Vol.2015-AL-151 No.3
2015/1/13

航空乗務員スケジューリング問題に対する列生成アプローチ

呉 偉[1,a)] 胡 艷楠[1,b)] 橋本 英樹[1,c)] 安藤 友人[2,d)] 白木 孝[2,e)] 柳浦 睦憲[1,f)]

Abstract:
In this paper, we consider the airline crew scheduling problem that calls for assigning the crew members in order to cover all the flights with the minimum total workdays 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 the real-world 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-cut method based upon a resource constrained dynamic programing for the column generation procedure. Computational results are given for a number of large-scale instances with up to 10,000 flights.

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 an airline crew scheduling problem 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 [1, 3].

For example, 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 [8].

Several approaches for the airline crew scheduling problem have been proposed in the literature, including exact algorithms such as tree search [2], branch-and-cut [6], as well as heuristic methods such as simulated annealing [4] and genetic algorithms [7].

In this paper, we model the problem as a set covering problem with costs of columns defined by the number of workdays, and present an efficient method to find promising columns through a graph representation that describes connections between flights, where the size of the graphs are kept small by using the cost structure. To solve this problem, we propose a branch-and-cut 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 current airline situation. In Section 3, we discuss a set covering model for this problem. In Section 4, a column generation approach is given, which utilizes the branch-and-cut framework and dynamic programing. Section 4 also proposes some ideas for solving this set covering problem (SCP) efficiently. We provide computational results and conclusion in Section 6.

2. Problem Description

First we give a few definitions for later description. A *flight leg* (sometimes also called *segment*) i is a single nonstop flight. A pairing k 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 an airport in order to cover a flight leg or to return to the departure city (crew base) at the end of a pairing.

Airline scheduling usually consists of five planning stages [5]. The last two stages, crew pairing and rostering, are usually referred to as crew scheduling problem.

As the input data of crew scheduling problem, a schedule consisting of all flight legs or segments 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 prob-

[1] Department of Computer Science and Mathematical Informatics, Graduate School of Information Science, Nagoya University, Nagoya, Japan
[2] NEC Corporation
[a)] goi@co.cm.is.nagoya-u.ac.jp
[b)] yannanhu@nagoya-u.jp
[c)] hasimoto@nagoya-u.jp
[d)] t-andou@cq.jp.nec.com
[e)] t-shiraki@bu.jp.nec.com
[f)] yagiura@nagoya-u.jp

lem, we define the number of workdays as the cost of a pairing. 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 Condition 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 Condition 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 Condition 3: The duration of a pairing must not exceed a specified limit on the number of workdays, usually 4-6 days. In our model, we assume that each crew member is unable to work more than 5 days, which means that the maximum number of workdays in a pairing is 5. Note that in our formulation, the cost of a pairing is defined to be the number of workdays, and hence each pairing has a cost of at most 5. 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 adjacent flight legs in a pairing, and t has to be at least 30 minutes as discussed above. A *break period t* 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_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 hours 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 a part of working time. The maximum working time w_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_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_{all} to be the set of all feasible pairings.

We define three additional constraints as follows:

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

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

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

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

A solution to the 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

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 half-month airline crew scheduling problem. For convenience, we assume that the time horizon of every instance from I, II and IV consists of 21 days with all the flight legs in the first and the last 3 days specified as deadhead. These deadheads in the beginning and ending help to cover the target flights in the middle core days. Instance III is a monthly flight data that is also sandwiched by 3 days with only deadheads before and after the core period. The information of the instances is shown in Table 1.

Table 1 Instance Information

Instance	#Days	#DummyDays	#AllFlights	#DeadHead
I	21	6	918	324
II	21	6	2203	651
III	36	6	2214	324
IV	21	6	11560	3400

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 will be discussed in Section 4. The second stage can be modeled as a set covering

or set partitioning problem. Since deadheads are allowed in our crew scheduling 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. This problem can be formulated as a set covering problem SCP(P) as follows:

$$\begin{aligned} \min \ & \sum_{k \in P} c_k x_k & \\ \text{s.t.} \ & \sum_{k \in P} a_{ik} x_k \geq 1, & \forall i \in F_{\text{all}} \\ & x_k \in \{0,1\}, & \forall k \in P, \end{aligned}$$

where P is a subset of P_{all} and F_{all} is the set of all non-deadhead flight legs. The binary decision variable x_k is a 0-1 variable associated with the kth pairing. If the pairing k is selected, then $x_k = 1$, and otherwise $x_k = 0$. The binary value a_{ik} equals 1 when pairing k covers flight leg i, otherwise, $a_{ik} = 0$. The cost c_j is the number of workdays in pairing j. When $P = P_{\text{all}}$ holds, the problem SCP(P_{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_{all}) of SCP(P_{all}) by iteratively solving SCP*(P) to optimality for subsets P whose sizes are relatively small. We call an 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_{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 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 an 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 workday 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 pth day to the $(p + q - 1)$st day for all $p \in \{1, 2, \ldots, N_d - q + 1\}$ and $q \in \{1, 2, \ldots, 5\}$, where N_d is the total days of the instance. Therefore, each instance has $(N_d - 2) \times 5$ subgraphs. For example, for instance I whose number of days is 21, we first create 21 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 20 graphs. Similar rules are applied to the cases of three, four and five consecutive days. As a result, we obtain 95 subgraphs in total.

For each subgraph, we connect a source node s to the flight legs whose departure city is Tokyo. A sink node t is linked from the flight legs whose arrival city is Tokyo.

Note that all the subgraphs are directed acyclic graphs (DAG) and any path starting from s to t represents a pairing. Although such an s-t path is not necessarily feasible, every such path generated from these subgraphs satisfies all the basic constraints.

4.2 Initial Pairing Set Generation

The set covering problem SCP(P) is available only when the initial pairing set P can cover all the non-DH flight legs. A warm start is very important to a column generation approach. Our initial pairing set generation starts from $P = \emptyset$ and consists of two steps.

The first step is that, for each subgraph, we define Q as the set of all the nodes directly connected from source node s and execute a depth first search (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 lege corresponding to the node and uncovered 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. The first step comes to an end when we a DFS has been executed from every node in Q. The time complexity for one graph is $O(|Q|E_{G(p,q)})$, where $E_{G(p,q)}$ denotes the the number of edges in graph $G(p, q)$.

Even this simple approach works effectively for instances as shown in Table 2. The first line represents the coverage (i.e., the ratio of nodes that are covered by the obtained set P) for each instance and each value in the second line shows the number of non-DH nodes that remain uncovered after the first step.

Table 2 Node Coverage by the Path Generation with DFS

Instance	I	II	III	IV
Node Coverage	94.95%	99.48%	97.20%	96.14%
#Uncovered Nodes	30	18	53	315

For those non-DH nodes that remain uncovered, the second step follows. Assume that node i is uncovered after the first step. For every graph $G(\cdot, \cdot)$, we generate a subgraph $K(i)$ induced by the set of all nodes reachable to or from node i. (Here we assume that every node in the graph is reachable from s and to t, because otherwise such a node is not necessary.) This ensures that any s-t path in $K(i)$ must cross node i. Furthermore, it is also clear that

any *s*-*t* path in $K(i)$ satisfies the basic constraints in Section 2.1. Hence, our problem reduces to the problem of finding an *s*-*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 $K(i)$, we maintain a matrix (w_{lf}) where the value w_{lf} 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 $N_{l_{\max}} \times N_{f_{\max}}$ because of Constraint 1 and 3. All the cells can be computed by a forward programming, and the value of the $(N_{l_{\max}}, N_{f_{\max}})$-element at the sink node t indicates if a feasible path exists in the current subgraph. If its value is lower than or equals to the maximum working time, a feasible path exists, and we can conclude that 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 $(N_{l_{\max}}, N_{f_{\max}})$-element at the sink node. The path generated with this procedure must be feasible, since this DP approach respects all the constraints.

The time complexity of this DP is $O(N_{l_{\max}} N_{f_{\max}} N_{K(i)} E_{K(i)})$, where $N_{K(i)}$ and $E_{K(i)}$ denote the number of nodes and edges in graph $K(i)$.

Since all the time-related input, including flying time, are divisible by 5, $N_{f_{\max}}$ can be shortened from the original value 720 to 144, which reduces leads also the computational cost to one fifth.

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

4.3 Column Generation

The SCP is known to be NP-hard. However, in a column generation approach, the SCP(P) associated with a pairing set P has to be solved in each iteration. We consider solving these master problems as their LP relaxation SCP*(P):

$$\begin{aligned} \min \ & \sum_{k \in P} c_k x_k & \\ \text{s.t.} \ & \sum_{k \in P} a_{ik} x_k \geq 1, & \forall i \in F_{\text{all}} \\ & 0 \leq x_k \leq 1, & \forall k \in P. \end{aligned}$$

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

$$\begin{aligned} \max \ & \sum_{i \in F_{\text{all}}} u_i & \\ \text{s.t.} \ & \sum_{i \in F_{\text{all}}} a_{ik} u_i \leq c_k, & \forall k \in P \\ & u_i \geq 0, & \forall i \in F_{\text{all}}. \end{aligned}$$

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 construct a pairing becomes a constrained shortest path problem in acyclic digraphs $G(\cdot, \cdot)$:

$$\max_{k \in P_{\text{all}}} \sum_{i \in F(k)} u_i^*,$$

where $F(k)$ is defied as the set of flight legs included in pairing k.

We solve this problem using a DP based branch-and-bound algorithm. We can use the DP similar to the one discussed in Section 4.2 by extending the matrix to a 3-dimensional table. However, we need to add one dimension and this can cause a great increase in computation time. If we add working time as a new dimension, the computation time will be increased by 162 times for each pricing problem. For this reason, and from preliminary experimental results, we decided to adopt a branch-and-bound framework using a relaxation of the above DP. 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. In the DP list of node j with respect to working time, the value $u_j^w(r)$ of the rth 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. Similarly, we prepare the DP lists $u_j^f(\cdot)$, $u_j^l(\cdot)$ 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 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 branch-and-bound framework. The algorithm generates partial paths from s by expanding the current path along an edge from the last node of the current path or by backtracking whenever the current path becomes infeasible or is concluded that it does not lead to a desirable path (i.e., a bounding operation). Suppose that the current path $\hat{k}$ is from source node s to a node i in a graph $G(\cdot, \alpha)$. For the next node j, we examine the following three conditions:

$$\begin{cases} \sum_{i \in F(\hat{k})} u_i^* + u_j^f(N_{f_{\max}}) < \alpha & \text{if edge } (i, j) \text{ is a sleep period} \\ \sum_{i \in \hat{k}} u_i^* + u_j^f(N_{f_{\max}} - f_{\hat{k}}) < \alpha & \text{otherwise;} \end{cases}$$

$$\begin{cases} \sum_{i \in \hat{k}} u_i^* + u_j^w(N_{w_{\max}}) < \alpha & \text{if edge } (i, j) \text{ is a sleep period} \\ \sum_{i \in \hat{k}} u_i^* + u_j^w(N_{w_{\max}} - w_{\hat{k}} - w_{(ij)}) < \alpha & \text{otherwise;} \end{cases}$$

$$\begin{cases} \sum_{i \in \hat{k}} u_i^* + u_j^l(N_{l_{\max}}) < \alpha & \text{if edge } (i, j) \text{ is a sleep period} \\ \sum_{i \in \hat{k}} u_i^* + u_j^l(N_{l_{\max}} - l_{\hat{k}}) < \alpha & \text{otherwise,} \end{cases}$$

where $w_{(ij)}$ is defined as the working time during the period between flight leg i and j, and $f_{\hat{k}}$, $w_{\hat{k}}$ and $l_{\hat{k}}$ denote the maximum flying time, working time and landing number of a pairing $\hat{k}$ as the definition in Section 2.1, and the only difference is that the pairing $\hat{k}$ is still under generation. If one of the above three conditions is satisfied, a bounding operation is applied to node j and a backtracking follows.

The computation times 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 4. We set the time limit to 7200 seconds for both algorithms. The branch-and-bound with DP has better performance for all instances.

Table 3 Comparison between simple tree search and DP-based branch-and-bound

Instance	I	II	III	IV
simple tree search	313	>7200	>7200	>7200
B&B with DP	181	6062	2496	>7200

4.4 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 arrive 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 times of the cases where the method of adding twins is adopted or not. The results are shown in Table 4 in seconds. The first line shows the time for the case without this idea, and the second line shows the results when the idea is incorporated in the column generation.

Table 4 The result of adding twins

Instance	I	II	III	IV
Without the addition of twins	181	6062	2496	>7200
With the addition of twins	85	8341	2608	>7200

From the computational point of view, this idea improves the speed of column generation for the smallest instance. This might be because the bottleneck of our approach for this instance is the time to find a good pairing, and it is advantageous to add more than one pairing at each iteration. On the other hand, it increases the computation time when it is applied to large instances. The reason might be that the additional twin pairings increase the size of SCP$^*(P)$, and the time to solve the resulting LP problems becomes the bottleneck.

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$^*(P_{\text{all}})$, 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 heuristic algorithms have been proposed. In this paper, we solve this integer programming problem by using a 3-flip neighborhood local search algorithm [9].

6. Computational Results and Conclusion

We now present computational results. The heuristic algorithms proposed for SCP in [9] were coded in C, and the other algorithms proposed in this paper were all coded in C++. All of them were built and compiled under Microsoft Visual Studio 2010. We used IBM ILOG CPLEX, version 12.4 for solving the LP relaxation of SCP. All experiments were carried out on a PC with two Core i5-2450M at 2.50 GHz and 4 GB RAM memory under Windows 7 operating system, where the computation was executed on a single core. For the 3-flip neighborhood local search, we set a time limit of 3600 seconds.

The result for all the instances are presented in Table 5. Computation times in Table 5 are expressed in seconds. As discussed in Section 4, the objective function value of LP relaxation, when the column generation stops normally with the stopping criterion, gives a lower bound on the optimal value of SCP(P_{all}). 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.

Table 5 The Results of our Set Covering Approach

Instance	#FlightLeg	#Pairing	Time(LP)	Value(LP)	Value(IP)
I	918	29406	85	142.54	147
II	2203	137039	8341	405.65	424
III	2214	65465	2608	432.51	449
IV	11560	30009	>7200	3348.71	3670

We also tested a software module designed for solving (almost) the same problem. We applied it to an experiment on 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 workdays and lower hotel expenses for crew members. On the other hand, the software obtained solutions with smaller number of deadheads and with more regularity.

References

[1] R. Anbil, E.L. Johnson and R. Tanga, “A Gobal Approach to Crew Pairing Optimization,” IBM Systems Journal 31 (1991) 62–74.
[2] J.E. Beasley and B. Cao, “A Tree Search Algorithm for the Crew Scheduling Problem,” European Journal of Operational Research 94 (1996) 517–526.
[3] J. Barutt and T. Hull, “Airline Crew Scheduling: Supercomputers and Algorithms,” SIAM News 23 (1990) 20–22.
[4] T. Emden-Weinert and M. Proksch, “Best Practice Simulated Annealing for the Airline Crew Scheduling Problem,” Journal of Heuristics 4 (1999) 419–436.
[5] B. Gopalakrishnan and E.L. Johnson, “Airline Crew Scheduling: State-of-the-Art,” Annals of Operations Research 140 (2005) 305-—337.
[6] K. Hoffman and M. W. Padberg, “Solving Airline Crew Scheduling Problems by Branch-and-Cut,” Management Science 39 (1993) 657-—682.
[7] D. Levine, “Application of a Hybrid Genetic Algorithm to Airline Crew Scheduling,” Computers & Operations Research 23 (1996) 547–558.
[8] X. Onodera and A. Mori “Cockpit Crew Scheduling and Supporting,” Proceedings of the World Congress on Expert Systems, 1–10, New York: Pergamon, 1991.
[9] M. Yagiura, M. Kishida and T. Ibaraki “A 3-flip Neighborhood Local Search for the Set Covering Problem,” European Journal of Operational Research 172 (2006) 472–499.