A column generation approach to the airline crew pairing problem to minimize the total person-days

Wei WU*, Yannan HU*, Hideki HASHIMOTO**, Tomohito ANDO***, Takashi SHIRAKI**** and Mutsunori YAGIURA**
* Graduate School of Information Science, Nagoya University
Furocho, Chikusa-Ku, Nagoya 464-8601, Japan
E-mail: outman@live.com
** Department of Logistics and Information Engineering, Tokyo University of Marine Science and Technology
Etcujima, Koto-Ku, Tokyo 135-8533, Japan
*** Transportation and City Infrastructure Division, NEC Corporation
5-7-1, Shib, Minato-Ku, Tokyo 108-8001, Japan
**** Data Science Research Laboratories, Central Research Laboratories, NEC Corporation
1753, Shimonumabe, Nakahara-Ku, Kawasaki, Kanagawa 211-8666, Japan

Received 1 April 2016

Abstract
In this paper, we consider the crew pairing problem in airline scheduling that calls for assigning crew members in order to cover all flights with the minimum total person-days under the constraints that the schedule of each crew member does not violate given constraints on the total working time, flying time, and the number of landings. In practical applications, it is difficult to create an efficient schedule satisfying all the constraints. We formulate the problem as a set covering problem and apply an LP-based column generation approach to generate a candidate set of schedules. We propose a branch-and-bound method based upon a resource constrained dynamic programming for the column generation procedure. Computational results are given for a number of large-scale instances with up to 10,000 flights.

Key words: Crew scheduling problem, Column generation, Set covering problem, Dynamic programming

1. Introduction

The crew scheduling problem frequently appears in real-world applications, such as those in bus and rail transit industry. In this paper, we consider a crew pairing problem in airline scheduling with a series of constraints and conditions particular to this industry. The crew costs constitute a high proportion (up to 20%) of total airline operation costs, and the number of airline flights increases with globalization. For this reason, a small percentage saving amounts to substantial reduction in expenses as mentioned by Anbil et al. (1992) and Barutt and Hull (1990).

Onodera and Mori (1991) raised an example that a Japanese airline company developed a knowledge-base system for crew scheduling in 1990, which cost about $4 million to build. However, it got paid for itself in direct cost savings only in about 18 months.

Several approaches for the airline crew scheduling problem have been proposed in the literature, including exact algorithms such as tree search by Beasley and Cao (1996), branch-and-cut by Hoffman and Padberg (1993), as well as heuristic methods such as simulated annealing by Emden-Weinert and Proksch (1999) and genetic algorithms by Levine (1996).

In this paper, we model the problem as a set covering problem with costs of columns defined by the number of person-days, and then we present an efficient method to find promising columns through a graph representation that
describes connections between flights, where the size of the graphs is kept small by using the cost structure. To solve the problem of finding a promising column, we propose a branch-and-bound method based upon a resource constrained dynamic programming, which enables the algorithm to solve large-scale instances. Moreover, regularity is exploited to quickly generate many columns just by repeating itineraries. Such repeated itineraries are preferable so that crew members will have a low risk of making mistakes in duty time.

The rest of this paper is organized as follows: In Section 2, we describe the crew scheduling problem in general, and we further present some requirements and constraints that are particular to some specific airline companies. In Section 3, we discuss a set covering model for this problem. In Section 4, we propose a column generation approach, which utilizes the branch-and-bound framework and dynamic programming. Section 4 also proposes some ideas for solving this set covering problem (SCP) efficiently. In Section 6, we present our computational results, and finally, we give concluding remarks in Section 7.

2. Problem Description

First we give a few definitions for later description. A flight leg (sometimes also called segment) is a single nonstop flight. A pairing is a sequence of flight legs that begins and ends at a crew base city, where the arrival airport of every flight leg in the sequence coincides with the departure airport of the next flight leg. A deadhead (DH) is a special flight leg such that the crew member assigned to it flies as a passenger to transport to the departure airport of another flight leg or to return to the departure city (crew base) at the end of a pairing.

Gopalakrishnan and Johnson (2005) described that airline scheduling usually consists of five planning stages. The last two stages among them, crew pairing and rostering, are usually referred to as crew scheduling problems.

As the input data of crew scheduling problem, a schedule consisting of all flight legs is provided before the crew pairing stage. A number of constraints also have to be satisfied for each pairing according to requirements from industrial applications. Each pairing has a cost associated with it. In our problem, we define the value of person-days as the cost of a pairing, where in our formulation, this value is defined to be $q$ for a pairing if it consists of flight legs from $q$ consecutive working days. One unit of person-day represents the amount of work done by one person in one working day. The objective of the crew pairing stage is to find a subset of all feasible pairings with the minimum total cost such that the subset contains every non-DH flight leg at least once (sometimes exactly once depending on the model definition).

In the rostering stage, a monthly (or weekly) schedule that can be operated by the crew is created by using the set of pairings generated at the crew pairing stage. Such a monthly (or weekly) schedule for the crew is called a roster. Although the exact number of crew members for the month (or week) becomes clear after crew rostering, this number is roughly determined after the pairing stage, and hence it is important to find a good solution in the pairing stage.

In this paper, we concentrate on the approach for the crew pairing stage.

2.1. Constraints for Pairing

Each airline company may have several basic and specific constraints for defining feasible pairings. The basic constraints are listed as follows:

**Basic Constraint 1:** The first departure city in a pairing has to be the same as the last arrival city. In our problem, we define such a city as Tokyo. It signifies that only the flight legs departing from NRT or HND airport can be the first flight leg in a pairing. Similarly, only the flight legs arriving at NRT or HND airport are allowed to be the last flight leg.

**Basic Constraint 2:** A specified time is required for a crew member to transfer from a flight leg to the next one. In our formulation, for any flight leg, its departure time has to be at least 30 minutes later than the arrival time of its previous flight.

**Basic Constraint 3:** The duration of a pairing must not exceed a specified limit on the value of person-days, which is usually 4–6 days. In our model, we assume that each crew member is unable to work more than $N_{\text{pass}}$ (a given input) days, which means that the maximum value of person-days in a pairing is $N_{\text{pass}}$. Recall that in our formulation, the cost of a pairing is defined to be the value of person-days, and hence each pairing has a cost of at most $N_{\text{pass}}$. We set $N_{\text{pass}} = 5$ unless otherwise stated.

As in many papers in the literature, we also have constraints regarding the aircraft types; we restrict our attention to a single aircraft type in our scheduling.

Before explaining specific constraints, we give several definitions. An interval time $t$ is defined as the time between the arrival and departure of two consecutive flight legs in a pairing, and $t$ has to be at least 30 minutes as discussed above.
A break period is defined as a short interval time satisfying $30 \leq t < 870$. If an interval time $t$ is above or equals to 870 minutes, it is regarded as a sleep period. A duty period consists of a sequence of flight legs without sleep periods between them, i.e., sleep periods divide a pairing into duty periods. The flying time of a duty period is the sum of actual flying times of the flights in the duty period except for the flying times in deadhead flights. The maximum flying time $f_{\text{path}}^k$ of a pairing $k$ is the maximum flying time among all the flying times of the duty periods in pairing $k$. The working time of a duty period is the total working minutes in the duty period. A break period also contributes to the working time by the duration of the break period when it is less than 330 minutes; otherwise, it is counted as a constant working time of 90 minutes. The time for a crew member to board a deadhead flight is also counted as part of working time. The maximum working time $w_{\text{path}}^k$ of a pairing $k$ is the maximum working time among all the working times of the duty periods in pairing $k$. A landing number of a duty period is the total number of landings in the duty period. The maximum landing number $l_{\text{path}}^k$ of a pairing $k$ is the maximum landing number among all the landing numbers of the duty periods in a pairing $k$.

For convenience, we define $P_{\text{all}}$ to be the set of all feasible pairings.

We define three additional constraints as follows:

**Specific Constraint 1:** $f_{\text{path}}^k \leq N_{f_{\text{max}}}$ for all $k \in P_{\text{all}}$;

**Specific Constraint 2:** $w_{\text{path}}^k \leq N_{w_{\text{max}}}$ for all $k \in P_{\text{all}}$;

**Specific Constraint 3:** $l_{\text{path}}^k \leq N_{l_{\text{max}}}$ for all $k \in P_{\text{all}}$,

where $N_{f_{\text{max}}}$, $N_{w_{\text{max}}}$ and $N_{l_{\text{max}}}$ are the given upper bounds on the maximum flying time, the maximum working time and the maximum number of landings, respectively, which are set to $N_{f_{\text{max}}} = 720$, $N_{w_{\text{max}}} = 810$ and $N_{l_{\text{max}}} = 5$ in this paper.

A solution to the crew scheduling problem is considered feasible only when all the pairings selected by the solution satisfy all the above mentioned constraints, and all the non-deadhead flight legs are covered by the selected pairings. Note that even though the model in this section was formulated based on real-world data from a company and is quite complicated, some of the constraints are simplified from real ones and the parameter values are not necessarily the same as those in the real-world applications.

### 2.2. Instances and Experimental Environment

All the computational experiments in this paper are conducted on four instances, named I, II, III and IV, that were generated based on real-world flight data. For I, II and IV, we aim to solve a weekly airline crew scheduling problem. For convenience, we assume that the time horizon of every instance from I, II and IV consists of 17 days with all the flight legs in the first and the last $N_{\text{dummy}}$ days specified as deadhead. These deadheads at the beginning and the end help to cover the target flights in the middle core days. Instance III is a monthly flight data that is also sandwiched by $N_{\text{dummy}}$ days with only deadheads before and after the core period. Since every pairing in any optimal solution includes at least one target flight, it is sufficient to set $N_{\text{dummy}}$ to $N_{\text{pmax}}-1$, where $N_{\text{pmax}}$ is the maximum value of person-days in a pairing. In this paper, we set $N_{\text{dummy}}$ to 5, because the largest value of $N_{\text{pmax}}$ we tested in Section 6.2 is 6. The information of the instances is shown in Table 1. The number of flights including deadhead flights for each instance is shown in column "#Flight.”

<table>
<thead>
<tr>
<th>Instance</th>
<th>#Day</th>
<th>#DummyDay</th>
<th>#Flight</th>
<th>#DH</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>17</td>
<td>10</td>
<td>918</td>
<td>540</td>
</tr>
<tr>
<td>II</td>
<td>17</td>
<td>10</td>
<td>2201</td>
<td>1295</td>
</tr>
<tr>
<td>III</td>
<td>41</td>
<td>10</td>
<td>2214</td>
<td>540</td>
</tr>
<tr>
<td>IV</td>
<td>17</td>
<td>10</td>
<td>11514</td>
<td>6777</td>
</tr>
</tbody>
</table>

All experiments in this paper are carried out on a PC with 3.30 GHz CPU and 32 GB RAM memory, where the computation is executed on a single core. The algorithms proposed in this paper are all coded in C++ unless otherwise stated. All of them were built and compiled under Microsoft Visual Studio 2012.

### 3. Set Covering Model

Our algorithm solves the crew pairing problem in two steps: the first stage generates feasible pairings, and the second stage selects a good subset of these pairings to cover all the flight legs. In most cases, it is impractical to generate all the feasible pairings in the first stage, since the number of such pairings grows exponentially with the number of flight legs. To deal with this, we propose a column generation approach that is discussed in Section 4. The second stage can be modelled as a set covering or set partitioning problem.
our objective is to find a set of pairings with minimum cost such that each flight leg is covered by at least one pairing. Let \( a_{ik} \) be a binary value that equals 1 when pairing \( k \) covers flight leg \( i \), otherwise, \( a_{ik} = 0 \). The cost \( c_k \) is the value of person-days of pairing \( k \). This problem can be formulated as a set covering problem SCP(\( P \)) as follows:

\[
\text{minimize} \quad \sum_{k \in P} c_k x_k \\
\text{subject to} \quad \sum_{k \in P} a_{ik} x_k \geq 1, \quad \forall i \in F_{\text{all}} \\
\quad \quad \quad \quad \quad \quad \quad x_k \in \{0, 1\}, \quad \forall k \in P.
\]

where \( P \) is a subset of \( P_{\text{all}} \) and \( F_{\text{all}} \) is the set of all non-deadhead flight legs. The binary decision variable \( x_k \) is a 0-1 variable associated with the \( k \)th pairing. If the pairing \( k \) is selected, then \( x_k = 1 \), and otherwise \( x_k = 0 \). When \( P = P_{\text{all}} \) holds, the problem SCP(\( P_{\text{all}} \)) becomes the original problem of finding an optimal set of pairings.

### 4. Column Generation Approach

Compared with the straightforward method of enumerating all the feasible pairings, a column generation approach has an advantage that it provides an optimal solution to the LP (linear programming) relaxation SCP\(^*(P_{\text{all}})\) of SCP(\( P_{\text{all}} \)) by iteratively solving SCP\(^*(P)\) to optimality for subsets \( P \) whose sizes are relatively small. We call SCP\(^*(P)\) a master problem. For any subset \( P \), the cost of an optimal solution to SCP\(^*(P)\) can be reduced further by adding good feasible pairings to \( P \). Such pairings can be found by solving a problem called the pricing problem that is defined based on an optimal solution to the dual of SCP\(^*(P)\) for the current subset \( P \). The approach stops when no good pairing can be found to improve the current solution. At this moment, the current solution is proved to be optimal to SCP\(^*(P_{\text{all}})\). Furthermore, in view of the experience over the past researches, the solution obtained by solving SCP(\( P \)) is known to be relatively good to SCP(\( P_{\text{all}} \)). In this section, we propose an efficient algorithm for finding good pairings. We also focus on the initial pairing generation and regularity.

#### 4.1. Column Generation

We consider the LP relaxation problem SCP\(^*(P)\):

\[
\text{minimize} \quad \sum_{k \in P} c_k x_k \\
\text{subject to} \quad \sum_{k \in P} a_{ik} x_k \geq 1, \quad \forall i \in F_{\text{all}} \\
\quad \quad \quad \quad \quad \quad \quad 0 \leq x_k \leq 1, \quad \forall k \in P.
\]

Its dual problem DSCP\(^*(P)\) is formulated as follows:

\[
\text{maximize} \quad \sum_{i \in F_{\text{all}}} u_i \\
\text{subject to} \quad \sum_{i \in F_{\text{all}}} a_{ik} u_i \leq c_k, \quad \forall k \in P \\
\quad \quad \quad \quad \quad \quad \quad u_i \geq 0, \quad \forall i \in F_{\text{all}}.
\]

We iteratively solve this LP problem with its dual problem. Denoting an optimal solution to DSCP\(^*(P)\) as \( u^* \), the pricing problem PRICE(\( u^* \)) to find a pairing \( k \) from \( P_{\text{all}} \) to be added to \( P \) can be defined as follows:

\[
\max_{i \in F_{\text{all}}} \frac{1}{c_k} \sum_{i \in F_{\text{all}}} a_{ik} u^*_i.
\]

Let \( \sigma(u^*) \) be the optimal value of PRICE(\( u^* \)) and let \( k^* \) be an optimal solution. If \( \sigma(u^*) > 1 \) holds, then the optimal value of DSCP\(^*(P)\) can be updated by adding pairing \( k^* \) to \( P \). The process is iterated until \( \sigma(u^*) \leq 1 \) holds.

#### 4.2. Graph Description

The problem of finding a sequence of flight legs can be formulated as a routing problem in digraphs, where the flight legs are associated to nodes. We link two nodes \((i, j)\) with a directed edge if the following two conditions are satisfied.

**Condition 1:** The arrival airport of flight leg \( i \) coincides with the departure airport of \( j \).
Condition 2: The departure time of flight leg $j$ is at least 30 minutes later than the arrival time of $i$.

For verifying the maximum person-day and reducing the computation time in column generation, we generate several subgraphs for one instance. We define $G(p, q)$ as the subgraph corresponding to the period from the $p$th day to the $(p - 1 + q)$th day for all $p \in \{1, 2, \ldots, N_d - q + 1\}$ and $q \in \{1, 2, \ldots, N_{\text{max}}\}$, where $N_d$ is the total days of the instance. Therefore, each instance has $(2N_d - N_{\text{max}} + 1)N_{\text{max}}/2$ subgraphs. For example, for instance I whose number of days is 17, we first create 17 subgraphs for each day involving only those flight legs whose departure and arrival times are both in this day. Then, for every pair of consecutive two days, we apply a similar rule to generate 16 graphs. Similar rules are applied to the cases of $3, 4, \ldots, N_{\text{max}}$ consecutive days. As a result, we obtain 75 subgraphs in total if $N_{\text{max}} = 5$.

For each subgraph, we connect a source node $s$ to the flight legs whose departure is on the first day in Tokyo. A sink node $t$ is linked from the flight legs whose arrival is on the last day in Tokyo. Finally, we remove the nodes that are not reachable from $s$ and those not reachable to $t$, because such nodes are not necessary.

Note that all the subgraphs are directed acyclic graphs (DAG) and any path from $s$ to $t$ represents a pairing with $q$ person-days. Although such an $s$–$t$ path generated from these subgraphs is not necessarily feasible, it must satisfy all the basic constraints.

With these subgraphs, the pricing problem $\text{PRICE}(u^*)$ can be decomposed into subproblems $\text{PRICE}(u^*, p, q)$ for all the subgraphs $G(p, q)$, where $\text{PRICE}(u^*, p, q)$ is the pricing problem to find a pairing that maximizes $\left(1/c_k\right) \sum_{i \in F_{pq}} a_{ik} u_{ik}^*$ among those that correspond to $s$–$t$ paths in $G(p, q)$. Because $c_k$ is the same for all such pairings $k$ (to be more precise, $c_k = q$ holds for every pairing $k$ that corresponds to an $s$–$t$ path in $G(p, q)$), this problem becomes a constrained longest path problem in directed acyclic digraphs $G(p, q)$. Let $\sigma(u^*, p, q)$ be the optimal solution value of $\text{PRICE}(u^*, p, q)$ and $k(u^*, p, q)$ be an optimal solution. If there is a graph $G(p, q)$ that satisfies $\sigma(u^*, p, q) > 1$, then the optimal value of $\text{DSCP}^*(P)$ can be updated by adding pairing $k(u^*, p, q)$ to $P$. Otherwise, i.e.,

$$\sigma(u^*, p, q) \leq 1$$

holds for all the subgraphs $G(p, q)$, we terminate the column generation.

In the column generation phase, we adopt the following strategy. We examine subgraphs $G(p, q)$ one by one for possible pairs of $p$ and $q$, solving the corresponding pricing problem, and whenever a pairing that violates (8) is found by solving the pricing problem $\text{PRICE}(u^*)$ for a subgraph $G(p, q)$, we add such a pairing to $P$ and immediately start a new iteration (i.e., we stop examining the remaining subgraphs and immediately move to the phase of solving $\text{SCP}^*(P)$ for the updated $P$).

### 4.3. Initial Pairing Set Generation

The set covering problem $\text{SCP}(P)$ is feasible only when the initial pairing set $P$ can cover all the non-DH flight legs. Our initial pairing set generation method to generate such a $P$ starts from $P = \emptyset$ and consists of two steps.

In the first step, we consider an iterative process based on depth first search (DFS). For each subgraph, we define $Q$ as the set of all the nodes directly connected from source node $s$ and execute a DFS from each node in $Q$. In choosing the next candidate node in DFS, we divide the unvisited nodes that are connected from the current node into two sets: currently uncovered nodes and covered nodes, where a node is covered if there is a pairing in $P$ containing the flight leg corresponding to the node and covered otherwise. If the set of uncovered nodes is not empty, we choose from this set the node whose departure time is closest to the arrival time of the current node; otherwise, we choose such a node from the set of covered nodes. Whenever the DFS reaches a node that is connected to the sink $t$, it checks if the current path from $s$ to $t$ satisfies the pairing constraints explained in Section 2.1, and if it does, we add the obtained $s$–$t$ path into $P$, terminate the current DFS, and start a new DFS from another node in $Q$ that has not been used as the starting node of DFS. Whenever we start a DFS from a node in $Q$, all the nodes are labeled unvisited, i.e., the new DFS can visit those nodes that have been visited by a former DFS. A set of such calls to DFS for a subgraph, which we call a DFS probe, comes to an end when DFS has been executed from every node in $Q$. We repeat the process of invoking DFS probes to all subgraphs, where one round consists of DFS probes with a DFS probe to every subgraph, and such rounds are repeated until no more uncovered nodes have become covered in a round. In each round, the time complexity for one graph is $O(|Q|E_{G(p,q)})$, where $E_{G(p,q)}$ denotes the number of edges in graph $G(p, q)$.

Even this simple approach works effectively for the tested instances as shown in Table 2. The first and the second columns express the instance name (“Instance”) and the number of non-DH nodes (“#Non-DH”) for each instance. Each value in the third column (“#Pairing”) shows the number of pairings in the obtained set $P$, and the fourth column (“#Round”) shows the number of rounds executed in the first step. The last two columns represent the number of uncov-
will be covered by such a path. This feasible path can be obtained by tracking back the DP cells through the path to \( h \). Recall that all nodes are reachable from \( s \) and to \( t \) in \( G(p, q) \), and the same also holds for \( G_i(p, q) \). This ensures that any \( s\rightarrow t \) path in \( G(p, q) \) must contain node \( i \). Furthermore, it is also clear that any \( s\rightarrow t \) path in \( G(p, q) \) satisfies the basic constraints in Section 2.1. Hence, our problem reduces to the problem of finding an \( s\rightarrow t \) path that satisfies all the specific pairing constraints.

We consider a dynamic programming (DP) for solving this problem. For each node \( h \) in graph \( G(p, q) \), we maintain a matrix \((w_{lj})\) where the value \( w_{lj}^h \) of the \((l, f)\)-element represents the minimum working time in a path among all feasible paths from source node \( s \) to \( h \) such that (after the latest sleep period if such a period exists) the number of landings is at most \( l \) and the total flying time is at most \( f \). Note that the matrix size is \( N_{\text{max}} \times N_{\text{max}} \), because of Specific Constraint 1 and 3. Denote by \( w_{(j)h} \) the working time during the period between flight leg \( j \) and \( h \), and if \( j = s \) or \( h = t \), let \( w_{(j)h} = 0 \). Similarly, we define \( w_h \) as the working time of flight leg \( h \), and if \( h = s \) or \( h = t \), let \( w_h = 0 \). Let all the values \( w_{lj}^h \) for node \( s \) be 0. Then the cells for all other nodes can be computed by a forward programming:

\[
\begin{align*}
 w_{lj}^h &= \begin{cases} 
 \min_{j \in V_h} w_{lj}^j + w_{(j)h} + w_h & \text{if min}_{j \in V_h} w_{lj}^j + w_{(j)h} \leq N_{\text{max}} \text{ and } (j, h) \text{ is not a sleep period} \\
 w_h & \text{if min}_{j \in V_h} w_{lj}^j + w_{(j)h} \leq N_{\text{max}} \text{ and } (j, h) \text{ is a sleep period} \\
 N_{\text{max}} + 1 & \text{otherwise}
\end{cases}
\end{align*}
\]

The value of the \((N_{\text{max}}, N_{\text{max}})\)-element at the sink node \( t \) indicates whether a feasible path exists in the current subgraph. If its value is lower than or equals to the maximum working time \( N_{\text{max}} \), a feasible path exists, and the target uncovered node \( i \) will be covered by such a path. This feasible path can be obtained by tracking back the DP cells through the path starting from the \((N_{\text{max}}, N_{\text{max}})\)-element at the sink node \( t \). The path generated with this procedure must be feasible, since this DP approach respects all the constraints.

The time complexity of this DP is \( O(N_{\text{max}}N_{\text{max}}N_{G(p,q)}E_{G(p,q)}) \), where \( N_{G(p,q)} \) and \( E_{G(p,q)} \) denote the number of nodes and edges in graph \( G(p, q) \). In practice, the computation time can further be reduced if the input values are multiples of an integer. For example, for the tested instances I to IV, all the time-related input, including flying time, is divisible by 5, and \( N_{\text{max}} \) can be reduced from the original value 720 to 144, which reduces the computational cost to one fifth.

We apply this scheme to every subgraph \( G(p, q) \) until a valid path is found. The instance is proved to be infeasible, if no feasible path is found for a target node \( i \).

### 4.4. Lower Bound

For large-scale instances, it becomes hard for the column generation approach to continue the search until all the necessary columns are generated, i.e., until the termination condition (8) is satisfied for all subgraphs \( G(p, q) \). In this section, we consider a lower bound on the optimal value of \( \text{SCP}^*(P_{\text{all}}) \) that can be obtained by solving the pricing problems \( \text{PRICE}(u^*, p, q) \) for all \( p \) and \( q \) even if the termination condition (8) is not satisfied in the current iteration.

Recall that \( P \) is a subset of \( P_{\text{all}} \), and \( u^* \) is an optimal solution to \( \text{DSCP}^*(P) \), and moreover, \( \sigma(u^*, p, q) \) denotes the optimal value of \( \text{PRICE}(u^*, p, q) \). If \( \sigma(u^*, p, q) \leq 1 \) holds for all the subgraphs \( G(p, q), \) \( \text{SCP}^*(P_{\text{all}}) \) has been exactly solved and the optimal value is \( \sum_{e \in F_{\text{all}}} u^*_e \) due to the duality theorem. Next we consider the situation where the termination condition (8) is not satisfied, i.e.,

\[
\sigma(u^*, p, q) > 1
\]  

\[(9)\]
holds for some $G(p,q)$. Recall that the flight legs are associated to nodes, and let $i$ also denote the node corresponding to flight $i$. Denote by $V(p,q)$ the node set of $G(p,q)$, and define $R(i)$ to be the set of subgraphs that contain flight leg $i$, i.e.,

$$R(i) = \{ G(p,q) \mid i \in V(p,q) \}.$$ 

Let $\Delta_i$ be defined as follows:

$$\Delta_i = \min_{G(p,q) \in R(i)} \frac{1}{\sigma(u^*, p, q)} \quad \forall i \in F_{\text{all}}.$$ 

We denote by $P_{\text{all}}^{(p,q)}$ the set of all feasible pairings ($s$–$t$ paths) in subgraph $G(p,q)$ and define $\hat{u}_i = \Delta_i u_i^*$ for each flight leg $i$. Then $\hat{u}$ is a feasible solution of DSCP$(P_{\text{all}})$ because the following inequality holds for all the subgraphs $G(p,q)$:

$$\sigma(\hat{u}, p, q) = \max_{k \in P_{\text{all}}^{(p,q)}} \frac{1}{c_k} \sum_{i \in F_{\text{all}}} a_k \Delta_i u_i^*$$

$$= \max_{k \in P_{\text{all}}^{(p,q)}} \frac{1}{c_k} \sum_{i \in F_{\text{all}}} G(p,q) \in R(i) \frac{1}{\sigma(u^*, p', q')} a_k u_i^*$$

$$\leq \max_{k \in P_{\text{all}}^{(p,q)}} \frac{1}{c_k} \sum_{i \in F_{\text{all}}} \frac{1}{\sigma(u^*, p, q)} a_k u_i^*$$

$$= \frac{1}{\sigma(u^*, p, q)} \max_{k \in P_{\text{all}}^{(p,q)}} \frac{1}{c_k} \sum_{i \in F_{\text{all}}} a_k u_i^* = 1.$$ 

The inequality from (11) to (12) holds for the following reason. For every $i$ and $k$, if $a_k$ equals zero, the term corresponding to $i$ and $k$ equals zero both in (11) and (12). Otherwise (i.e., if $a_k$ equals one), it means that path $k$ contains node $i$. Accordingly, $G(p,q)$ is in set $R(i)$, which indicates that the term in (12) is a candidate among the minimize function range in (11), and the corresponding term in (11) is less than or equals to the one in (12).

As a result, the objective function value $\sum_{i \in F_{\text{all}}} \hat{u}_i$ gives a lower bound on the optimal value of SCP$(P_{\text{all}})$ as well as that of SCP$(P_{\text{all}})$.

### 4.5. DP-based Algorithm for Pricing Problem

We solve the pricing problem using a DP-based branch-and-bound algorithm. Before devising this method, we considered a DP approach similar to the one discussed in Section 4.3 by extending the matrix at each node to a 3-dimensional table. However, such an additional dimension causes a great increase in both computation time and memory. For this reason, and from preliminary experimental results, we decided to adopt a branch-and-bound framework using a relaxation of the above DP for bounding operations. Our method consists of two phases and can be outlined as follows.

In the first phase, it creates three independent DP lists for each node corresponding to the three constraints regarding working time, flying time and landings, respectively. In the DP list of node $j$ with respect to working time, the value $u_j^w(r)$ of the $r$th cell represents the maximum obtainable price along a path among all paths from node $j$ to the sink node such that the total working time (before the first sleep period if such a period exists) is at most $r$, where the price of a path is the sum of $u_i^*$ for all nodes $i$ on the path. Similarly, we prepare the DP lists $u_j^f(\cdot)$ and $u_j^l(\cdot)$ for both flying time and landings, respectively. Note that by reducing the 3-dimensional table into three independent lists, each value in the DP list only indicates an upper bound, since only the constraint associated with the list is guaranteed, and the path realizing the value in the list may violate one or both of the other two constraints. These kind of DP list cannot provide us with a feasible pairing, but with an upper bound on the price value, which is important for bounding operations.

In the second phase, we use a depth-first branch-and-bound search to generate an optimal path. The algorithm generates partial paths from $s$ by expanding the current path along an edge from the last node of the path or by backtracking whenever the current path becomes infeasible or it is concluded that it does not lead to a desirable path (i.e., a bounding operation). Suppose that the current path $\hat{k}$ is a partial path from source node $s$ to a node $i$ in a graph $G(p,q)$. Denote by $F(\hat{k})$ the node set of path $\hat{k}$, and we define $f_i^j$, $w_i^*$ and $l_i^j$ to be the flying time, working time and landing number of path $\hat{k}$ from the last sleep period to node $i$, or from $s$ to $i$ if $\hat{k}$ contains no sleep period. In expanding the current path $\hat{k}$ by appending a node $j$ at the end of $\hat{k}$, we choose a node whose departure time is earliest among those in $V_j^*$ that have not been examined yet as a candidate to append to the current path $\hat{k}$. If the path expanded by appending such a node $j$ is
feasible, we examine the following three conditions:

\[
\begin{align*}
\sum_{i \in F(k)} u^i + u^j(N_{\text{max}}) & \leq q \quad \text{if } (i, j) \text{ is a sleep period} \\
\sum_{i \in F(k)} u^i + u^j(N_{\text{max}} - f^k) & \leq q \quad \text{otherwise}; \\
\sum_{i \in F(k)} u^i + u^j(N_{\text{max}}) & \leq q \quad \text{if } (i, j) \text{ is a sleep period} \\
\sum_{i \in F(k)} u^i + u^j(N_{\text{max}} - w^k) & \leq q \quad \text{otherwise}; \\
\sum_{i \in F(k)} u^i + u^j(N_{\text{max}} - l^k) & \leq q \quad \text{otherwise}.
\end{align*}
\]

Recall that our objective is to find a path in \( G(p, q) \) that satisfies the inequality (9), which means that the total of prices \( u^i \) on the nodes in the path has to be greater than \( c_k = q \). If one of the above three conditions is satisfied, we can conclude that no path with total price greater than \( q \) can be generated by further extending from node \( j \) the current partial path \((k \rightarrow j)\), i.e., a bounding operation can be applied to node \( j \) and a backtracking follows.

The computational results of a simple tree search (that backtracks only if the current path becomes infeasible) and the proposed DP-based branch-and-bound method are shown in Table 3. We set the time limit to 7200 seconds for both algorithms. The computation time in seconds for solving the LP relaxation \( \text{SCP}'(P_{\text{all}}) \) by column generation (“Time”) and the objective values (“ObjectValue”) to \( \text{SCP}'(P) \) for the set \( P \) when the column generation terminated are shown in Table 3, where “t.l.” signifies that the time limit was reached before the column generation stopped with condition (8) satisfied for all graphs. The column “#Iteration” expresses the number of LP relaxation problems \( \text{SCP}'(P) \) that have been solved during the execution.

The branch-and-bound with DP has better performance for all instances.

### Table 3 The results of simple tree search and DP-based branch-and-bound method

<table>
<thead>
<tr>
<th>Instance</th>
<th>Simple tree search</th>
<th>B&amp;B with DP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#Iteration</td>
<td>Objective</td>
</tr>
<tr>
<td>I</td>
<td>151</td>
<td>228.0</td>
</tr>
<tr>
<td>II</td>
<td>342</td>
<td>749.0</td>
</tr>
<tr>
<td>III</td>
<td>273</td>
<td>1115.0</td>
</tr>
<tr>
<td>IV</td>
<td>53</td>
<td>5769.0</td>
</tr>
</tbody>
</table>

#### 4.6. Regularity

Most flight legs are regularly scheduled, e.g., a flight from an airport to another is scheduled with the same departure and arrival times for every weekday. This section considers a method to exploit such regularity. We call two pairings twins if for every corresponding pair of flight legs, the departure and arrival airports are the same, and the departure and arrival times for every weekday. This section considers a method to exploit such regularity. We call two pairings twins if for every corresponding pair of flight legs, the departure and arrival airports are the same, and the departure and arrival times are the same but not on the same day, where the intervals between the two corresponding flight legs are the same for all pairs. When the column generation obtains a valid good pairing to be added to \( P \), all of its twin pairings are also added to \( P \).

We compared the computation time of the cases where the method of adding twins is adopted or not. The results for the case without this idea are shown in Table 4, and Table 5 shows the results when the idea is incorporated.

### Table 4 The results of the algorithm without twins

<table>
<thead>
<tr>
<th>Instance</th>
<th>#Iteration</th>
<th>#Pairing</th>
<th>Objective</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>3135</td>
<td>3201</td>
<td>92.8</td>
<td>56</td>
</tr>
<tr>
<td>II</td>
<td>23309</td>
<td>23560</td>
<td>247.4</td>
<td>t.l.</td>
</tr>
<tr>
<td>III</td>
<td>14688</td>
<td>14947</td>
<td>382.4</td>
<td>t.l.</td>
</tr>
<tr>
<td>IV</td>
<td>3679</td>
<td>4994</td>
<td>2433.3</td>
<td>t.l.</td>
</tr>
</tbody>
</table>

We can observe from the computational results that this idea improves the speed of column generation for all the instances. This might be because the bottleneck of our approach for these instances is the time to find a good pairing, and it is advantageous to add more than one pairing at each iteration. Note however that more than 20,000 twin pairings were generated only in 653 iterations for instance III, and a different conclusion might be drawn for larger instances if the time horizon becomes much longer.
Table 5 The results of the algorithm with twins

<table>
<thead>
<tr>
<th>Instance</th>
<th>#Iteration</th>
<th>#Pairing</th>
<th>ObjectValue</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>878</td>
<td>8228</td>
<td>92.8</td>
<td>37</td>
</tr>
<tr>
<td>II</td>
<td>10944</td>
<td>97582</td>
<td>247.2</td>
<td>5697</td>
</tr>
<tr>
<td>III</td>
<td>653</td>
<td>22064</td>
<td>380.8</td>
<td>1140</td>
</tr>
<tr>
<td>IV</td>
<td>821</td>
<td>9042</td>
<td>2238.1</td>
<td>t.l.</td>
</tr>
</tbody>
</table>

5. SCP Heuristic Approach

The column generation approach stops the iteration if no good pairing can be generated to improve the SCP*(P), and if this stopping criterion is satisfied, the LP relaxation of the original SCP, SCP*(P_all), has been solved to optimality. However, the obtained solution to SCP*(P) can have fractional elements.

As the last step, we solve the SCP in its integer programming (IP) formulation described in Section 3. The SCP is known to be NP-hard, and many good exact and heuristic algorithms have been proposed. In this paper, we consider two approaches: One is to solve this integer programming problem by using an IP solver, IBM ILOG CPLEX, and the other is to solve it with a heuristic approach based on a 3-flip neighborhood local search algorithm (3FNLS) proposed by Yagiura et al. (2006). If sufficient time and memory were available, CPLEX would be able to solve SCP(P) to optimality. However, the 3FNLS obtained better results for all instances under a time limit of one hour. Table 6 shows the results of solving SCP(P) by CPLEX and 3FNLS, where P is obtained by the proposed column generation approach.

Table 6 The objective values of SCP(P) obtained by CPLEX and 3FNLS in one hour

<table>
<thead>
<tr>
<th></th>
<th>CPLEX</th>
<th>3FNLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>97</td>
<td>96</td>
</tr>
<tr>
<td>II</td>
<td>288</td>
<td>255</td>
</tr>
<tr>
<td>III</td>
<td>454</td>
<td>406</td>
</tr>
<tr>
<td>IV</td>
<td>2533</td>
<td>2456</td>
</tr>
</tbody>
</table>

6. Computational Results

We now present computational results. The heuristic algorithms proposed for SCP by Yagiura et al. (2006) were coded in C, and we set a time limit of 3600 seconds.

6.1. Results of the Proposed Approach

Table 7 shows the results of the proposed column generation approach, where the columns show the number of pairings generated by the algorithm (“#Pairing”), the computation time in seconds for solving the LP relaxation by column generation (“Time (LP)”) and the objective values (“Value (LP)”) to SCP*(P). The objective values obtained by solving the resulting SCP(P) by the 3FNLS are listed in the last column (“Value (SCP)”). As discussed in Section 4, the rounded up value of the objective value of LP relaxation, when the column generation stops normally with the termination condition (8), gives a lower bound on the optimal value of SCP(P_all), which are shown in column “LB” in the table. Note that for instance IV, the column generation was stopped with a time limit of 7200 seconds (denoted “t.l.” in the table), and the value in column “LB” shows a lower bound obtained by the method proposed in Section 4.4. We can observe that the gap between the objective value of SCP (as IP) and its lower bound is within 10% for all the instances except IV.

Table 7 Computation times in seconds of our set covering approach

<table>
<thead>
<tr>
<th>Instance</th>
<th>#Pairing</th>
<th>Time (LP)</th>
<th>Value (LP)</th>
<th>LB</th>
<th>Value (SCP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>8228</td>
<td>37</td>
<td>92.8</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>II</td>
<td>97582</td>
<td>5897</td>
<td>247.2</td>
<td>248</td>
<td>255</td>
</tr>
<tr>
<td>III</td>
<td>22064</td>
<td>1140</td>
<td>380.8</td>
<td>381</td>
<td>406</td>
</tr>
<tr>
<td>IV</td>
<td>8899</td>
<td>t.l.</td>
<td>2267.5</td>
<td>837</td>
<td>2456</td>
</tr>
</tbody>
</table>

We also tested an existing software module developed by a company and designed for solving (almost) the same problem. We applied it to the same instances and analyzed the two approaches from various aspects, including some criteria that are not explicitly considered in our formulation. It was observed that solutions obtained by our approach have less number of total person-days and lower hotel expenses for crew members. On the other hand, the software obtained solutions with smaller number of deadheads and with more regularity.
6.2. Maximum Person-Days

Recall that we defined the maximum value of person-days $N_{p_{\text{max}}}$ in Section 2.1. This input parameter restricts the search space $P_{\text{all}}$ of pricing problem $\text{PRICE}(u^*)$. Let $P_{\text{all}}^\alpha$ be the set of all feasible pairings under the condition $N_{p_{\text{max}}} = \alpha$. We can easily prove the following lemma from the fact that $\alpha > \beta$ implies $P_{\text{all}}^\alpha \subseteq P_{\text{all}}^\beta$.

**Lemma 6.1** If $\alpha > \beta$ holds, the optimal value of $\text{SCP}^*(P_{\text{all}}^\alpha)$ is not more than that of $\text{SCP}^*(P_{\text{all}}^\beta)$.

The same result also holds for $\text{SCP}(P_{\text{all}}^\alpha)$ and $\text{SCP}(P_{\text{all}}^\beta)$.

The parameter $N_{p_{\text{max}}}$ is usually decided by practical reasons and is given as an input, but from computational point of view, it gives a trade-off between the quality of obtained solutions and computational cost in solving $\text{SCP}^*(P_{\text{all}})$ by our column generation approach. Intuitively, we might consider that if we use greater $N_{p_{\text{max}}}$, a better objective value of $\text{SCP}(P)$ can be obtained by a set $P$ generated by the proposed column generation provided that it stops normally with the termination condition (8) (no good pairing can be added to improve the $\text{SCP}^*(P)$). However, this is not always the case as observed through the experiments.

<table>
<thead>
<tr>
<th>$N_{p_{\text{max}}}$</th>
<th>#Pairing</th>
<th>Time (LP)</th>
<th>Value (LP)</th>
<th>LB Value (SCP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>1860</td>
<td>1</td>
<td>98.1</td>
<td>99</td>
</tr>
<tr>
<td>3</td>
<td>5567</td>
<td>11</td>
<td>93.2</td>
<td>94</td>
</tr>
<tr>
<td>4</td>
<td>8063</td>
<td>34</td>
<td>92.9</td>
<td>93</td>
</tr>
<tr>
<td>5</td>
<td>8228</td>
<td>37</td>
<td>92.8</td>
<td>93</td>
</tr>
<tr>
<td>6</td>
<td>11155</td>
<td>80</td>
<td>92.6</td>
<td>93</td>
</tr>
</tbody>
</table>

Table 8 Results of Instance I

<table>
<thead>
<tr>
<th>$N_{p_{\text{max}}}$</th>
<th>#Pairing</th>
<th>Time (LP)</th>
<th>Value (LP)</th>
<th>LB Value (SCP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>33466</td>
<td>1390</td>
<td>251.1</td>
<td>252</td>
</tr>
<tr>
<td>3</td>
<td>69679</td>
<td>3670</td>
<td>247.7</td>
<td>248</td>
</tr>
<tr>
<td>4</td>
<td>97582</td>
<td>5897</td>
<td>247.2</td>
<td>248</td>
</tr>
<tr>
<td>5</td>
<td>98736</td>
<td>t.l.</td>
<td>247.2</td>
<td>248</td>
</tr>
</tbody>
</table>

Table 9 Results of Instance II

<table>
<thead>
<tr>
<th>$N_{p_{\text{max}}}$</th>
<th>#Pairing</th>
<th>Time (LP)</th>
<th>Value (LP)</th>
<th>LB Value (SCP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>7945</td>
<td>43</td>
<td>407.6</td>
<td>408</td>
</tr>
<tr>
<td>3</td>
<td>20892</td>
<td>535</td>
<td>382.6</td>
<td>383</td>
</tr>
<tr>
<td>4</td>
<td>29303</td>
<td>1161</td>
<td>381.9</td>
<td>382</td>
</tr>
<tr>
<td>5</td>
<td>22064</td>
<td>1140</td>
<td>380.8</td>
<td>381</td>
</tr>
<tr>
<td>6</td>
<td>27655</td>
<td>1712</td>
<td>380.6</td>
<td>381</td>
</tr>
</tbody>
</table>

Table 10 Results of Instance III

<table>
<thead>
<tr>
<th>$N_{p_{\text{max}}}$</th>
<th>#Pairing</th>
<th>Time (LP)</th>
<th>Value (LP)</th>
<th>LB Value (SCP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>8899</td>
<td>t.l.</td>
<td>2267.5</td>
<td>837</td>
</tr>
<tr>
<td>3</td>
<td>10134</td>
<td>t.l.</td>
<td>2350.9</td>
<td>809</td>
</tr>
</tbody>
</table>

Table 8–11 show the computational results for each instance under different $N_{p_{\text{max}}}$ values. The methods and parameter settings are the same as the experiments shown in Table 7, as well as the meaning of each column. The mark “-” signifies that no feasible solution exists under the corresponding value of $N_{p_{\text{max}}}$. For all instances, when we only consider crew pairings with up to 2 person-days (i.e., when $N_{p_{\text{max}}} \leq 2$), the problem becomes infeasible or the results are much worse than other cases. The values in columns “Time (LP)” and “Value (LP)” in Table 8–10 show that there is a trade-off between the quality of obtained solutions and computational costs during the column generation process. However, the final results in “Value (SCP)” do not always decrease as the values of $N_{p_{\text{max}}}$ increase for some instances such as Instance
I and III, even though its LP relaxation problem SCP*(P_all) is solved to optimality by column generation. The results in Table 8–11 suggest that our column generation heuristic can lead to a better objective value of SCP(P) by applying it with a smaller value of \( N_{p_{\text{max}}} \) than its given value. Moreover, although the number of all feasible pairings can exponentially increase with \( N_{p_{\text{max}}} \), the total computation time does not grow with \( N_{p_{\text{max}}} \) so drastically. Hence, it might be worth trying to invoke the proposed algorithm iteratively with different values of \( N_{p_{\text{max}}} \), e.g., from 2 to the given maximum value of person-days.

7. Conclusion

We studied a crew pairing problem in which the objective is to minimize the total person-days subject to some basic and specific constraints. We modelled it as a set covering problem.

We presented a column generation approach that incorporates several ideas. We modelled the problem of finding a desirable pairing, in both initial pairing generation and pricing problem, as a problem of finding a longest path in a graph under some resource constraints. We proposed a two-step method for generating initial pairings, which consists of a depth first search, and a dynamic programming (DP) algorithm for covering target flight legs. We then proposed a DP-based branch-and-bound method in solving the pricing problem. This method outperformed a branch-and-bound algorithm without bounding operations by DP. We also considered regularity based on the fact that most flight legs are regularly scheduled. For large-scale instances, we presented a lower bound which can be obtained even if the column generation approach is stopped before the termination condition is satisfied.

Through computational experiments, we confirmed that the proposed approach successfully obtained good solutions for most of the tested instances with up to 10,000 flights.

References