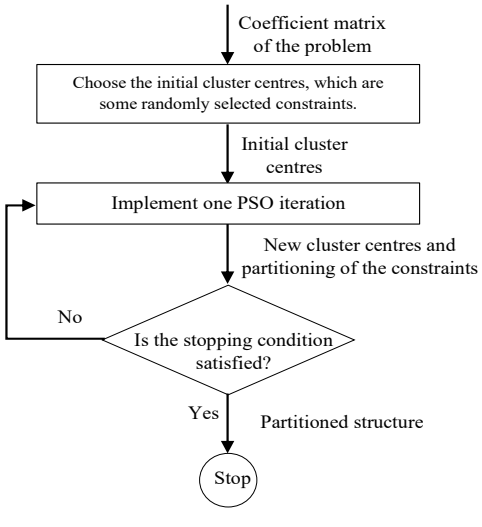
Step 5: If $CurrentFlight$ ends at home base, add $CurrentPairing$ to $PairingList$ and move on to the next step. Otherwise, move on to step 7.

Step 6: Update $CoveredFlights$. If all flights are covered, stop. Otherwise; go to the next step.

Step 7: Creating $CandidateList$:
For each flight j in $FlightList$:
If (start city of flight j is the same as finish city of $CurrentFlight$)
If (the days of flight j and $CurrentFlight$ are the same) and ($connection\ time(CurrentFlight, j) \geq$ minimum connection time) and ($FlightTime + flight\ time(j) \leq$ maximum flight time, then add flight j to $CandidateList$.
Else if the day of flight j and the day of $CurrentFlight$ are consecutive and the difference of that is at most two days with the day of first flight of $CurrentPairing$, then add flight j to $CandidatList$.

Step 8: If $CandidatList = \emptyset$, go to step 2. Otherwise; go to step 3.

Figure 8 The flowchart of the Partition_Alg to form the partitioned structure of the problem



It is worth mentioning that the reason to choose this constraint partitioning algorithm is its approach to automatically determine the appropriate number of sub-problems, so it is not necessary to set it as an input parameter beforehand. More details about the PSO-clustering algorithm can be found in Omran et al. (2006).

5.1.4 Other conditions of the algorithm

1 The knapsack problem in the proposed algorithm is solved by the greedy method of Dantzig (1957), which was explained in Section 4.

2 The termination condition of CPP_V1 is considered as the infeasibility of the knapsack problem, which happens when the right-hand side of its constraint ($A - C$) gets less than zero. CPP_V2 stops when X_0 contains all variables (or $X_1 = \varnothing$). Note that, other conditions such as the special number of iterations or solution time can also be considered.

3 The coefficients of the variables in the objective function of the knapsack problem (w_j) is considered as the number of flights covered by them.

4 The final solution of the algorithm is improved using a neighbouring search method. In each step of this method, two variables of S^f with the maximum value of index w_j/n_j, where w_j and n_j are the coefficient of the objective function of x_j and the number of constraints which are just satisfied by x_j, respectively, are found. Then, it is checked whether they can be replaced by other variables which improve the value of the objective function.

The solutions of the proposed algorithm, related to both versions, are shown in Table 2, parts (a) to (d). In this table, the first and the second columns show the name and the days of each pairing, respectively. The next five columns represent the flight sequence of each pairing. The last two columns show the overall connection time and FT of the pairings in each day, respectively. In addition, the deadheading flight of each solution is shown in a darker format in the table of each part. The performance of the two versions is compared in Subsection 5.2. Let us note that Table 2, part (a), is also the solution of Agustín et al. (2016). By comparing the results of the proposed algorithm in this paper and those in the paper authored by Agustín et al. (2016) on the case study, the following points should be made:

1 The overall FT for all solutions in parts (a) to (d), which shows the cost of each solution, is 4,060. In addition, they contain five pairings and one deadheading flight.

2 In addition to the solution of Agustín et al. (2016), which is shown in part (a), the proposed algorithm produces three other alternative solutions with the same properties (overall FT of 4,060 and five pairings). Airline companies are willing to improve several indicators for the CPP, but using a multi-objective function for the problem renders it too complex. Therefore, generating various final solutions with the same objective value increases their flexibility to choose the one that better meets their desired indicators.

3 The average solution time of the two versions of the proposed algorithm is 13.28 and 2.08 seconds, respectively. For the second version, the solution time is much better

than the time reported for the algorithm of Agustín et al. (2016), which was about 20 seconds.

4 As all solutions in parts (a) to (d) have the same overall FT, that is 4,060, and contain five pairings and one deadheading flight, in order to compare these multiple solutions, some indicators are defined in Table 3. As each pairing of a solution is assigned to a flight crew, these indicators show the level of workload among the crews. They measure the maximum value of flight numbers (FNs), daily flight numbers (DFN), FT, DFT, time away from base (TAFB) and daily total time (DTT) for the pairings of each solution. There is also another indicator that measures the maximum value of the ratio of the daily connection time (DCT) between flights and the DFT which is defined as DCT/DFT for the pairings of each solution. This indicator is defined in order to ensure a balance between the total daily resting and working time of the crews.

All these indicators are defined in such a way that the lower amount of them is more desirable. According to Table 3, the solution of part (d) outperforms the other solutions.

Table 2 The solutions of the proposed algorithm, parts (a) to (d)

(a)

Pairing	*Day*	*Flight sequence*	*Connection time*	*Flight time*
A	2	MAD-SCQ	0h00m	1h10m
	3	SCQ-MAD MAD-BCN	6h45m	2h05m
	4	BCN-ORY ORY-BCN BCN-MAD	2h15m	4h15m
B	3	MAD-BCN BCN-SCQ SCQ-BCN	3h30m	4h05m
	4	BCN-FCO FCO-BCN BCN-MXP MXP-BCN	4h00m	6h35m
	5	BCN-MUC MUC-MAD	0h45m	4h35m
C	3	MAD-BCN BCN-FCO FCO-BCN	2h00m	4h20m
	4	BCN-MAD MAD-NCE NCE-MAD MAD-LPA	3h15m	7h15m
	5	LPA-MAD MAD-FRA FRA-MAD	2h05m	7h30m
D	3	MAD-BCN BCN-BRU	1h00m	3h05m
	4	BRU-MAD MAD-SCQ SCQ-MAD MAD-AMS	7h20m	6h55m
	5	AMS-MAD	0h00m	2h25m
E	1	MAD-BCN BCN-FCO FCO-BCN	2h00m	4h20m
	2	BCN-ORY ORY-BCN	0h50m	3h15m
	3	BCN-PMI PMI-BCN BCN-MAD MAD-NCE NCE-MAD	3h25m	5h50m

(b)

Pairing	*Day*	*Flight sequence*	*Connection time*	*Flight time*
A	3	MAD-BCN BCN-BRU	1h00m	3h05m
	4	BRU-MAD MAD-SCQ SCQ-MAD MAD-AMS	7h20m	6h55m
	5	AMS-MAD	0h00m	2h25m

Note: Part (a) is the solution of the algorithm developed by Agustín et al. (2016) as well.

Table 2 The solutions of the proposed algorithm, parts (a) to (d) (continued)

(b)

Pairing	*Day*	*Flight sequence*	*Connection time*	*Flight time*
B	3	MAD-BCN BCN-FCO FCO-BCN BCN-SCQ SCQ-BCN	4h40m	7h25m
	4	BCN-MAD MAD-NCE NCE-MAD MAD-LPA	3h15m	7h15m
	5	LPA-MAD MAD-FRA FRA-MAD	2h05m	7h30m
C	1	MAD-BCN BCN-FCO FCO-BCN	2h00m	4h20m
	2	BCN-ORY ORY-BCN	0h50m	3h15m
	3	BCN-PMI PMI-BCN BCN-MAD MAD-NCE NCE-MAD	3h25m	5h50m
D	3	MAD-BCN	0h00m	1h00m
	4	BCN-FCO FCO-BCN BCN-MXP MXP-BCN	4h00m	6h35m
	5	BCN-MUC MUC-MAD	0h45m	4h35m
E	2	MAD-SCQ	0h00m	1h10m
	3	SCQ-MAD MAD-BCN	6h45m	2h05m
	4	BCN-ORY ORY-BCN BCN-MAD	2h15m	4h15m

(c)

Pairing	*Day*	*Flight sequence*	*Connection time*	*Flight time*
A	3	MAD-BCN BCN-BRU	1h00m	3h05m
	4	BRU-MAD MAD-SCQ SCQ-MAD MAD-AMS	7h20m	6h55m
	5	AMS-MAD	0h00m	2h25m
B	2	MAD-SCQ	0h00m	1h10m
	3	SCQ-MAD MAD-BCN BCN-SCQ SCQ-BCN	5h45m	5h10m
	4	BCN-ORY ORY-BCN BCN-MAD	2h15m	4h15m
C	3	MAD-BCN BCN-FCO FCO-BCN	2h00m	4h20m
	4	BCN-MAD MAD-NCE NCE-MAD MAD-LPA	3h15m	7h15m
	5	LPA-MAD MAD-FRA FRA-MAD	2h05m	7h30m
D	3	MAD-NCE NCE-MAD MAD-BCN	1h45m	4h25m
	4	BCN-FCO FCO-BCN BCN-MXP MXP-BCN	4h00m	6h35m
	5	BCN-MUC MUC-MAD	0h45m	4h35m
E	1	MAD-BCN BCN-FCO FCO-BCN	2h00m	4h20m
	2	BCN-ORY ORY-BCN	3h15m	7h15m
	3	BCN-PMI PMI-BCN BCN-MAD	1h15m	4h05m

(d)

Pairing	*Day*	*Flight sequence*	*Connection time*	*Flight time*
A	2	MAD-SCQ	0h00m	1h10m
	3	SCQ-MAD MAD-NCE NCE-MAD MAD-BCN	3h20m	5h30m
	4	BCN-ORY ORY-BCN BCN-MAD	2h15m	4h15m

Note: Part (a) is the solution of the algorithm developed by Agustín et al. (2016) as well.

Table 2 The solutions of the proposed algorithm, parts (a) to (d) (continued)

(d)				
Pairing	*Day*	*Flight sequence*	*Connection time*	*Flight time*
B	3	MAD-BCN BCN-SCQ SCQ-BCN	3h30m	4h05m
	4	BCN-MAD MAD-NCE NCE-MAD MAD-LPA	3h15m	7h15m
	5	LPA-MAD MAD-FRA FRA-MAD	2h05m	7h30m
C	1	MAD-BCN BCN-FCO FCO-BCN	2h00m	4h20m
	2	BCN-ORY ORY-BCN	3h15m	7h15m
	3	BCN-PMI PMI-BCN BCN-MAD	1h15m	4h05m
D	3	MAD-BCN BCN-BRU	1h00m	3h05m
	4	BRU-MAD MAD-SCQ SCQ-MAD MAD-AMS	7h20m	6h55m
	5	AMS-MAD	0h00m	2h25m
E	3	MAD-BCN BCN-FCO FCO-BCN	2h00m	4h20m
	4	BCN-FCO FCO-BCN BCN-MXP MXP-BCN	4h00m	6h35m
	5	BCN-MUC MUC-MAD	0h45m	4h35m

Note: Part (a) is the solution of the algorithm developed by Agustín et al. (2016) as well.

Table 3 Values of the indicators to compare the solutions of the proposed algorithm in Table 2

	(a)	*(b)*	*(c)*	*(d)*
Max FN of pairings	10	12	10	10
Max DFN of pairings	5	5	4	4
Max FT of pairings	19h5m	22h10m	19h5m	19h5m
Max DFT of pairings	7h30m	7h30m	7h30m	7h30m
Max TAFB of pairings	26h25m	32h10m	26h25m	25h35m
Max DTT of pairings	14h15m	14h15m	14h15m	14h15m
Max DCT/DFT	3.24	3.24	1.11	1.06

5.2 Results for randomly generated CPPs

In order to evaluate the performance of the proposed algorithm on CPPs with different sizes, some randomly-generated test problems, whose structure is similar to the real case of the previous section, are examined. These problems contain 22 to 320 flights.

5.2.1 The comparison of the two versions of the proposed algorithm

This section reports the computational results for the two versions of the proposed algorithm, denoted by CPP_V1 and CPP_V2, respectively. To evaluate the efficiency of the proposed algorithms, a case study of CPP, presented in Subsection 5.1, is modelled in the appendix, and its optimal solutions for the problems is obtained by the commercial solver CPLEX (version 12.6).

Table 4 The comparison of the two versions of the proposed algorithm to solve CPPs with different sizes

#Flights	CPLEX			CPP_V1				CPP_V2			
	Overall flight time	*#Pairings*	*Solution time (s)*	*Overall flight time*	*#Pairings*	*#Iterations*	*Solution time (s)*	*Overall flight time*	*#Pairings*	*#Iterations*	*Solution time (s)*
22	1,835*	7	5.03	1,835	7	1	2.20	1,835	7	1	1.25
33	2,555*	15	5.06	2,555	12	2	7.37	2,555	12	2	1.37
41	4,060*	7	7.50	4,060	5	2	13.28	4,060	5	2	2.8
60	5,380*	28	17.42	5,380	24	2	38.69	5,380	25	1	1.84
69	6,420*	32	22.49	6,420	25	4	101.54	6,420	25	4	3.22
101	8,680*	43	311.00	8,680	36	9	654.96	8,680	36	6	12.56
103	8,480*	44	30.75	8,480	36	4	282.67	8,480	36	3	26.62
139	12,085	63	TL	12,085	56	7	2,118.47	12,085	56	10	50.98
147	17,190	61	TL	17,200	53	4	347.22	17,190	52	5	41.01
168	14,215*	63	536.50	14,215	52	12	2,458.68	14,215	53	12	75.20
197	NA	-	-	21,850	74	19	TL	21,850	74	15	273.68
247	NA	-	-	19,070	99	6	TL	18,495	81	24	723.94
250	NA	-	-	20,365	107	13	TL	19,730	88	39	1,563.02
251	NA	-	-	28,000	82	38	4,162.79	28,000	83	11	318.83
316	NA	-	-	25,410	138	8	TL	24,080	107	36	2,671.34
320	NA	-	-	42,415	144	11	TL	40,105	120	46	1,737.06

Notes: *shows optimal solution. 'TL' shows the time limit of two hours. 'NA' stands for not available.

The results are presented in Table 4. In this table, the first column represents the number of flights for each problem. Note that the CPP with 41 flights is the real case studied in the previous section. Columns two through four, five through eight, and nine through 12 refer to the results obtained by CPLEX, CPP_V1, and CPP_V2, respectively. The columns 'overall FT', '#Pairings', and 'solution time' show the overall FT of the resulting pairings, the number of final pairings, and the solution running time (in seconds), respectively. As can be seen, the two versions of the proposed algorithm perform almost equally in terms of overall FT and the number of final pairings for problems up to 197 flights. The main difference between them is the solution time, which is on average almost 26 times larger for CPP_V1 than for CPP_V2. For the remaining problems, which include 197, 247, 250, 316, and 320 flights, a time limit (TL) of two hours is considered because the running time of CPP_V1 takes too long, so the best solution obtained during this time period is reported in the table. According to the table, for these problems, on average, the objective values and the number of final pairings generated by CPP_V1 are 4.40% and 23.20% larger than those of CPP_V2, respectively.

According to Table 4, we have also considered a TL of two hours for CPLEX. During this period, CPLEX has generated optimal solutions of problems containing 22, 33, 41, 60, 69, 101, 103, and 168 flights, which are marked with * in the table. As can be seen, both versions of the algorithm have also produced the optimal values.

For problems with 139 and 147 flights, CPLEX could not provide final optimal solutions in two hours. In addition, for problems with 197 or more flights, CPLEX could not solve the problem and even provide a feasible solution due to the large size of the problems. These problems are marked not available (NA).

5.2.2 The effect of the initial feasible solution

In Table 4, the advantage of CPP_V2 vis-à-vis CPP_V1 is shown. Therefore, in what follows, this version of the algorithm is used. In Table 5, the effect of the initial feasible solution on the performance of the proposed algorithm is reported for the CPP with 103 flights. The first initial solution is obtained by the *Initial_Pairings_Alg* of Figure 7. The second initial solution is generated by the depth first search (DFS) method until all flights are covered. This method is explained in detail in Deveci and Demirel (2018b).

Table 5 The comparison of the effect of the initial feasible solution on the performance of the proposed algorithm to solve the CPP with 103 flights

	Initial overall flight time	*#Initial pairings*	*Final overall flight time*	*# Final pairings*	*#Iterations*	*Solution time (s)*
Initial_Pairings_Alg	8,800	51	8,480	36	3	26.62
DFS method	10,705	38	8,480	37	12	82.72

According to Table 5, the objective function of the initial solutions generated by *Initial_Pairings_Alg* and the DFS method are 8,800 and 10,705, respectively. In addition, they contain 51 and 38 pairings, respectively. Both initial solutions lead to the final overall FT of 8,480. Due to the better initial solution generated by *Initial_Pairings_Alg* than the DFS method, the algorithm reached the final solution by three iterations in 26.62 seconds. However, for the initial solution generated by the DFS method, it reached the

final solution by 12 iterations in 82.72 seconds. As a result, the better the initial solution, the fewer the number of iterations and the less the solution time.

5.2.3 The effect of sorting variables on the performance of CPP_V2

As stated, in CPP_V2, the variables (pairings) of the problem are sorted based on the ratio of their coefficients in the objective function to their coefficients in the knapsack constraint. As a result, the values of the coefficients in the objective function affect the sorting result. So far, these values have been the number of flights covered by the pairings.

In this part, in addition to this rule (Sort_1), another rule (Sort_2) is introduced. In Sort_2, the coefficient of the objective function of a pairing is calculated as the sum of the values of the flights covered by it. The value of each flight is directly proportional to the difficulty of its coverage by the available pairings. This means that the fewer pairings a flight is present in, the more its value. Therefore, the value of a flight is calculated as the inverse of the number of pairings involving that flight.

In addition to these two rules, another pair of rules are introduced, which are called Sort_3 and Sort_4. In Sort_3, at each iteration, X_0 is calculated through the sorting of Sort_1. If $L_i \leq$ *sum of the costs* (*new_P*) $< L_{i-1}$ in Figure 6, is not satisfied, X_0 is calculated through the sorting of Sort_2. In Sort_4, in odd and even iterations, X_0 is calculated through the sorting of Sort_1 and Sort_2, respectively. The effect of these four rules on the performance of the proposed algorithm is evaluated on the CPP with 251 flights in Table 6.

In this table, the overall FT and the number of the resultant pairings, the number of iterations and solution time of the algorithm are reported. The algorithm is run 5 times for each rule and the best result is reported in this table. As it can be seen, different types of sorting lead to a special solution. However, none of them is superior to others for any columns of the table.

Table 6 The comparison of the effect of sorting of the variables on the performance of the proposed algorithm to solve the CPP with 251 flights

	Overall flight time	*#Pairings*	*#Iterations*	*Solution time (s)*
Sort_1	28,000	83	17	442.66
Sort_2	28,045	83	18	382.63
Sort_3	28,065	84	12	431.21
Sort_4	28,000	82	17	678.68

5.2.4 The effectiveness of knapsack problem solutions on the performance of CPP_V1

The knapsack problem is an NP-hard problem and, therefore, as the size of the problem increases, so does its solution time. For this reason, a heuristic method was used in this work to solve it in CPP_V1. Table 7 shows the difference between optimal solutions of knapsack problem and the solutions generated by the greedy method of Dantzig (1957) on the performance of CPP_V1 when tested on problems with 41 and 103 flights.

Table 7 The effect of the exact and inexact knapsack problem solutions on the performance of CPP_V1

#Flights	*CPP_V1 (heuristic solution)*					*CPP_V1 (exact solution)*				
	Overall flight time	*#Pairings*	*#Iterations*	*Solution time (s)*		*Overall flight time*	*#Pairings*	*#Iterations*	*Solution time (s)*	
				Iteration average	*Total*				*Iteration average*	*Total*
41	4,060	5	2	6.60	13.2	4,060	5	3	8.81	26.43
103	8,480	36	4	70.67	282.66	8,480	37	4	168.53	674.12

Table 7 shows that when using exact and heuristic methods to solve the knapsack problem, the overall FT and the number of final pairings are almost the same, but the average solution time of each iteration is almost 1.33 and 2.38 times larger for problems with 41 and 103 flights, respectively, when using the exact method compared to the heuristic one.

As can be seen, as the size of the problem increases with the exact method, due to the NP-hardness of the knapsack problem, the solution time also increases, but the solution quality is almost the same as the heuristic method. The result is that exact solutions of the knapsack problem do not necessarily guarantee the lowest overall FT due to the existence of random steps in each iteration of the algorithm. Therefore, the greedy method of Dantzig (1957) is used to simplify the implementation of CPP_V1 in Section 4 so that CPP_V2 takes less time while maintaining the quality of the generated solutions.

6 Conclusions

In this paper, a new decomposition algorithm based on constraint partitioning is developed for solving CPPs. The new proposed method is based on the removal/addition of some pairings from/to the feasible solution of the 'partitioned structure' of the problem. The removal and addition sub-problems are a knapsack model for determining the removing pairings and a problem for generating new pairings to improve the current solution of a CPP, respectively. The second version of the proposed method is obtained by simplifying the implementation of the first version, when the knapsack problem is solved by a greedy method.

In this paper, we introduce an original technique not explored previously in the literature. The proposed method removes some of the current pairings, in addition to adding new pairings. Therefore, this procedure does not increase the problem size.

The proposed algorithm is applied to a case study whose data was available in the literature of CPPs. Among others, the wider variety of the final solutions, potentially enabling airline companies to choose the right one more easily and flexibly, and the lower solution time are the main benefits of the proposed algorithm vis-à-vis the previous method for the case. In addition, the new solution outperforms its previous one on some important indicators of airline companies. In addition, the performance of the algorithm is evaluated on some randomly-generated test problems, involving up to 320 flights.

Apart from a 'partitioned structure', examined in this paper, the authors of the present study have worked on other structures including a 'block-angular structure' for SCPs and have proved the relation between the optimal value of an SCP with a 'block-angular structure' and its sub-problems through lower and upper bounds. It is our sincere hope that the method proposed in this paper for 'partitioned structures' can be extended to 'block-angular structures'.

The comparison or combination of the proposed method with others such as column generation algorithm is also suggested as a promising area of study for future researchers. Furthermore, use of other methods for *Partition_Alg* and *Conditioned_Pairings_Alg* are other proposed future directions of this paper.

References

Aggarwal, D., Saxena, D.K., Bäck, T. and Emmerich, M. (2020a) *Airline Crew Pairing Optimization Framework for Large Networks with Multiple Crew Bases and Hub-And-Spoke Subnetworks*, arXiv preprint arXiv:2003.03994.

Aggarwal, D., Saxena, D.K., Bäck, T. and Emmerich, M. (2020b) *A Novel Column Generation Heuristic for Airline Crew Pairing Optimization with Large-Scale Complex Flight Networks*, arXiv preprint arXiv:2005.08636.

Aggarwal, D., Saxena, D.K., Bäck, T. and Emmerich, M. (2020c) *On Initializing Airline Crew Pairing Optimization for Large-Scale Complex Flight Networks*, arXiv preprint arXiv:2003. 06423.

Aggarwal, D., Saxena, D.K., Back, T. and Emmerich, M. (2020d) *Real-World Airline Crew Pairing Optimization: Customized Genetic Algorithm Versus Column Generation Method*, arXiv preprint arXiv:2003.03792.

Aggarwal, D., Singh, Y.K. and Saxena, D.K. (2020e) *On Learning Combinatorial Patterns to Assist Large-Scale Airline Crew Pairing Optimization*, arXiv preprint arXiv:2004.13714.

Aggarwal, D., Saxena, D.K., Emmerich, M. and Paulose, S. (2018) 'On large-scale airline crew pairing generation', in *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, November, pp.593–600 [online] https://doi.org/10.1109/SSCI.2018.8628699.

Agustín, A., Gruler, A., de Armas, J. and Juan, A.A. (2016) 'Optimizing airline crew scheduling using biased randomization: a case study', *Lecture Notes in Computer Science*, pp.331–340 [online] https://doi.org/10.1007/978-3-319-44636-3_31.

AhmadBeygi, S., Cohn, A. and Weir, M. (2009) 'An integer programming approach to generating airline crew pairings', *Computers & Operations Research*, Vol. 36, No. 4, pp.1284–1298 [online] https://doi.org/10.1016/j.cor.2008.02.001.

Aksoy, N. (2010) *Pricing by Local Search in Column Generation for the Airline Crew Pairing Problem*, MSc dissertation.

Aydemir-Karadag, A., Dengiz, B. and Bolat, A. (2013) 'Crew pairing optimization based on hybrid approaches', *Computers & Industrial Engineering*, Vol. 65, No. 1, pp.87–96 [online] https://doi.org/10.1016/j.cie.2011.12.005.

Dantzig, G.B. (1957) 'Discrete-variable extremum problems', *Operations Research*, Vol. 5, No. 2, pp.266–288 [online] https://doi.org/10.1287/opre.5.2.266.

Demirel, N.Ç. and Deveci, M. (2017) 'Novel search space updating heuristics-based genetic algorithm for optimizing medium-scale airline crew pairing problems', *International Journal of Computational Intelligence Systems*, Vol. 10, No. 1, pp.1082–1101 [online] https://doi.org/ 10.2991/ijcis.2017.10.1.72.

Desaulniers, G., Lessard, F., Saddoune, M. and Soumis, F. (2020) 'Dynamic constraint aggregation for solving very large-scale airline crew pairing problems', *SN Operations Research Forum*, September, Vol. 1, No. 3, pp.1–23, Springer International Publishing [online] https://doi.org/ 10.1007/s43069-020-00016-1.

Deveci, M. and Demirel, N.Ç. (2018a) 'A survey of the literature on airline crew scheduling', *Engineering Applications of Artificial Intelligence*, Vol. 74, pp.54–69 [online] https://doi.org/ 10.1016/j.engappai.2018.05.008.

Deveci, M. and Demirel, N.Ç. (2018b) 'Evolutionary algorithms for solving the airline crew pairing problem', *Computers & Industrial Engineering*, Vol. 115, pp.389–406 [online] https://doi.org/ 10.1016/j.cie.2017.11.022.

Dück, V., Wesselmann, F. and Suhl, L. (2011) 'Implementing a branch and price and cut method for the airline crew pairing optimization problem', *Public Transport*, Vol. 3, No. 1, p.43 [online] https://doi.org/10.1007/s12469-011-0038-9.

Elfeky, E.Z. (2009) *Evolutionary Algorithms for Constrained Optimization*, Doctoral dissertation, University of New South Wales, Canberra, Australia.

Erdoğan, G., Haouari, M., Matoglu, M.Ö. and Özener, O.Ö. (2015) 'Solving a large-scale crew pairing problem', *Journal of the Operational Research Society*, Vol. 66, No. 10, pp.1742–1754 [online] https://doi.org/10.1057/jors.2015.2.

Haouari, M., Zeghal Mansour, F. and Sherali, H.D. (2019) 'A new compact formulation for the daily crew pairing problem', *Transportation Science*, Vol. 53, No. 3, pp.811–828 [online] https://doi.org/10.1287/trsc.2018.0860.

Kasirzadeh, A., Saddoune, M. and Soumis, F. (2017) 'Airline crew scheduling: models, algorithms, and data sets', *EURO Journal on Transportation and Logistics*, Vol. 6, No. 2, pp.111–137 [online] https://doi.org/10.1007/s13676-015-0080-x.

Kato, K. and Sakawa, M. (2003) 'Genetic algorithms with decomposition procedures for multidimensional 0–1 knapsack problems with block angular structures', *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, Vol. 33, No. 3, pp.410–419, DOI: 10.1109/TSMCB.2003.811126.

Klabjan, D., Johnson, E.L., Nemhauser, G.L., Gelman, E. and Ramaswamy, S. (2001) 'Solving large airline crew scheduling problems: random pairing generation and strong branching', *Computational Optimization and Applications*, Vol. 20, No. 1, pp.73–91 [online] https://doi.org/10.1023/A:1011223523191.

Kornilakis, H. and Stamatopoulos, P. (2002) 'Crew pairing optimization with genetic algorithms', in *Hellenic Conference on Artificial Intelligence*, Springer, Berlin, Heidelberg, pp.109–120 [online] https://doi.org/10.1007/3-540-46014-4_11.

Masipa, L. (2019) *A Heuristic Approach to the Deterministic and Stochastic Air Crew Pairing Problem*, MSc dissertation, Stellenbosch, Stellenbosch University.

Novianingsih, K. and Hadianti, R. (2018) 'A heuristic method for solving airline crew pairing problems', in *MATEC Web of Conferences*, EDP Sciences, Vol. 204 [online] https://doi.org/10.1051/matecconf/201820402006.

Omran, M.G., Salman, A. and Engelbrecht, A.P. (2006) 'Dynamic clustering using particle swarm optimization with application in image segmentation', *Pattern Analysis and Applications*, Vol. 8, No. 4, pp.332 [online] https://doi.org/10.1007/s10044-005-0015-5.

Quesnel, F., Desaulniers, G. and Soumis, F. (2020) 'A branch-and-price heuristic for the crew pairing problem with language constraints', *European Journal of Operational Research*, Vol. 283, No. 3, pp.1040–1054 [online] https://doi.org/10.1016/j.ejor.2019.11.043.

Rasmussen, M.S., Lusby, R.M., Ryan, D.M. and Larsen, J. (2011) *Subsequence Generation for the Airline Crew Pairing Problem*, Technical Report, Technical University of Denmark, DTU Management Engineering.

Reisi, N.M. and Moslehi, G. (2013) 'Cockpit crew pairing problem in airline scheduling: shortest path with resources constraints approach', *International Journal of Industrial Engineering and Production Research*, Vol. 24, No. 4, pp.259–268.

Saddoune, M., Desaulniers, G. and Soumis, F. (2013) 'Aircrew pairings with possible repetitions of the same flight number', *Computers & Operations Research*, Vol. 40, No. 3, pp.805–814 [online] https://doi.org/10.1016/j.cor.2010.11.003.

Wu, W., Hu, Y., Hashimoto, H., Ando, T., Shiraki, T. and Yagiura, M. (2016) 'A column generation approach to the airline crew pairing problem to minimize the total person-days', *Journal of Advanced Mechanical Design, Systems, and Manufacturing*, Vol. 10, No. 3 [online] https://doi.org/10.1299/jamdsm.2016jamdsm0040.

Yaakoubi, Y., Soumis, F. and Lacoste-Julien, S. (2020) 'Machine learning in airline crew pairing to construct initial clusters for dynamic constraint aggregation', *EURO Journal on Transportation and Logistics*, Vol. 9, No. 4, p.100020 [online] https://doi.org/10.1016/j.ejtl.2020.100020.

Zeren, B. and Özkol, İ. (2012) 'An improved genetic algorithm for crew pairing optimization', *Journal of Intelligent Learning Systems and Applications*, Vol. 4, No. 1, p.70, DOI: 10.4236/jilsa.2012.41007.

Zeren, B. and Özkol, I. (2016) 'A novel column generation strategy for large scale airline crew pairing problems', *Expert Systems with Applications*, Vol. 55, pp.133–144 [online] https://doi.org/10.1016/j.eswa.2016.01.045.

Appendix

The proposed mathematical model for CPP of the case study

In this appendix, a mathematical model for the case study of CPP presented in Subsection 5.1 is proposed. The problem and its assumptions are fully explained before. The notations of the model are shown in Table A1.

Table A1 The notations for the proposed mathematical model of the CPP of the case study

Notation	*Description*
Model sets	
F	The set of flights
P	The set of all feasible pairings
W	The set of start days of the flights (in the case study, $W = \{1, 2, 3, 4, 5\}$)
F^O	The set of flights start at home base (or their origin city is home base)
F^D	The set of flights end at home base (or their destination city is home base)
F^w	The set of flights starting at day $w \in W$
A^S	The set of flight pairs (f, l) which can be sequential according to the regulations of the airline company
Model indices	
f, f', l	Index of a flight, $f, f', l \in F$
p	Index of a pairing, $p \in P$
w	Index of a start day of a flight, $w \in W$
Model parameters	
O_f	The origin city of flight f
D_f	The destination city of flight f
Day_f	The start day of flight f
t_f	The duration of flight f
S_f	The start time of flight f (which is considered among 0 to 1,440 for a 24-hour day)
E_f	The end time of flight f (which is considered among 0 to 1,440 for a 24-hour day)
Model variables	
x_{fp}	1 if flight f is in pairing p 0 otherwise
y_{flp}	1 if flight pair (f, l) is in pairing p 0 otherwise
u_{fp}	1 if flight f is the start flight of pairing p 0 otherwise
v_{fp}	1 if flight f is the end flight of pairing p 0 otherwise
z_p	1 pairing p is chosen 0 otherwise

Note that according to assumptions of Subsection 5.1, A^S contains a flight pair (f, l) if:

- $D_f = O_l$, meaning that the destination of flight f is the same as the origin of flight l.
- $S_l - E_f \geq 45$ if (f, $l \in F^w \; \forall w \in W$), meaning that the start time of flight l is at least 45 minutes later than the end time of flight f if their start day is the same.
- $0 \leq Day_l - Day_f \leq 1$, meaning that flight l should start on the same day or at most one day after flight f.

According to the above definitions, the mathematical model of the problem is as follows:

$$\min Z = \sum_f \sum_p x_{fp} t_f \tag{A1}$$

s.t.

$$\sum_{f \in F^O} u_{fp} = z_p \qquad \forall p \tag{A2}$$

$$\sum_f u_{fp} \leq z_p \qquad \forall p \tag{A3}$$

$$\sum_{f:(f,l) \in A^S} y_{flp} - v_{lp} = \sum_{f':(l,f') \in A^S} y_{lf'p} - u_{lp} \qquad \forall l, p \tag{A4}$$

$$\sum_{f \in F^D} v_{fp} = z_p \qquad \forall p \tag{A5}$$

$$\sum_f V_{fp} \leq z_p \qquad \forall p \tag{A6}$$

$$\sum_f V_{flp} \leq x_{fp} \qquad \forall f, p \tag{A7}$$

$$\sum_f y_{flp} \leq x_{lp} \qquad \forall l, p \tag{A8}$$

$$x_{fp} \leq \sum_{l:(f,l) \in A^S} y_{flp} + v_{fp} \qquad \forall f, p \tag{A9}$$

$$x_{lp} \leq \sum_{f:(f,l) \in A^S} y_{flp} + u_{fp} \qquad \forall l, p \tag{A10}$$

$$\sum_p x_{fp} \geq 1 \qquad \forall f \tag{A11}$$

$$\sum_{f \in F^w} t_f x_{fp} \leq 480 \qquad \forall p, w \tag{A12}$$

$$\sum_f v_{fp} Day_f - \sum_f u_{fp} Day_f \leq 2 \qquad \forall p \tag{A13}$$

$$x_{fp}, y_{flp}, u_{fp}, v_{fp}, z_p \in \{0,1\} \qquad \forall f, l \in F, p \in P \qquad \text{(A14)}$$

In the above model, objective function (A1) minimises the cost of the selected pairings, calculated as their overall FT. Constraints (A2) and (A3) state that each pairing, if chosen, should contain one flight departing from the home base as a takeoff flight and that at most one flight can start the pairing, respectively. Constraint (A4) is the balancing constraint, which states that each flight of a pairing should have one preceding flight and one following flight unless it is the first or last flight of a pairing. Constraints (A5) and (A6) are similar to equations (A2) and (A3) and force the model to choose a final flight from the set of flights that terminate at home base, and at most one flight can play the role of the final flight for each pairing. Constraints (A7) through (A10) define the relationship between variables *x* and *y*. According to equations (A7) and (A8), if a flight is not in a pairing, none of the variables *y* containing that flight should receive a value. According to equation (A9), if a flight is in a pairing, it should either have a subsequent flight (the first summation) or be the last flight of that pairing. Similarly, according to equation (A10), if a flight occurs in a pairing, it should have a preceding flight (the first summation) or be the first flight of that pairing. Constraint (A11) states that each flight should be in at least one pairing. Note that this constraint is not in the form of equality since deadheading is allowed.

Constraints (A12) and (13) are directly related to airline regulations regarding limitations on overall DFT and the number of days in a pairing, respectively. According to equation (A12), the overall FT of flights belonging to the same pairing and the same day must not exceed eight hours (or 480 minutes). Constraint (A13) states that the day of the last flight of a pairing should be two days larger than the day of its initial flight so that the length of a pairing is at most three days. Binary requirements on the variables are expressed through equation (A14).